

Computer Vision:

Stochastic Grammars for Parsing Objects, Scenes, and Events

Song-Chun Zhu, Siyuan Huang

October 12, 2021

Contents

Preface	vii
1 Introduction	1
1.1 Vision as Joint Parsing of Objects, Scenes, and Events	1
1.2 Unified Representation for Models, Algorithms, and Attributes	4
1.2.1 Three Families of Probabilistic Models	4
1.2.2 Dynamics of Three Inferential Channels	5
1.2.3 Three Attributes associated with each Node	6
1.3 Missing Themes in Popular Data-Driven Approaches	6
1.3.1 Top-Down Inference in Space and Time	6
1.3.2 Vision is to be a Continuous Computational Process	7
1.3.3 Resolving Ambiguities and Preserving Distinct Solutions	9
1.3.4 Vision is Driven by a Large Number of Tasks	11
1.4 Scope of Book: Compositional Patterns in the Middle Entropy Regime	11
1.4.1 Information Scaling and Three Entropy Regimes	11
1.4.2 Organization of Book	12
2 Overview of Stochastic Grammar	15
2.1 Grammar as a Universal Representation of Intelligence	15
2.2 An Empiricist's View of Grammars	16
2.3 The Formalism of Grammars	18
2.4 The Mathematical Structure of Grammars	19
2.5 Stochastic Grammar	21
2.6 Ambiguity and Overlapping Reusable Parts	23
2.7 Stochastic Grammar with Context	25
3 Spatial And-Or Graph	27
3.1 Three New Issues in Image Grammars in Contrast to Language	27
3.2 Visual Vocabulary	30
3.2.1 The Hierarchical Visual Vocabulary – the "Lego Land"	30
3.2.2 Image Primitives	30
3.2.3 Basic Geometric Groupings	32
3.2.4 Parts and Objects	33
3.3 Relations and configurations	36
3.3.1 Relations	36
3.3.2 Configurations	38

3.3.3	The Reconfigurable Graphs	40
3.4	Parse Graph for Objects and Scenes	41
3.5	Knowledge Representation with And-Or Graph	43
3.5.1	And-Or graph	43
3.5.2	Stochastic Models on the And-Or graph	47
3.6	Examples in Related Work	49
3.6.1	Probabilistic Geometric Grammars	49
3.6.2	Mixture of Deformable Part-based Models and Object Detection Grammars	49
3.6.3	Probabilistic Program Induction	50
3.6.4	Recursive Cortical Networks	51
4	Learning the And-Or Graph	53
4.1	Overview of the Learning Problem	53
4.1.1	Learning Parameters in Stochastic Context-Free Grammar	54
4.1.2	Probability Model for AOG	55
4.2	Learning Parameters in AOG	56
4.2.1	Maximum Likelihood Learning of Θ	57
4.2.2	Learning and Pursuing the Relation Set	58
4.2.3	Examples of Sampling and Synthesis from AOG	59
4.2.4	Summary of the Parameter Learning	60
4.3	Structure Learning: Block Pursuit and Graph Compression	62
4.3.1	Hybrid Image Templates (HIT) as Terminal Nodes	67
4.3.2	AOT: Reconfigurable Object Templates	70
4.3.3	Learning AOT from Images	71
4.3.4	Inference on AOTs	77
4.3.5	Example: The Synthesized 1D Text AOT Learning	79
4.4	Structure Learning: Unsupervised Structure Learning	83
4.4.1	Algorithm Framework	84
4.4.2	And-Or Fragments	84
4.5	Structure Learning: Pruning from Full Graph	87
4.5.1	General Framework	87
4.5.2	Example: Learning Image Tangram Model	88
5	Parsing Algorithms for Inference in And-Or Graphs	99
5.1	Classic Search and Parsing Algorithms	99
5.1.1	Heuristic Search in And-Graph, Or-Graph and And-Or-Graph	99
5.1.2	Bottom-Up Chart Parsing	118
5.1.3	Top-Down Earley Parser and Generalization	126
5.1.4	Inside-Outside Algorithm for Parsing and Learning	129
5.1.5	Figure of Merit Parsing	133
5.2	Scheduling Top-down and Bottom-up Processes for Object Parsing	145
5.2.1	Integrating α - β - γ Processes in Inference	147
5.2.2	Learning the α , β and γ Processes	158
5.3	Example I: Integrating the α , β and γ for Image Parsing	165
5.3.1	Experiment I: Evaluating Information Contributions of the α , β and γ Processes Individually	166

5.3.2	Experiment II: Object Parsing in a Greedy Pursuit Manner by Integrating the α , β and γ Processes	168
5.4	Example II: Recognition on Object Categories	169
6	Attributed And-Or Graph	176
6.1	Introduction of Attribute Grammar	176
6.2	Attributed Graph Grammar Model	177
6.3	Example I: Parsing the Perspective Man-made World	179
6.4	Example II: Single-View 3D Scene Reconstruction and Parsing	180
6.4.1	Attribute Hierarchy	182
6.4.2	Attribute Scene Grammar	186
6.4.3	Probabilistic Formulation for 3D Scene Parsing	187
6.4.4	Inference	191
6.5	Example III: Human-Centric Indoor Scene Synthesis Using Stochastic Grammar	193
6.5.1	Representation	193
6.5.2	Probabilistic Formulation	194
6.5.3	Synthesizing Scene Configurations	197
6.6	Example IV: Joint Parsing of Human Attributes, Parts and Pose	198
6.7	Summarization	202
7	Temporal And-Or Graph	204
7.1	Introduction	204
7.2	Atomic Action Models	205
7.2.1	2D HOI in Time - A Simplified Atomic Action Model	206
7.2.2	Modeling Human-Object Interaction in 3D and Time	208
7.2.3	Part-Level 3DHOI	213
7.2.4	Hand-Object Interaction	213
7.2.5	Concurrent HOI's and hoi's in STC-AOG	215
7.3	Event representation by T-AOG	216
7.3.1	The T-AOG for Events	216
7.3.2	Parse Graph	218
7.3.3	Example I: Synthesizing New Events by T-AoG	218
7.3.4	Example II: Group Activity Parsing by ST-AoG	220
7.4	Parsing with Event Grammars	225
7.4.1	Formulation of Event Parsing	225
7.4.2	Generating Parse Graphs of Single Events	227
7.4.3	Runtime Incremental Parsing	228
7.4.4	Generalized Earley Parser	228
7.4.5	Multi-agent Event Parsing	235
7.5	Learning the T-AoG	235
7.5.1	Information Projection	235
7.5.2	Block Pursuit on Data Matrix	237
7.5.3	Detecting Grounded Spatial Relations	239
7.5.4	Learning Atomic Actions	240
7.5.5	Learning Longer Events and T-AOG	241
7.5.6	Learning the Parameters of T-AOG	242
7.5.7	Example: Learning Event Grammars	243

8	Fluent and Causal And-Or Graph	245
8.1	Perceptual Causality and Fluent	245
8.1.1	What are Cause and Fluent?	245
8.1.2	Perceptual Causality	246
8.2	Vision and Causality	247
8.2.1	Converting Perceptual Causality to Heuristics	247
8.2.2	Assumptions and Structural Equation Models	248
8.2.3	Potential Effects: The Space of Fluent Changes	248
8.2.4	Potential Causes: The Space of Action Detections	248
8.3	Perceptual Causal Relations	248
8.3.1	Defining Perceptual Causal Relations	249
8.3.2	Preparing the Data: Creating Clips from the Video	249
8.3.3	Evaluating Causal Relations	250
8.4	Pursuit of the Causal Relations	250
8.4.1	Fitting the Causal Relation	252
8.4.2	Pursuing Causal Relations by Information Projection	253
8.4.3	Precise Selection of cr When Actions are Hierarchical	254
8.5	The Causal And-Or Graph	255
8.5.1	The Probability Model on the Causal And-Or Graph	257
8.6	C-AoG Inference and Reasoning	257
8.6.1	Reasoning over Time	260
8.6.2	Evaluating the Inference Model	262
8.7	Discussion	263
9	Joint Parsing	268
9.1	Introduction	268
9.2	Holistic ⁺⁺ 3D Scene understanding	270
9.2.1	Representation	271
9.2.2	Probabilistic Formulation	272
9.2.3	Joint Inference	273
9.3	From View Centered Parsing to Scene Centered Parsing	276
9.3.1	Representation	277
9.3.2	Bayesian Formulation	278
9.3.3	Inference	284
9.4	Joint Video and Text Parsing	285
9.4.1	Representation	285
9.4.2	Joint Parsing and Inference	287
9.5	Query Answering from Joint Parse Graphs	290
9.5.1	Answering Natural Language Queries	290
9.5.2	Visual Turing Test	291
10	Grounded Grammar Induction	294
10.1	Introduction	294
10.2	Grounded Grammar Induction	294
10.2.1	Compound PCFG for Language	295
10.2.2	Compound PCFG for Image	297
10.2.3	Joint Learning by Alignment	298

10.2.4 Qualitative Study	299
Index	300

Authors

Song-Chun Zhu received his Ph.D. degree in Computer Science from Harvard University in 1996 (advised by Dr. David Mumford). He is currently professor of Statistics and Computer Science, and director of the Center for Vision, Learning, Cognition, and Autonomy, at the University of California, Los Angeles. His research interest has been to pursue a common statistical framework for vision, and broadly intelligence. He has published over 300 papers in computer vision, statistical learning, cognition, and AI, and robot autonomy, and received a number of honors, including the David Marr Prize in 2003 for image parsing with Z. Tu *et al.*, the Marr Prize honorary nominations in 1999 for texture modeling and 2007 for object modeling with Dr. Yingnian Wu *et al.* As a junior faculty he received in 2001 the Sloan Fellow in Computer Science, NSF Career Award, and ONR Young Investigator Award. In 2008 he received the Aggarwal prize from the Intl Association of Pattern Recognition for “contributions to a unified foundation for visual pattern conceptualization, modeling, learning, and inference.” In 2013 he received the Helmholtz Test-of-time prize for a paper on image segmentation. He is a fellow of IEEE Computer Society since 2011. He has been the principal investigator leading several ONR MURI and DARPA teams working on scene and event understanding and cognitive robots under a unified mathematical framework. In late 2020, he returned to China to establish a non-profit organization – Beijing Institute for General Artificial Intelligence, and holds joint positions as chair professor at both Peking University and Tsinghua University.



Siyuan Huang received his Ph.D. degree in Statistics from the Department of Statistics at the University of California, Los Angeles (UCLA) in 2021 (advised by Dr. Song-Chun Zhu). His research interests lie in computer vision, machine learning, cognition, and robotics. He received the best paper award at ICML 2020 workshop on Bridge Between Perception and Reasoning. He is a co-organizer of 3D Scene Understanding for Vision, Graphics, and Robotics workshops at CVPR 2019-2021. He currently focuses on the problem of human-like holistic 3D scene understanding, which solves the problems in perception, interaction, learning, and reasoning.



Preface

“The highest activity a human being can attain is learning for understanding, because to understand is to be free.”

– Baruch Spinoza, *Philosopher*, 1632-1677.

This textbook is intended for researchers and graduate students in statistics, computer science, and engineering. It is based on the work of the author and his collaborators in the past 30 years and the materials have been taught in a graduate course in the Department of Statistics and the Department of Computer Science at the University of California, Los Angeles. It may be used as a reference in the fields of computer vision and pattern recognition, machine learning, graphics, robotics, and artificial intelligence.

Introducing the Book Series

The book series consists of three parts.

The first book covers David Marr’s paradigm and various underlying statistical models for vision. The mathematical foundations herein integrate three regimes of models (low-, mid-, and high-entropy regimes) and provide essential foundation for research in visual coding, recognition, cognition, and reasoning. Concepts in this book are first explained for understanding and then supported by findings in psychology and neuroscience, after which they are established by statistical models and further linked to research in other fields such as physics. A reader of this book will gain a unified, cross-disciplinary view of artificial intelligence research in vision and will accrue knowledge spanning from psychology to neuroscience to statistics.

The second book defines a stochastic grammar for parsing objects, scenes, and events, posing computer vision as a joint parsing problem. It summarizes research efforts over the past 20 years that have worked to extend King-Sun Fu’s paradigm of syntactic pattern recognition. Similar to David Marr, King-Sun Fu was a pioneer and influential figure in the vision and pattern recognition community. Specifically, Zhu and Mumford first published a long survey article or booklet on "A Stochastic Grammar of Images" in 2006 and advocated for compositional and grammatical models as a general framework for visual representation. During 2005-2020, Zhu and collaborators have extended the models to attribute spatial, temporal, and causal and-or graphs for parsing objects, scenes, and events, which are covered in the current book.

The third book discusses visual commonsense reasoning, including subjects such as functionality, physics, intentionality, causality, and values. The third book connects vision to cognition and artificial intelligence.

The author would like to thank many current and former Ph.D. students at UCLA, to name a few, Luyao Yuan, Baoxiong Jia, Qing Li, Sirui Xie, Feng Shi, Tengyu Liu, Shu Wang, Yuxin Qiu, Zeyu Zhang, Zilong Zheng, Chao Xu, Tianfu Wu for their contributions to this book, especially Jiangyong Huang for polishing the latex, Eric Fischer for editing chapters 1 and 2, and polishing the book.

1

Introduction

The emphasis of this book, the second in the series, is on image parsing and semantic understanding. Specifically, images may be parsed into objects, scenes, and events by a stochastic And-Or graph. This book follows the first in the series, which discusses statistical models for Marr’s paradigm, and precedes the third in the series, which discusses visual commonsense reasoning, including functionality, physics, intent, causality, and values.

This book expounds modeling, synthesis, analysis, and learning principles for image parsing. In regard to modeling, the models laid out here, although they may capture extremely unique visual phenomena, are all derived under a common mathematical framework. Models possess spatial (*e.g.*, objects and scenes), temporal (*e.g.*, actions, activities, and events), causal (*e.g.*, causal effects), and attribute (*e.g.*, appearance and geometry) features, all of which may be explained by a unified framework. In regard to synthesis, it may be accomplished by sampling from the models. To perform analysis, a parse graph may be inferred and the image interpreted; this process is referred to as “parsing”. To accomplish learning, models may learn from data in various learning modes, such as parameter learning or structure learning and pursuit.

1.1 Vision as Joint Parsing of Objects, Scenes, and Events

In this book, computer vision is posed as a problem of parsing images to derive semantic meaning. Images and videos are parsed in each of the spatial, temporal, and causal domains, so as to derive the maximal amount of semantic meaning from the raw input data. An image or video may be represented through all three of these domains using a spatial-temporal-causal And-Or graph (STC-AOG), discussed in later chapters.

Fig. 1.1 displays a formal interpretation of a natural image in the form of a simple parse graph. The nodes in a parse graph are organized in a tree-like hierarchical structure, which contains possible connections between sibling nodes. In “From Image Parsing to Text Generation” by Yao *et al.* [177], it is shown that the parse graph in Fig. 1.1 may actually be used to generate the following text description: “It is a scene with 2 people, 1 table, 2 chairs, vending machines, ground (paved), walls (red, brick), and vegetation (green). The person 1 (male) is carrying a backpack (red), walking on the ground towards the vending machines. The person 2 (male) is holding a (news)paper, sitting on chair 1.”

The text description of the image reveals that natural images portray more than just objects. The parsing of natural images and video will require, in more detail, the joint parsing of objects, scenes, and events within the image or video, which encompasses many possible states captured by various nodes in the parse graph. Overall, there may be several types of nodes in a parse graph, as displayed by Fig. 1.2, including scenes or objects, minds or intents, hidden objects, actions, imagined actions, attributes, and fluents, *i.e.*, how things

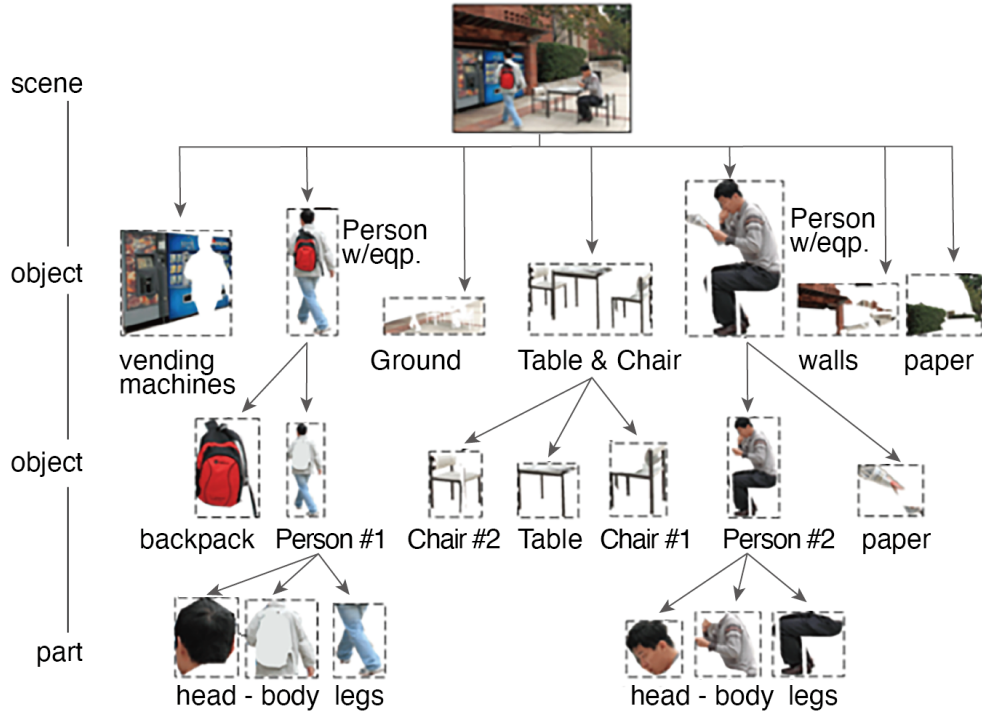


Figure 1.1: This parse graph of a natural image uses a hierarchical structure to represent the raw input data of the image in a semantically meaningful way. The image of a student walking to a vending machine, while another student reads a newspaper at a table, is decomposed into various object parts.

change over time. Understanding natural images such as Fig. 1.2 is complex and, in general, accomplished by performing joint parsing and subsequent higher-level cognitive reasoning from parsed image data. From a single image, a dense 3D scene may ultimately be reconstructed. Camera parameters, materials, and illumination may be estimated, and the scene may be parsed hierarchically in terms of attributes, fluents, relations, intents and beliefs of the relevant agents. Actions may be predicted, and invisible, hidden states may be recovered, such as the hidden state of water boiling within a metal kettle. Lastly, the functionalities of objects within the scene may also be determined, such as the functionality of a chair or table.

Fig. 1.3 further shows an example of joint parsing in three domains: spatial, temporal, and causal domains. In the spatial hierarchy, the scene, functional areas, objects, parts, and primitives are determined. In the temporal hierarchy, group activities, events, actions, and movements are determined. In the causal hierarchy, causal chains in the fluent space are determined. The parse graph in Fig. 1.3 may actually be used not only for generating text descriptions but also for answering human queries.

How does joint parsing (in the spatial, temporal, and causal dimensions) compare to other well-known goals for computer vision problems, *e.g.*, recognition and classification? In general, there are two types of objectives in visual inference. One objective is to seek the joint optimal solution for all nodes in the parse graph simultaneously, as is the goal in image segmentation and scene labeling on flat graphs and image parsing and event parsing on hierarchical graphs. Another objective is to seek the marginal optimal solution for a certain node in the parse graph, as is the goal in the classification or recognition of objects. So, in short, joint parsing produces a joint optimal solution, while recognition or classification produces marginal optimal solutions. In seeking marginal solutions for only a certain node in the parse graph, algorithms may use all features of the object and its surroundings, but they do not explicitly solve for other variables other

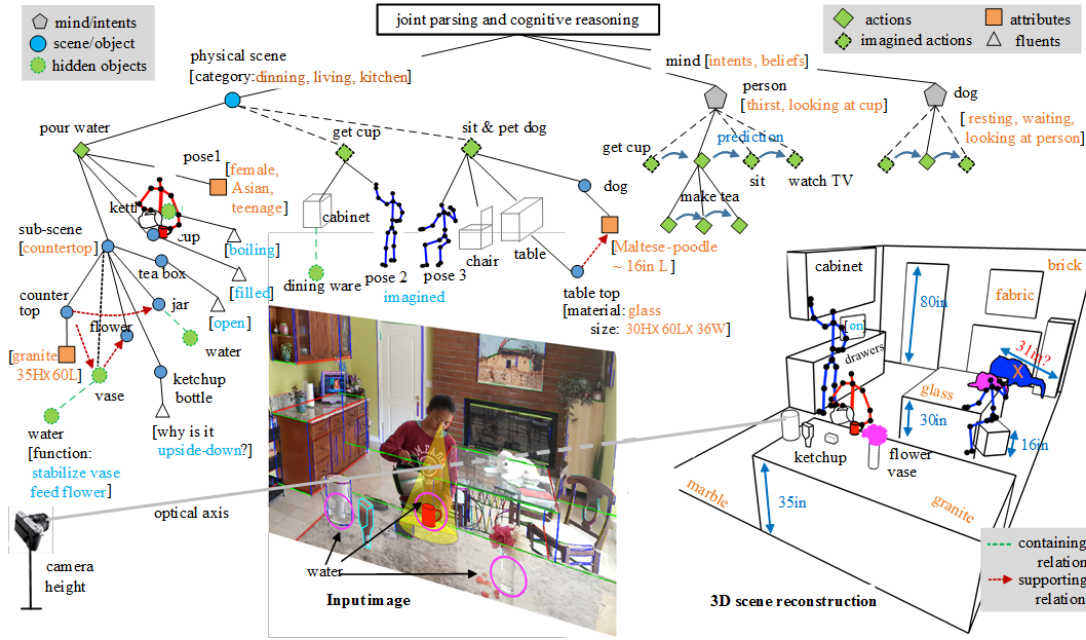


Figure 1.2: The parse graph of a natural image involves joint parsing and subsequent cognitive reasoning to determine the nodes in the parse graph, which represent, *e.g.*, minds or intents, scenes or objects, hidden objects, actions, imagined actions, attributes, and fluents, *i.e.*, how things change over time. Reproduced from Ref. [187] with permission of Elsevier, © 2020.

than the object of interest for classification or recognition. Unlike in joint optimal solutions, in marginal optimal solutions, nodes in the parse graph only interact through local non-maximum suppression.

In the past decade or two, the computer vision community has largely focused on obtaining marginal optimal solutions to problems. Solutions are often highly tailored to the specific problem, in such a way that the solutions are not generalizable to different but similar problems. Additionally, these specific solutions are often dependent on massive amounts of data. This machine learning paradigm is referred to as “big data for small tasks.” Contrary to this paradigm, in this book, just as in the entire book series, the focus is on “small data for big tasks.” Universal, generalizable solutions are desired, which depend on small amounts of data, *e.g.*, just a few training examples. The “small data for big tasks” paradigm is more reminiscent of human learning, in that it exploits prior knowledge and creates generalizable solutions.

Algorithms for the three families of models, *i.e.*, descriptive, generative, and discriminative models, maybe be organized according to their underlying representation. Algorithms for descriptive models operate on various graphs in which the nodes represent states of the same semantic level. Such algorithms include constraint-satisfaction, relaxation-labeling, dynamic programming, Viterbi, belief propagation, and Markov chain Monte Carlo (MCMC), *e.g.*, the Gibbs sampler and Swendsen-Wang cuts. Algorithms for generative models operate on hierarchical graph representations in which one level of nodes semantically generates the nodes at the level below as parts or components. Such algorithms include heuristic search, MCMC such as Metropolis Hastings and reversible jumps, and parsing algorithms, *e.g.*, Cocke–Younger–Kasami (CYK) algorithm for bottom-up inference, Earley parser for top-down inference, and the inside-outside algorithm. Lastly, algorithms for discriminative models operate by selecting features to best discriminate between various classes. Such algorithms include decision trees, boosting, support vector machines (SVM), and deep neural networks (DNN).

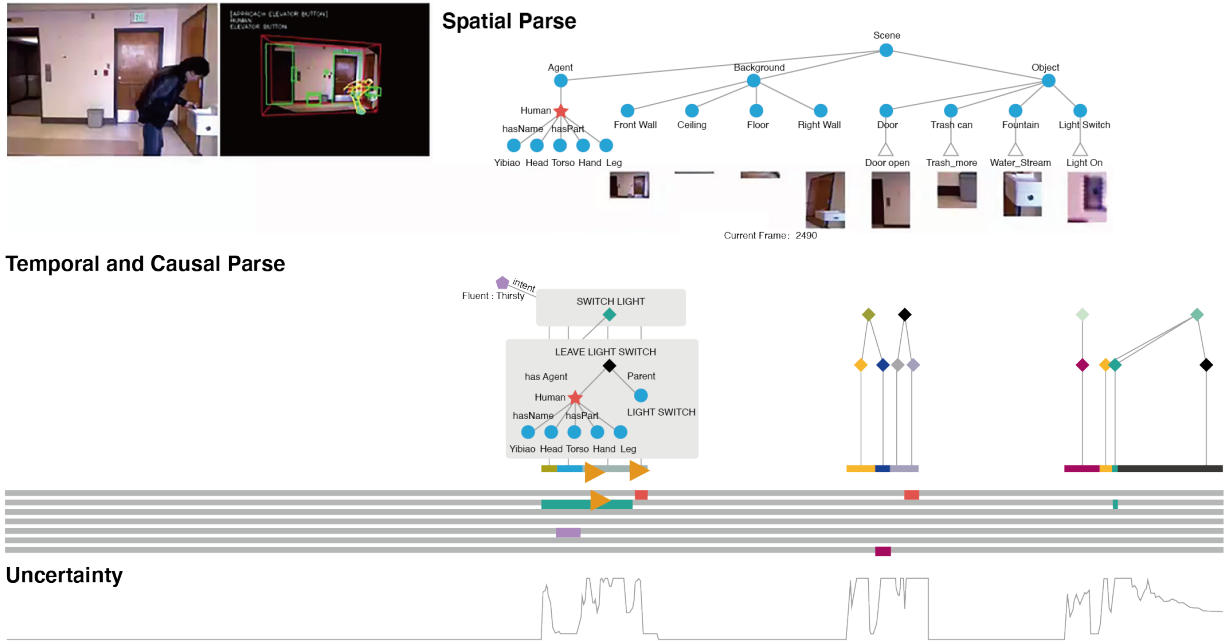


Figure 1.3: Illustration of a STC-AOG. The STC-AOG in a parse graph represents spatial properties such as objects and primitives, temporal properties such as activities and actions, and causal properties, *i.e.*, causal chains in the fluent space.

1.2 Unified Representation for Models, Algorithms, and Attributes

The unified representation for models, algorithms, and attributes for vision presented in this book is oriented around several key aspects. With regard to models, the unified representation in this book takes into account interactions and roles between the three probabilistic families of models: descriptive, discriminative, and generative models—more specifically, descriptive models on flat graphs for context, generative models with layers of dictionaries, and discriminative models for performing inference on marginal spaces. Examples of flat descriptive models, hierarchic generative models, and discriminative models are listed in Fig. 1.4. With regard to algorithms, the unified representation in this book takes into account the dynamics of three inferential channels. Namely, the three channels include α , *i.e.*, direct detection from image features; β , *i.e.*, bottom-up binding; and γ , *i.e.*, top-down prediction in space and time. With regard to attributes, the unified representation in this book takes into account the three attributes associated with each node in a parse graph. These include appearance, *i.e.*, color, age, dress styles, *etc.*; geometry, *i.e.*, camera geometry, scene layout, human poses, actions, and interactions with objects; and topology. In regard to topology, the decomposition sub-parse graph under each node in the parse graph is actually the given node’s topology attribute.

1.2.1 Three Families of Probabilistic Models

Flat descriptive models, in existence since the 1960s, are constraint-satisfaction models and include Markov random fields, graphical models, and the Filters, Random Fields, and Maximum Entropy (FRAME) model by Zhu. Hierarchic generative models, introduced in the 1970s by Fu and others, are concerned with syntactic pattern recognition and include stochastic context-free grammars, sparse coding models, wavelets and harmonic analysis, and image grammars. Discriminative models, originating in 1959 with the invention

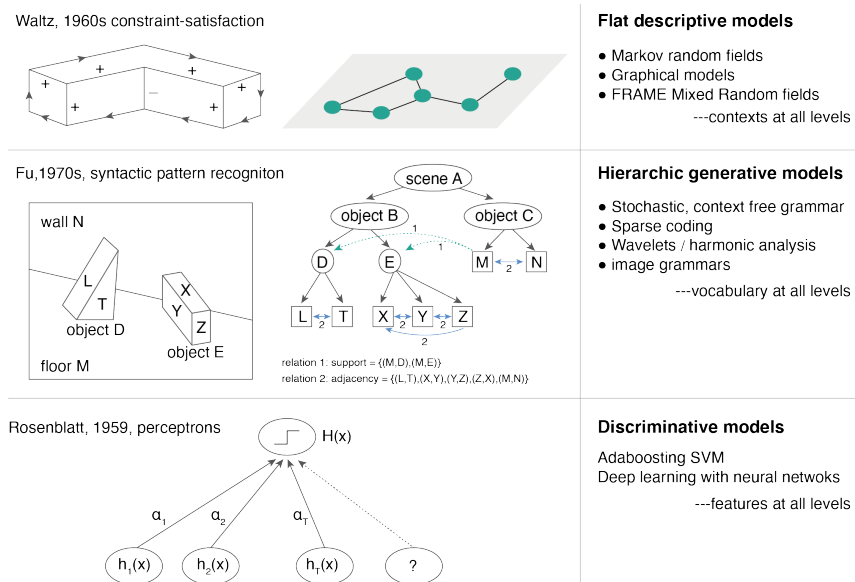


Figure 1.4: Flat descriptive models, hierarchic generative models, and discriminative models are three probabilistic families of models used to facilitate a unified representation for models, algorithms, and attributes in vision.

of the perceptron from Rosenblatt, include AdaBoosting, support vector machines (SVM), and deep learning models with neural networks. Flat descriptive models have contexts at all levels, hierarchic generative models have vocabulary at all levels, and discriminative models have features at all levels.

To clarify terminology, take the representation of a table as an example, shown in Fig. 1.5. Descriptive or declarative models, such as Markov random fields, the Gibbs model, and the Julesz ensemble, are constraint-satisfaction models and are flat, undirected graphs. Variants of descriptive models, such as Markov chains, may be directed graphs, but are still flat. Models that are both descriptive and generative, such as hidden Markov models, allow for hierarchical relations. In hierarchical models, a whole may be decomposed into its parts. Contrarily, in discriminative models, the whole is discriminated using its parts, meaning various features of the image could be used to help perform correct discrimination.

1.2.2 Dynamics of Three Inferential Channels

Recall that, with regard to algorithms, the unified representation in this book takes into account the dynamics of three inferential channels. Namely, the three channels include α , which consists of direct feature detection from images; β , which consists of bottom-up binding; and γ , which consists of top-down prediction in space and time.

Each node in a parse graph has its own α , β , and γ computing processes. By monitoring, at each node in the parse graph, the dynamics of three inferential channels for (a) direct feature detection, (b) bottom-up inference, and (c) top-down inference, a more unified representation with respect to algorithms may be accomplished.

Remember that, in this book, a unified representation with respect to models, algorithms, and attributes is offered. Next, the unified representation with respect to attributes is discussed.

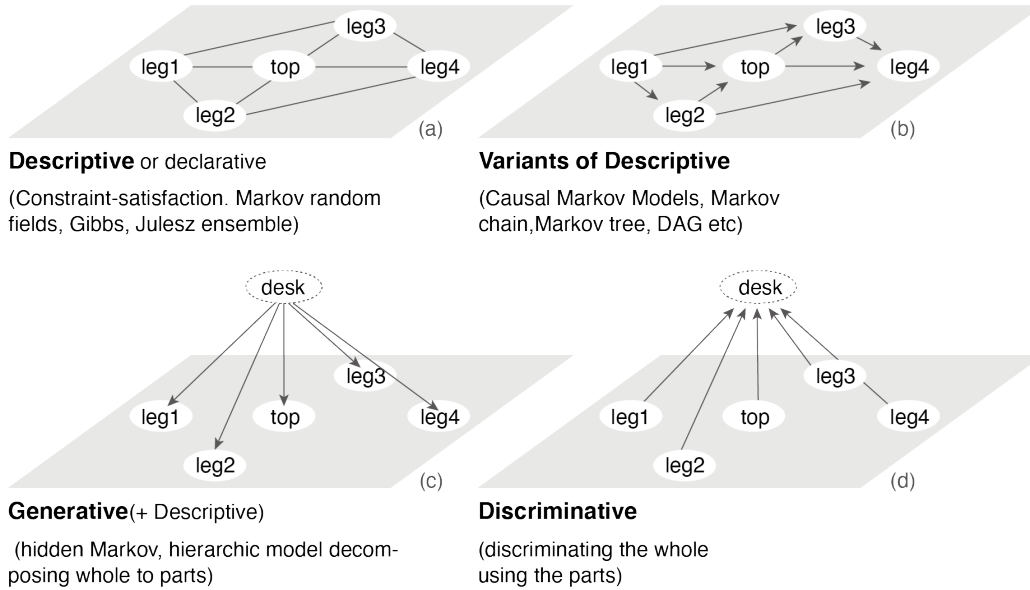


Figure 1.5: Flat descriptive models, hierarchic generative models, and discriminative models are the three probabilistic families of models which must be considered in creating a unified representation for models, algorithms, and attributes in vision.

1.2.3 Three Attributes associated with each Node

Recall that, with regard to attributes, the unified representation in this book takes into account the three attributes associated with each node in a parse graph. These three attributes include appearance, geometry, and topology. Appearance attributes may include texture, illumination, color, age, and dress styles. Geometry attributes may include 3D frame, camera parameters, scene layout, poses, actions, and interactions with objects. In regard to topology, the decomposed sub-parse graph under each node in a parse graph is actually the given node's topology attribute.

1.3 Missing Themes in Popular Data-Driven Approaches

Often missing themes in popular data-driven approaches to computer vision include top-down inference in space and time, vision as a continuous computational process, retaining the ability to resolve ambiguities in vision problems and hence preserve distinct solutions, and the notion that vision is driven by a large number of tasks. These crucial themes should be reflected in data-driven computer vision algorithms, as they are motivated by empirical observations about human vision. These themes are discussed extensively in this section.

1.3.1 Top-Down Inference in Space and Time

In regard to human vision, understanding an image requires a vast amount of prior knowledge about the world. Most objects cannot be detected by their local appearance alone, as shown by the coffee cup, bear, and tea box in Fig. 1.6. Visual phenomena often exhibit qualities such as low resolution or occlusion that make it considerably more difficult to construct representations of some stimuli or images. Prior knowledge plays a large role in the interpretation of some visual phenomena.



Figure 1.6: Replicating the faculty in human vision, top-down inference in computer vision can help identify unclear, *e.g.*, low-resolution or occluded, objects in natural images. By using the information conveyed by surrounding objects, the identities of a coffee cup, bear, and tea box in this case can be inferred.

Top-down inference using prior knowledge should play a large role in computer vision, just as it does in human vision. Top-down inference is, unfortunately, a missing theme in many popular data-driven approaches. In human vision, the prior knowledge of, *e.g.*, surrounding objects may help to predict an object in a scene such as a pen, phone, or key, which is not fully detectable on its own due to distance or occlusion, as in Fig. 1.6. Top-down inference in computer vision utilizes a hierarchical representation, employing a parse graph to represent and facilitate the dynamics of both top-down and bottom-up inferential processes. Facilitating the dynamics of top-down and bottom-up inference is, more specifically, a scheduling problem and is introduced in a later chapter in this book. Note in Fig. 1.6 how top-down inference may be used to infer the identity of the coffee cup, bear, and tea box, which in this case exhibit both low resolution and occlusion.

Also noteworthy is that in the visual pathways in the brain, there are many more top-down and lateral connections than bottom-up connections. In fact, the ratio is roughly 10 top-down or lateral connections for each bottom-up connection. This simply signifies that, in comparison to bottom-up inference, top-down inference, which again is based on an agent's prior knowledge, is especially crucial to vision. Computer vision algorithms should thus prioritize accurate top-down inference.

1.3.2 Vision is to be a Continuous Computational Process

The idea that computer vision is a continuous computational process is also a lost component in popular data-driven approaches. This continuous computational process involves a continual ordering or construction of the joint parse graph representation of an image. In the beginning, computer vision researchers generally thought that a parse graph should be initially constructed from terminal nodes, *i.e.*, image primitives, to then construct object parts, objects, and eventually entire scenes. However, a psychological study by Thorp *et al.* in 1995 showed that human vision can classify higher-level nodes in a hierarchy, such as whether an image is indoors or outdoors, in as early as 40 milliseconds, which is roughly the time needed for edge detection. This pointed to the fact that human vision does not employ bottom-up inference exclusively, moving from primitives to eventually entire scenes. Rather, human vision employs a continual process


Image shown to subjects	40ms	80ms	107ms	500ms
	" Possibly outdoor scene, maybe a farm. I could not tell for sure."	" There seem to be two people in the center of the scene."	" People playing rugby. Two people in close contact, wrestling, on grass. Another man more distant. Goal in sight."	" Some kind of game or fight. Two groups of two men. One in the foreground was getting a fist in the face. Outdoors, because I see grass and maybe lines on th grass? That is why I think of a game, rough game though. More like rugby than football because they weren't in pads and helmets..."

Figure 1.7: A psychological study by Thorp *et al.* demonstrated that vision is a continuous computational process, for which interpretation is subject to change over time based on continual top-down and bottom-up inferences made by the perceiving agent.

of both top-down and bottom-up inference over time to form interpretations of visual stimuli. Note in Fig. 1.7 how subjects could indicate in roughly 80 milliseconds how many people are in the main scene, in roughly 107 milliseconds the activities of the people in the scene, and in roughly 500ms almost all relatively important information about the full scene. This points to the fact that vision is a continuous computational process and subject to change over time based on new inferences made by the agent. The more one looks, the more one sees.

So, the psychological study by Thorp *et al.* demonstrated that vision is a continuous computational process, making inferences over time in both the top-down and bottom-up directions to interpret visual stimuli. As human visual interpretation is subject to change over time, a parse graph should be constructed similarly as a computational process that employs continuous top-down and bottom-up inference. The word “continually”, to be clear, denotes that the parse graph is subject to change based on, *e.g.*, new inferences made by the agent or changes to the prior knowledge of the agent.

In another study in psychophysics by Lin Chen in 1985, an argument was made that human vision in the early stages of perception, *e.g.*, under 50 milliseconds, is very sensitive to global “topology features” as a means to justify interpreted textons. Topology in this case denotes the configuration, or graph, of the layout of the scene. The configuration of the scene layout can help to justify or, conversely, cast doubt on textons interpreted by the agent from the earliest stage of their perception. This is another psychological experiment demonstrating the prevalent top-down inferential nature of human vision.

In moving from parsing some visual stimuli to full understanding, then, what types of concepts are relevant and hence will be discussed in this book? That is, what takes an agent from syntactic parsing to semantic understanding? Note Fig. 1.8, which displays core concepts involved in the transition from image parsing to image understanding. Image understanding includes image parsing and cognitive reasoning. Image parsing is posed as a joint inference problem involving hierarchical and contextual parse graph representations, still only on a syntactic level. This is the focus of this book. Cognitive reasoning augments the parse graph with meaningful semantics, defined by various tasks, functions, physics, intent, and so forth. This is beyond the syntactic level and will be covered in the third book.

Previously, it was mentioned that visual interpretation may be broken down into the interpretation of objects, scenes, and events. With regard to objects, the transition from syntax, the result of image parsing, to meaningful semantics, the result of image understanding, consists of a transition from primitives to parts to objects to finally object groups. Learning these object properties, an agent gains a deeper understanding

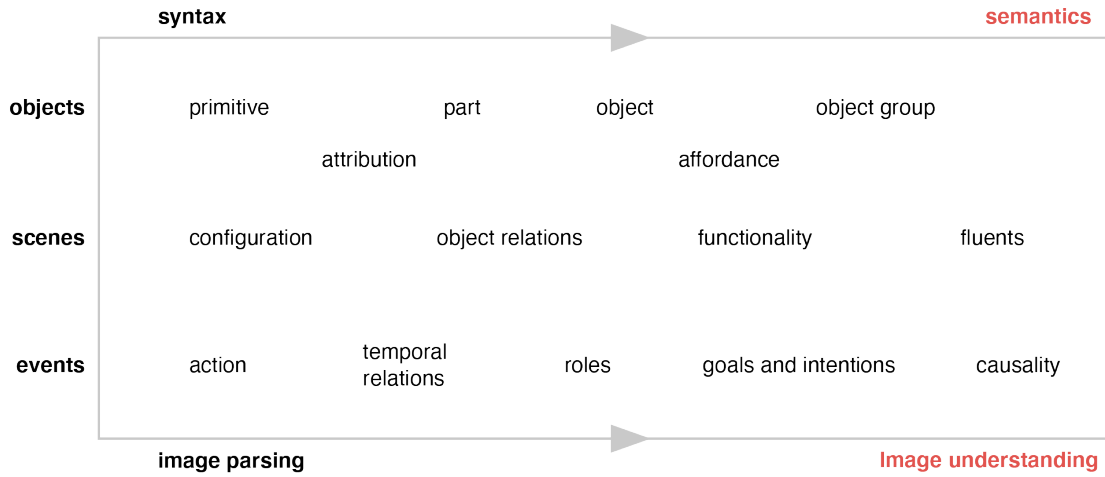


Figure 1.8: The transition from syntax, the result of image parsing, to meaningful semantics, the result of image understanding, rests on the visual interpretation of increasingly higher-order properties of visual stimuli, which can be categorically broken down into objects, scenes, and events. For example, in understanding an event, the relevant actions or temporal relations can be deduced from image parsing and minimal exercising, if any, of inference. However, understanding the goals and intentions of agents, or the causality, of an event involves more exercising of inference but also leads to a much richer understanding of the event.

of some images. With regard to scenes, the transition from syntax to semantics consists of a transition from configurations to object relations to functionality to fluents, *i.e.*, how things change over time. Similarly, learning these scene properties, an agent gains a deeper understanding of some images. With regard to events, the transition from syntax to semantics consists of a transition from actions to temporal relations to roles to goals and intentions to causality. Again, learning these event properties, an agent gains a deeper understanding of some images.

1.3.3 Resolving Ambiguities and Preserving Distinct Solutions

Another missing component in data-driven vision approaches to vision is retaining the algorithmic ability to resolve ambiguities and, furthermore, preserve multiple distinct solutions for an image. Images have local and global ambiguities, and an agent's perception may switch or jump between valid perceptions in some structured state space. Both of these valid perceptions need to be preserved as valid solutions.

Fig. 1.9 shows some classic examples in vision of ambiguous interpretations. The image in the upper-left corner may be perceived as a young woman looking away, or an old woman looking down and forward. The mind may alternate between visual perceptions, and in general, agents may also have different initial perceptions of the image. In this case, a vision algorithm should seek to preserve these two distinct solutions, one a young woman and the other an old woman, as valid interpretations of the image. Similarly, the image in the upper-right corner may be perceived as an individual playing a saxophone-like instrument, or a woman's face, and a vision algorithm should preserve these distinct solutions. This is a difficult problem in computer vision, as the distinct solutions in these cases correspond to extremely different parse graphs for the given input image. An individual playing a saxophone is an entirely different visual interpretation and parse graph than a woman's face. Lastly, in the bottom image showing the duck-rabbit ambiguity, the two distinct solutions shown are a duck and a rabbit. For the solution with a duck, the derived parse graph is made up of three key parts: the beak, an eye, and a head. For the solution with a rabbit, the derived parse graph is

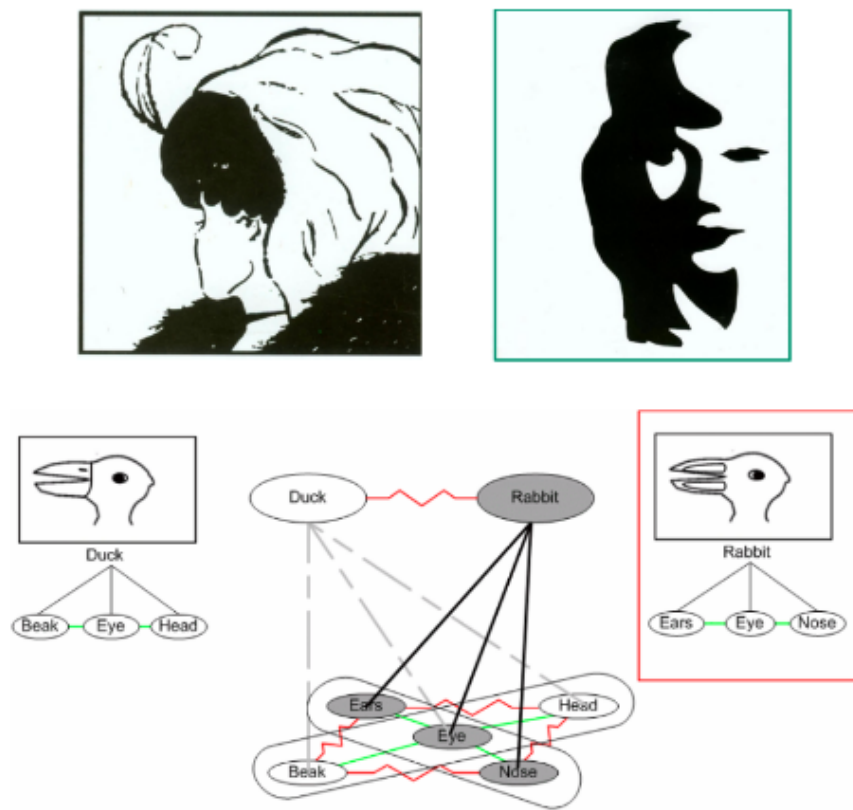


Figure 1.9: These classic examples in vision demonstrate how it is possible to have two different interpretations of the same image, *e.g.*, a duck and a rabbit. Accordingly, computer vision algorithms should resolve these ambiguities so as to preserve both images, *e.g.*, the duck and the rabbit, as valid interpretations, or solutions, to the vision problem.

made up of three different parts: the ears, an eye, and a nose. This example most clearly shows how one object part could be perceived as either of one or two parts, a duck's beak or a rabbit's ears, depending on the perception had by the agent. J. Porway and Zhu in "Computing Multiple Solutions in Graphical Models by Cluster Sampling" address and solve the duck-rabbit ambiguity, so the system successfully switches or jumps between the duck and rabbit perceptions.

It remains a significant challenge for vision algorithms to preserve multiple distinct solutions or interpretations for an image, as this inherently "delays" the determination of the identity of the object until a later point in time. The algorithm must decide how long this waiting time can be, depending on the relevant task of the vision problem and any risks posed by waiting. The alternative to waiting to resolve the object ambiguity is to make an early determination based on a generic machine learning loss function, which may lead to a break-down in the validity of the overall system, so this approach is certainly not without faults. In any case, resolving image ambiguities and preserving distinct solutions is often a missing component of popular data-driven approaches for vision.

1.3.4 Vision is Driven by a Large Number of Tasks

A last concept often not reflected by data-driven approaches is that, from the perspective of an autonomous agent, vision should very often be driven by a large number of tasks. This is because most vision problems are not general but rather quite task-dependent. Fig. 1.10 shows the many distinct ways in which vision is used for the task of making coffee. A human's vision is very much task-driven during this task—one becomes focused on the tools and steps required to make coffee so as to make the process more efficient. A computer vision algorithm should seek to replicate this behavior, applying a task-driven nature a vision problem where appropriate, as a means to increase the realisticness and therefore the success of the relevant system seeking to model human vision.

Applying task-driven behavior to a vision algorithm will assist with a series of difficult problems, such as the ones portrayed by Fig. 1.10. Examples of such problems include looking for objects and detecting them in cluttered scenes and also guiding the body and hand to approach objects to facilitate grasping. To give an example, grasping the cup with a robotic hand necessitates real-time feedback about the status. Other difficult problems may include guiding the approach of one object toward another, *e.g.*, placing a cup under a faucet, or checking the status of an object that is progressing toward a goal, *e.g.*, the water level of a cup that is currently being filled. Each of these tasks utilizes only a small portion of the parse graph, which is constructed in real-time in a task-driven way based on a hierarchy of relevant tasks. If a vision algorithm can similarly find and utilize a small portion of a parse graph for the various subtasks it has to solve as part of a task like making coffee, it will most closely resemble human vision and hence be the best algorithm.

1.4 Scope of Book: Compositional Patterns in the Middle Entropy Regime

The scope of this book is compositional patterns in the mid-entropy regime. The first book discussed and presented various models for the low- and high-entropy regimes, and this book does the same for the mid-entropy regime. The mid-entropy regime consists of compositional patterns with elements of both the low- and high-entropy regimes.

1.4.1 Information Scaling and Three Entropy Regimes

In the first book in this series, entropy regimes and information scaling were introduced. The continuous entropy rate, *i.e.*, image complexity, was divided into three regimes: the low-entropy regime, the high-entropy regime, and the mid-entropy regime. Recall that, in the low-entropy regime, the complexity of the underlying world structure W is much smaller than the dimension of the image I . Accordingly, to model visual phenomena in this regime, such as the images on the left in Fig. 1.11, sparse coding models are common. In the high-entropy regime, exemplified by the images on the right in Fig. 1.11, the complexity of the underlying world structure is much higher than the dimension of the image I . For example, dozens of leaves may be projected onto a 7×7 -pixel patch, which forms the impression of texture, as in the upper-right image. Appropriate for modeling the high-entropy regime are statistical physics models like the Gibbs model, Markov random fields, and the FRAME model.

In the mid-entropy regime, a composition of the low- and high-entropy regimes, compositional models such as And-Or graphs are appropriate. In Fig. 1.12 and Fig. 1.13, gradual transitions can be observed from the low-entropy regime, *e.g.*, image primitives, to the high-entropy regime, *e.g.*, textures or random noise. The mid-entropy regime consists of objects and parts that are compositions of elements from the low- and high-entropy regimes.

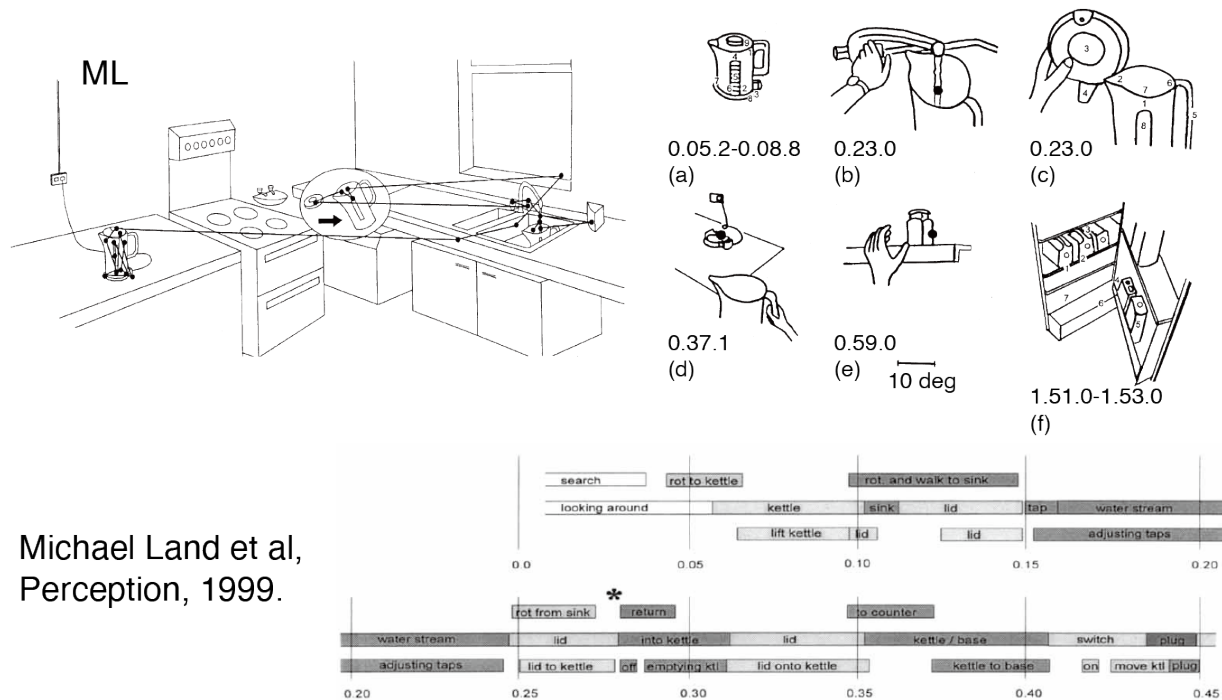


Figure 1.10: Many subtasks are involved in the task of making coffee, but humans can more efficiently perform this complex task due to the task-driven nature of vision. In the upper-right corner, diagrams (a) to (f) portray the specific focus of human vision as the task of making coffee is carried out. Human vision quickly shifts focus from subtask to subtask and in this sense is extremely task-driven. Computer vision models should seek to replicate this task-driven nature of human vision. Reproduced from Ref. [81] with permission of SAGE Publication, © 1999.

1.4.2 Organization of Book

The remaining chapters of this book discuss a spatial, temporal, causal, and attributed stochastic grammar with context embedded in a unified And-Or graph (AOG) representation.

The spatial And-Or graph (S-AOG) serves as a unified framework of representation, learning, and recognition for objects and scenes. With the attribute AOG (A-AOG), constraints may be integrated into a recursive bottom-up and top-down process. This corresponds to the idea of relations between geons in Recognition-by-Components (RBC) theory. The stochastic temporal And-Or graph (T-AOG) can be used to represent hierarchical compositions of events and temporal relations between subevents. The terminal node of a T-AOG is an atomic action, which may be decomposed into a human pose, one or multiple objects, and the relations between the pose and the objects. The causal And-Or Graph (C-AOG) is suited for unsupervised learning of causal relations from video, as they learn which actions can cause objects to change status. With the spatial-temporal-causal AOG (STC-AOG), joint parsing of the world may be performed to transfer observed data into a structured hierarchical knowledge representation. Finally, with grounded grammar induction of vision and language, we can learn the grammar without supervision or pre-defined hierarchies from the alignment of two modalities.



Figure 1.11: Entropy increases moving from left to right, which corresponds to the world complexity W increasing and the image complexity decreasing. Models suited for the low-entropy regime, such as sparse coding models, are appropriate for images like the ones on the left. Models suited for the high-entropy regime, such as Markov random fields, are appropriate for images like the ones on the right. Compositional models suited for the mid-entropy regime, such as And-Or graphs, are appropriate for images like the ones in the middle.

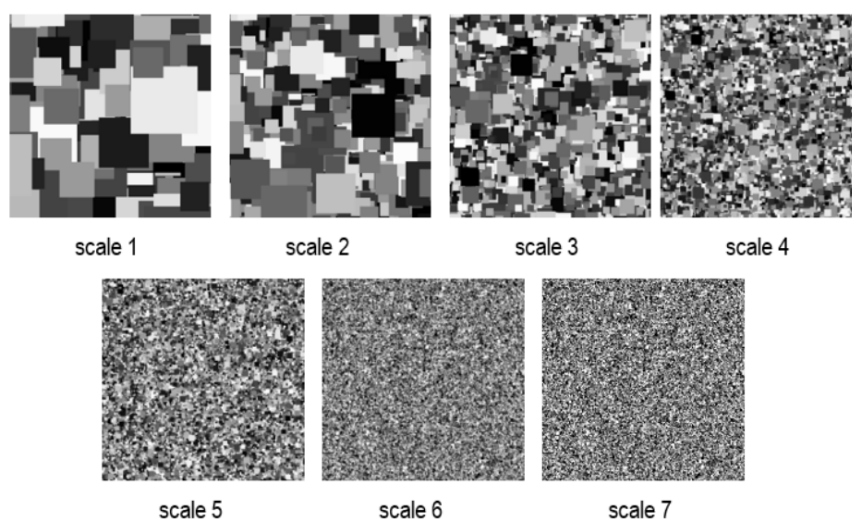


Figure 1.12: Moving from scales 1 to 7, entropy increases. This corresponds to world complexity increasing and image complexity decreasing.

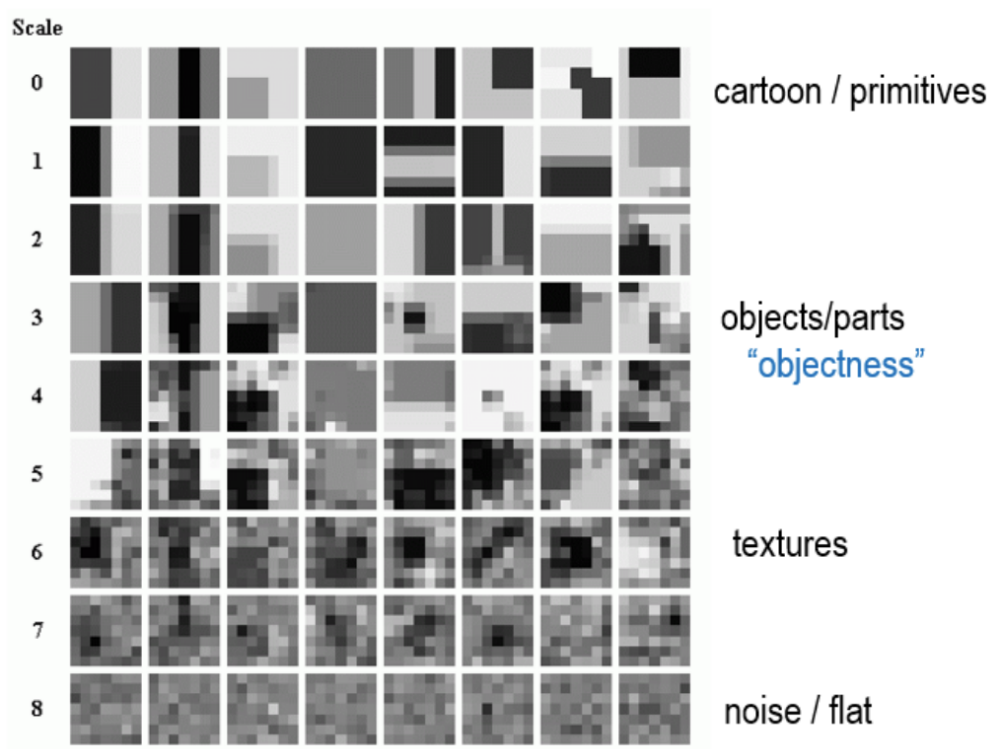


Figure 1.13: Moving from scales 0 to 8, entropy increases. This corresponds to world complexity increasing and image complexity decreasing. Compositional models are appropriate for modeling objects and parts in the mid-entropy regime, which exhibits qualities of both the low- and high-entropy regimes.

2

Overview of Stochastic Grammar

Before diving into the grammatical modeling in computer vision, we provide an overview of grammar in the general sense. The hope is that readers can obtain an impression of its universality, particularly its central role in cognitive modeling.

2.1 Grammar as a Universal Representation of Intelligence

Grammar in language is merely a recent extension of much older grammars that are built into the brains of all intelligent animals to analyze sensory input, to structure their actions and even formulate their thoughts. All of these abilities, beyond the simplest level, are structured in hierarchical patterns built up from interchangeable units but obeying constraints, just as speech is.

— David Mumford

Indeed, grammar captures the structuralism in the world so well that nativists insist that we humans acquire it as an evolutionary gift in our mind to become a highly intelligent creature. Anthropologists like Daniel Everett [35] even argue that some of our ancestor sapiens relatives were also endowed with this cognitive capability to collectively sail across the sea. Essentially, complex systems have these common principles of organization:

1. **Compositionality:** Complex configurations in 1D (DNA, mRNA, unfolded proteins), 2D (images or a layer of materials) and 3D (folded proteins, objects and scenes in vision) are composed of small dictionaries, in a layered, hierarchical structure.
2. **Reconfigurability:** The compositions allow alternative elements and are reconfigurable with part sharing and reuse. This flexibility yields a large number of configurations in space and time.
3. **Emerging functions and properties:** Each element in the system has its own graph structures. There are also bonds to interact with other elements, which collectively give rise to emerging properties at multiple scales.

Originally aiming at reverse engineering human intelligence, pioneers in cognitive science and artificial intelligence adopted the very same set of principles when formulating the representation of information in their core disciplines. Borrow the terminology from grammar, a mental state of an agent can be hierarchically represented with a *parse graph* that is valid in its grammatical modeling of the world. Specifically, parse graphs for *vision* represent the hierarchical organization and composition of objects, scenes and events in the *spacetime* of the agent. This representation is further augmented in *cognition* for functionality,

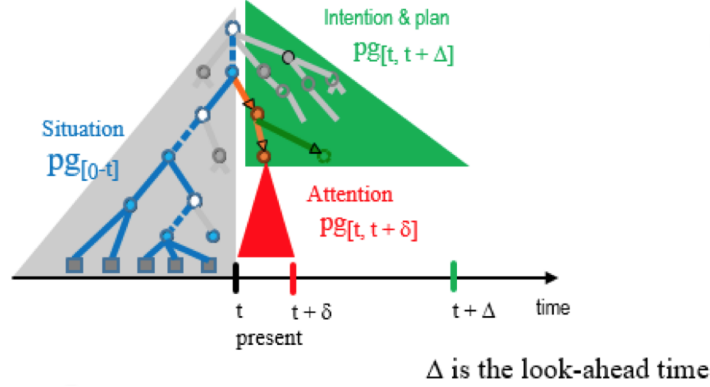


Figure 2.1: The situation, intention, and attention in a mental parse graph.

physics, causality and intentions. That is, elements in visual parse graphs provide the basis for high-level cognitive activities. Basically, there are two kinds of manifesto of these activities: motion and communication. Tasks and motions are represented in a coarse-to-fine hierarchy with parse graphs in the classic field of *robotics*. Finally, for communication with the device of language, for instance nature language, parse graphs can represent the syntactical hierarchy of phrases, sentences, discourses and stories. It can also represent the semantic hierarchy, following the same grammar above. In sum, Fig. 2.1 provides a holistic view of the joint cognitive representation. Note that *Situations* (with *beliefs*) and *Intents* (with *plans*) are two partial parse graph, the boundary between which is *agency*. The *attention* mechanism attends on a sub-parse graph.

2.2 An Empiricist's View of Grammars

The origin of grammar in real-world signals, either in natural language or vision, comes from the observation that certain parts of a signal s tend to occur together more frequently than by chance. One evidence discovered by Mumford in his early statistical endeavor is that when he removes punctuation, numbers and capitalization from Mark Twain's and counted the letter frequency only, 74% word boundaries can be recovered as minima of the binding strength, as shown in Fig 2.2. Such co-occurring elements can be grouped together to form higher order parts of the signal, and this process can be repeated, forming increasingly larger parts. Because of their higher probability, these parts are found to re-occur in other similar signals. So they form a vocabulary of "reusable" parts. A basic statistical measure, which indicates whether a grouping is a good part, is a quantity that measures, in bits, the binding strength of two parts $s|_A$ and $s|_B$ of signal $s : D \rightarrow I$ where $A \subset D$ and $B \subset D$:

$$\log_2 \left(\frac{p(s|_{A \cup B})}{(p(s|_A) \cdot p(s|_B))} \right). \quad (2.1)$$

Two parts of a signal are bound together if the probability of their co-occurrence is significantly greater than the probability of their independent occurrence. A classic example showing similar phenomena like Mumford goes back to Laplace, who found that the sequence of 14 letters "CONSTANTINOPLE" occurs much more frequently in normal text than in random sequences of the 26 letters in which the letters are chosen independently, even with their standard frequencies. In this example, the composite part is a word and its constituents are letters. A more elaborate example from vision is shown in Figure 2.3. The left figure illustrates how nearby lines tend to be parallel more often than at other mutual orientations and hence a pair

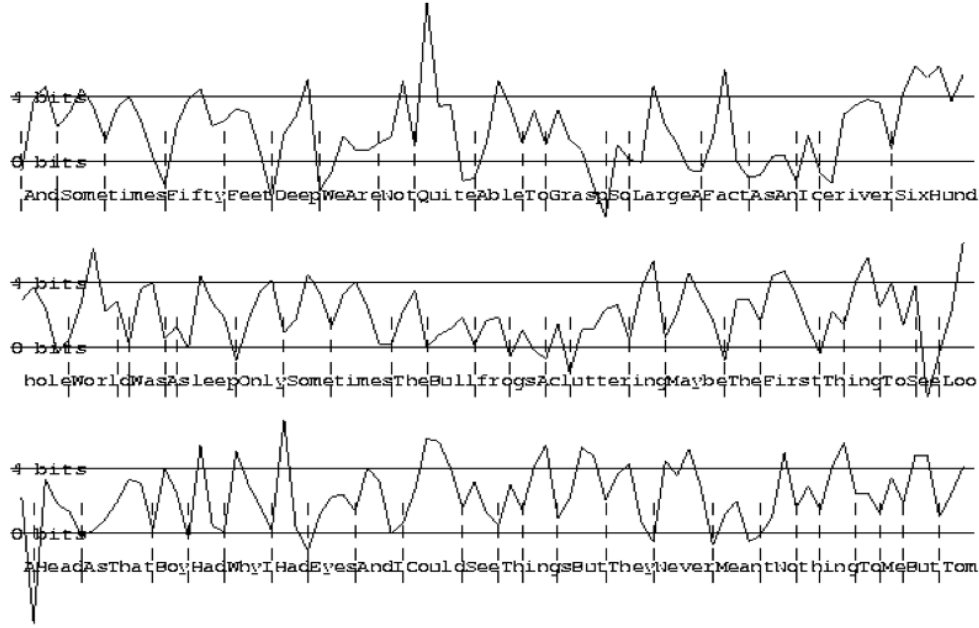


Figure 2.2: Recovering word boundaries of Mark Twain’s novels with statistical learning. Reproduced from Ref. [99] with permission of CRC Press, © 2010.

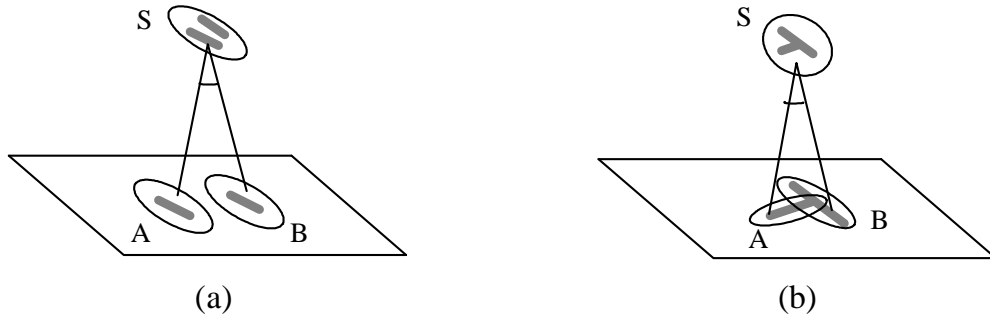


Figure 2.3: (a) Two parallel lines form a reusable part containing as its constituents the two lines (b) A T-junction is another reusable part formed from two lines. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

of parallel lines forms a reusable part. On the right, we see how another frequent configuration is when two lines are intersecting and roughly perpendicular forming a "T-junction".

The set of reusable parts that one identifies in some class of signals, *e.g.* in images, is called the *vocabulary* for this class of signals. Each such reusable part has a name or label. In language, a noun phrase, whose label is "NP", is a common reusable part and an element of the linguistic vocabulary. In vision, a face is a clear candidate for such a very high-level reusable part. The set of such parts which one encounters in analyzing statistically a specific signal is called the *parse graph* of the signal. Abstractly, one first associates the set of subsets $\{A_i\}$ of D with a signal s such that $s|_{A_i}$ is a reusable part. Then these subsets are made into the vertices or nodes $\langle A_i \rangle$ of the parse graph. In the graph, the proper inclusion of one subset in another, $A_i \subsetneq A_j$, is shown by a "vertical" directed edge $\langle A_j \rangle \rightarrow \langle A_i \rangle$. For simplicity, we prune redundant edges in this graph by adding edges only when $A_i \subsetneq A_j$ and there is no A_k such that $A_i \subsetneq A_k \subsetneq A_j$.

In the ideal situation, parse graph is a tree with the whole signal at the top and the domain D (the letters of the text or the pixels of the image) at the bottom. Moreover, each node $\langle A_i \rangle$ should be the disjoint union of its children, *i.e.*, the parts $\{A_j | A_j \subsetneq A_i\}$ such that $\cup_j A_j = A_i$. This is the case for the simple parse trees of Figure 2.3 or in most sentences, such as the ones shown below in Figure 2.9.

2.3 The Formalism of Grammars

In last section we introduce an empiricists' perspective of grammar. However, the most impactful school of thought in the linguistics literature argues at a viewpoint of nativism. This formal idea of grammar goes back to Panini's Sanskrit grammar in the first millenium BCE. But its modern formalization can be attributed to Chomsky [84]. Normally, grammar is defined as a 4-tuple $\mathcal{G} = (V_N, V_T, \mathbf{R}, S)$, where V_N is a finite set of non-terminal nodes, V_T a finite set of terminal nodes, $S \in V_N$ is a start symbol at the root, and \mathbf{R} is a set of production rules,

$$\mathbf{R} = \{\gamma : \alpha \rightarrow \beta\}. \quad (2.2)$$

$\alpha, \beta \in (V_N \cup V_T)^+$ are strings of terminal or non-terminal symbols, with α, β including at least one non-terminal symbol*. Chomsky classified languages into 4 types according to the form of their production rules. A type 3 grammar has rules $A \rightarrow aB$ or $A \rightarrow a$, where $a \in V_T$ and $A, B \in V_N$. It is also called a finite state or regular grammar. A type 2 grammar has rules $A \rightarrow \beta$ and is called a context-free grammar. A type 1 grammar is context sensitive with rules $\xi A \eta \rightarrow \xi \beta \eta$ where a non-terminal node A is rewritten by β in the context of two strings ξ and η . The type 0 grammar is called a phrase structure or free grammar with no constraint on α and β . One of the greatest contributions of this Chomsky hierarchy is that it bridges grammar structure in linguistics with the general computation theory traced back to mathematicians like Alan Turing. A summary is provides in Table 2.1.

Grammar	Language	Automation	Production rules
Type 0	Recursively enumerable	Turing machines	$\alpha A \beta \rightarrow \gamma$
Type 1	Context-sensitive	Linear-bounded non-deterministic Turing machines	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type 2	Context-free	Non-deterministic pushdown automata	$A \rightarrow \alpha$
Type 3	Regular	Finite state automata	$A \rightarrow a, A \rightarrow aB$

Table 2.1: The Chomsky hierarchy for language and automata.

The set of all possible strings of terminals ω derived from a grammar \mathcal{G} is called its *language*, denoted by

$$\mathbf{L}(\mathcal{G}) = \{\omega : S \xRightarrow{\mathbf{R}^*} \omega, \omega \in V_T^*\}. \quad (2.3)$$

\mathbf{R}^* means a sequence of production rules deriving ω from S , *i.e.*

$$S \xRightarrow{\gamma_1, \gamma_2, \dots, \gamma_{n(\omega)}} \omega \quad (2.4)$$

If the grammar is of type 1, 2 or 3, then given a sequence of rules generating the terminal string ω , we obtain a parse tree for ω , denoted by

$$\mathbf{pt}(\omega) = (\gamma_1, \gamma_2, \dots, \gamma_{n(\omega)}). \quad (2.5)$$

* V^* means a string consisting of $n \geq 0$ symbols from V , and V^+ means a string with $n \geq 1$ symbols from V .

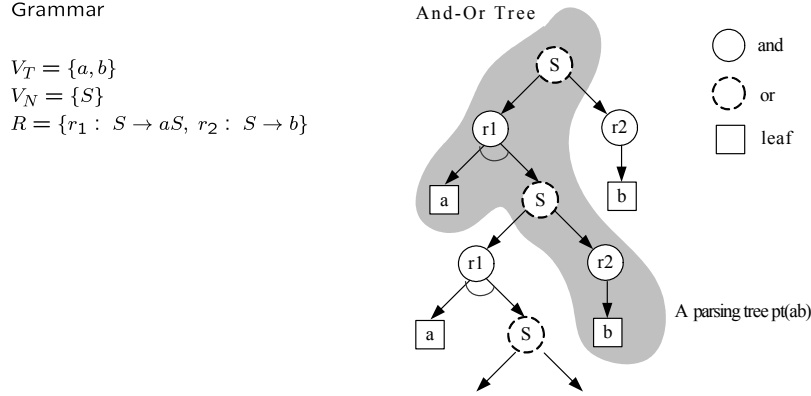


Figure 2.4: A very simple grammar, its universal And-Or tree and a specific parse tree in shadow. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

If each production rule creates one node labeled by its head A and a set of vertical arrows between A and each symbol in the string β . To relate this to the general setup of the previous section, note that each node has a set of ultimate descendants in the string ω . This is to be a reusable part. If we give this part the label $A \in V_N$, we see that the tree can equally well be generated by taking these parts as nodes and putting in vertical arrows when one part contains another with no intermediate part. Thus the standard Chomskian formulation is a special case of our general setup.

As is illustrated in Figure 2.6, the virtue of grammar lies in its expressive power of generating a very large set of valid sentences (or strings), *i.e.* its language, through a relatively much smaller vocabulary V_T, V_N and production rules \mathbf{R} . Generally speaking, the following inequality is often true in practice,

$$|\mathbf{L}(\mathcal{G})| \gg |V_N|, |V_T|, |\mathbf{R}|. \quad (2.6)$$

In images, V_T can be pixels, but here we will find it more convenient to make it a simple set of local structures in the image, such as textons and other image primitives [56, 57]. Then V_N will be reusable parts and objects in the image, and a production rule $A \rightarrow \beta$ is a template which enables you to expand A . Then the $\mathbf{L}(\mathcal{G})$ will be the set of all valid object *configurations*. The grammar rules represent both structural regularity and flexibility. The structural regularity is enforced by the template which decomposes an entity A , such as an object into certain elements in β . The structural flexibility is reflected by the fact that each structure A has many alternative decompositions.

2.4 The Mathematical Structure of Grammars

In the following chapters, we will find it convenient to describe the entire grammar by one universal *And-Or Tree*, which contains all parsings as subtrees. In a parse tree, the Or-nodes are labelled by $V_N \cup V_T$ and the And-nodes are labelled by production rules \mathbf{R} . We generate this subtree recursively, starting by taking start symbol as a root which is an Or-node. We proceed as follows: wherever we have an Or-node with non-terminal label A , we consider all rules which have A on the left and create children which are And-nodes labelled by the corresponding rules. These in turn expand to a set of Or-nodes labelled by the symbols on the right of the rule. An Or-node labelled by a terminal does not expand further. Clearly, all specific parse trees will be contained in the universal And-Or tree by selecting specific children for each Or-node reached when descending the tree. This tree is often infinite. An example is shown in Figure 2.4.

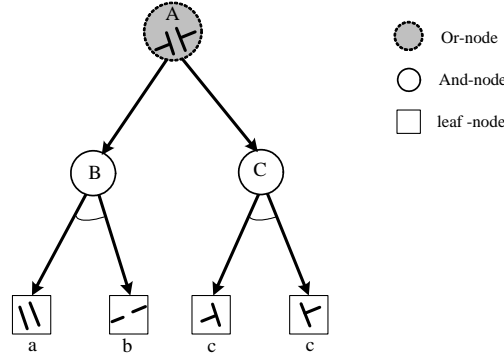


Figure 2.5: An example of binding elements a,b,c into a larger structures A in two alternative ways, represented by an And-Or tree. Reproduced from Ref. [185] with permission of , © 2007.

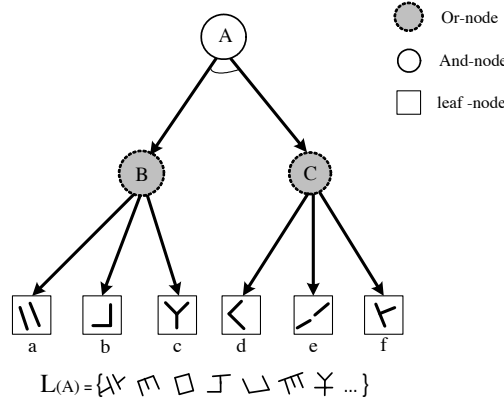


Figure 2.6: An And-node A is composed of two Or-nodes B and C , each of which includes three alternative leaf nodes. The 6 leaf nodes can compose a set of configurations for node A , which is called the "language" of A . Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

A vision example of an And-Or tree, using the reusable parts in Figure 2.3, is shown in Figure 2.5.

A, B, C are non-terminal nodes and a, b, c are terminal or leaf nodes. B, C are the two ambiguous ways to interpret A . B represents an occlusion configuration with two layers while C represents a butting/alignment configuration at one layer. The node A in Figure 2.5 is a frequently observed local structure in natural images when a long bar (*e.g.* a tree trunk) occludes a surface boundary (*e.g.* a fence). Note that in this example we can see the flexibility of grammar in representing both 2D visual representation (production rule at C) and 2.1D representation (production rule at B).

The expressive power of an And-Or tree is illustrated Figure 2.6 and Figure 2.7. Figure 2.6 is an And-node A which has two components B and C . Both B and C are Or-nodes with three alternatives shown by the six leaf nodes. The 6 leaf nodes can compose a set of configurations for node A , which is called the "language" of A – denoted by $L(A)$. Some of the valid configurations are shown at the bottom. The power of composition is crucial for representing visual concepts which have varying structures. For example, if A is an object category, such as car or chair, then $L(A)$ is a set of valid designs of cars or chairs. The expressive power of the And-Or tree rooted at A is reflected in the ratio of the total number of configurations that it can compose over the number of nodes in the And-Or tree. For example, Figure 2.7 shows two levels of And-nodes and two levels of Or-nodes. Both have branch factor $b = 3$. This tree has a total of 10 And-nodes, 30

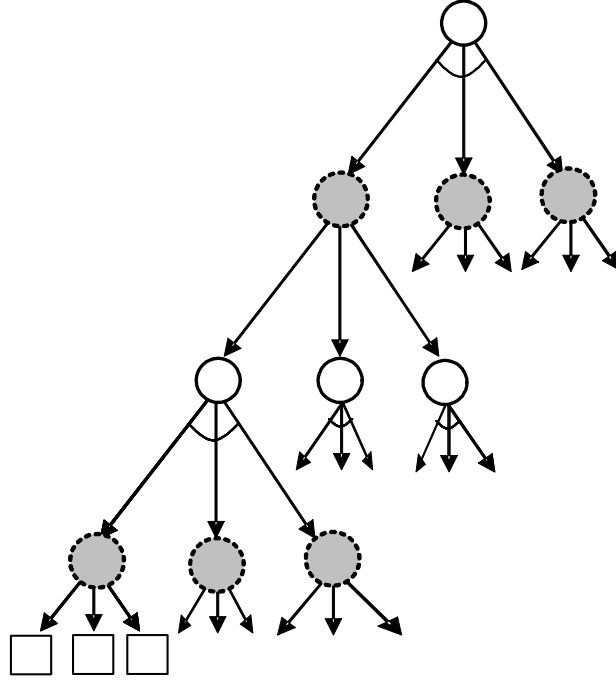


Figure 2.7: Expressiveness of grammar: An And-Or tree (5-level branch number = 3) with 10 And-nodes, 30 Or-nodes, and 81 leaf nodes, can produce $3^{12} = 531,441$ possible configurations. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

Or-nodes, and 81 leaf nodes while the number of possible structures is $(3 \times 3^3)^3 = 531,441$, though some structures may be repeated.

We shall discuss three major differences between vision grammars and language grammars in the next Chapter.

2.5 Stochastic Grammar

To connect with real world signals, we must augment grammars with a set of probabilities \mathcal{P} as a fifth component. For example, stochastic context-free grammar (SCFG) – the most common stochastic grammar in the literature – supposes $A \in V_N$ has a number of alternative rewriting rules,

$$A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_{n(A)}, \quad \gamma_i : A \rightarrow \beta_i. \quad (2.7)$$

Each production rule is associated with a probability $p(\gamma_i) = p(A \rightarrow \beta_i)$ such that:

$$\sum_{i=1}^{n(A)} p(A \rightarrow \beta_i) = 1. \quad (2.8)$$

This corresponds to what is called a random branching process in statistics [3]. Similarly, a stochastic regular grammar corresponds to a Markov chain process.

The probability of a parse tree is defined as the product,

$$p(\mathbf{pt}(\omega)) = \prod_{j=1}^{n(\omega)} p(\gamma_j). \quad (2.9)$$

The probability for a string (in language) or configuration (in image) $\omega \in \mathbf{L}(\mathcal{G})$ sums over the probabilities of all its possible parse trees.

$$p(\omega) = \sum_{\mathbf{pt}(\omega)} p(\mathbf{pt}(\omega)). \quad (2.10)$$

Therefore a stochastic grammar $\mathcal{G} = (V_N, V_T, \mathbf{R}, S, \mathcal{P})$ produces a probability distribution on its language

$$\mathbf{L}(\mathcal{G}) = \{(\omega, p(\omega)) : S \xrightarrow{\mathbf{R}^*} \omega, \omega \in V_T^*\}. \quad (2.11)$$

A stochastic grammar is said to be *consistent* if $\sum_{\omega \in \mathbf{L}(\mathcal{G})} p(\omega) = 1$. This is not necessarily true even when Equation 2.8 is satisfied for each non-terminal node $A \in V_N$. The complication is caused by cases when there is a positive probability that the parse tree may not end in a finite number of steps. For example, if we have a production rule that expands A to AA or terminates to a respectively,

$$A \rightarrow AA \mid a \quad \text{with prob. } \rho \mid (1 - \rho)$$

If $\rho > \frac{1}{2}$, then node A expands faster than it terminates, and it keeps replicating. This poses some constraints for designing the set of probabilities \mathcal{P} .

The set of probabilities \mathcal{P} can be learned in a supervised way from a set of observed parse trees $\{\mathbf{pt}_m, m = 1, 2, \dots, M\}$ by maximum likelihood estimation,

$$\mathcal{P}^* = \arg \max \prod_{m=1}^M p(\mathbf{pt}_m). \quad (2.12)$$

The solution is quite intuitive: the probability for each non-terminal node A in (2.7) is

$$p(A \rightarrow \beta_i) = \frac{\#(A \rightarrow \beta_i)}{\sum_{j=1}^{n(A)} \#(A \rightarrow \beta_j)}. \quad (2.13)$$

In the above equation, $\#(A \rightarrow \beta_i)$ is the number of times a rule $A \rightarrow \beta_i$ is used in all the M parse trees. In an unsupervised learning case, when the observation is a set of strings without parse trees, one can still follow the ML-estimation above with an EM-algorithm. It was shown in [20] that the ML-estimation of \mathcal{P} can rule out infinite expansion and produce a consistent grammar.

In Figure 2.5, one can augment the two parses by probabilities ρ and $1 - \rho$ respectively. We write this as a stochastic production rule:

$$A \rightarrow a \cdot b \mid c \cdot c; \quad \rho \mid (1 - \rho). \quad (2.14)$$

Here " \mid " means an alternative choice and is represented by an Or-node. " \cdot " means composition and is represented by an And-node with an arc underneath. One may guess that the interpretation B has a higher probability than C , *i.e.*, $\rho > 1 - \rho$ in natural images.

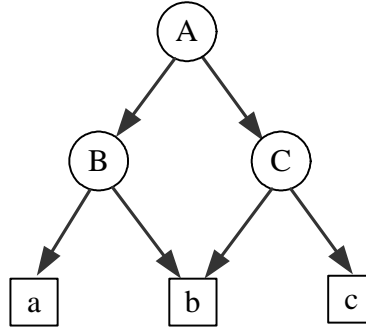


Figure 2.8: Parts sharing and the diamond structure in And-Or graphs. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

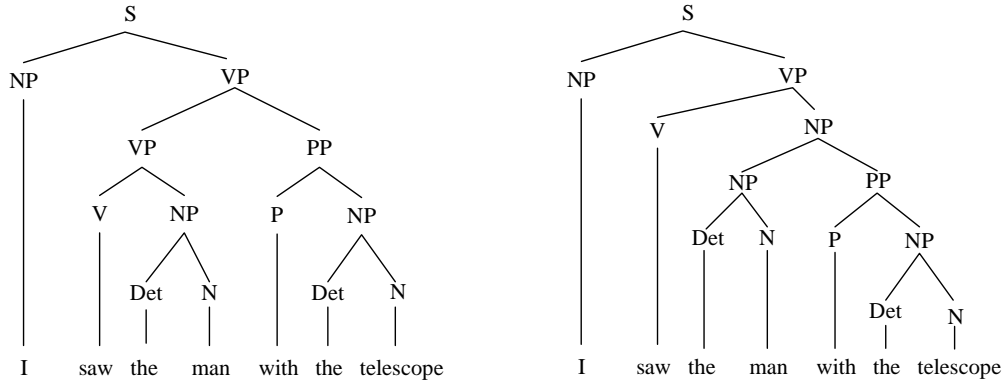


Figure 2.9: An example of ambiguous sentence with two parse trees. The non-terminal nodes S, V, NP, VP denotes sentence, verbal, noun phrase, and verbal phrase respectively. Note that if the two parses are merged, we obtain a graph, not a tree, with a "diamond" in it as above. Reproduced from Ref. [53] with permission of Springer, © 2004.

2.6 Ambiguity and Overlapping Reusable Parts

As mentioned, in good cases, there are no overlapping reusable parts in the base signal, and each part is the disjoint union of its children. But this need not to be the case. If two reusable parts do overlap, typically this leads to parse structures with a diamond in them (Figure 2.8 is an example). Many sentences, for example, are ambiguous and admit two reasonable parses. If there exists a string $\omega \in \mathbf{L}(\mathcal{G})$ that has more than one parse tree, then \mathcal{G} is said to be an *ambiguous grammar*. For example, Figure 2.9 shows two parse trees for a classic ambiguous sentence (discussed in [53]). Note that in the first parse, the reusable part ‘saw the man’ is singled out as a verb phrase or VP; in the second, one finds instead the noun phrase (NP) ‘the man with the telescope.’ Thus, the base sentence has two distinct reusable parts which overlap in ‘the man.’ Fixing a specific parse eliminates this complication. In context, the sentence is always spoken with only one of these meanings, so one parse is right while the other one is wrong. One reusable part is accepted and the other one is rejected. If we reject one, the remaining parts do not overlap.

The above is, however, only the simplest case where reusable parts overlap. Taking image parsing in vision, there seem to be four ways overlap can occur.

1. Ambiguous scenes where distinct parses suggest themselves.

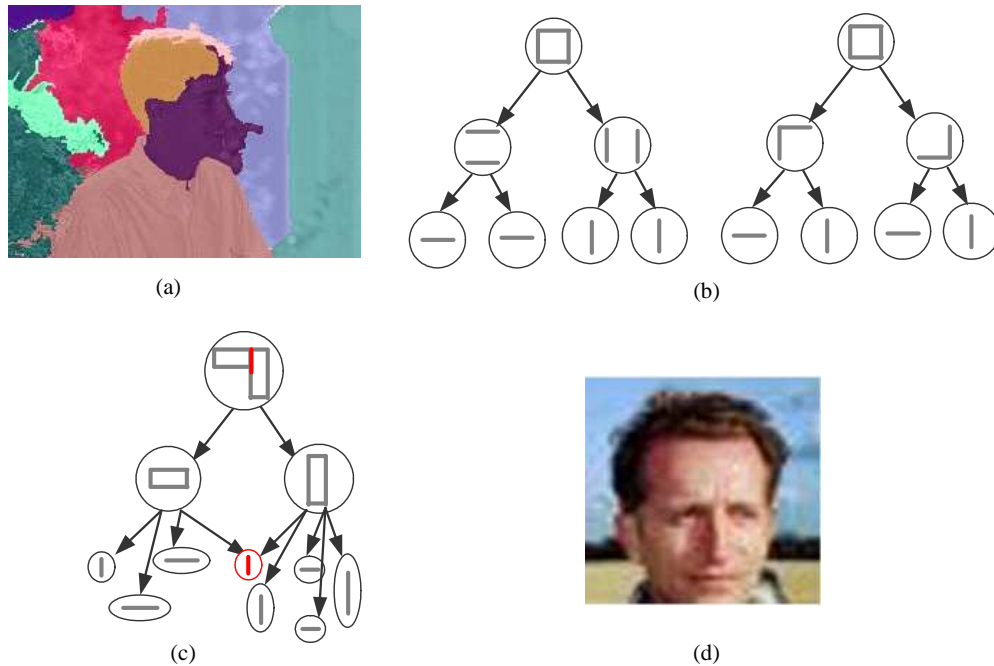


Figure 2.10: Four types of images in which "reusable parts" overlap. (a) The Pinocchio nose is a part of the background whose gray level is close to the face, so it can be grouped with the face or the background. This algorithm chose the wrong parse. (b) The square can be parsed in two different ways depending on which partial patterns are singled out. Neither parse is wrong but the mid-level units overlap. (c) The two halves of a butt joint have a common small edge. (d) The reconstructed complete sky, trees and field overlap with the face. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

2. High-level patterns which incorporate multiple partial patterns.
3. "Joints" between two high-level parts where some sharing of pixels or edges occurs.
4. Occlusion where a background object is completed behind a foreground object, so the two objects overlap.

A common cause of ambiguity in images is when there is an accidental match of color across the edge of an object. An example is shown in Figure 2.10a: the man's face has similar color to the background and, in fact, the segmenter decided the man had a Pinocchio-like nose. The true background and the false head with a large nose overlap. As in the linguistic examples, there is only a "true" parse and the large nose parse should be rejected.

An example of the second is given by a square (or by many alpha-numeric characters). A square may be broken up into two pairs of parallel lines. A pair of parallel lines is a common reusable part in its own right, so we may parse the square as having two child nodes, each such a pair. But the square is also built up from 4 line pairs meeting in a right angle. Such pairs of lines also form common reusable parts. The two resulting parses are shown in Figure 2.10.b. One "solution" to this issue is to choose, once and for all, one of these as the preferred parse for a square. In analyzing an image, both parses may occur but, in order to give the consistent and determinate "square" label, one parse is chosen and the other parts representing partial structures are rejected.

"Joints" will be studied below: often two parts of the image are combined in characteristic geometric ways. For example, two thin rectangles may butt against each other and then form a compound part. But clearly, they share a small line segment which is common to both their boundaries: see Figure 2.10.c. If the parsing begins at the pixel level, such sharing between adjacent parts is almost inevitable. The simplest way to restore the treelike nature of the parse seems to be to duplicate the overlapping part. For example, an edge is often part of the structure on each side and it seems very natural to allocate to the edge two nodes — the edge attached to side 1 and the edge attached to side 2.

The most vision-specific case of overlap is caused by occlusion. Occlusion is seen in virtually every image. It can be modeled by the 2.1D sketch [103]. Mentally, humans (and presumably other visual animals) are quite aware that two complete objects exist in space but that certain parts of the two objects project to the same image pixels, with only one being visible. Here we consciously form duplicate image planes carrying the two objects: this is crucial when we actually want to use our priors to reconstruct as much as possible of the occluded object. It seems clear that the right parse for such objects should add extra leaves at the bottom to represent the occluded object. The new leaves carry colors, textures *etc.*, extrapolated from the visible parts of the object. Their occluded boundaries were what the gestalt school called *amodal* contours. The gestalt school demonstrated that people often make very precise predictions for such amodal contours.

When we talk about Spatial And-or Graph in the following Chapters, we will assume that the reusable parts do not overlap so that inclusion gives us a treelike parse structure. This simplifies immensely the computational algorithms. Future work may require dealing with diamonds more carefully (REF Geman).

2.7 Stochastic Grammar with Context

As will be seen in the next chapter, the image grammar we propose is an And-Or tree defined by a stochastic grammar but we further augment it to an And-Or graph by adding relations and contexts constraints as horizontal links. The resulting probabilistic models are defined on the And-Or graph to represent a stochastic context *sensitive* grammar for images.

A simple example of this in language, due to Mark, Miller and Grenander, augments the stochastic grammar models with word co-occurrence probabilities. Let $\omega = (\omega_1, \omega_2, \dots, \omega_n)$ be a sentence with n words, then bi-gram statistics counts the frequency $h(\omega_i, \omega_{i+1})$ and all word pairs, and therefore leads to a simple Markov chain model for the string ω :

$$p(\omega) = h(\omega_1) \prod_{i=1}^{n-1} h(\omega_{i+1}|\omega_i). \quad (2.15)$$

In [95], a probabilistic model was proposed to integrate parse tree model in (2.9) and the bi-gram model in (2.15) for the terminal string, by adding factors $h^*(\omega_{i+1}|\omega_i)$ and re-normalizing the probability:

$$p(\mathbf{pt}(\omega)) = \frac{1}{Z} h^*(\omega_1) \prod_{i=1}^{n-1} h^*(\omega_{i+1}|\omega_i) \cdot \prod_{j=1}^{n(\omega)} p(\gamma_j). \quad (2.16)$$

The factors are chosen so that the marginal probability on word pairs matches the given bi-gram model. Note that one can always rewrite the probability in a Gibbs form for the whole parse tree and strings,

$$p(\mathbf{pt}(\omega); \Theta) = \frac{1}{Z} \exp\left\{-\sum_{j=1}^{n(\omega)} \lambda(\gamma_j) - \sum_{i=1}^{n-1} \lambda(\omega_{i+1}, \omega_i)\right\}, \quad (2.17)$$

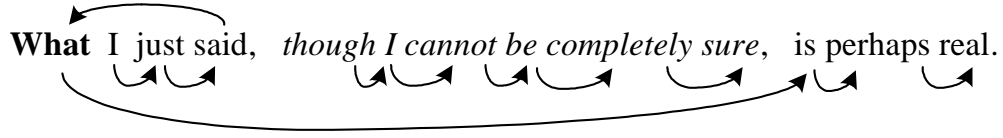


Figure 2.11: An English sentence with non-local 'next' relations shown by the arrows and the word 'what' is a joint to link two clauses. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

where $\lambda(\gamma_j) = -\log p(\gamma_j)$ and $\lambda(\omega_{i+1}, \omega_i) = -\log h^*(\omega_{i+1}|\omega_i)$ are parameters included in Θ . Thus the existence of the h^* is a consequence of the existence of exponential models matching given expectations.

However, the left-to-right sequence of words may not express the strongest contextual effects. There are non-local relations as the arrows in Figure 2.11 show. Firstly interjections mess up phrases in language. The italicized words in the sentence split the text flow. Thus the "next" relation in the bi-gram is not deterministically decided by the word order but has to be inferred. Secondly the word "what" is both the object of the verb "said" and the subject of the verb "is". It connects the two clauses together. Quite generally, all pronouns indicate long-range dependencies, link two reusable parts and carry context from one part of an utterance or text to another. In images, one shall see many different types of joints that combine parts of objects, such as butting, hinge, and various alignments that similarly link two reusable parts.

As we shall discuss in a later section, each node may have many types of relations in the way it interacts with other nodes. These relations are often hidden or cannot be deterministically decided and thus we shall represent these potential connections through some "address variables" associated with each node. The value of an address variable in a node ω_i is an index towards another node ω_j , and the node pair (ω_i, ω_j) observes a certain relation. These address variables have to be computed along with the parse tree in inference.

In vision, these non-local relations occur much more frequently. These relationships represent the spatial context at all levels of vision from pixels, primitives to parts, objects and scenes, and lead to various graphical models, such as Markov random fields. Gestalt organizations are popular examples in the middle level and low-level vision. For example, whenever a foreground object occludes part of a background object, with this background object being visible on both sides of the foreground one, these two visible parts of the background object constrain each other. Other non-local connections may reflect functional relations, such as object X is "supporting" object Y.

3

Spatial And-Or Graph

In this chapter, we study a stochastic and context-sensitive grammar—Spatial And-or Graph, which serves as a unified framework of representation, learning and recognition for a large number of object categories and scene understanding. The proposed method is a stochastic grammar embedded in an And-Or graph representation and this representation supports recursive top-down/bottom-up procedures for image parsing under Bayesian framework, making it convenient to scale up in complexity. The proposal grammar integrates three prominent representations in the literature: stochastic grammars for composition, Markov (or graphical) models for contexts, and sparse coding with primitives (wavelets). We start with discovering the comparison between image grammar and language and then the basic concepts: visual vocabulary, relations and configurations. Finally we shall arrive at the knowledge representation with And-or Graph.

3.1 Three New Issues in Image Grammars in Contrast to Language

As is revealed in last chapter, an image grammar should include two aspects: (i) The hierarchical structures (the grammar \mathcal{G}) which generate a large set of valid image *configurations* (*i.e.* the language $\mathbf{L}(\mathcal{G})$). This is especially important for modeling object categories with large intra-class structural variabilities. (ii) The context information which makes sure that the components in a configuration observe good spatial relationships between object parts, for example, relative positions, ratio of sizes, and consistency of colors. Both aspects encode important parts of our visual knowledge.

Going from 1D language grammars to 2D image grammars is non-trivial and requires a major leap in technology. Perhaps more important than anything else, one faces enormous complexity, although the principles are still simple. The following section summarizes three major differences (and difficulties) between the language grammars and image grammars.

The first huge problem is the loss of the left-to-right ordering in language. In language, every production rule $A \rightarrow \beta$ is assumed to generate a linearly ordered sequence of nodes β and following this down to the leaves, we get a linearly ordered sequence of terminal words. In vision, we have to replace the implicit links of words to their left and right neighbors by the edges of a more complex "region adjacency graph", *i.e.*, RAG. To make this precise, let the domain D of an image I have a decomposition $D = \cup_{k \in S} R_k$ into disjoint regions. Then we make a RAG with nodes $\langle R_i \rangle$ and edges $\langle R_k \rangle - \langle R_l \rangle$ whenever R_k and R_l are adjacent. This means we must *explicitly add horizontal edges to our parse tree to represent adjacency*. In a production rule $A \rightarrow \beta$, we no longer assume the nodes of β are linearly ordered. Instead, we should make β into a *configuration*, that is, a set of nodes from $V_N \cup V_T$ plus horizontal edges representing adjacency. We shall make this precise below.

Ideas to deal with the loss of left-to-right ordering have been proposed by the K.S. Fu school of "syntactic

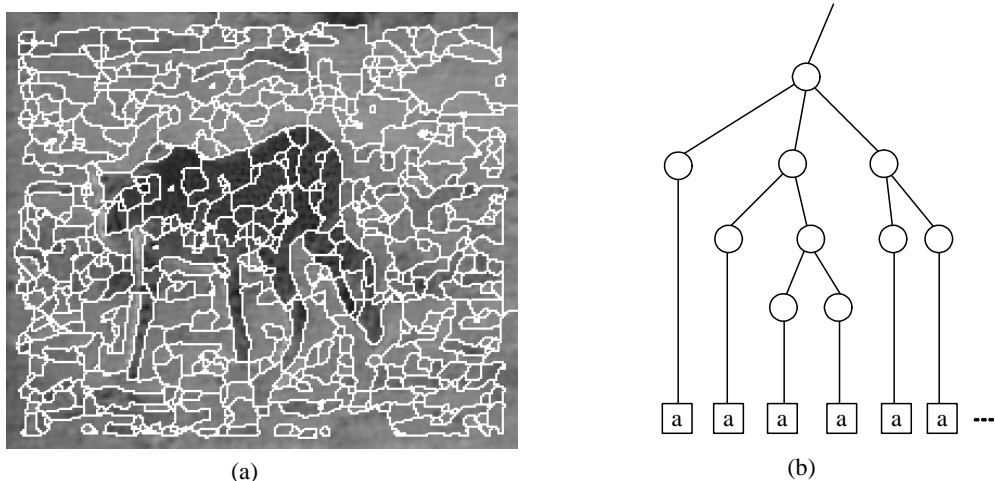


Figure 3.1: A cheetah and the background after local segmentation: both can be described by a RAG. Without the left-to-right order, if the regions are to be merged one at a time, they have a combinatorially explosive number of parse trees. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

pattern recognition" under the names "web grammars" and "plex grammars" [49], by Grenander in his pattern theory [75], and more recently by graph grammars for diagram interpretation in computer science [124]. These ideas have not received enough attention in vision. We need to study the much richer spatial relations for how objects and parts are connected. Making matters more complex, due to occlusions and other non-local groupings, non-adjacent spatial relations often have to be added in the course of parsing.

One immediate consequence of the lack of natural ordering is that a region has very ambiguous production rules. Let A be a region and a an atomic region, and let the production rules be $A \rightarrow aA \mid a$. A linear region $\omega = (a, a, a, \dots, a)$ has a unique parse graph in left-to-right ordering. With the order removed, it has a combinatorial number of parse trees. Figure 3.1 shows an example of parsing an image with a cheetah. It becomes infeasible to estimate the probability $p(\omega)$ by summing over all these parse trees in a grammar.

Therefore we must avoid these recursively defined grammar rules $A \rightarrow aA$, and treat the grouping of atomic regions into one large region A as a single computational step, such as the grouping and partitioning in a graph space [7]. Thus the probability $p(\omega)$ is assigned to each object as a whole instead of the production rules. In the literature, there are a number of hierarchic representations by an adaptive image pyramid, for example, the work by Rosenfeld and Hong in the early 80's [67], and the multi-scale segmentation by Galun *et al.* [50]. Though generic elements are grouped in these works, there are no explicit grammar rules. We shall distinguish such multi-scale pyramid representation from parse trees.

The second issue, unseen in language grammar, is the issue of image scaling [88, 162, 166]. It is a unique property of vision that objects appear at arbitrary scales in an image when the 3D object lies nearer or farther from the camera. You cannot hear or read an English sentence at multiple scales, but the image grammar must be a multi-resolution representation. This implies that the parse tree can terminate immediately at any node because no more detail is visible.

Figure 3.2 shows a human face in three levels from [174]. The left column shows face images at three resolutions. The middle column shows three configurations (graphs) of increasing detail and the right column shows the dictionaries (terminals) used at each resolution respectively. At a lower resolution, a face is represented by patches as a whole (for example, by principal component analysis) whereas at a middle

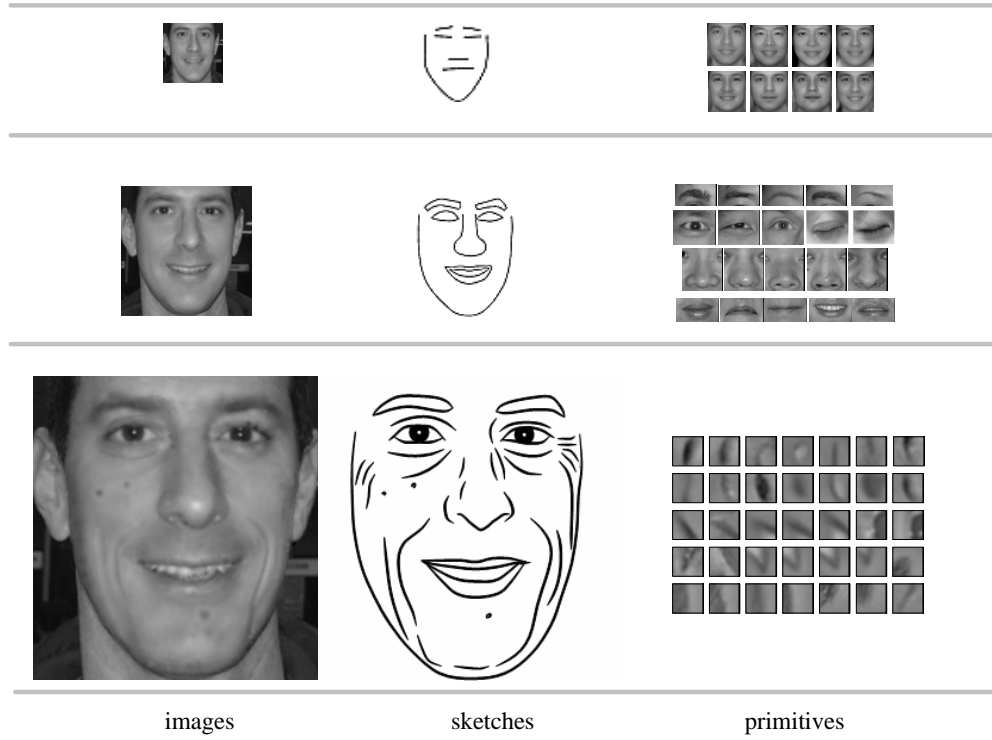


Figure 3.2: A face appears at three resolutions is represented by graph configurations in three scales. The right column shows the primitives used at the three levels. From Ref. [174].

resolution, it is represented by a number of parts. At a higher resolution, the face is represented by a sketch graph using smaller image primitives. The sketch graphs shown in the middle of Figure 3.2 expand with increasing resolution.

One can account for this by adding some termination rules to each non-terminal node, *e.g.*, each non-terminal node may exit the production for a low resolution case.

$$\forall A \in V_N, A \rightarrow \beta_1 | \cdots | \beta_{n(A)} | t_1 | t_2 | \cdot \quad (3.1)$$

where $t_1, t_2, \in V_T$ are image primitives or image templates for A at certain scales. For example, if A is a car, then t_1, t_2 are typical views (small patches) of the car at low resolution. As they are in low resolution, the parts of the cars are not very distinguishable and thus are not represented separately. The decompositions $\beta_i, i = 1, 2, \dots, n(A)$ represent the production rules for higher resolutions, so this new issue does not complicate the grammar design, except that one must learn the image primitives at multiple scales in developing the visual vocabulary.

The third issue with image grammars is that natural images contain a much wider spectrum of quite irregular local patterns than speech signals. Images not only have very regular and highly structured objects which could be composed by production rules, but also contain very stochastic patterns, such as clutter and texture which are better represented by Markov random field models. In fact, the spectrum is continuous. The structured and textured patterns can transfer from one to the other through continuous scaling [162, 169]. The two categories of models ought to be integrated more intimately and melded into a common model. This raises numerous challenges in modeling and learning at all levels of vision. For example, how do we decide when we should develop an image primitive (texon) for a specific element or use a texture description

(for example, a Markov Random Field)? How do we decide when we should group objects in a scene by a production rule or by a Markov random field for context? These questions shall be answered in the following.

3.2 Visual Vocabulary

3.2.1 The Hierarchical Visual Vocabulary – the "Lego Land"

In English dictionaries, a word not only has a few attributes, such as meanings, number, tense, and part of speech, but also a number of ways to connect with other words in a context. Sometimes the connections are so strong that compound words are created. For example, the word "apple" can be bound with "pine" or "Fuji" to the left, or "pie" and "cart" to the right. For slightly weaker connections, phrases are used. For instance, the word "make" can be connected with "something" using the prepositions "of" or "from", or connected with "somebody" through the prepositions "at" or "against". Thus a word is very much like a piece of Legos for building toy objects.

The bonds exist more explicitly and are much more necessary in the 2D image domain. We define the visual vocabulary as follows.

Definition I. Visual vocabulary

The visual vocabulary is a set of pairs, each consisting of an image function $\Phi_i(x, y; \alpha_i)$ and a set of $d(i)$ bonds (*i.e.*, its degree), to be eventually connected with other elements, which are denoted by a vector $\beta_i = (\beta_{i,1}, \dots, \beta_{i,d(i)})$. We think of $\beta_{i,k}$ as an address variable or pointer. α_i is a vector of attributes for (a) a geometric transformation, *e.g.* the central position, scale, orientation and plastic deformation, and (b) appearance, such as intensity contrast, profile or surface albedo. In particular, α_i determines a domain $\Lambda_i(\alpha_i)$ and Φ_i is then defined for $(x, y) \in \Lambda_i$ with values in R (a gray-valued template) or R^3 (a color template). Often each $\beta_{i,k}$ is associated with a subset of the boundary of $\Lambda_i(\alpha_i)$. The whole vocabulary is thus a set:

$$\Delta = \{(\Phi_i(x, y; \alpha_i), \beta_i) : (x, y) \in \Lambda_i(\alpha_i) \subset \Lambda\}, \quad (3.2)$$

where i indexes the type of the primitives.

The conventional wavelets, Gabor image bases, image patches, and image fragments are possible examples of this visual vocabulary except that they don't have bonds. As an image grammar must adopt a multi-resolution representation, the elements in its vocabulary represent visual concepts at all levels of abstraction and complexity. In the following, we introduce some examples of the visual vocabulary at the low, middle, and high levels respectively.

3.2.2 Image Primitives

As we have seen in Part I, Julesz conjectured that textons (blobs, bars, terminators, crosses) are the atomic elements in the early stage of visual perception for local structures [72]. Marr extended Julesz's texton concept to image primitives which he called "symbolic tokens" in his primal sketch representation [1]. An essential criterion in selecting a dictionary in low-level vision is to ensure that they are parsimonious and sufficient in representing real-world images, and more importantly they should have the necessary structures

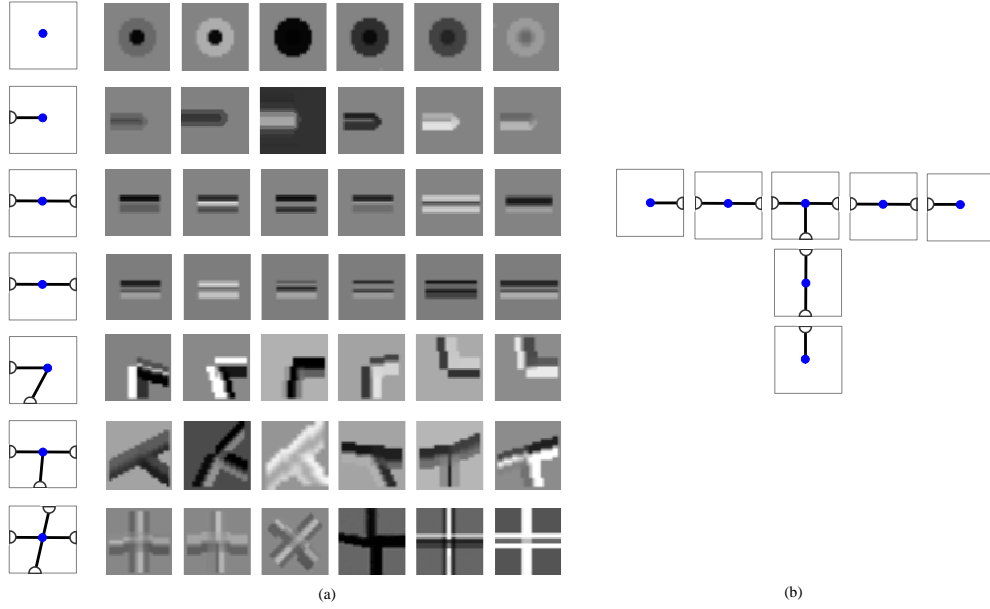


Figure 3.3: Low level visual vocabulary – image primitives. (a). Some examples of image primitives: blobs, terminators, edges, ridges, "L"-junctions, "T"-junction, and cross junction *etc.* These primitives are the elements for composing a bigger graph structure at the upper level of the hierarchy. (b) is an example of composing a big "T"-shape image using 7 primitives. From Ref. [185] with permission of Now Publishers Inc, © 2007.

to allow composition into higher-level parts. An example of dictionary of image primitives is the primal sketch(see Part I Chapter X for detail).

As illustrated in Figure 3.3.(a), an image primitive is a small image patch with a degree d connections or bonds which are illustrated by the half circles. The primitives are called blobs, terminators, edges or ridges, "L"-junctions, "T"-junctions, and cross junctions for $d = 0, 1, 2, 3, 4$ respectively. Each primitive has a number of attributes for its geometry and appearance. These primitives can be used to represent images. For example in Figure 3.4, the boundaries of the two rectangles are covered by 4 "T"-junctions, 8 "L"-junctions, and 20 step edges. We denote the domain covered by an image primitive Φ_i^{sk} by $\Lambda_{sk,i}$, and the pixels covered by these primitives, which are called the "sketchable part" in [57], are denoted by

$$\Lambda_{sk} = \bigcup_{i=1}^{n_{sk}} \Lambda_{sk,i}. \quad (3.3)$$

The image I on Λ_{sk} is denoted by I_{sk} and is modeled by the image primitives through their intensity profiles. Let ϵ be the residual noise.

$$\mathbf{I}_{sk}(x, y) = \Phi_i^{sk}(x, y; \alpha_i, \beta_i) + \epsilon(x, y), \quad (x, y) \in \Lambda_{sk,i}, \quad i = 1, 2, \dots, n_{sk}. \quad (3.4)$$

The remaining pixels are flat or stochastic texture areas, called non-sketchable, and are clustered into a few homogeneous texture areas

$$\Lambda_{nsk} = \Lambda \setminus \Lambda_{sk} = \bigcup_{j=1}^{n_{nsk}} \Lambda_{nsk,j}. \quad (3.5)$$

They can be reconstructed through Markov random field models conditional on \mathbf{I}_{sk} ,

$$\mathbf{I}_{nsk,j} | \mathbf{I}_{sk} \sim p(\mathbf{I}_{nsk} | \mathbf{I}_{sk}; \Theta_j). \quad (3.6)$$

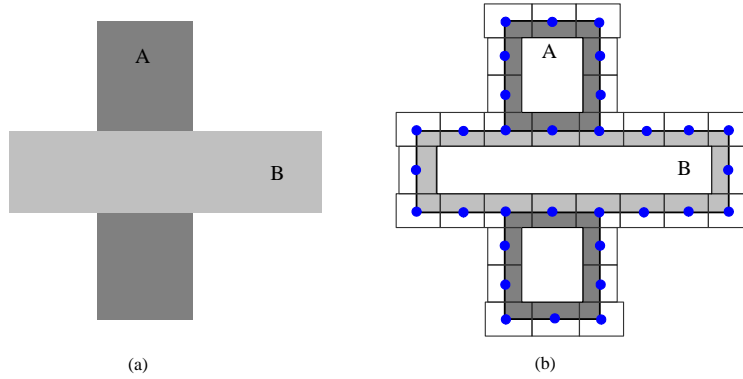


Figure 3.4: An illustrative example for composing primitives into a graph configuration. (a) is a simple image, and (b) is a number of primitives represented by rectangles which cover the structured parts of the image. The remaining part of the image can be reconstructed through simple heat diffusion. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

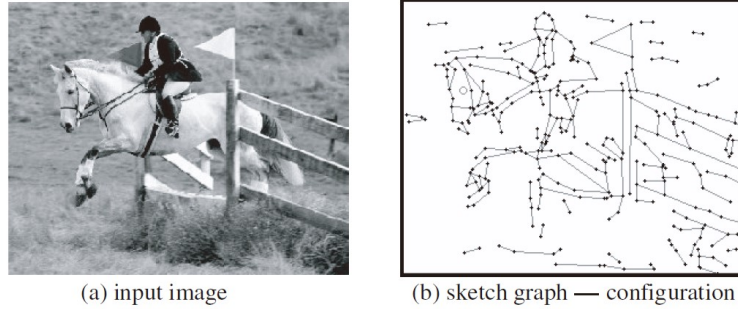


Figure 3.5: An example of the primal sketch model. (a) An input image I . (b) The sketch graph – configuration computed from the image I . Reproduced from Ref. [57] with permission of Elsevier, © 2007.

Θ_j is a vector-valued parameter for the Gibbs model, for example, the FRAME model [186].

To summarize, image primitives can compose a planar attribute graph configuration to generate the structured part of the image. These primitives are transformed, warped, and aligned to each other to have a tight fit. Adjacent primitives are connected through their bonds. The explicit use of bonds distinguishes the image primitives from other basic image representations, such as wavelets and sparse image coding [94, 104] mentioned before, and other image patches and fragments in the recent vision literature [156]. The bonds encode the topological information, in addition to the geometry and appearance, and enable the composition of bigger structures in the hierarchy.

3.2.3 Basic Geometric Groupings

If by analogy, image primitives are like English letters or phonemes in language, then one naturally wonders what are the visual words and visual phrases. This is the central question addressed by the gestalt school of psychophysicists [73, 182]. One may summarize their work by saying that the geometric relations of alignment, parallelism and symmetry, especially as created by occlusions, are the driving forces behind the grouping of lower-level parts into larger parts. A set of these composite parts is shown in Figure 3.6 and briefly described in the caption.

It is important to realize that these groupings occur at every scale. Many of them occur in local groupings

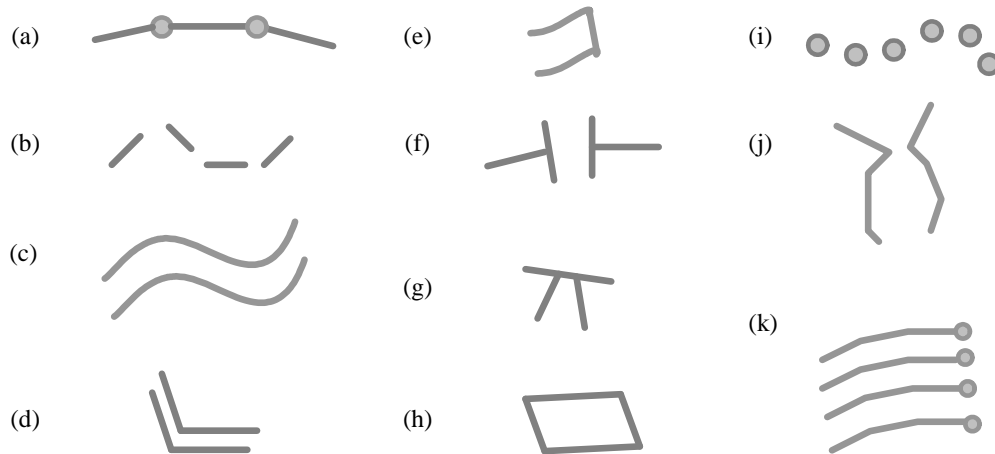


Figure 3.6: Middle level visual vocabulary: common groupings found in images. (a) extended curves, (b) curves with breaks and imperfect alignment, (c) parallel curves, (d) parallels continuing past corners, (e) ends of bars formed by parallels and corners, (f) curves continuing across paired T-junctions (the most frequent indication of occlusion), (g) a bar occluded by some edge, (h) a square, (i) a curve created by repetition of discrete similar elements, (j) symmetric curves and (k) parallel lines ending at terminators forming a curve. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

containing as few as 2 to 8 image primitives as in the previous section. We will call these "graphlets" [167]. But extended curves, parallels and symmetric structures may span the whole image. Notably, symmetry is always a larger scale feature but one occurring very often in nature (*e.g.* in faces) and which is highly detectable by people even in cluttered scenes. Parallel lines also occur frequently in nature, *e.g.* in tree trunks. The occlusion clue shown in Figure 3.6 is especially important because it is not only common but is the strongest clue in a static 2D image to the 3D structure of the scene. Moreover, it implies the existence of an "amodal" or occluded contour representing the continuation of the left and right edges behind the central bar. This necessitates a special-purpose algorithm to be discussed below. Figure 3.7 shows an image with its primal sketch on the right side with its graphlets shown in dark line segments.

These graphlets are learned through clustering and binding the image primitives in a way discussed at the beginning of this book. Each cluster in this space is an equivalence class subject to an affine transform, some deformation, as well as minor topological editing. These graphlets are generic 2D patterns, and some of them could be interpreted as object parts.

3.2.4 Parts and Objects

If one is only interested in certain object categories segmented from the background, such as bicycles, cars, iPods, chairs or clothes, the dictionary will be object parts. Although these object parts are significant within each category or reusable by a few categories, their overall frequency is low and they are often rare events in a big database of real-world images. Thus the object parts are less significant as contributors to lowering image entropy than the graphlets presented above, and the latter are, in turn, less entropically significant than the image primitives at the lower level.

We take one complex object category – clothes as an example. Figure 3.9 shows how a shirt is composed of three parts: a collar, and a left and a right short sleeves. In this figure, each part is represented by an attribute graph with open bonds, like the graphlets. For example, the collar part has 5 bonds, and the two short sleeves have 3 bonds to be connected with the arms and collar. By decomposing a number of instances

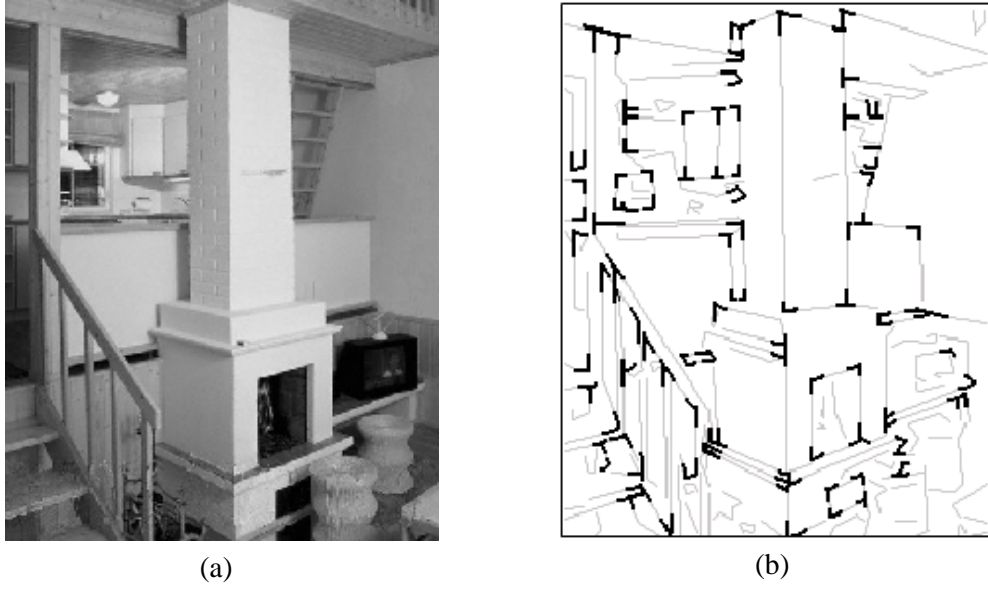


Figure 3.7: An example of graphlets in natural image. The graphlets are highlighted in the primal sketch. These graphlets can be viewed as larger pieces of lego. From Ref. [51].

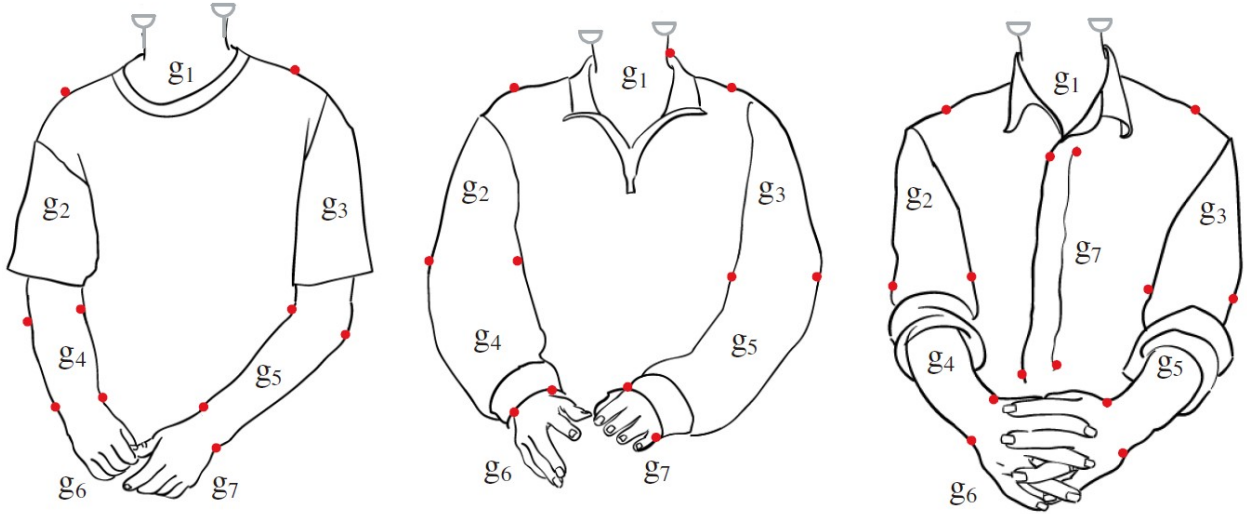


Figure 3.8: Three novel configurations composed of 6,5,7 sub-templates in the categories, respectively. The bonds are shown by the red dots. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

in the clothes category together with upper body and shoes, one can obtain a dictionary of parts. Figure 3.10 shows some examples for each category.

Thus we denote the dictionary by

$$\Delta_{\text{cloth}} = \{(\Phi_i^{\text{cloth}}(x, y; \alpha_i), \beta_i) : \forall i, \alpha_i, \beta_i.\} \quad (3.7)$$

As before, Φ_i^{cloth} is an image patch defined in a domain Λ_i^{cloth} which does not have to be compact or

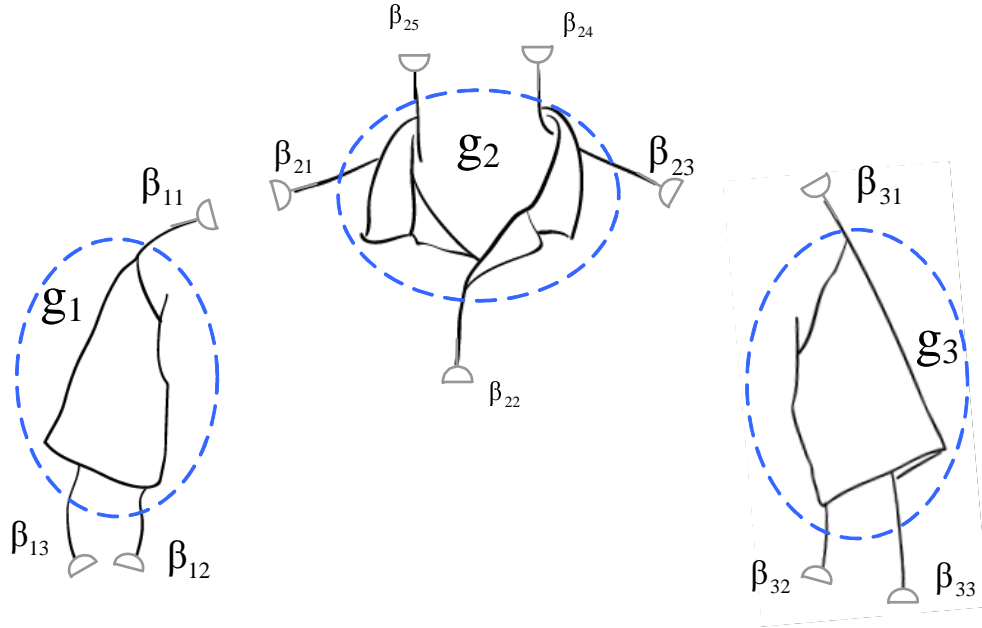


Figure 3.9: High level visual vocabulary – the objects and parts. We show an example of upper body clothes made of three parts: a collar, a left and a right short sleeves. Each part is again represented by a graph with bonds. A vocabulary of part for human clothes is shown in Figure 3.10. From Ref. [16].

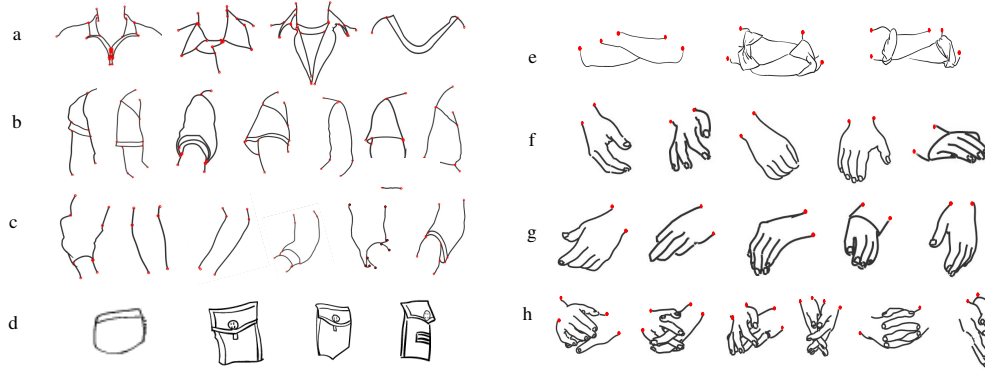


Figure 3.10: The dictionary of object parts for cloth and body components. Each element is a small graph composed of primitives and graphlets and has open-bonds for collecting with other parts. Modified from Ref. [16].

connected. α_i controls the geometric and photometric attributes, and $\beta_i = (\beta_{i1}, \beta_{i2}, \dots, \beta_{id(i)})$ is a set of open bonds. These bonds shall be represented as address variables that point to other bonds.

In fact, the object parts defined above are not much different from the dictionaries of image primitives or graphlets, except that they are bigger and more structured. Indeed they form a continuous spectrum for the vision vocabulary from low to high levels of vision.

By analogy, each part is like a class in object-oriented programming, such as C++. The inner structures of the class are encapsulated, only the bonds are visible to other classes. These bonds are used for communication between different object instances.

In the literature, Biederman [8] proposes a set of "geons" as 3D object elements, which are generalized cylinders for representing 3D man-made objects. In practice, it is very difficult to compute these generalized cylinders from images. In comparison, we adopt a view-based representation for the primitives, graphlets and parts, which can be inferred relatively reliably.

3.3 Relations and configurations

While the hierarchical visual vocabulary represents the vertical compositional structures, the relations in this section represent the horizontal links for contextual information between nodes in the hierarchy at all levels. The vocabulary and relations are the ingredients for constructing a large number of image configurations at various levels of abstractions. The set of valid configurations constitutes the language of an image grammar.

3.3.1 Relations

We start with a set of nodes $V = \{A_i : i = 1, 2, \dots, n\}$ where $A_i = (\Phi_i(x, y; \alpha_i), \beta_i) \in \Delta$ is an entity representing an image primitive, a grouping, or an object part as defined in the previous section. A number of spatial and functional relations must be defined between the nodes in V to form a graph with colored edges where the color indexes the type of relation.

Definition II Attributed Relation

A binary relation defined on an arbitrary set S is a subset of the product set $S \times S$

$$\{(s, t)\} \subset S \times S. \quad (3.8)$$

An attributed binary relation is augmented with a vector of attributes γ and ρ ,

$$E = \{(s, t; \gamma, \rho) : s, t \in S\}, \quad (3.9)$$

where $\gamma = \gamma(s, t)$ represents the structure that binds s and t , and $\rho = \rho(s, t)$ is a real number measuring the compatibility between s and t . Then $\langle S, E \rangle$ is a graph expressing the relation E on S . A k -way attributed relation is defined in a similar way as a subset of S^k .

There are three types of relations of increasing abstraction for the horizontal links and context. The first type is the bond type that connects image primitives into bigger and bigger graphs. The second type includes various joints and grouping rules for organizing the parts and objects in a planar layout. The third type is the functional and semantic relation between objects in a scene.

Relation type 1: bonds and connections. For a set of nodes $V = \{A_i : i = 1, 2, \dots, n\}$ defined above, each node $A_i \in V$ has a number of open bonds $\{\beta_{ij} : j = 1, 2, \dots, n(i)\}$ shown by the half disks in the previous section. We collect all these bonds as a set,

$$S_{\text{bond}} = \{\beta_{ij} : i = 1, 2, \dots, n, j = 1, 2, \dots, n(i)\}. \quad (3.10)$$

Two bonds β_{ij} and β_{kl} are said to be connected if they are aligned in position and orientation. Therefore the bonding relation is a set of pairs of bonds with attributes:

$$E_{\text{bond}}(S) = \{(\beta_{ij}, \beta_{kl}; \gamma, \rho)\} \quad (3.11)$$

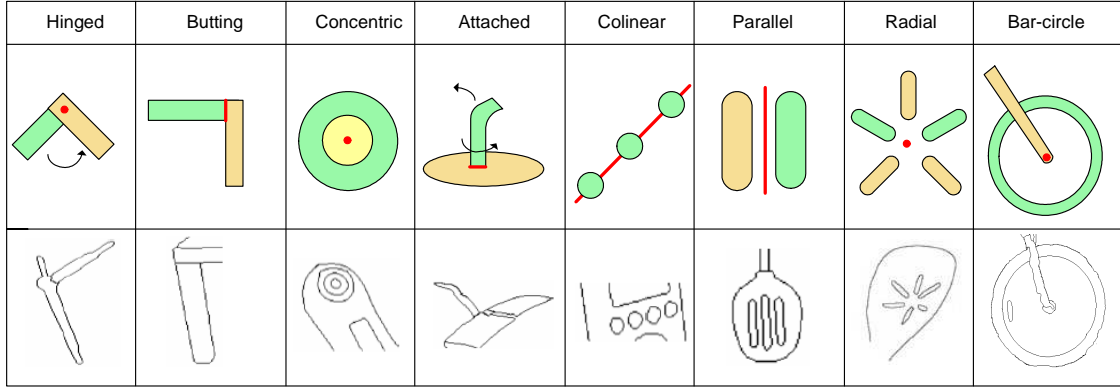


Figure 3.11: Examples of spatial relations for binding object parts. The red dots or lines are the attributes $\gamma(s, t)$ of joint relation (s, t) which form the ‘glue’ in this relation. From Ref. [116].

where $\gamma = (x, y, \theta)$ denote the position and orientation of the bond. The latter is the tangent direction at the bond for the two connected primitives. ρ is a function to check the consistency of intensity profile or color between two connected primitives.

A trivial example is the image lattice. The primitives $A_i, i = 1, \dots, |\Lambda|$ are the pixels. Each pixel has 4 bonds $\beta_{ij}, j = 1, 2, 3, 4$. Then $E_{\text{bond}}(S)$ is the set of 4-nearest neighbor connections. In this case, $\gamma = \text{nil}$ is empty, and ρ is a pair-clique function for the intensities at pixels i and j . Figures 3.6 and 3.9 show more examples of bonds for composing graphlets from primitives, and composing clothes from parts. Very often people use graphical models, such as templates, with fixed structures where the bonds are decided deterministically and thus become transparent. In the next subsection, we shall define the bonds as random variables to reconfigure the graph structures.

Relation type 2: joints and junctions. When image primitives are connected into larger parts, some spatial and functional relations must be found. Besides its open bonds to connect with others, usually its immediate neighbors, a part may be bound with other parts in various ways. The gestalt groupings discussed in the previous section are the best examples: parts can be linked over possibly large distances by being collinear, parallel or symmetric. To identify these groupings, connections must be created flagging this non-accidental relationship. Figure 3.11 displays some typical relations of this type between object parts.

Some of these relations also contribute to 3D interpretations. For example, an ellipse is a part that has multiple possible compositions. If it is recognized as a bike wheel, its center can function as an axis and thus can be connected to the tip of a bar (see the rightmost of Figure 3.11). It could also be the rim of a tea cup, and then the two ends of its long axis will be joined to a pair of parallel lines to form a cylinder. In Figure 2.11 we discussed a phenomenon that occurred in language where the word "what" is shared by two clauses. Similarly, we have many such joints in images, such as hinge joints, and butt joints.

As Figure 3.11 shows, two parts can be hinged at a point. For example, two hands of a clock have a common axis. For a set of parts in an image $S = V$, the hinge relation is a set

$$E_{\text{hinge}}(S) = \{(A_i, A_j; \gamma(A_i, A_j), \rho(A_i, A_j))\}. \quad (3.12)$$

Here γ is the hinge point and $\rho = \text{nil}$. In a butting relation, $\gamma(A_i, A_j)$ represents the line segment(s) shared by the two parts. The line segment is shown in red in Figure 3.11. Sometimes, two parts may share two line segments. For example, the handle of a teapot or cup share two line segments with the body.

Relation type 3: object interactions and semantics. When letters are grouped into words, semantic meanings emerge. When parts are grouped into objects, semantic relations are created for their interactions.

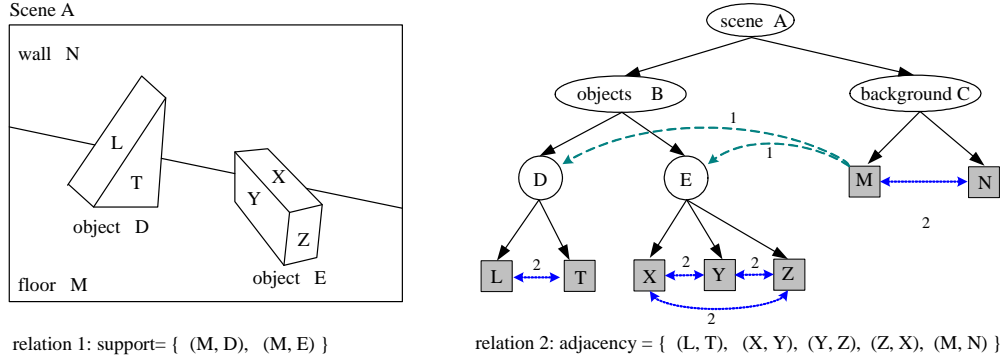


Figure 3.12: A parser tree for a block world from Ref. [49]. The ellipses represents non-terminal nodes and the squares are for terminal nodes. The parse tree is augmented into a parse graph with horizontal connections for relations, such as one object supporting the other, or two adjacent objects sharing a boundary.

Very often these relations are directed. For example, the occluding relation is a viewpoint dependent binary relation between object or surfaces, and it is important for figure-ground segregation. A view point independent relation is a supporting relation. A simple example is shown in Figure 3.12. Let $S = V$ be a set of objects,

$$E_{\text{supp}} = \{ \langle M, D \rangle, \langle M, E \rangle \} \quad (3.13)$$

$$E_{\text{occlld}} = \{ \langle D, M \rangle, \langle E, M \rangle, \langle D, N \rangle, \langle E, N \rangle \}. \quad (3.14)$$

The $\langle \rangle$ represents directed relation and the attributes γ, ρ are omitted. There are other functional relations among objects in a scene. For example, a person A is eating an apple B $E_{\text{edible}}(S) = \{ \langle A, B \rangle \}$, and a person is riding a bike $E_{\text{ride}}(S) = \{ \langle A, C \rangle \}$. These directed relations usually are partially ordered.

It is worth mentioning that the relations are dense at low level, such as the bonds, in the sense that the size $|E(S)|$ is in the order of $|S|$, and that they become very sparse (or rare) and diverse at high level. At the high level, we may find many interesting relations but each relation may only have a few occurrences in the image.

3.3.2 Configurations

So far, we have introduced the visual dictionaries and relations at various levels of abstractions. The two components are integrated into what we call the visual configuration in the following.

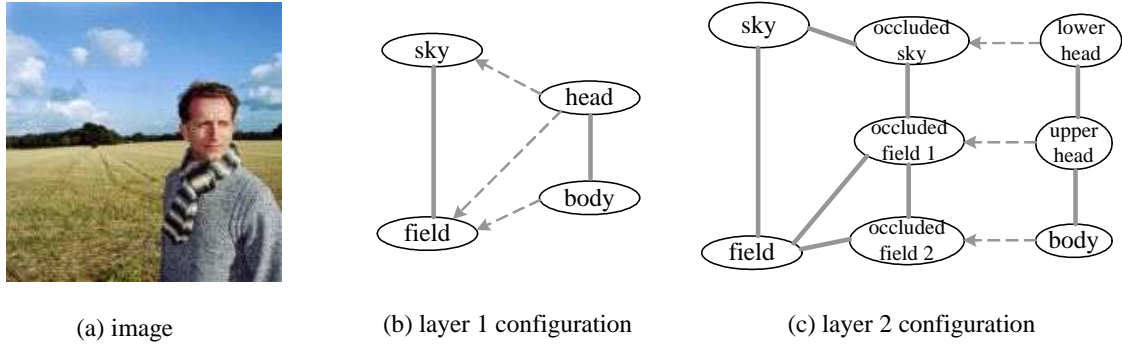


Figure 3.13: An illustration of scene configuration. (a) is a scene of a man in a field. (b) is the graph for the highest level configuration $\mathcal{C} = \langle V, E \rangle$, V is the set of 4 objects $\{sky, field, head, body\}$ and $E = E_{adj} \cup E_{occlude}$ includes two relations: "adjacency" (solid lines) and "occlusion" (dotted arrows). (c) is the configuration at an intermediate level in which the occlusion relation is unpacked: now the dotted arrows indicate two *identical* sets of pixels but on separate layers. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

Definition III. Configuration

A configuration \mathcal{C} is a spatial layout of entities in a scene at a certain level of abstraction. It is a one layer graph, often flattened from hierarchic representation,

$$\mathcal{C} = \langle V, E \rangle. \quad (3.15)$$

$V = \{A_i, i = 1, 2, \dots, n\}$ is a set of attributed image structures at the same semantic level, such as primitives, parts, or objects and E is a relation. If V is a set of sketches and $E = E_{bonds}$, then \mathcal{C} is a primal sketch configuration. If E is a union of several relations $E = E_{r_1} \cup \dots \cup E_{r_k}$, which often occurs at the object level, then \mathcal{C} is called a "mixed configuration". For a generative model, the image on a lattice is the ultimate "terminal configuration", and its primal sketch is called the "pre-terminal configuration". Note that E will close some of the bonds in V and leave others open; thus we may speak of the open bonds in a configuration.

We briefly present examples of configurations at three levels.

Firstly, for early vision, the scene configuration \mathcal{C} is a primal sketch graph where V is a set of image primitives with bonds and $E = E_{bonds}$ is the bond relation. For example, Figure 3.4.(b) illustrates a configuration for a simple image in Figure 3.4.(a), and Figure 3.5.(b) is a configuration for the image in Figure 3.5.(a). These configurations are attributed graphs because each primitive v_i is associated with variables α_i for its geometric properties and photometric appearance. The primal sketch graph is a parsimonious "token" representation in Marr's words [1], and thus it is a crucial stage connecting the raw image signal and the symbolic representation above it. It can reconstruct the original image with perceptually equivalent texture appearance.

Secondly, for the parts to object level, Figure 3.8 displays three possible upper body configurations composed of a number of clothes' parts shown in Figure 3.10. In these examples, each configuration \mathcal{C} is a graph with vertices being 6-7 parts and $E = E_{bond}$ is a set of bonds connecting the parts, as it was shown in Figure 3.9.

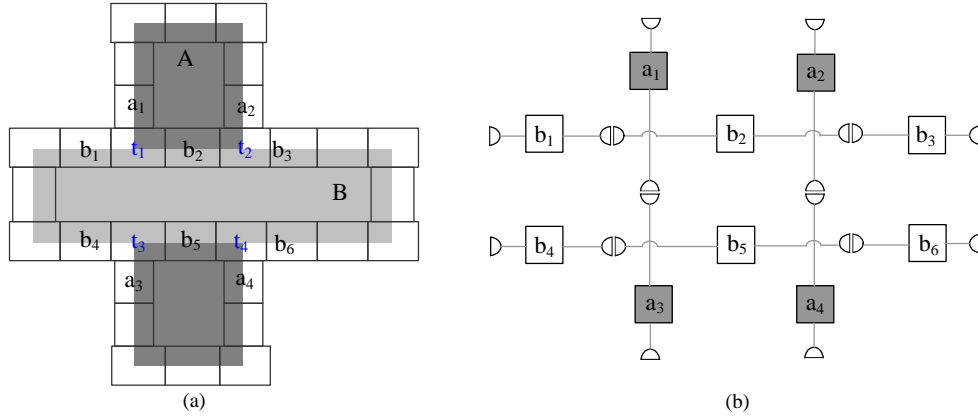


Figure 3.14: (a) A primal sketch configuration for a simple image. It has four primitives for "T"-junctions – t_1, t_2, t_3, t_4 . It is a planar graph formed by bonding the adjacent primitives. (b) A layered (2.1D sketch) representation with two occluding surfaces. The four "T"-junctions are broken. The bonds are reorganized. a_1 is bonded with a_3 , and a_2 is bonded with a_4 . From Ref. [102].

Thirdly, Figure 3.13 (a) and (b) illustrate a scene configuration at the highest level of abstraction. V is a set of objects, and E included two relations: an "adjacency" relation in solid lines

$$E_{\text{adj}} = \{(sky, field), (head, body)\}, \quad (3.16)$$

and a directed "occlusion" relation in dotted arrows,

$$E_{\text{contain}} = \{< head, sky >, < head, field >, < body, field >\}. \quad (3.17)$$

In summary, the image grammar which shall be presented in the following section is also called a "layered grammar". That is, it can generate configurations as its "language" at different levels of detail.

3.3.3 The Reconfigurable Graphs

In vision, the configurations are inferred from images. For example, in a Bayesian framework, the graph $\mathcal{C} = \langle V, E \rangle$ will not be pre-determined but reconfigurable on-the-fly. That is, the set of vertices may change, so does the set of edges (relations). Therefore the configurations must be made flexible to meet the demand of various visual tasks. Figure 3.14 shows such an example.

On the left of the figure is a primal sketch configuration \mathcal{C}_{sk} for the simple image shown in Figure 3.4. This is a planar graph with 4 "T"-junctions. In this configuration, two adjacent primitives are connected by the bond relation E_{bond} . The four "T"-junctions are then broken in the right configuration, which is called the 2.1D sketch [102] and denoted by $\mathcal{C}_{2.1sk}$. The bonds are reorganized with a_1 being connected with a_3 and a_2 with a_4 . $\mathcal{C}_{2.1sk}$ includes two disjoint subgraphs for the two rectangles in two layers. From this example, we can see that both the vertices and the bonds must be treated as random variables. Figure 3.15 shows a real application of this sort of reconfiguration in computing a 2.1D sketch from a 2D primal sketch. This example is from [57]. It decomposes an input image in 3.15.(a) into three layers in 3.15.(d), (e) and (f), found after reconfiguring the bonds by completing the contours (red line segments in 3.15.(b) and (c)) behind and filling-in the occluded areas using the Markov random field region descriptor in the primal sketch model. From the point of view of parse structures, we need to add new nodes to represent the extra layers present

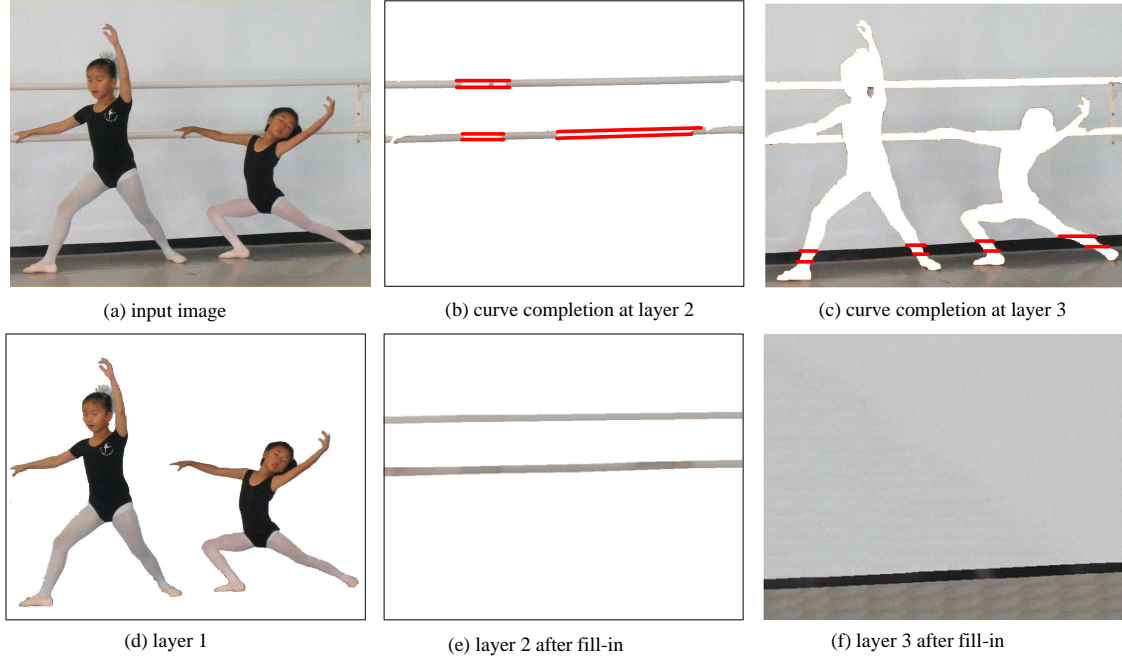


Figure 3.15: From a 2D sketch to a 2.1D layered representation by reconfiguring the bond relations. (a) is an input image from which a 2D sketch is computed. This is transferred to a 2.1D sketch representations with three layers shown in (d), (e) and (f) respectively. The inference process reconfigures the bonds of the image primitives shown in red in (b) and (c). From Ref. [185] with permission of Now Publishers Inc, © 2007.

behind the observed surfaces together with "occluded by" relations. This is illustrated by Figure 3.13, (c). This is a configuration which has duplicated three regions to represent missing parts of the background layer.

A mathematical model for the reconfigurable graph is called the *mixed Markov model* in [48]. In a mixed Markov model, the bonds are treated as nodes. Therefore, the vertex set V of a configuration has two types of nodes – $V = V_x \cup V_a$. V_x includes the usual nodes for image entities, and V_a is a set of address nodes, for example, the bonds. The latter are like the pointers in the C language. These address nodes reconfigure the graphical structure and realize non-local relations. It was shown that a probability model defined on such reconfigurable graphs still observes a suitable form of the Hammersley-Clifford theorem and can be simulated by Gibbs sampler.

By analogy to language, the bonds in this example correspond to the arrows in the English sentence discussed in Figure 2.11 for non-local context. As there are many possible (bond, joint, functional, and semantic) relations, each image entity (primitives, parts, objects) may have many random variables as the "pointers". Many of them could be empty, and will be instantiated in the inference process.

3.4 Parse Graph for Objects and Scenes

In this section, we define parse graphs as image interpretations. Then we will show in the next section that these parse graphs are generated as instances by an And-Or graph. The latter is a general representation that embeds the image grammar.

Recall that in Section 2.3 a language grammar is a 4-tuple $\mathcal{G} = (V_N, V_T, \mathbf{R}, S)$, and that a sentence ω is

derived (or generated) by a sequence of production rules from a starting symbol S ,

$$S \xrightarrow{\gamma_1, \gamma_2, \dots, \gamma_{n(\omega)}} \omega \quad (3.18)$$

These production rules form a parse tree for ω ,

$$\mathbf{pt}(\omega) = (\gamma_1, \gamma_2, \dots, \gamma_{n(\omega)}). \quad (3.19)$$

For example, Figure 2.9 shows two possible parse trees for a sentence "I saw the man with the telescope".

This grammar is a generative model, and the inference is an inverse process that computes a parse tree for a given sentence as its interpretation or one of its best interpretations. Back to image grammars, a configuration \mathcal{C} is a flat attributed graph corresponding to a sentence ω , and a parse tree \mathbf{pt} is augmented to a parse graph \mathbf{pg} by adding horizontal links for various relations. In previous sections, Figure 3.12.(b) has shown a parse graph for a block work scene.

In the following, we define a parse graph as an interpretation of image.

Definition IV. Parse graph

A *parse graph* \mathbf{pg} consists of a hierarchic parse tree (defining "vertical" edges) and a number of relations E (defining "horizontal edges"):

$$\mathbf{pg} = (\mathbf{pt}, E). \quad (3.20)$$

The parse tree \mathbf{pt} is also an And-tree whose non-terminal nodes are all And-nodes. The decomposition of each And-Node A into its parts is given by a production rule which now produces not a string but a configuration:

$$\gamma : A \rightarrow \mathcal{C} = \langle V, E \rangle. \quad (3.21)$$

A production should also associate the open bonds of A with open bonds in \mathcal{C} . The whole parse tree is a sequence of production rules

$$\mathbf{pt}(\omega) = (\gamma_1, \gamma_2, \dots, \gamma_n). \quad (3.22)$$

The horizontal links E consists of a number of directed or undirected relations among the terminal or non-terminal nodes, such as bonds, junctions, functional and semantic relations,

$$E = E_{r_1} \cup E_{r_2} \cup \dots \cup E_{r_k}. \quad (3.23)$$

A parse graph \mathbf{pg} , when collapsed, produces a series of flat configurations at each level of abstraction/detail,

$$\mathbf{pg} \implies \mathcal{C}. \quad (3.24)$$

Depending on the type of relation, there may be special rules for producing relations at a lower level from higher-level relations in the collapsing process. The finest configuration is the image itself in which every pixel is explained by the parse graph. The next finest configuration is the primal sketch graph.

The parse graph, augmented with spatial context and possible functional relations, is a comprehensive interpretation of the observed image \mathbf{I} . The task of image parsing is to compute the parse graph from input image(s). In the Bayesian framework, this is to either maximize the posterior probability for an optimal solution,

$$ParsingGraph^* = \arg \max p(ParsingGraph|\mathbf{I}), \quad (3.25)$$

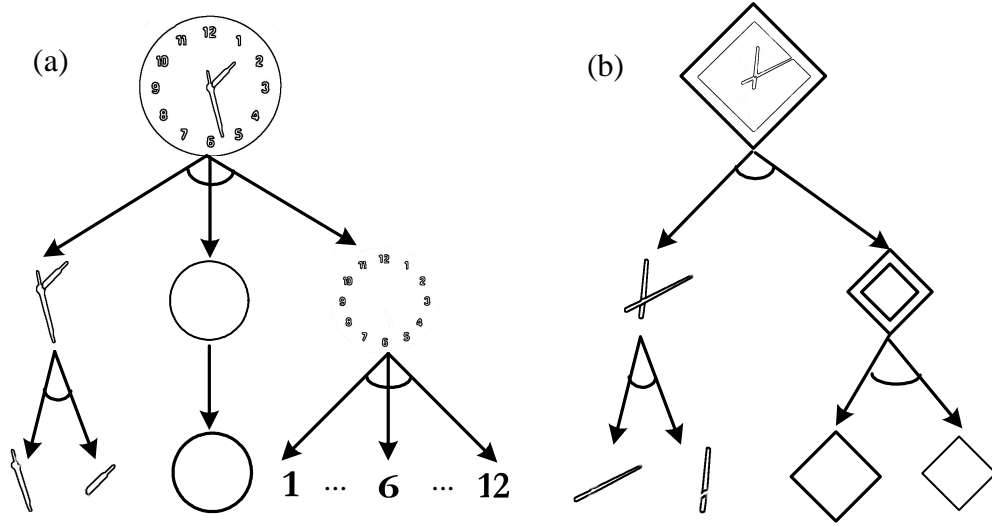


Figure 3.16: Two parse graph examples for clocks which are generated from the And-Or-graph in Figure 3.18. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

or sampling the posterior probability for a set of distinct solutions,

$$ParsingGraph_i : i = 1, 2, \dots, K \sim p(ParsingGraph|\mathbf{I}). \quad (3.26)$$

Object instances in the same category may have very different configurations and thus distinct parse graphs. Figure 3.16 displays two parse graphs for two clock instances. It has three levels and the components are connected through three types of relations: the hinge joint to connect clock hands, a co-centric relation to align the frames, and a radial relation to align the numbers.

As it was mentioned in Chapter 3.1, objects appear at arbitrary scales in images. As shown in Figure 3.2, a face can be decomposed into facial elements at higher resolution, and it may terminate as a whole face for low resolution. Therefore, one remarkable property that distinguishes an image parse graph is that a parse graph may stop at any level of abstraction, while the parse tree in language must stop at the word level. This is the reason for defining visual vocabulary at multiple levels of resolution, and defining the image grammar as a layered grammar.

3.5 Knowledge Representation with And-Or Graph

This section addresses the central idea of this chapter – developing a consistent representation framework for the vast amount of visual knowledge at all levels of abstraction. The proposed representation is the And-Or graph embedding image grammars. The And-Or graph representation was first explicitly used in [16] for representing and recognizing a complex object category of clothes.

3.5.1 And-Or graph

While a parse graph is an interpretation of a specific image, an And-Or graph embeds the whole image grammar and contains all the valid parse graphs. Before introducing the And-Or graph, we revisit the origin of grammar and its Chomsky formulation in Sections 2.2 and 2.3.

Firstly, we know each production rule in the SCFG (Stochastic Context-Free Grammar) can be written as

$$A \rightarrow \beta_1 | \beta_2 \cdots | \beta_{n(A)}, \quad \text{with } A \in V_N, \beta \in (V_N \cup V_T)^+. \quad (3.27)$$

Therefore each non-terminal node A can be represented by an Or-node with $n(A)$ alternative structures, each of which is an And-node composed of a number of substructures. For example, the following rule is represented by a two-level And-Or tree in Figure 2.5.

$$A \rightarrow a \cdot b | c \cdot c; \quad \rho | (1 - \rho). \quad (3.28)$$

The two alternatives branching at the Or-node are assigned probabilities $(\rho, 1 - \rho)$. Thus an SCFG can be understood as an And-Or tree.

Secondly, we have shown in Figure 2.6 that a small And-Or tree can produce a combinatorial number of configurations – called its language. To represent contextual information in the following, we augment the And-Or tree into an And-Or graph producing a context-sensitive image grammar.

In [183], it has been shown that any visual pattern can be conceptualized as a statistical ensemble that observes a certain statistical description. For a complex object pattern, its statistical ensemble must include a large number of distinct configurations. Thus our objective is to define an And-Or graph, thus its image grammar, such that its language, *i.e.* the set of valid configurations that it produces, reproduces the ensemble of instances for the visual pattern.

An And-Or graph augments an And-Or tree with three new features.

1. Horizontal lines indicate to show relations, bonds, junctions and semantic relations.
2. Relations at all levels are augmented on the And-Or graph to represent hard (compatibility) or soft (statistical) constraints.
3. The children of an Or-node may share Or-node children. It represents a reusable part shared by several production rules. The sharing of nodes reduces the complexity of the representation and thus the size of dictionary. Other possible sharing may be useful: see, for example, Section 2.6.

Figure 3.17.(a) has shown a simple example of an And-Or graph. An And-Or graph includes three types of nodes: And-nodes (solid circles), Or-nodes (dashed circles), and terminal nodes (squares). The Or-nodes have labels for classification at various levels, such as scene category, object classes, and parts *etc.* Due to this recursive definition, one may merge the And-Or graphs for many objects or scene categories into a larger graph. In theory, the whole natural image ensemble can be represented by a huge And-Or graph, as it is for language.

By assigning values to these labels on the Or-node, one obtains an And-graph – *i.e.* a parse graph. The bold arrows and shaded nodes in Figure 3.17.(a) constitute a parse graph **pg** embedded in the And-Or graph. This parse graph is shown in Figure 3.17.(b) and produces a configuration shown Figure 3.17.(d). It has four terminal nodes (for primitives, parts, or objects): 1, 6, 8, 10 and the edges are inherited from their parent relations. Both nodes 6 and 8 have a common ancestor node C . Therefore the relation $\langle B, C \rangle$ is propagated to $\langle 1, 6 \rangle$ and $\langle 1, 8 \rangle$. For example, if $\langle B, C \rangle$ includes three bonds, two bonds may be inherited by $\langle 1, 8 \rangle$ and one by $\langle 1, 6 \rangle$. Similarly the links $\langle 6, 10 \rangle$ and $\langle 8, 10 \rangle$ are inherited from $\langle C, D \rangle$.

Figure 3.17.(c) is a second parse graph and it produces a configuration in Figure 3.17.(e). It has 4 terminal nodes 2, 4, 9, 9'. The node 9 is a reusable part shared by nodes C and D . It is worth mentioning that a shared node may appear as multiple instances.

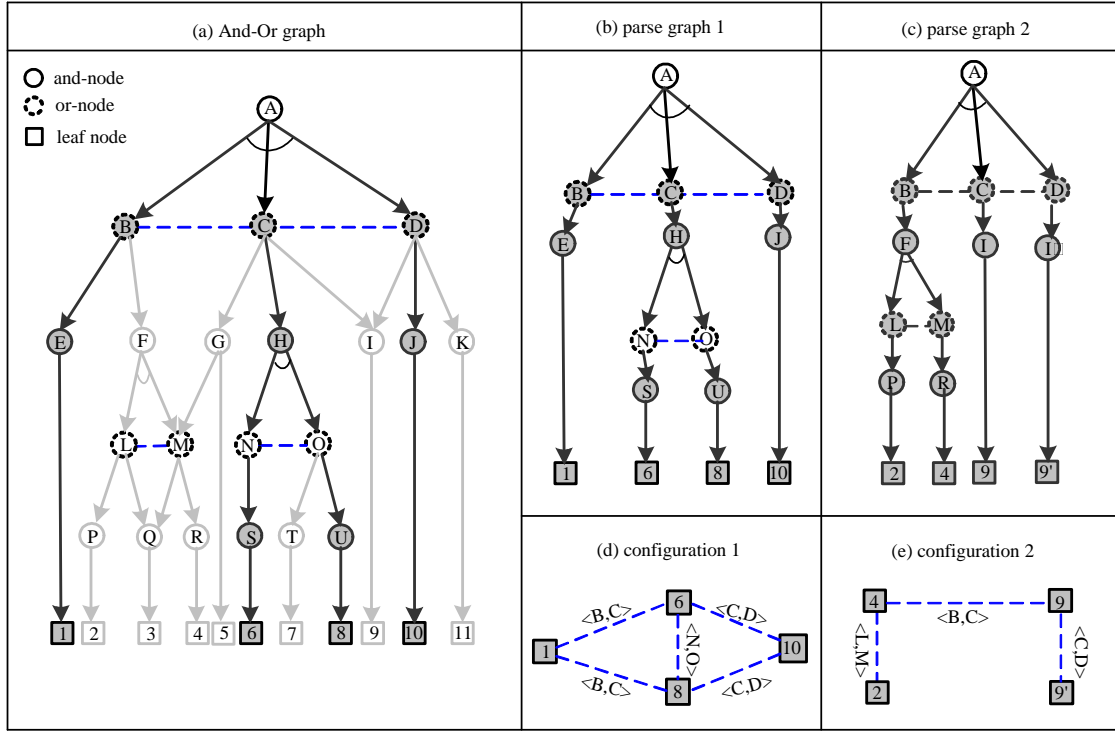


Figure 3.17: Illustrating the And-Or graph representation. (a) An And-Or graph embodies the grammar productions rules and contexts. It contains many parse graphs, one of which is shown in bold arrows. (b) and (c) are two distinct parse graphs by selecting the switches at related Or-nodes. (d) and (e) are two graphical configurations produced by the two parse graphs respectively. The links of these configurations are inherited from the And-or graph relations. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

Definition V. And-Or Graph

An And-Or graph is a 5-tuple for representing an image grammar \mathcal{G} .

$$G_{\text{and-or}} = \langle S, V_N, V_T, \mathcal{R}, \mathcal{P} \rangle. \quad (3.29)$$

S is a root node for a scene or object category, $V_N = V^{\text{and}} \cup V^{\text{or}}$ is a set of non-terminal nodes including an And-node set V^{and} and an Or-node set V^{or} . The And-nodes plus the graph formed by their children are the productions and the Or-nodes are the vocabulary items. V_T is a set of terminal nodes for primitives, parts and objects (note that an object at low resolution may terminate without decomposition directly), \mathcal{R} is a number of relations between the nodes, and \mathcal{P} is the probability model defined on the And-Or graph.

The following is a more detailed explanation of the components in the And-Or graph.

1. *The Non-terminal nodes* include both And-nodes and Or-nodes $V_N = V^{\text{and}} \cup V^{\text{or}}$,

$$V^{\text{and}} = \{u_1, \dots, u_{m(u)}\}, \quad V^{\text{or}} = \{v_1, \dots, v_{m(v)}\}. \quad (3.30)$$

An Or-node $v \in V^{\text{or}}$ is a switch pointing to a number of possible And-Nodes, the productions whose head

is v .

$$v \rightarrow u_1 | u_2 \cdots | u_{n(v)}, \quad u_1, \dots, u_n \in V^{\text{and}}. \quad (3.31)$$

We define a switch variable $\omega(v)$ for $v \in V$, that takes an integer value to index the child node.

$$\omega(v) \in \{\emptyset, 1, 2, \dots, n(v)\} \quad (3.32)$$

By choosing the switch variables in the Or-nodes, one obtains a parse graph from the And-Or graph. The switch variable is set to empty $\omega(v) = \emptyset$ if v is not part of the parse graph. In fact the assignments of Or-nodes at various levels of the And-Or graph corresponds to scene classification and object recognition. In practice, when an Or-node has a large $n(v)$, *i.e.* too fat, one may replace it by a small Or-tree that has $n(v)$ leaves. We omit the discussion of such cases for clarity.

An And-node $u \in V^{\text{and}}$ either terminates as a template $t \in V_T$ or it can be decomposed into a set of Or-nodes. In the latter case, the relations between these child nodes are specified by some relations $r_1, \dots, r_{k(u)} \in \mathcal{R}$ shown by the dashed horizontal lines in Figure 3.17. We adopt the symbol $::$ for representing the relations associated with the production rule or the And-node.

$$\begin{aligned} u &\rightarrow t \in V_T; \quad \text{or} \\ u &\rightarrow \mathcal{C} = (v_1, \dots, v_{n(v)}) :: (r_1, \dots, r_{k(v)}), \quad v_i \in V, r_j \in \mathcal{R}. \end{aligned}$$

The termination rule reflects the multi-scale representation. That is, the node u may be instantiated by a template at a relatively lower image resolution.

2. *The Terminal node set* $V_T = \{t_1, \dots, t_{m(T)}\}$ is a set of instances from the image dictionary Δ . Usually it is a graphical template $(\Phi(x, y; \alpha), \beta)$ with attributes α and open bonds β . Usually, each $t \in V_T$ is a sketch graph, such as the image primitives.

3. *The Configurations* produced from the root node S are the language of the grammar: $G_{\text{and-or}}$,

$$\mathbf{L}(G_{\text{and-or}}) = \Sigma = \{\mathcal{C}_k : S \xrightarrow{G_{\text{and-or}}} \mathcal{C}_k \mid k = 1, 2, \dots, N\}. \quad (3.33)$$

Each configuration $\mathcal{C} \in \Sigma$ is a composite template, for example, the cloth shown in Figure 3.9. The And-Or graph in Figure 3.17.(a) contains a combinatorial number of valid configurations, *e.g.*

$$\Sigma = \{(1, 6, 8, 10), (2, 4, 9, 9), (1, 5, 11), (2, 4, 6, 7, 9), \dots\} \quad (3.34)$$

The first two configurations are shown on the right side of Figure 3.17.

4. *The relation set* \mathcal{R} pools over all the relations between nodes at all levels.

$$\mathcal{R} = \cup_m E_m = \{e_{st} = (v_s, v_t; \gamma_{st}, \rho_{st})\}. \quad (3.35)$$

These relations become the pair-cliques in the composite graphical template. When a node v_s is split later, the link e_{st} may be split as well or may descend to specific pairs of children. For example, in Figure 3.17 node C is split into two leaf nodes 6 and 8, then the relation (B, C) is split into two subsets between $(1, 6)$ and $(1, 8)$,

5. \mathcal{P} is a probability model defined on the And-Or graph. It includes many local probabilities – one at each Or-node to account for the relative frequency of each alternative, and local energies associated with each link $e \in \mathcal{R}$. The former is like the SCFG and the latter is like the Markov random fields or graphical models. We will discuss the probability component in the next subsection.

Before concluding this section, we show an And-Or graph for a clock category [87] in Figure 3.18. Figure 3.18 has shown two parse graphs as instances of this And-Or graph. The dark bold arrows in Figure 3.18 are the parse tree shown in Figure 3.16.(a).

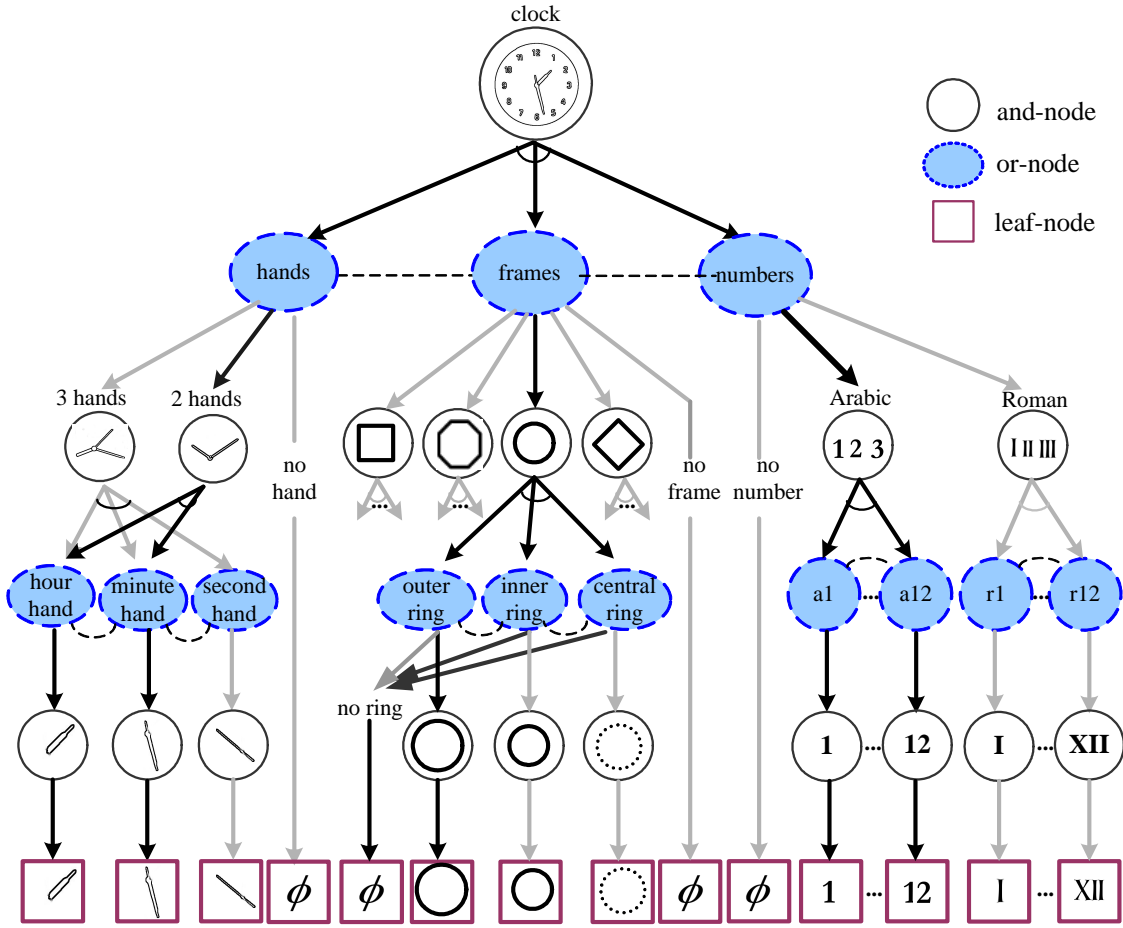


Figure 3.18: An And-Or graph example for the object category – clock. It has two parse graphs shown in Figure 3.16, one of which is illustrated in dark arrows. Some leaf nodes are omitted from the graph for clarity. From [87].

In summary, an And-Or graph $G_{\text{and-or}}$ defines a *context sensitive graph grammar* with V_T being its vocabulary, V_N the production rules, Σ its language, \mathcal{R} the contexts. $G_{\text{and-or}}$ contains all the possible parse graphs which in turns produce a combinatorial number of configurations. Again, the number of configurations is far larger than the vocabulary, *i.e.*

$$|V_N \cup V_T| \ll |\Sigma|. \quad (3.36)$$

This is a crucial aspect for representing the large intra-category structural variations.

Our next task is to define a probability model on $G_{\text{and-or}}$ to make it a stochastic grammar.

3.5.2 Stochastic Models on the And-Or graph

The probability model for the And-Or graph $G_{\text{and-or}}$ must integrate the Markov tree model (SCFG) for the Or-nodes and the Markov graphical (SCSG) models for the And-nodes. Together a probability model is defined on the parse graphs. The objective of this probability model is to match the frequency of parse graphs in an observed training set (supervised learning will be discussed in the next section).

Just as the language model in equation (2.17) defined probabilities on each parse tree $\mathbf{pt}(\omega)$ of each sentence ω , the new model should define probabilities on each parse graphs \mathbf{pg} . As \mathbf{pg} produces a final configuration \mathcal{C} deterministically when it is collapsed, thus $p(\mathbf{pg}; \Theta)$ produces a marginal probability on the final configurations with Θ being its parameters. A configuration \mathcal{C} is assumed to be directly observable, *i.e.* the input, and parse graph \mathbf{pg} are hidden variables and have to be inferred.

By definition IV, a parse graph \mathbf{pg} is a parse tree \mathbf{pt} augmented with relations E ,

$$\mathbf{pg} = (\mathbf{pt}, E). \quad (3.37)$$

For notational convenience, we denote the following components in \mathbf{pg} .

- $T(\mathbf{pg}) = \{t_1, \dots, t_n(\mathbf{pg})\}$ is the set of leaf nodes in \mathbf{pg} . For example, $T(\mathbf{pg}) = \{1, 6, 8, 10\}$ for the parse graph shown by the dark arrows in Figure 3.17. In applications, $T(\mathbf{pg})$ is often the pre-terminal nodes with each $t \in T(\mathbf{pg})$ being an image primitive in the primal sketch.
- $V^{\text{or}}(\mathbf{pg})$ is the set of non-empty Or-nodes (switches) that are used \mathbf{pg} . For instance, $V^{\text{or}}(\mathbf{pg}) = \{B, C, D, N, O\}$. These switch variables selected the path to decide the parse tree $\mathbf{pt} = (\gamma_1, \gamma_2, \dots, \gamma_n)$.
- $E(\mathbf{pg})$ is the set of links in \mathbf{pg} .

The probability for \mathbf{pg} is of the following Gibbs form, similar to equation (2.17),

$$p(\mathbf{pg}; \Theta, \mathcal{R}, \Delta) = \frac{1}{Z(\Theta)} \exp\{-\mathcal{E}(\mathbf{pg})\} \quad (3.38)$$

where $\mathcal{E}(\mathbf{pg})$ is the total energy,

$$\mathcal{E}(\mathbf{pg}) = \sum_{v \in V^{\text{or}}(\mathbf{pg})} \lambda_v(\omega(v)) + \sum_{t \in T(\mathbf{pg}) \cup V^{\text{and}}(\mathbf{pg})} \lambda_t(\alpha(t)) + \sum_{(i,j) \in E(\mathbf{pg})} \lambda_{ij}(v_i, v_j, \gamma_{ij}, \rho_{ij}) \quad (3.39)$$

The model is specified by a number of parameters Θ , the relations set \mathcal{R} and the vocabulary Δ . The first term in the energy is the same as the SCFG. It assigns different weights $\lambda_v()$ to the switch variables $\omega(v)$ at the or-nodes v . The weight should account for how frequently a child node appears. Removing the 2nd and 3rd terms, this reduces to a SCFG in eqn 2.9. The second and third terms are typical singleton and pair-clique energy for graphical models. The second term is defined on the geometric and appearance attributes of the image primitives. The third term models the compatibility constraint, such as the spatial and appearance constraint between the primitives, graphlets, parts, and objects.

This model can be derived from a maximum entropy principle under two types of constraints on the statistics of training image ensembles. One is to match the frequency at each Or-node, just like the SCFG, and the other is to match the statistics, such as histograms or co-occurrence frequency as in standard graphical models. Θ is the set of parameters in the energy,

$$\Theta = \{\lambda_v(), \lambda_t(), \lambda_{ij}(); \forall v \in V^{\text{or}}, \forall t \in V_T, \forall (i, j) \in \mathcal{R}\}. \quad (3.40)$$

Each $\lambda()$ above is a potential function, not a scalar, and is represented by a vector through discretizing the function in a non-parametric way, as it was done in the FRAME model for texture [186]. Δ is the vocabulary for the generative model. The partition function is summed over all parse graphs in the And-Or graph $G_{\text{and-or}}$ or the grammar \mathcal{G} .

$$Z = Z(\Theta) = \sum_{\mathbf{pg}} \exp\{-\mathcal{E}(\mathbf{pg})\}. \quad (3.41)$$

chair:	
1.0	chair \rightarrow top (ϕ_{000}) base (ϕ_{001})
top:	
0.55	top \rightarrow seat (ϕ_{100}) back (ϕ_{101})
0.45	top \rightarrow seat (ϕ_{110}) back (ϕ_{111}) arm (ϕ_{112}) arm (ϕ_{113})
base:	
0.65	base \rightarrow leg (ϕ_{200}) leg (ϕ_{201}) leg (ϕ_{202}) leg (ϕ_{203})
0.35	base \rightarrow axle (ϕ_{210}) wheel-leg (ϕ_{211}) wheel-leg (ϕ_{212}) wheel-leg (ϕ_{213})
seat (A_3)	
back (A_4)	
arm (A_5)	
leg (A_6)	
axle (A_7)	
wheel-leg (A_8)	

Figure 3.19: An example of a PGG for chairs. The symbols ϕ_{crk} represent conditional geometry models which define distributions on the geometry properties of a subpart k given its parent c subject to a rule r , while A_c represent appearance models for linking a part c to image data. Courtesy of [5].

3.6 Examples in Related Work

In this section, we introduce several examples in the computer vision literature which can be alternatively viewed under the S-AOG framework.

3.6.1 Probabilistic Geometric Grammars

Probabilistic geometric grammars (PGGs) were proposed by M. Aycinena, *et al.* [5], which represents object classes recursively in terms of their parts, thereby exploiting the hierarchical and substitutive structure inherent to many types of objects. The PGG model was inspired by the AOG framework [185]. It allows Or-nodes in addition to the And-nodes that exist in simple part hierarchies; this difference is what makes it a grammar.

As illustrated in Figure 3.19, A PGG is a set of part models, each of which is either primitive or composite (*i.e.*, terminal nodes and nonterminal nodes in AOGs). A composite part model (*i.e.*, an Or-node) consists of a set of rules which define how the part can be broken down into subparts (*i.e.*, And-nodes). A primitive part model (*i.e.*, Terminal-node) consists of an appearance model which describes a distribution over the part’s image appearance. More details are referred to [5].

3.6.2 Mixture of Deformable Part-based Models and Object Detection Grammars

Deformable part-based models (DPMs) were proposed by P. Felzenszwalb, *et al.* [38] which were discriminatively trained for object detection and were a dominant method in the field of object detection such as on the PASCAL VOC datasets [34]. A DPM for an object with n parts is formally defined by a $(n + 2)$ -tuple $(F_0, P_1, \dots, P_n, b)$ where F_0 is a root filter template for the entire object, P_i is a model for the i -th part and b is a real-valued bias term. Each part model P_i is defined by a 3-tuple (F_i, v_i, d_i) where F_i is a filter template for the i -th part, v_i is a 2D vector specifying an “anchor” position for part i relative to the root position, and d_i is a 4D vector specifying coefficients of a quadratic function defining a deformation cost for each possible placement of the part relative to the anchor position. A mixture model with m components is

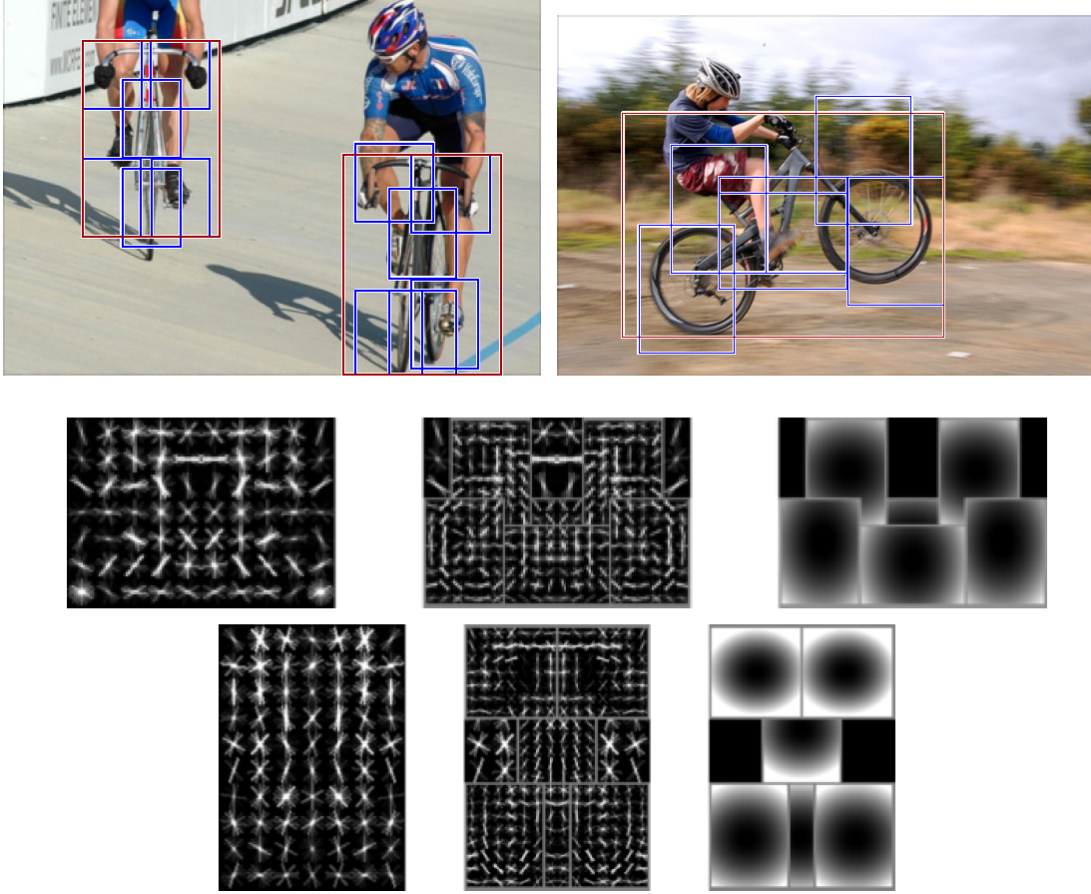


Figure 3.20: An example of a mixture of DPMs for bicycles. *Top*: Some typical detection results using the 2-component bicycle DPMs. *Bottom*: Two bicycle DPMs for the side viewpoint and the frontal viewpoint respectively. Courtesy of [38].

defined by a m -tuple, $M = (M_1, \dots, M_m)$, where M_c is the DPM for the c -th component. See Figure 3.20 for an example.

A mixture of DPMs can be viewed as a 3-layer Or-And Tree model: the root Or-node represents the mixture, the child And-nodes of the root represent the mixture components (i.e., different DPMs), and the child Terminal-nodes of an And-node represents parts. More details are referred to [38].

Felzenszwalb and McAllester extended the mixture of DPMs to a more general object detection grammar [42] which is similar in spirit to the S-AOG, but is formulated with discriminative training.

3.6.3 Probabilistic Program Induction

Lake, *et al.* presented a Bayesian program learning (BPL) model which represents concepts (*e.g.*, handwritten characters) as simple programs that best explain observed examples under a Bayesian criterion [80]. The model is capable of learning a large class of handwritten characters from just a single example and generalizing in ways that are mostly indistinguishable from people. The BPL approach learns simple stochastic programs to represent concepts, building them compositionally from parts (Figure 3.21, iii), subparts (Figure 3.21, ii), and spatial relations (Figure 3.21, iv). BPL defines a generative model that can sample new

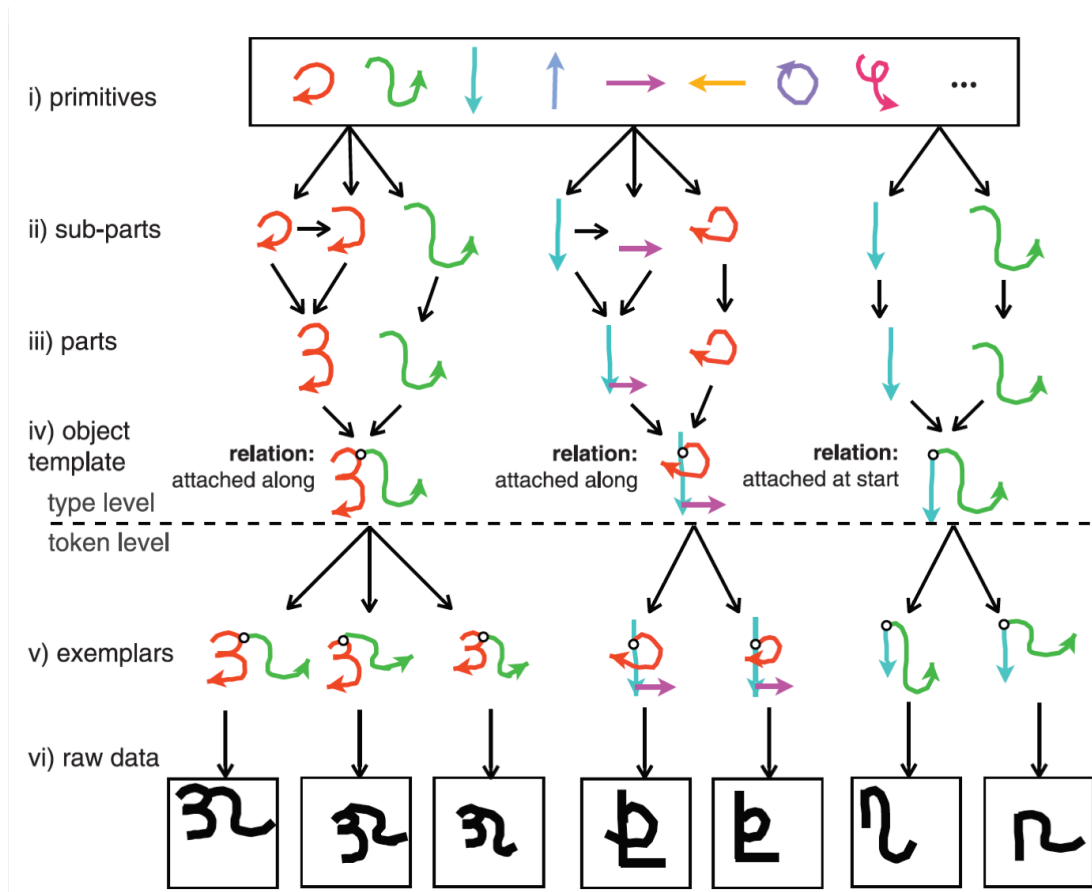


Figure 3.21: An example of BPL on handwritten characters. New types are generated by choosing primitive actions (color coded) from a library (i), combining these subparts (ii) to make parts (iii), and combining parts with relations to define simple programs (iv). New tokens are generated by running these programs (v), which are then rendered as raw data (vi). Courtesy of [80].

types of concepts (an “A,” “B,” *etc.*) by combining parts and subparts in new ways. Each new type is also represented as a generative model, and this lower-level generative model produces new examples (or tokens) of the concept (Figure 3.21, v), making BPL a generative model for generative models. The final step renders the token-level variables in the format of the raw data (Figure 3.21, vi).

The hierarchical and compositional structure of BPL is the same as that of AOGs, which expresses object-part-subpart-primitive hierarchy in a compositional way. Instead of using the commonly adopted filter templates as primitives as done in the DPMs, BPL developed a small generative program which can be interpreted as a continuous Or-node accounting for local structural variations in handwritten characters. BPL also used a small motor program accounting for relationships between entities at the same level (primitive, subpart and part), which can be interpreted as top-down context-sensitive spatial modeling.

3.6.4 Recursive Cortical Networks

George *et al.* proposed a recursive cortical network (RCN) for learning And-Or like models on CAPTCHAs from small examples. As illustrated in Figure 3.22, in RCN, objects, CAPTCHA letters, are modeled as a combination of contours and surfaces (Figure 3.22A). Contours appear at the boundaries of surfaces, both at

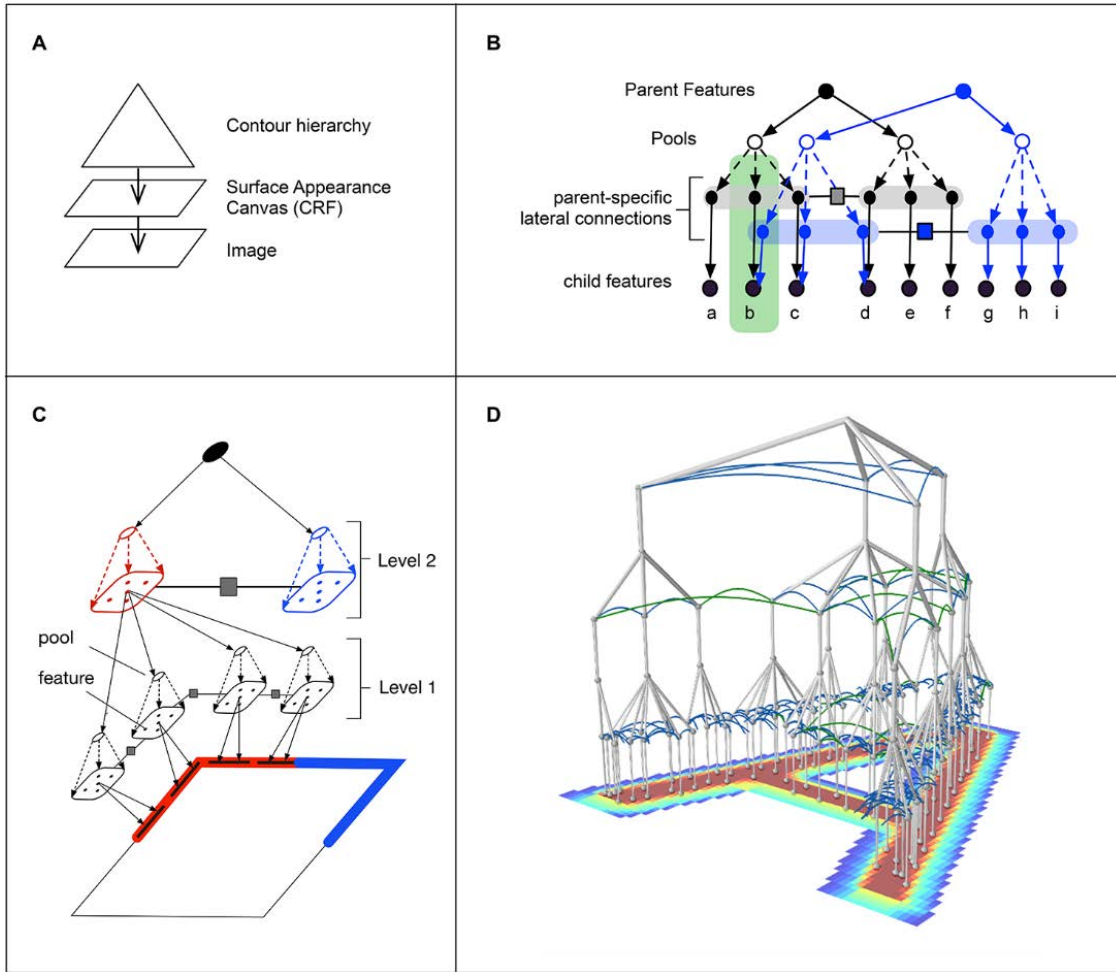


Figure 3.22: Structure of the RCN. (A) A hierarchy generates the contours of an object, and a Conditional Random Field (CRF) generates its surface appearance. (B) Two subnetworks at the same level of the contour hierarchy keep separate lateral connections by making parent-specific copies of child features and connecting them with parent-specific laterals; nodes within the green rectangle are copies of the feature marked “e”. (C) A three-level RCN representing the contours of a square. Features at Level 2 represent the four corners, and each corner is represented as a conjunction of four line-segment features. (D) Four-level network representing an “A”. Courtesy of [54].

the outline of objects and at the border between the surfaces that compose the object. Surfaces are modeled using a Conditional Random Field (CRF) which captures the smoothness of variations of surface properties. Contours are modeled using a compositional hierarchy of features. Factored representation of contours (shape) and surfaces (appearance) enables the model to recognize object shapes with dramatically different appearances without training exhaustively on every possible shape and appearance combination.

RCN integrates and builds upon various ideas from compositional models, especially S-AOG — hierarchical composition, gradual building of invariances, lateral connections for selectivity, contour-surface factorization and joint-explanation based parsing — into a structured probabilistic graphical model such that Belief Propagation can be used as the primary approximate inference engine.

4

Learning the And-Or Graph

In previous chapters, we studied the definition and applications of And-Or Graphs (AOG). In this chapter, we move forward to some learning algorithms of the AOG. An AOG is a model defined by its parameters and structure. In the following sections, we discuss parameter and structure learning separately. As the space of the structures and relations are considerably huge, learning AOGs exactly is usually intractable. We utilize maximize likelihood estimation (MLE) as the backbone of our algorithms and accommodate the pursuit algorithm to learn AOG.

4.1 Overview of the Learning Problem

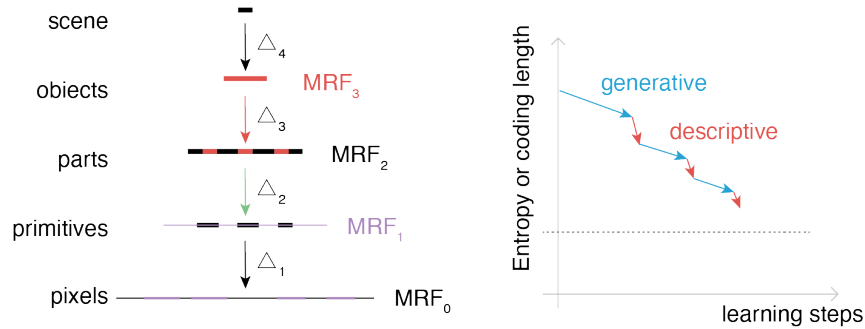


Figure 4.1: Pursuit of AOG models. (Left) Hierarchical structure of the dictionaries. Here we use a computer vision representation as an example. AOG is a model to interpret images. The basic-level units of an image are the pixels, which form the vocabulary of Δ_1 . Then, we establish higher-level vocabularies from the composition of lower-level vocabularies. For example, the compositions of pixels form primitives and the compositions of parts form objects. (Right) The decrease of the entropy/coding length as the learning goes on. In descriptive process, filters and relations are added into AOG. In generative process, elements are adding in the dictionaries. The decrease of entropy/coding length is equivalent to the increase of likelihood.

The goal of learning an AOG is to find an AOG that best explains the training data. This can be modeled as an MLE problem. In many cases, an MLE problem can be solved with optimization approaches such as stochastic gradient descent. However, because AOG learning is a non-differentiable process, we have to use another method to tackle the optimization. We pursuit an AOG model from two directions to maximize the information gain (the mutual information between the old and new models) in every step. The first is

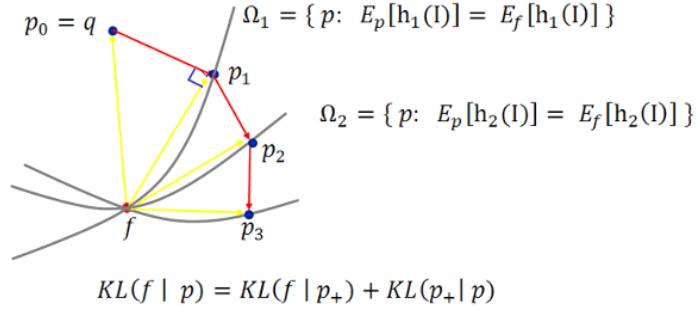


Figure 4.2: Pursuit as information projection. We start from a reference model q and want to approximate the ground truth distribution f that generates the data. The pursuit can be done by adding constraints or active basis [18]. Every time a constraint is added the current model needs to be updated to satisfy the constraint and the difference (measured by KL-divergence) between the current model and the ground truth model will be monotonically decreased. Usually, constraints are statistics that we care about. Given a new constraint, we want to choose the model that satisfies the constraint and is close to the previous model to avoid adding unnecessary features [184]. Pythagorean theorem of KL divergence comes from [27]. Figure reproduced/modified from [140].

the descriptive direction, which adds filter and relations to the AOG. The second is the generative direction, which adds an element to the AOG dictionary by combining existing elements to form hierarchies. Both of these operations guarantee an increment of the likelihood. In the rest of the chapter, we parameterize an AOG with $\Theta, \mathcal{R}, \mathcal{F}$ and Δ . Θ represents all the parameters in an AOG, including the or-node weights and parameters in potential functions. \mathcal{F}, \mathcal{R} and Δ represent all the filters, relations and the hierarchical dictionaries in the AOG respectively. In most of the cases, we can consider filters as the relations defined on the elements in the basic level dictionaries, so we can also merge \mathcal{F} into \mathcal{R} and consider them together in our algorithms. The gist of our learning algorithms is illustrated in Figure 4.1.

4.1.1 Learning Parameters in Stochastic Context-Free Grammar

Before we get into AOG, let's first study the algorithm of learning parameters in a stochastic context-free grammar (SCFG). The difference between an SCFG and an AOG is the former doesn't have relations and is context-free. The only thing we need to learn is the probability of the production rules.

Define V_N and V_T as the non-terminal and terminal nodes in an SCFG G . $A \in V_N$ has a number of alternative rewriting rules,

$$A \rightarrow \beta_1 | \dots | \beta_{n(A)}, \quad \gamma_{A,i} : A \rightarrow \beta_i \\ \text{with } A \in V_N, \beta \in (V_N \cup V_T)^+, \\ n(A) \text{ as the number of alternative rules of } A.$$

Each production rule is associated with a probability $p(\gamma_{A,i}) = p(A \rightarrow \beta_i)$ and $\sum_{i=1}^{n(A)} p(\gamma_{A,i}) = 1$. Given a sentence, ω (a sample generated from G) and its parse-tree, $\mathbf{pt}(\omega)$ (it is a tree as G is context-free), the probability of this parse tree is:

$$p(\mathbf{pt}(\omega)) = \prod_{j=1}^{n(\omega)} p(\gamma_j).$$

Here we abuse the notation and use $n(\omega)$ to denote the number of non-terminal nodes included in the parsing tree. In a supervised way, suppose we have a set of observed parse trees:

$$\{\mathbf{pt}_m, m = 1, 2, \dots, M\}$$

Denote $\theta_{A,i} = p(A \rightarrow \beta_i), \forall A, i$, we have a likelihood,

$$l(\Theta) = \sum_{m=1}^M \log p(\mathbf{pt}_m; \Theta) = \sum_A \#(A \rightarrow \beta_i) \log \theta_{A,i} \quad (4.1)$$

where $\#(A \rightarrow \beta_i)$ denotes the total number of occurrence of this production rule in the dataset. We also have the normalization constraints to make sure all the probabilities are valid distributions. Thus, we can solve a conditional optimization problem with Lagrange multipliers:

$$\begin{aligned} L(\Theta) &= l(\Theta) - \sum_{A \in V_N} \lambda_A \left(\sum_{i=1}^{n(A)} p(A \rightarrow \beta_i) - 1 \right) \\ &= \sum_A \#(A \rightarrow \beta_i) \log \theta_{A,i} - \sum_{A \in V_N} \lambda_A \left(\sum_{i=1}^{n(A)} p(A \rightarrow \beta_i) - 1 \right) \end{aligned} \quad (4.2)$$

$$(4.3)$$

We can calculate $\frac{\partial L}{\partial \theta_{A,i}}$ and set it to 0. Then we can get:

$$\begin{aligned} \frac{\partial L}{\partial \theta_{A,i}} &= \frac{\sum_A \#(A \rightarrow \beta_i)}{\theta_{A,i}} - \lambda_A = 0 \\ \therefore \theta_{A,i} &= \frac{\sum_A \#(A \rightarrow \beta_i)}{\lambda_A} \end{aligned}$$

λ_A can be calculated with the normalization constraints as $\sum_{i=1}^{n(A)} \#(A \rightarrow \beta_i)$. Thus, together, we have $\theta_{A,i} = \frac{\#(A \rightarrow \beta_i)}{\sum_{j=1}^{n(A)} \#(A \rightarrow \beta_j)}$, which is the empirical frequency of the rule occurrence. The grammar we learned is a consistent (the total probability of all possible samples sums to one) SCFG. Notice that here we are learning SCFG in a supervised manner, given samples and their parsing trees. If we want to learn unsupervisedly, we need to use inside-outside algorithms 5.1.4 and integrate out all possible parsing trees.

4.1.2 Probability Model for AOG

We can step forward from SCFG to AOG by adding constraints into the grammar and obtain a context-sensitive grammar. Take a bi-gram Markov chain of sentences for an example. Consider $w = (w_1, \dots, w_n)$, where w_i represents a word in the sentence. Given the SCFG, we can have a parsing tree of w , $\mathbf{pt}(w)$. The probability of $\mathbf{pt}(w)$ is:

$$p(\mathbf{pt}(w)) = p(\gamma_1)p(\gamma_2) \dots p(\gamma_K) \quad (4.4)$$

$$= \frac{1}{Z} \exp\left\{- \sum_{\gamma_i \in \mathbf{pt}(w)} \lambda(\gamma_i)\right\} \quad (4.5)$$

Here we use $\lambda(\gamma_i)$ to represent the switch variable of γ_i .

The probability of a bi-gram Markov chain of w is:

$$p(w) = p(w_1) \prod_{i=1}^{n-1} p(w_{i+1}|w_i) \quad (4.6)$$

$$= \frac{1}{Z} \exp\left\{-\sum_i \phi(w_i) - \sum_{\langle i,j \rangle} \psi(w_i, w_j)\right\} \quad (4.7)$$

The unary terms can be better understood in an image parsing scenario. w_i can be the terminal nodes of the grammar, but they are still symbolic. We also need to calculate the compatibility between the terminal grammar nodes and the input raw data, such as an image.

A model for the AOG is defined on the parse graphs \mathbf{pg} . We want to:

- minimizes a KL-divergence between the SCFG model and the AOG model (for hierarchical structures and grammar consistency)
- matches the statistical constraints (for contexts)

$$p(\mathbf{pg}(w); \Theta) = \arg \min KL(p(\mathbf{pg}(w); \Theta) || p(\mathbf{pt}(w); \Theta_0)) \quad (4.8)$$

$$\text{s.t. } E_p[h_i(\mathbf{pg}(w))] = E_f[h_i(\mathbf{pg}(w))], \forall i \quad (4.9)$$

Here we abuse the notation and use Θ to represent all the parameters in the AOG, including or-node weights, energy function parameters and relations. Θ_0 represents the parameters in the SCFG, which are the production rule probabilities. h are unary and binary constraints that consider the compatibility between the grammar and the input data and the context among the grammar. f is the ground truth model that we want to recover from the data. The parsing trees are available given the SCFG, and the expectations wrt. f can be approximated with the training data. What we want to learn is the Θ , which makes our AOG not only closest to the SCFG but also satisfy all the statistical constraints that we care about. We can write a general formula for the probability of the parsing graphs:

$$p(\mathbf{pg}(w); \Theta) = \frac{1}{Z(\Theta)} \exp\left\{-\sum_{\gamma \in \mathbf{pt}(w)} \lambda(\gamma) - \sum_{g_i \in T(\mathbf{pg}(w))} E(g_i(w); \Theta) - \sum_{e_{ik,jl} \in \mathcal{R}(\mathbf{pg}(w))} E(\beta_{ik}(w), \beta_{jl}(w); \Theta)\right\}, \quad (4.10)$$

where $T(\mathbf{pg})$ is the set of the terminal notes in this \mathbf{pg} and $\mathcal{R}(\mathbf{pg})$ is the set of relations in this \mathbf{pg} . The first term alone in equation 4.10 stands for a SCFG. The second and third terms are energy terms on the terminal notes and relations. In the following sections, we'll elaborate the learning algorithms for AOGs in detail, starting from the parameter learning.

4.2 Learning Parameters in AOG

Suppose we have a training set sampled from an underlying distribution f governing the objects.

$$D^{\text{obs}} = \{(\mathbf{I}_i^{\text{obs}}, \mathbf{pg}_i^{\text{obs}}) : i = 1, 2, \dots, N\} \sim f(\mathbf{I}, \mathbf{pg}). \quad (4.11)$$

The parse graphs $\mathbf{pg}_i^{\text{obs}}$ can be obtained from the ground truth database, like [176], or considered missing in unsupervised case. The objective is to learn a model p which approaches f by minimizing a Kullback-Leibler divergence,

$$p^* = \arg \min KL(f||p) = \arg \min \sum_{\mathbf{pg} \in \Omega_{\mathbf{pg}}} \int_{\Omega_{\mathbf{I}}} f(\mathbf{I}, \mathbf{pg}) \log \frac{f(\mathbf{I}, \mathbf{pg})}{p(\mathbf{I}, \mathbf{pg}; \Theta, \mathcal{R}, \Delta)} d\mathbf{I}$$

$$= \arg \min -\text{entropy}(f(\mathbf{I}, \mathbf{p}g)) - \mathbb{E}_{(\mathbf{I}, \mathbf{p}g) \sim f} [\log p(\mathbf{I}, \mathbf{p}g; \Theta, \mathcal{R}, \Delta)]. \quad (4.12)$$

The entropy of f is constant and we can approximate the expectation with empirical sampling. Then, this is equivalent to the MLE for the optimal vocabulary Δ , relation \mathcal{R} , and parameter Θ ,

$$(\Delta, \mathcal{R}, \Theta)^* = \arg \max \sum_{i=1}^N \log p(\mathbf{I}_i^{\text{obs}}, \mathbf{p}g_i^{\text{obs}}; \Theta, \mathcal{R}, \Delta) - \ell(V_T, V_N, N) \quad (4.13)$$

$$\log p(\mathbf{I}_i^{\text{obs}}, \mathbf{p}g_i^{\text{obs}}; \Theta, \mathcal{R}, \Delta) \propto - \sum_{v \in V^{\text{or}}} \lambda_v(\omega(v)) - \sum_{t \in V^T} \lambda_t(\alpha(t)) - \sum_{(i,j) \in \mathcal{R}} \lambda_{mn}(v_m, v_n) \quad (4.14)$$

where $\ell(V_T, V_N, N)$ is a regularization term that shall balance the model complexity wrt. sample size N but also account for the semantic significance of each elements for the vision purpose (human guided here). The latter is often reflected by utility or cost functions in Bayesian decision theory. Other notations will be explained in next section.

Learning the probability model includes three phases and all three phases follow the same principle above.

1. Estimating the parameters Θ from training data D^{obs} for given \mathcal{R} and Δ .
2. Learning and pursuing the relation set \mathcal{R} for nodes in \mathcal{G} given Δ .
3. Discovering and binding the vocabulary Δ and the hierarchy of the And-Or tree automatically.

4.2.1 Maximum Likelihood Learning of Θ

For a given And-Or Graph hierarchy and relations, the estimation of Θ follows the MLE learning process. Let $\mathcal{L}(\Theta) = \sum_{i=1}^N \log p(\mathbf{I}_i^{\text{obs}}, \mathbf{p}g_i^{\text{obs}}; \Theta, \mathcal{R}, \Delta)$ be the log-likelihood, by setting $\frac{\partial \mathcal{L}(\Theta)}{\partial \Theta} = 0$, we have the following three learning steps.

1. Learning the λ_v at each Or-node $v \in V^{\text{or}}$ accounts for the frequency of each alternative choice. The switch variable at v has $n(v)$ choices $\omega(v) \in \{\emptyset, 1, 2, \dots, n(v)\}$ and it is \emptyset when v is not included in the $\mathbf{p}g$. The derivation is similar to what we have in section 4.1.1. We compute the histogram,

$$\mathbf{h}_v^{\text{obs}}(\omega(v) = k) = \frac{\#(\omega(v) = k)}{\sum_{j=1}^{n(v)} \#(\omega(v) = j)}, \quad k = 1, 2, \dots, n(v). \quad (4.15)$$

$\#(\omega(v) = k)$ is the number of times that node v appears with $\omega(v) = k$ in all the parse graphs in $\Omega_{\mathbf{p}g}^{\text{obs}}$. Thus,

$$\lambda_v(\omega(v) = k) = -\log \mathbf{h}_v^{\text{obs}}(\omega(v) = k), \quad \forall v \in V^{\text{or}}. \quad (4.16)$$

2. Learning the potential function λ_t at the terminal node $t \in V^T$. $\frac{\partial \mathcal{L}(\Theta)}{\partial \lambda_t} = 0$ leads to the statistical constraints,

$$E_{p(\mathbf{p}g; \Theta, \mathcal{R}, \Delta)}[\mathbf{h}(\alpha(t))] = \mathbf{h}_t^{\text{obs}}, \quad \forall t \in V^T. \quad (4.17)$$

In the above equation, $\alpha(t)$ are the attributes of t and $\mathbf{h}(\alpha(t))$ is a statistical measure of the attributes, such as the histogram. $\mathbf{h}_t^{\text{obs}}$ is the observed histogram pooled over all the occurrences of t in $\Omega_{\mathbf{p}g}^{\text{obs}}$.

$$\mathbf{h}_t^{\text{obs}}(z) = \frac{1}{\#t} \sum_t \mathbf{1}(z - \frac{\epsilon}{2} < \alpha(t) \leq z + \frac{\epsilon}{2}). \quad (4.18)$$

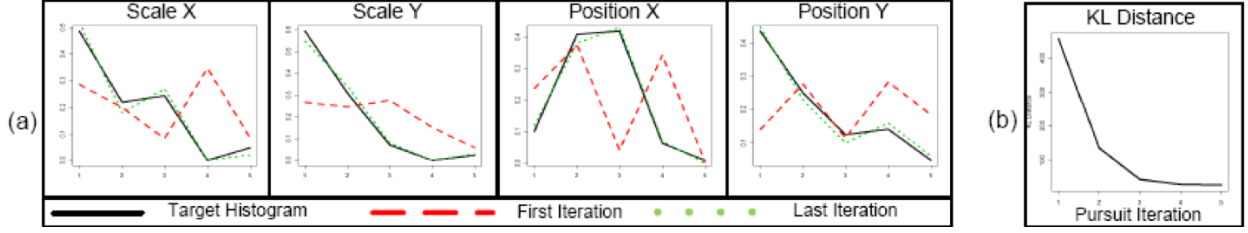


Figure 4.3: Results of the learning procedure. In the left, we show histograms for four pairwise relationships at different iterations. The last iteration matches the observed histogram quite closely. In the right, we can see the decrease of KL divergence between the current and target model as the relationship pursuit is performed.

$\#t$ is the total number of times, a terminal node t appears in the data Ω_{pg}^{obs} . z indexes the bins in the histogram and ϵ is the length of a bin.

3. Learning the potential function λ_{mn} for each pair relation $(m, n) \in \mathcal{R}$. $\frac{\partial \mathcal{L}(\Theta)}{\partial \lambda_{mn}} = 0$ leads to the following implicit function,

$$E_{p(\mathbf{pg}; \Theta, \mathcal{R}, \Delta)}[\mathbf{h}(v_m, v_n)] = \mathbf{h}_{mn}^{\text{obs}}, \forall (m, n) \in \mathcal{R}. \quad (4.19)$$

Again, $\mathbf{h}(v_m, v_n)$ is a statistic on v_m, v_n , for example, a histogram on the relative size, position, and orientation, appearance *etc.* $\mathbf{h}_{mn}^{\text{obs}}$ is the histogram summed over all the occurrence of (v_m, v_n) in D^{obs} .

The equations (4.16), (4.17) and (4.19) are the constraints for deriving the Gibbs model $p(\mathbf{pg}; \Theta, \mathcal{R}, \Delta)$ in equation (4.14) through the maximum entropy principle.

Due to the coupling of the energy terms, both equations (4.17) and (4.19) are solved iteratively through a gradient method. In a general case, we follow the stochastic gradient method adopted in learning the FRAME model [186], which approximates the expectations $E_p[\mathbf{h}(\alpha(t))]$ in equation (4.17) and $E_p[\mathbf{h}(v_i, v_j)]$ in (4.19) by sample means from a set of synthesized examples. This is the method of analysis-by-synthesis adopted in our texture modeling paper [186]. In the end of this section, we show the sampling and synthesis experiments on two object categories – clock and bike in Figures 4.5 and 4.7.

4.2.2 Learning and Pursuing the Relation Set

Besides the learning of parameters Θ , we can also augment the relation sets \mathcal{R} in an And-Or Graph, and thus pursue the energy terms in $\sum_{(m,n) \in \mathbf{pg}} \lambda_{mn}(v_m, v_n)$ in the same way as pursuing the filters and statistics in texture modeling by the minimax entropy principle [186].

Suppose we start with an empty relation set $\mathcal{R} = \emptyset$ and thus $p = p(\mathbf{pg}; \lambda, \emptyset, \Delta)$ is a SCFG model. The learning procedure is a greedy pursuit. In each step, we add a relation e_+ to \mathcal{R} and thus augment model $p(\mathbf{pg}; \Theta, \mathcal{R}, \Delta)$ to $p_+(\mathbf{pg}; \Theta, \mathcal{R}_+, \Delta)$, where $\mathcal{R}_+ = \mathcal{R} \cup \{e_+\}$. e_+ is selected from a large pool $\Delta_{\mathcal{R}}$ so as to maximally reduce KL-divergence, given Pythagorean theorem of KL divergence from Ref. [27],

$$e_+ = \arg \max KL(f||p) - KL(f||p_+) = \arg \max KL(p_+||p), \quad (4.20)$$

Thus we denote the information gain of e_+ by

$$\delta(e_+) \stackrel{\text{def}}{=} KL(p_+||p) \approx f^{\text{obs}}(e_+) d_{\text{manh}}(\mathbf{h}^{\text{obs}}(e_+), \mathbf{h}_p^{\text{syn}}(e_+)). \quad (4.21)$$

In the above formula, $f^{\text{obs}}(e_+)$ is the frequency that relation e_+ is observed in the training data, $\mathbf{h}^{\text{obs}}(e_+)$ is the histogram for relation e_+ over training data D^{obs} , and $\mathbf{h}_p^{\text{syn}}(e_+)$ is the histogram for relation e_+ over

the synthesized parse graphs according to the current model p . d_{manh} is the Mahalanobis distance between the two histograms.

Intuitively, $\delta(e_+)$ is large if e_+ occurs frequently and tells a large difference between the histograms of the observed and the synthesized parse graphs. Large information gain means a significant relation e_+ .

Algorithm 1: Learning Θ by Stochastic Gradients

Input: $D^{\text{obs}} = \{\mathbf{pg}_i^{\text{obs}}; i = 1, 2, \dots, M\}$.

- 1 Compute histograms $\mathbf{h}_v^{\text{obs}}, \mathbf{h}_t^{\text{obs}}, \mathbf{h}_{mn}^{\text{obs}}$ from D^{obs} for all feature/relations.
- 2 Learn the parameters λ_v at the Or-nodes by equation (4.16).
- 3 **repeat**
- 4 Sample a set of parse graphs from the current model $p(\mathbf{pg}; \Theta, \mathcal{R}, \Delta)$
- 5 $D^{\text{syn}} = \{\mathbf{pg}_i^{\text{syn}}; i = 1, 2, \dots, M'\}$
- 6 Compute histograms $\mathbf{h}_t^{\text{syn}}, \mathbf{h}_{ij}^{\text{syn}}$ from D^{syn} for all feature/relations
- 7 Select a feature/relation that maximizes the diff. between obs. vs. syn. histograms.
- 8 Set $\lambda = 0$ for the newly selected feature/relation.
- 9 **repeat**
- 10 Update the parameters with step size η
- 11 $\delta\lambda_t = \eta_t (\mathbf{h}_t^{\text{syn}} - \mathbf{h}_t^{\text{obs}}), \forall t \in V_T,$
- 12 $\delta\lambda_{mn} = \eta_{mn} (\mathbf{h}_{mn}^{\text{syn}} - \mathbf{h}_{mn}^{\text{obs}}), \forall (m, n) \in \mathcal{R}.$
- 13 Sample a set of parse graphs and update the histograms.
- 14 **until** $|\mathbf{h}_t^{\text{syn}} - \mathbf{h}_t^{\text{obs}}| \leq \epsilon$ and $|\mathbf{h}_{mn}^{\text{syn}} - \mathbf{h}_{mn}^{\text{obs}}| \leq \epsilon$ for the selected feature/relations.;
- 15 **until** $|\mathbf{h}_t^{\text{syn}} - \mathbf{h}_t^{\text{obs}}| \leq \epsilon$ and $|\mathbf{h}_{mn}^{\text{syn}} - \mathbf{h}_{mn}^{\text{obs}}| \leq \epsilon$ for all features and relations;
- 16 Equations (4.17) and (4.19) are then satisfied to certain precision.

4.2.3 Examples of Sampling and Synthesis from AOG

In this section, we show some examples of sampling and synthesizing images from AOGs. Our examples illustrate the improvement of the samples as we pursuit the model and add relations into the initial SCFG. In [183], Zhu *et al.* showed a range of image synthesis experiments by sampling the image model (ensembles) for various visual patterns, such as textures, texton, shape contours, faces *etc.* to verify the learned model in the spirit of analysis-by-synthesis.

In Figure 4.4, we compare faces synthesized from SCFG and AOG. We can see that context-sensitive grammar generates much more realistic faces. Figure 4.5 illustrates the synthesis process for a clock category whose AOG is shown previously in Figure 3.18. The experiment is from [116]. Each row in Figure 4.5 shows five typical examples from the synthesis set $\Omega_{\mathbf{pg}}^{\text{syn}}$ in different iterations. In the first row, the clocks are sampled from the SCFG (Markov tree) in a window. These examples have valid parts for clocks shown in different colors, but there are no spatial relations or features to constrain the attributes of the component or layouts. Thus the instances look quite wrong. In the second row, the relative positions of the components (in terms of their centers) are considered. After matching the statistics of the synthesized and observed sets, the sampled instances look more reasonable. In the third, fourth, and fifth rows, the statistics on the relative scale, the hinge relation between clock hands, and a containing relation are added one by one. The synthesized instances become more realistic.

Figure 4.7 shows the same random sampling and synthesis experiment on bicycles. With more spatial relations included and statistics matched, the sampled bikes from the learning models become more realistic from (a) to (d). The set of relations we chose from are included in Figure 4.6.

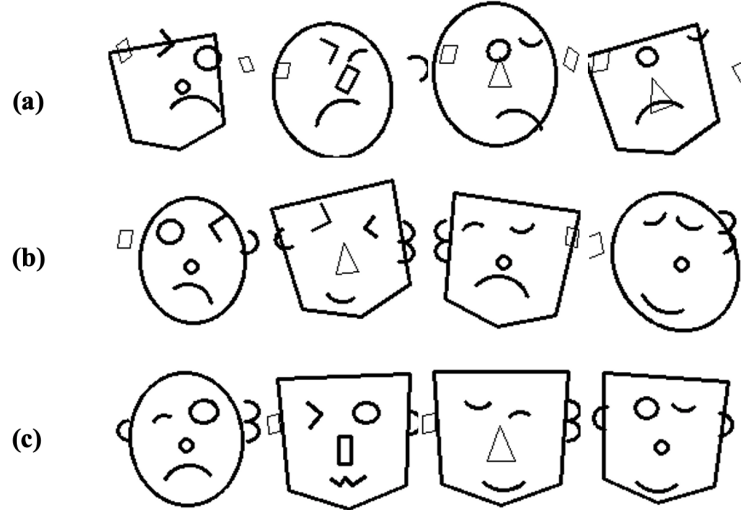


Figure 4.4: Synthesized cartoon faces. All the facial organs are generated from an SCFG. The attributes, however, are determined by an AOG, which considers the context, such as the relative distance between the nose and the mouth, the symmetry of the eyes and ears, *etc.* (a) No relations are added. Although all components are included, the layouts are random. (b) Only unary relations are added. All components are anchored in a rough area in the face, but the relative distances are not considered. (c) Binary relations are included. Now the relative positions of all the organs are correct, giving realistic face layouts. From Ref. [116].

The synthesis process produces novel configurations not seen in the observed set and also demonstrates that the spatial relations captured by the And-Or graph will provide information for top-down prediction of object components. Figure 4.8 shows an example of top-down prediction and hallucination of occluded parts using the learned bike model above.

We also show the application of AOG models in human pose parsing. In Figure 4.9, we define an AOG for human posing and combine it with the learned appearance model in Figure 4.10 to parse pedestrian poses from natural scenes [128]. The compositionality of the AOGs enables us to parse diverse human poses and handle significant deformation/variation of the poses. In Figure 4.11 we show samples generated from the learned AOG. The context-sensitive grammar can generate enriched samples w/o constraints assigned to certain parts of the body.

4.2.4 Summary of the Parameter Learning

In summary, the learning algorithm starts with an SCFG (Markov tree) and a number of observed parse graphs for training D^{obs} . It first learns the SCFG model by counting the occurrence frequency at the Or-nodes. Then by sampling this SCFG, it synthesizes a set of instances D^{syn} . The sampled instances in D^{syn} will have the proper components but often have wrong spatial relations among the parts as there are no relations specified in SCFG. Then the algorithm chooses a relation that has the most different statistics (histogram) over some measurement between the sets D^{obs} and D^{syn} . The model is then learned to reproduce the observed statistics over the chosen relation. A new set of synthesized instances is sampled. This iterative process continues until no more significant differences are observed between the observed and synthesized sets.

Remark 1. At the initial step, the synthesized parse graphs will match the frequency counts on all

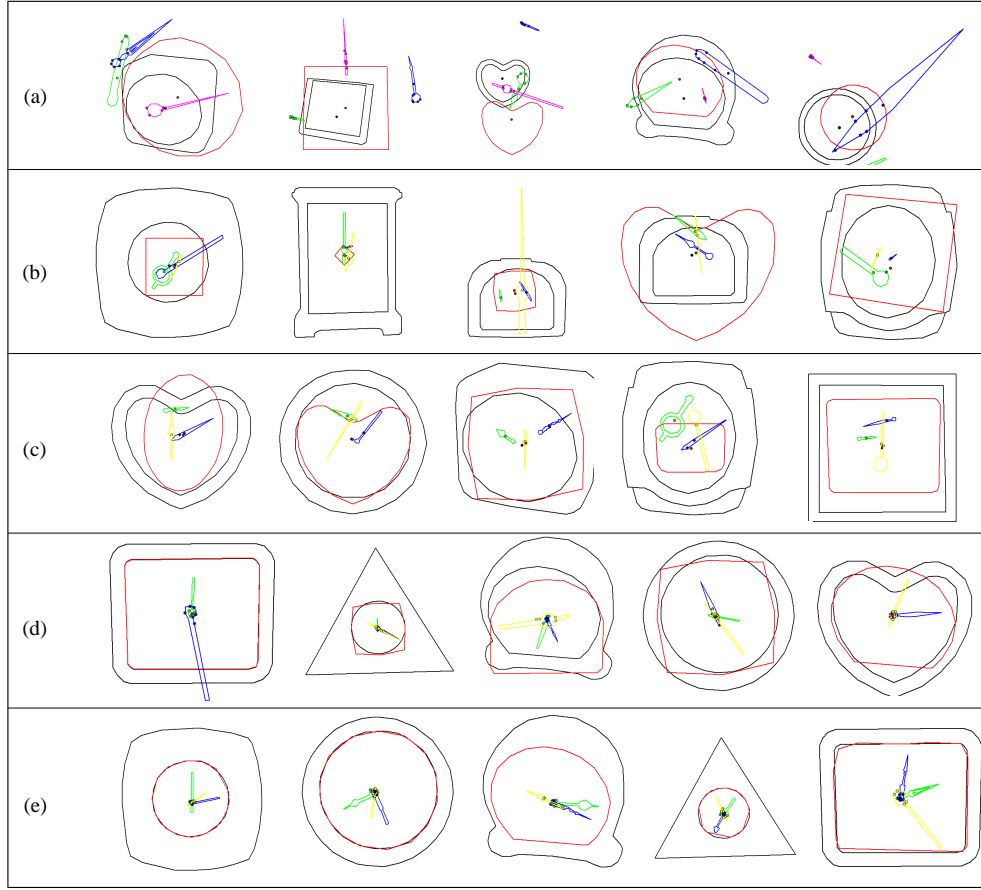


Figure 4.5: Learning the And-Or graph parameters for the clock category. (a) Sampled clock examples (synthesis) based on SCFG (Markov tree) that accounts for the frequency of occurrence. (b-e) Synthesis examples at four incremental stages of the minimax entropy pursuit process. (b) Matching the relation positions between parts, (c) further matching the relative scales, (d) further pursuing the hinge relation, (e) further matching the containing relation. Reprinted/modified, with permission, from (Porway, Yao and Zhu, 2008) [115].

Or-nodes first, but the synthesized parse graphs and their configurations will not look realistic. Parts of the objects will be in wrong positions and have wrong relations. The iterative steps will make improvements. Ideally, if the features and statistical constraints selected in equations (4.17) and (4.19) are sufficient, then the synthesized configurations

$$\Omega_C^{\text{syn}} = \{C_i^{\text{syn}} : \mathbf{pg}_i^{\text{syn}} \longrightarrow C_i^{\text{obs}}, i = 1, 2, \dots, M'\}. \quad (4.22)$$

should resemble the observed configurations. This is what people did in texture synthesis.

Remark 2. In the above learning process, a parse graph $\mathbf{pg}_i^{\text{obs}}$ contributes to some parameters only when the corresponding nodes and relations are present in $\mathbf{pg}_i^{\text{obs}}$.



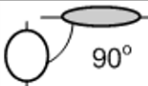













Soft Relationships			Hard Relationships				
Position	Scale	Orientation	Contained	Hinged	Attached	Butting	Concentric
							
							
Low Level Relationships			High Level Relationships				

Figure 4.6: Relation examples. Here we only include binary relations. They can be applied to selected grammar nodes and return a value. The input of a relation function is the attributes of two nodes. Then it outputs the value, indicating if the two nodes have certain relation. Reprinted/modified, with permission, from (Porway, Yao and Zhu, 2008) [115].

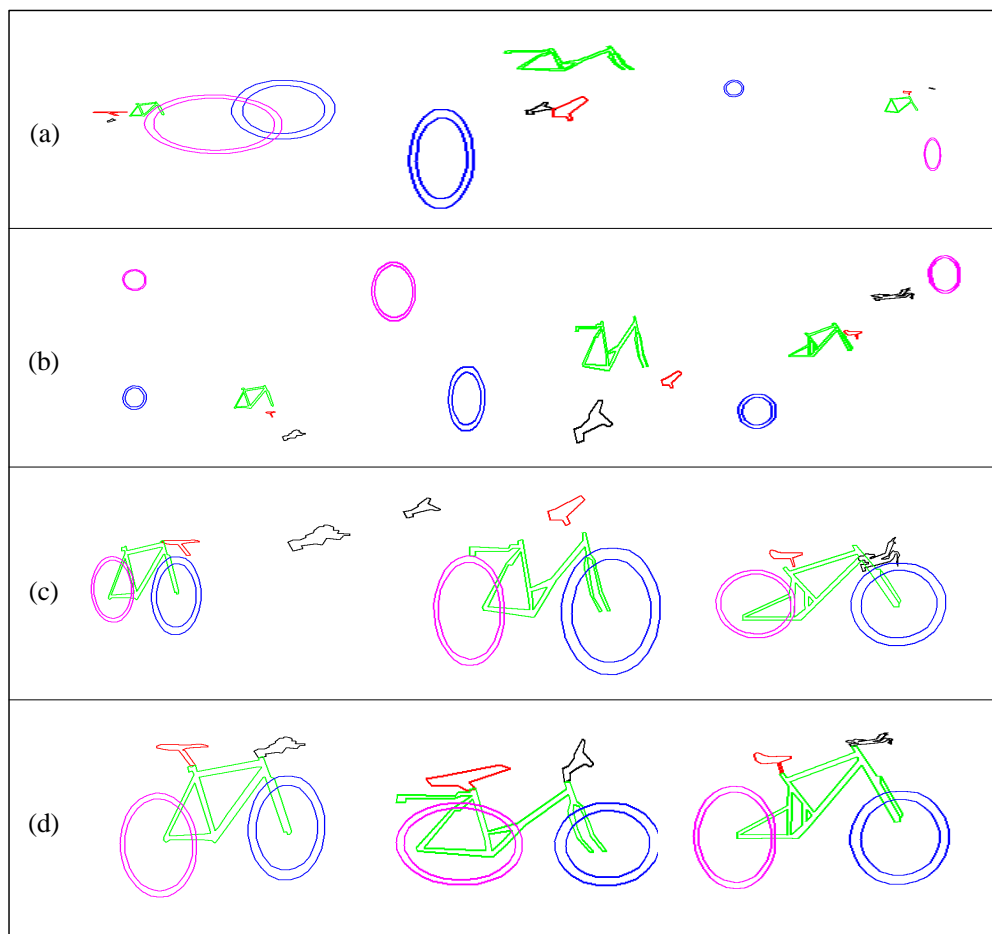


Figure 4.7: Random sampling and synthesis of the bike category. Reprinted/modified, with permission, from (Porway, Yao and Zhu, 2008) [115].

4.3 Structure Learning: Block Pursuit and Graph Compression

In the coming sections, we study the structure learning of AOG. Recall that learning structure aims to obtain the hierarchical vocabulary for the grammar. The hierarchy enables semantic meaningful components to be

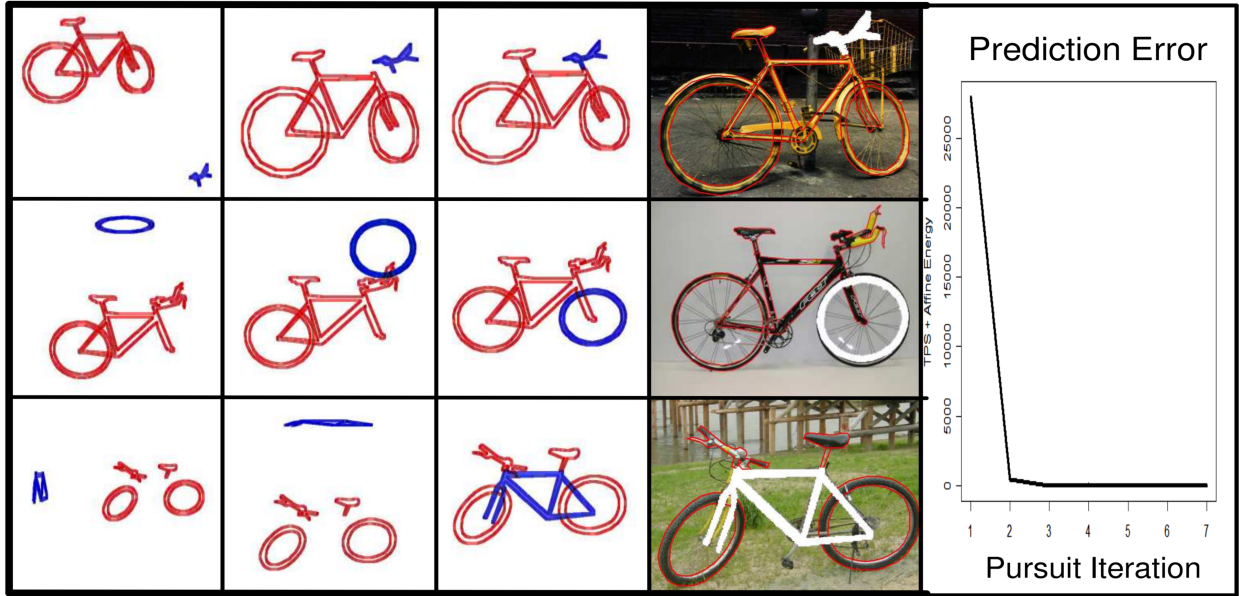


Figure 4.8: Top-down prediction by sampling the missing part. A neighborhood of parts is fixed and the remaining parts are Gibbs sampled. The accuracy of the prediction is measured by the Thin-plate-spline + affine transformation (Y-Axis of the right curve) needed to move the predicted part to its true position. We can see that this energy decreases drastically as we add more relations to the model [115]. Reprinted/modified, with permission, from (Porway, Yao and Zhu, 2008) [115].

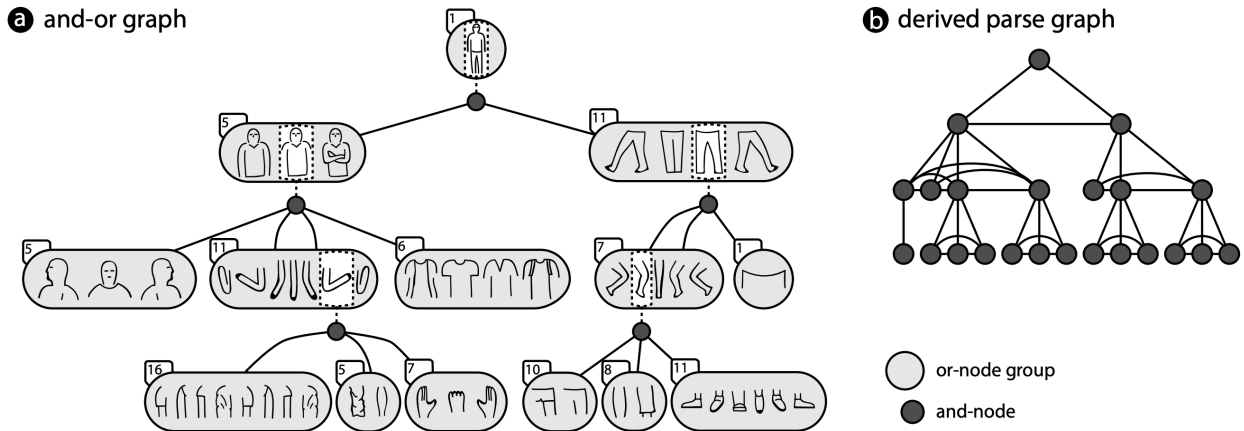


Figure 4.9: And-Or graph grammar model for human pose parsing: (a) Pictorially represent the and-or graph, representing or-nodes as ovals containing a selection from multiple forms for a given part type. For each of these forms, there is a corresponding and-node, represented as black circles, which specify the composition of that part from smaller subparts. (b) A parse graph is a derivation of the and-or graph, and contains instantiations for each and-node corresponding to every or-node selected in the derivation. The edges of the parse graph represent local geometric and type constraints between parts. Reprinted/modified, with permission, from (Rothrock, Park and Zhu, 2013) [128].

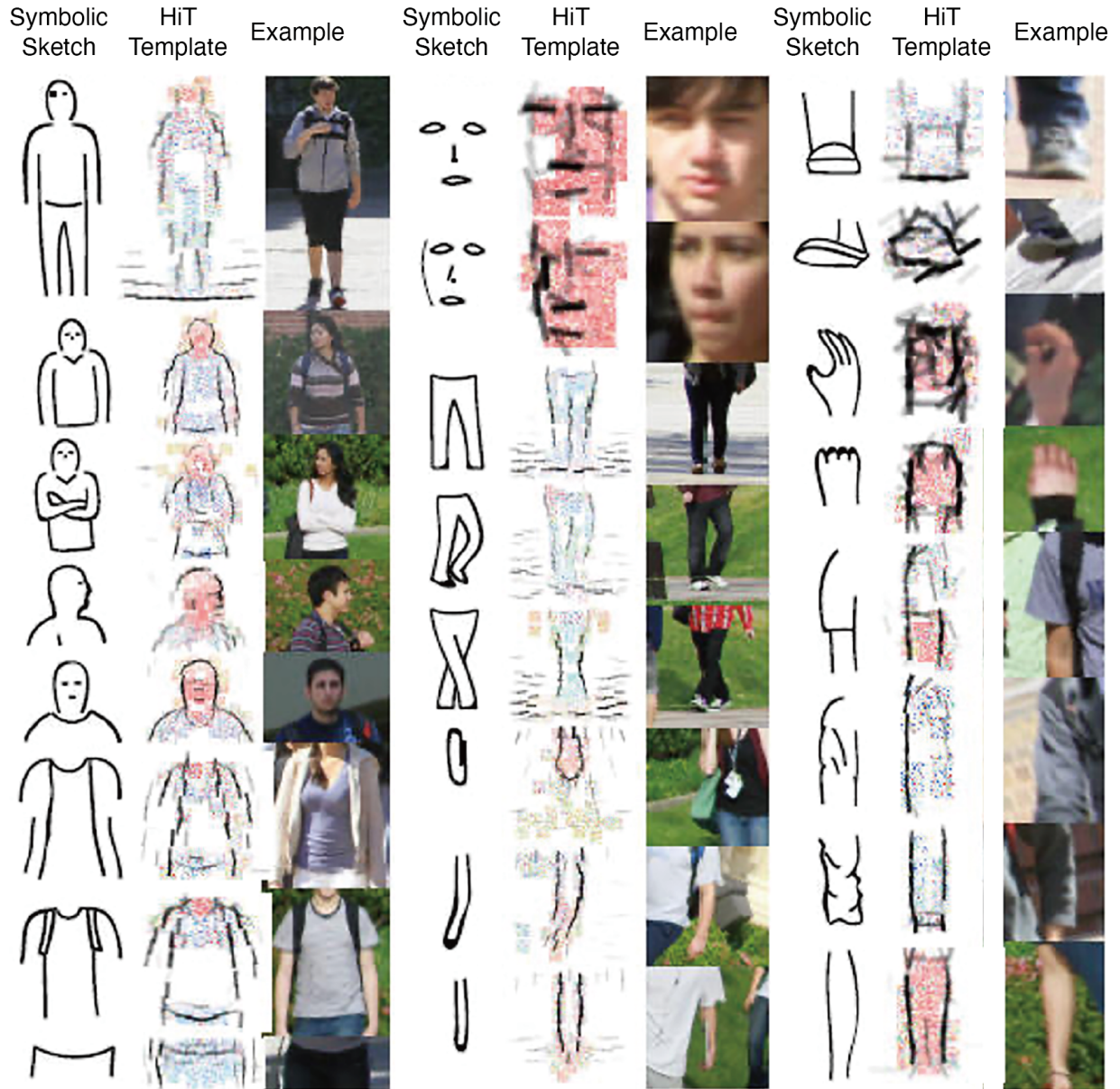


Figure 4.10: Learned HIT appearance templates for 25 of the 110 productions in the model. Each production corresponds to a prototypical part configuration which captures a particular perceptual aspect of the part’s appearance such as clothing, color, geometry, pose, and lighting. The color features are rendered by sampling pixels from their corresponding prototype histograms.

reconfigured and composed, expressing large-scale languages while maintaining concise grammar. Visual objects are fundamentally compositional and exhibit rich structural variations: cats may have sharp or round ears; desks may have long or short legs. Therefore object templates must be reconfigurable to account for structural variabilities. In this section, we introduce a framework for learning AND-OR templates (AOT) which combine compositionality represented as AND nodes, and reconfigurability represented as OR nodes.

An AOT is a stochastic reconfigurable template that generates a set of valid configurations or object

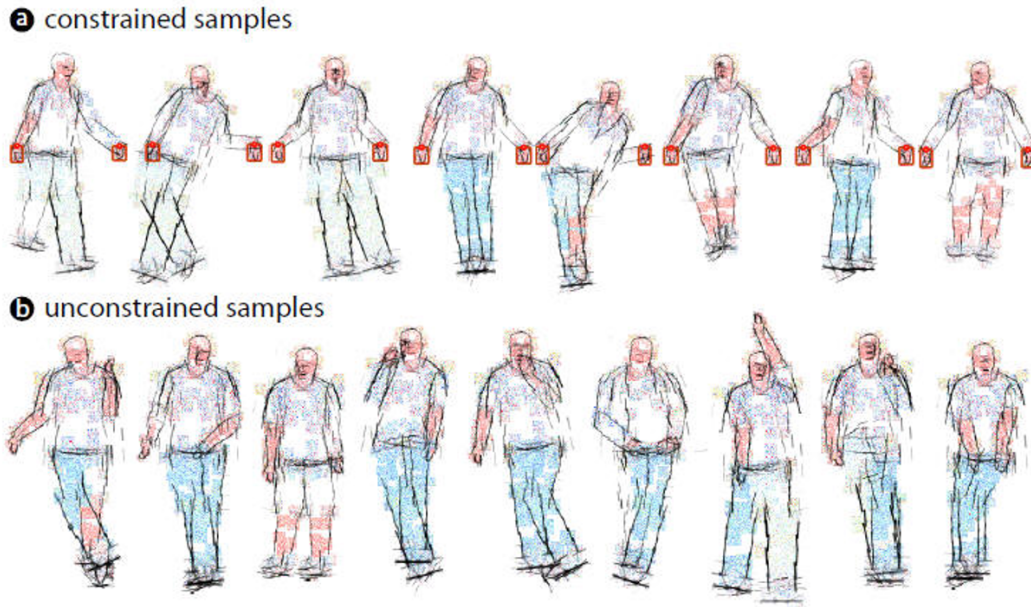


Figure 4.11: Random samples from the prior model are synthesized by sampling a random parse graph from prior model, then sampling the appearance template for each part. To illustrate the flexibility of the model, samples can be conditioned on any set of parts. Constrained samples conditioned on fixed hand positions are shown in (a), and unconditioned samples are shown in (b).

templates. The AND nodes represent compositions of parts, while the OR nodes account for articulation and structural variation of parts. As an example, Figure 4.12 shows a learned AOT from 320 animal face images without manual labeling. The solid circles denote AND nodes and the hollow ones denote OR nodes. The branching probabilities are also shown at each OR node. The rectangles denote terminal nodes, where the numbers index the animal facial parts (*e.g.* eyes, ears) shown as Hybrid Image Templates (HITs) in the third row of the figure. HITs are used as reusable building blocks of AOT and will be introduced in the following. The hierarchical AOT generates a large number of valid configurations.

The learning algorithm for AOT consists of two steps:

i) Block pursuit for hierarchical dictionaries of reusable parts. Firstly we set up a data matrix (*e.g.* Figure 4.15) using feature responses of positive training examples. Each row in the data matrix corresponds to one training example; and each column corresponds to one feature (*e.g.* primitive, texture and color). The data matrix is usually flat, *i.e.* with a small number of rows (*e.g.* 100) and a large number of columns (*e.g.* 10^6). The entries in the data matrix, *i.e.* the feature responses, are normalized to real numbers between 0 and 1. A response of 1 indicates that a certain feature or part is present in the image, while 0 indicates absence of such feature. Each visual part corresponds to a rectangular *block* in the data matrix. The block is specified by a set of common features (columns) shared by a set of examples (rows). The shared features form a *template* of the block members. The summation over entries in the block measures how significant and how frequent it is. We find that pursuing large blocks with a lot of 1's directly links to the information projection principle [170, 186] and maximum likelihood estimation. Once we pursue the blocks, *i.e.* the parts, we augment the data matrix with new columns measuring the responses of the parts. This procedure is done recursively until we reach the scale of whole objects. It is worth noting that the pervasive ambiguity of parts (*i.e.* where to segment objects into parts) can be reduced or eliminated by jointly pursuing for both objects and parts. We show this in the one-dimensional text example in Section 4.3.5.

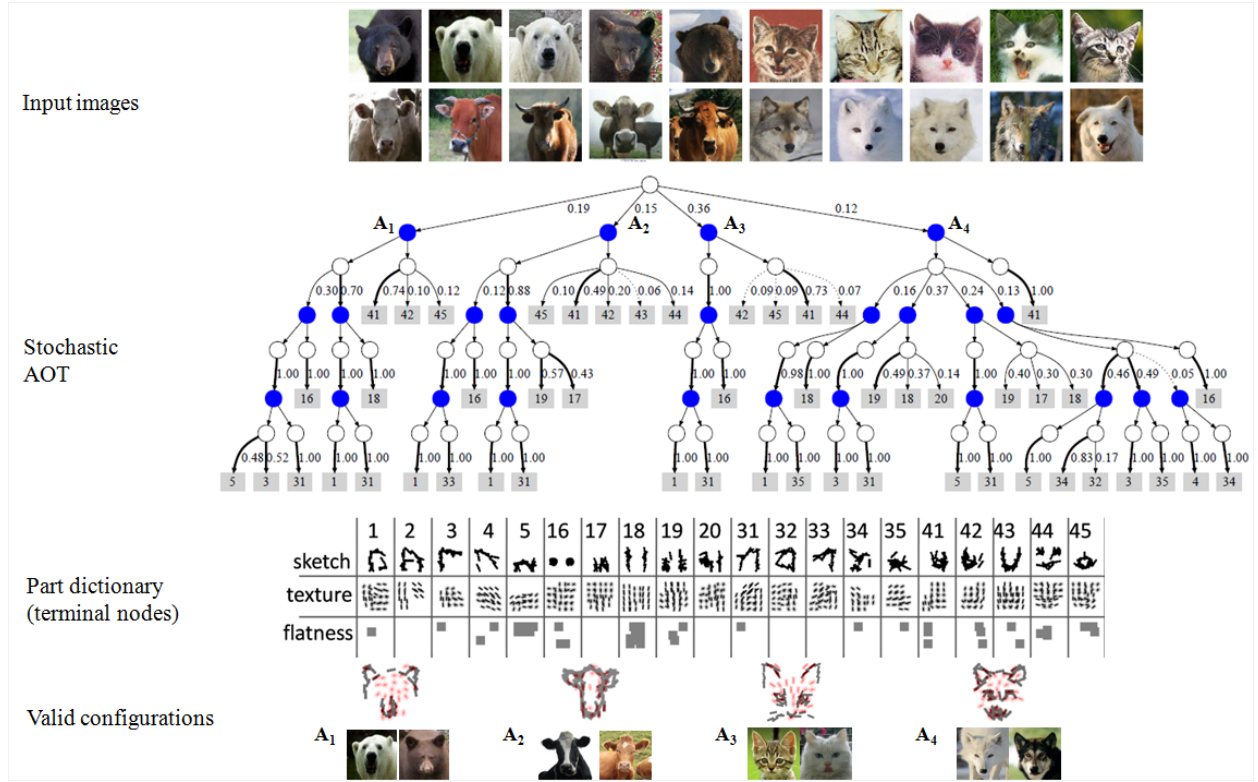


Figure 4.12: An AND-OR Template (AOT) learned from 320 animal face images of four categories, with no manual labeling. Shaded circles denote AND nodes, which are combinations of terminal nodes or children OR nodes. Empty circles mean structural OR nodes. Shaded rectangles are terminal nodes, which are hybrid image templates (HITs) for part appearances. Each terminal node is also associated with a geometric OR node which accounts for its deformation and articulation. For clarity, we removed the geometric OR nodes and OR branches with probability less than 0.1. Reprinted/modified, with permission, from (Zhang and Zhu, 2011) [141].

ii) Graph compression on the AOT. After block pursuit, we encode the training images by a set of configurations of parts (*i.e.* which parts appear and where they appear). It is a flat and large AOT whose root node has many branches. This flat AOT simply memorizes data, so it suffers from large model complexity and thus poor generalizability. We propose a graph compression procedure to produce a compact AOT. There are two operators in graph compression:

- **Sharing.** This operator restructures the AOT by sharing parts (*e.g.* $(A \cap B) \cup (A \cap C) \Rightarrow A \cap (B \cup C)$).
- **Merging.** This operator merges OR nodes with similar branching probabilities, and re-estimates the merged probabilities.

By applying the two operators, we reduce the model complexity, which is measured by the number of nodes in the AOT. This is accompanied by a slight loss in likelihood. We use a parameter to control the trade-off between data likelihood and model complexity. This is closely related to Bayesian model selection methods like BIC [133]. Though in this section we introduce block pursuit and graph compression in AOT learning as an example, the methodology is universal and can be extended to other applications, *e.g.*, pursuit of atomic actions or causal relationships.

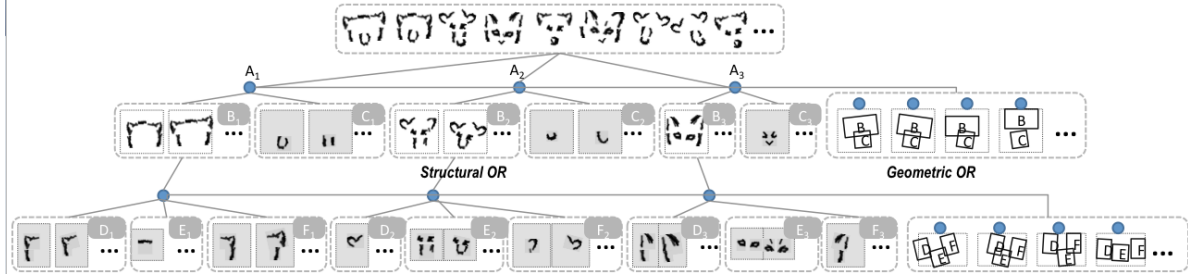


Figure 4.13: A more detailed illustration of the animal AOT shown in Figure 4.12. For OR nodes, we illustrate typical templates with structural variations and geometric transforms. The AND nodes are denoted by solid blue circles. The terminal nodes are individual parts (e.g. ear, mouth) which are represented by automatically learned hybrid image templates. Reprinted/modified, with permission, from (Zhang and Zhu, 2011) [141].

4.3.1 Hybrid Image Templates (HIT) as Terminal Nodes

In our representation, the terminal nodes of the AOT are HITs (hybrid image templates) [140], which we explain in the following.

Image alphabet. An image \mathbf{I} is divided into many small image patches $\{\mathbf{I}_\Lambda\}$, where $\{\Lambda\}$ are domains of local regions (e.g. $11^2 \sim 19^2$ pixels). The space of small image patches is quantized into four categories: sketch, texture, flatness and color.

Definition 1 (Definition 1). *The image alphabet, denoted as $\Delta^{(1)}$, is the set of feature prototypes that are typical and appear frequently in small image patches. They include sketch, texture, flatness and color:*

$$\Delta^{(1)} = \{B_j\} \cup \{\mathbf{h}_j^{\text{txt}}\} \cup \{\mathbf{h}_j^{\text{ftt}}\} \cup \{\mathbf{h}_j^{\text{clr}}\}$$

$\{B_j\}$ are image primitives or sketches (e.g. Gabor wavelets), which often appear near strong edges and object boundaries. $\{\mathbf{h}_j^{\text{txt}}\}$ are histograms of gradient orientations (e.g. HoG) which are suitable to represent more complex textures inside objects (e.g. fur and hair). $\{\mathbf{h}_j^{\text{ftt}}\}$ are flatness features which often appear in empty image patches like cloudless sky. $\{\mathbf{h}_j^{\text{clr}}\}$ are histograms of color, which are most capable of describing objects with distinctive colors (e.g. tomatoes).

HIT model. A HIT is a fully generative probabilistic image model, which consists of a small number of atomic feature prototypes at selected locations and orientations. See Figure 4.14 for two HITs learned from tomato and pear images. A HIT is specified by a list:

$$\text{HIT} = \{(B_1, x_1, y_1, o_1), (\mathbf{h}_2, x_2, y_2), (\mathbf{h}_3, x_3, y_3), (B_4, x_4, y_4, o_4), \dots\}$$

where B_1, B_4, \dots are image primitives and $\mathbf{h}_2, \mathbf{h}_3, \dots$ are histogram descriptors for texture, flatness and color. $\{(x_j, y_j)\}$ denote the selected locations and $\{o_j\}$ are the selected orientations. Though local deformation is allowed for its individual elements, the HIT itself does not deal with large articulation or structural variation. The proposed AOT is aimed at addressing this issue, and the HITs serve as terminal nodes of AOT. Each AOT is an object template consist of a few HITs as parts.

Feature responses. Within a window (e.g. 150 by 150 pixels) of visual object, the image is divided into small image patches using a regular grid. For each patch, we measure a one-dimensional response $r(\mathbf{I})$ which indicates how likely each feature prototype in $\Delta^{(1)}$ appears in this patch. r measures the similarity

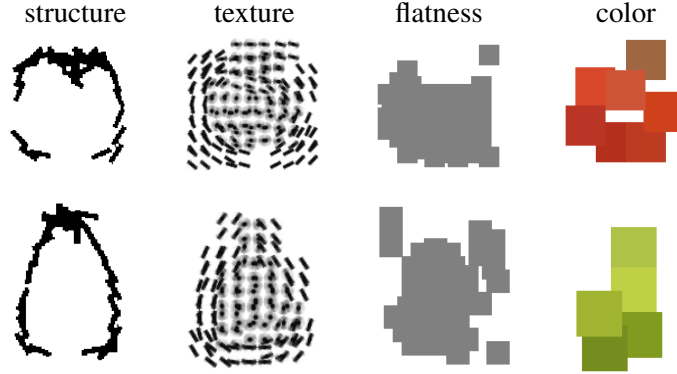
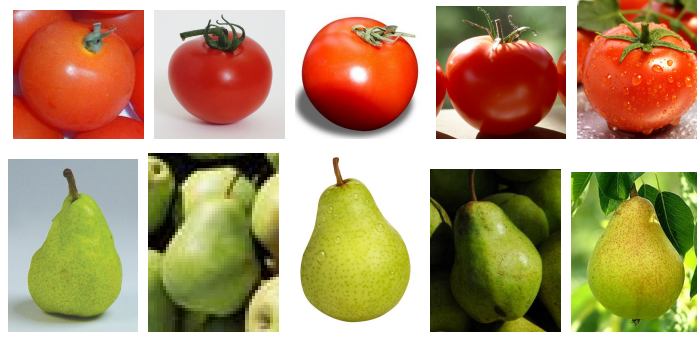


Figure 4.14: HITs for tomato and pear. Best viewed in color. Reprinted/modified, with permission, from (Zhang and Zhu, 2011) [140].

between the image patch and feature prototype, and it is normalized to a value between 0 and 1. Larger value of r means this feature prototype appears with higher probability.

For image primitives $\{B_j\}$, we compute feature response r as the Euclidean distance between the image patch and the primitive. For texture, color and flatness, we compute locally pooled histograms in the image patch. We refer to [140] for parameters of atomic features and how to compute the one dimensional response $r(\mathbf{I})$.

Let D be the total number of feature responses (*i.e.* number of patches times the dictionary size $|\Delta^{(1)}|$) on one image. D is often on the order of 10^6 or more. The feature responses of image \mathbf{I} are organized in a vector:

$$\mathbf{R}(\mathbf{I}) = (r_1(\mathbf{I}), \dots, r_D(\mathbf{I})).$$

From the D candidate features, we select a small subset to compose a HIT. Let $\{j_1, \dots, j_T\} \subset \{1, \dots, D\}$ be indexes of the selected features. T is the number of selected atomic features, and it is usually on the order of 10. To simplify notation, we will use r_t to denote r_{j_t} when there is no ambiguity.

Probability model. Let $\mathcal{X}_+ = \{\mathbf{I}_1, \dots, \mathbf{I}_N\}$ be positive example images (*e.g.* animal faces) governed by the underlying target distribution $f(\mathbf{I})$. Let \mathcal{X}_- be a large set of generic natural images governed by the reference distribution $q(\mathbf{I})$. Our objective of learning is to pursue a model $p(\mathbf{I})$ to approximate $f(\mathbf{I})$ in a series of steps:

$$q(\mathbf{I}) = p_0(\mathbf{I}) \rightarrow p_1(\mathbf{I}) \rightarrow \dots p_T(\mathbf{I}) = p(\mathbf{I}) \approx f(\mathbf{I})$$

starting from q .

The model p after T iterations contains T selected features $\{r_t : t = 1, \dots, T\}$. If the selected feature responses capture all information about image \mathbf{I} , it can be shown by variable transformation [170] that:

$$\frac{p(\mathbf{I})}{q(\mathbf{I})} = \frac{p(r_1, \dots, r_T)}{q(r_1, \dots, r_T)}. \quad (4.23)$$

So p can be constructed by reweighting q with the marginal likelihood ratio on selected features.

Under the maximum entropy principle, $p(\mathbf{I})$ can be expressed in the following log-linear form:

$$p(\mathbf{I}) = q(\mathbf{I}) \prod_{t=1}^T \left[\frac{1}{z_t} \exp \{ \beta_t r_t(\mathbf{I}) \} \right]. \quad (4.24)$$

where β_t is the parameter for the t -th selected feature r_t and z_t ($z_t > 0$) is the individual normalization constant determined by β_t :

$$z_t = \sum_{r_t} q(r_t) \exp \{ \beta_t r_t \}. \quad (4.25)$$

It is an expectation over the reference distribution $q(r_t)$. In practice, as a pre-processing step prior to model learning, we estimate $q(r_t)$ in the form of a histogram using a large number of random training examples. And we can compute z_t using Monte Carlo estimation.

For simplicity, we will use an aggregated normalizing constant $Z = \prod_{t=1}^T z_t$ when there is no ambiguity.

By the information projection principle [27, 170, 186], we adopt a step-wise procedure for feature selection. In particular, the t -th feature r_t is selected and model p_t is updated by:

$$\begin{aligned} p_t &= \arg \max \mathcal{K}(p_t | p_{t-1}) \\ s.t. \quad E_{p_t}[r_t] &= \frac{1}{N} \sum_{i=1}^N r_t(\mathbf{I}_i) \end{aligned} \quad (4.26)$$

where \mathcal{K} denotes the Kullback-Leibler divergence, and by maximizing it over all candidate features, we select a most informative feature r_t to augment p_{t-1} towards p_t . The constraint equation in Eq. (4.26) ensures that the updated model is consistent with the observed training examples on marginal statistics. The optimal β_t can be found by a simple line search or gradient descent to satisfy the constraint in Eq. (4.26).

It is convenient to rewrite Eq. (4.24) using a long and sparse vector β of length D , with only few non-zero entries at the indexes $\{j_1, \dots, j_T\}$ corresponding to selected features:

$$p(\mathbf{I}) = q(\mathbf{I}) \prod_{j=1}^D \left[\frac{1}{z_j} \exp \{ \beta_j r_j(\mathbf{I}) \} \right] \quad (4.27)$$

And the logarithm of normalizing constants $(\log z_1, \dots, \log z_D)$ is also sparse, with $\log z_j = 0$ (*i.e.* $z_j = 1$) whenever $\beta_j = 0$. Since β encodes both indexes and multiplicative weights of selected features, we may simply consider β itself as an HIT.

Definition 2 (Definition 2). *The terminal nodes of AOT, denoted as $\Delta^{(2)}$, is the set of HITs,*

$$\Delta^{(2)} = \{ \text{HIT}_k : k = 1, \dots, K \},$$

which are automatically learned from images. Each entry in $\Delta^{(2)}$ is a part template composed of elements in $\Delta^{(1)}$ at selected locations and orientations. K can also be learned, which will be explained in Section 4.3.3. See the third row of Figure 4.12 for an example of $\Delta^{(2)}$.

In [140] it is shown that the HIT performs well on object categorization. With fewer features and parameters, it achieves on par or better accuracy compared with state-of-the-art methods like HoG + SVM [25] on public benchmarks, especially when there are a small number of training examples.

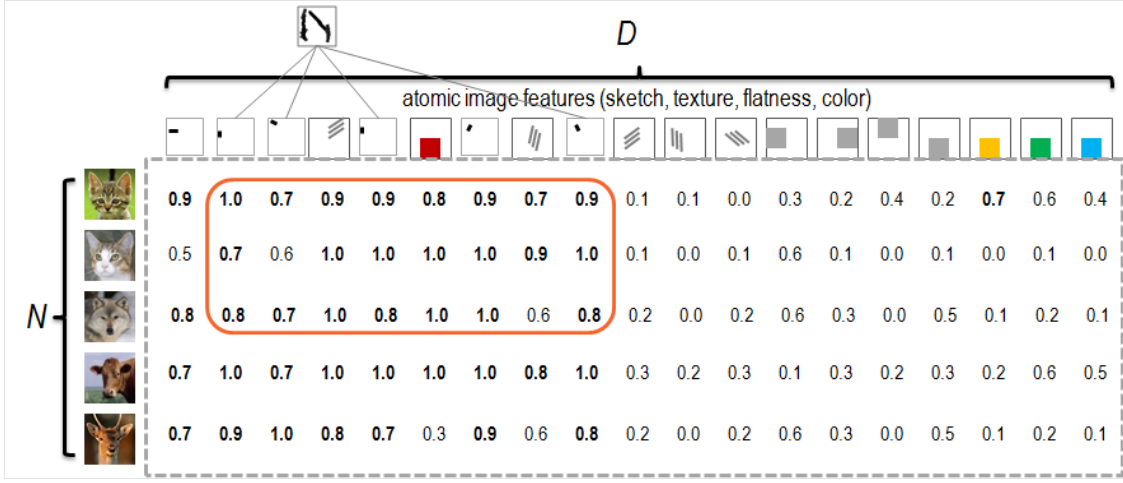


Figure 4.15: The data matrix \mathbf{R} measured on images. Reprinted/modified, with permission, from (Zhang and Zhu, 2011) [141].

4.3.2 AOT: Reconfigurable Object Templates

An AOT consists of a number of configurations of parts, which include

- i) structural variabilities (*i.e.* what parts appear);
- ii) geometric variabilities (*i.e.* where they appear).

The AOT embodies a stochastic context-free grammar to regulate the structural and geometric variabilities. It can efficiently capture high-order interaction of parts and compositionality.

Figure 4.13 illustrates an AOT for animal faces. The terminal nodes are shown as shaded rectangles. AND nodes are denoted as blue solid circles. OR nodes (for both geometric and structural variabilities) are drawn with dashed boxes together with typical configurations. The root node is an OR node with all the valid configurations of animal faces. It is branched into several sets of valid structural configurations as well as geometric configurations (represented as AND nodes) of two sub-parts: upper face (B) and mouth (C). As we move down the AOT, the upper face is in turn decomposed into left ear (D), right ear (E) and forehead (F). The structural and geometric configurations are not observed in training images, and thus are modeled by two separate sets of latent random variables:

Definition 3 (Definition 3). *The structural configuration \mathbf{b} of AOT is a binary activation vector of length K ($K = |\Delta^{(2)}|$), indicating which parts in $\Delta^{(2)}$ are activated. $\mathbf{b}_k = 1$ means HIT_k is activated and appears in the image.*

Definition 4 (Definition 4). *The geometric configuration τ of AOT is a list of transforms (translation, rotation and scaling) applied to the parts in $\Delta^{(2)}$.*

The AOT (*e.g.* in Figure 4.13) defines a set of valid configurations for \mathbf{b} and τ , and puts a probability distribution $p(\tau, \mathbf{b}; \text{AOT})$ on this set.

The complete likelihood for an AOT is defined as

$$p(\mathbf{I}, \tau, \mathbf{b} | \text{AOT}, \beta) = p(\tau, \mathbf{b} | \text{AOT}) \cdot p(\mathbf{I} | \tau, \mathbf{b}, \beta), \quad (4.28)$$

and the image likelihood conditioned on the configuration $(\boldsymbol{\tau}, \mathbf{b})$ is a log-linear form following Eq. (4.27):

$$p(\mathbf{I}|\boldsymbol{\tau}, \mathbf{b}, \boldsymbol{\beta}) = \exp \left\{ \sum_{k=1}^K \mathbf{b}_k \left(\sum_{j=1}^D \beta_{k,j} r_{\boldsymbol{\tau}(j)}(\mathbf{I}) - \log Z_k \right) \right\} q(\mathbf{I}), \quad (4.29)$$

Here we slightly abuse the notation and use $\boldsymbol{\tau}$ as a warping function that maps $\{1, \dots, D\}$ to an integer-valued index. This in effects matches a location (and orientation, scale) in the template to a location in the image. For out-of-bound situation, $r_{\boldsymbol{\tau}(j)} = 0$ if $\boldsymbol{\tau}(j) < 1$ or $\boldsymbol{\tau}(j) > D$. $\boldsymbol{\beta}$ is a $K \times D$ real-valued matrix denoting a set of K HITs for part templates. $\boldsymbol{\beta}$ is extremely sparse, and there are only around $10 \sim 30$ out of D (D can easily exceed 10^6) non-zero entries in each row. Z_k 's are normalizing constants.

We shall call the log-likelihood ratio $\log \frac{p}{q}$ as a *template matching score*, which measures the information gain of explaining the image by the foreground object model instead of the background model:

$$\text{Score}(\mathbf{I}) = \log \frac{p(\mathbf{I}|\boldsymbol{\tau}, \mathbf{b}, \boldsymbol{\beta})}{q(\mathbf{I})} = \sum_{k=1}^K \text{Score}(\text{HIT}_k, \mathbf{I}) \quad (4.30)$$

where

$$\text{Score}(\text{HIT}_k, \mathbf{I}) = \mathbf{b}_k \left(\sum_{j=1}^D \beta_{k,j} r_{\boldsymbol{\tau}(j)}(\mathbf{I}) - \log Z_k \right). \quad (4.31)$$

We assume that the structural configuration is independent from the geometric configuration, and thus

$$p(\boldsymbol{\tau}, \mathbf{b}|\text{AOT}) = p(\boldsymbol{\tau}|\text{AOT}^{\text{geo}}) \cdot p(\mathbf{b}|\text{AOT}^{\text{str}}), \quad (4.32)$$

where AOT^{geo} is a sub-AOT consisting of only geometric OR nodes (see Figure 4.13), and AOT^{str} consisting of only structural OR nodes. This facilitates fast inference. In our work, AOT^{geo} is a hierarchical deformation tree similar to the active basis model [170].

4.3.3 Learning AOT from Images

The terminal nodes $\Delta^{(2)}$ and the non-terminal nodes in AOT are learned automatically from training images. We first describe how to learn $\Delta^{(2)}$ (the HITs for part templates) using an EM-type block pursuit algorithm. The same algorithm is applied recursively to learn $\Delta^{(3)}$ (compositions of HITs). Then we introduce a graph compression algorithm to learn a compact set of non-terminal AND/OR nodes.

4.3.3.1 Block Pursuit on Data Matrix

Data matrix. The learning is performed on the data matrix \mathbf{R} as shown in Figure 4.15. Each row of \mathbf{R} is a feature vector for an image in \mathcal{X}_+ . \mathbf{R} is not necessarily a rectangular matrix, as images of different sizes produce feature vectors of varying lengths. But for simplicity, we assume all positive training images are roughly aligned and have the same size as the object template (this assumption is released in 4.3.3.2). Therefore \mathbf{R} is a matrix with N (number of positive examples) rows and D (number of all candidate features) columns, and each entry $\mathbf{R}_{ij} = r_j(\mathbf{I}_i)$ is a feature response ($0 \leq \mathbf{R}_{ij} \leq 1$). Larger value of \mathbf{R}_{ij} means feature j appears in image \mathbf{I}_i with higher probability.

On the data matrix, we pursue large blocks $\{\mathcal{B}_k : k = 1, \dots, K\}$ with lots of 1's, which correspond to HITs that appear frequently and with high confidence. A block is specified by a set of common features (columns) shared by a set of examples (rows). The significance of block \mathcal{B}_k is measured by the summation over the block:

$$\text{Score}(\mathcal{B}_k) = \sum_{\substack{i \in \text{rows}(\mathcal{B}_k) \\ j \in \text{cols}(\mathcal{B}_k)}} (\beta_{k,j} \mathbf{R}_{i,j} - \log z_{k,j}) \quad (4.33)$$

where $\text{rows}(\cdot)$ and $\text{cols}(\cdot)$ denote the rows and columns of block \mathcal{B}_k . $\text{cols}(\mathcal{B}_k)$ corresponds to the selected features in HIT_k ; and $\text{rows}(\mathcal{B}_k)$ are the examples on which HIT_k is activated. $\beta_{k,j}$ is the multiplicative parameter of feature j in HIT_k , and $z_{k,j}$ is the individual normalizing constant determined by $\beta_{k,j}$. See Eq. 4.25 for estimation of $z_{k,j}$.

The score of \mathcal{B}_k is equal with the summation of Eq. (4.31) over positive examples $\{\mathbf{I}_i : i = 1, \dots, N\}$:

$$\text{Score}(\mathcal{B}_k) = \sum_{i=1}^N \text{Score}(\text{HIT}_k, \mathbf{I}_i). \quad (4.34)$$

If we have already identified K blocks $\{\mathcal{B}_k : k = 1, \dots, K\}$, then the total score for all the blocks is equal with the summation of Eq. (4.30) over positive examples:

$$\sum_{k=1}^K \text{Score}(\mathcal{B}_k) = \sum_{i=1}^N \text{Score}(\mathbf{I}_i) = \sum_{i=1}^N \log \frac{p(\mathbf{I}_i)}{q(\mathbf{I}_i)}. \quad (4.35)$$

So pursuing blocks by maximizing Eq. (4.35) corresponds to maximum likelihood estimation and the information projection principle.

Information projection. Recall that we pursue a series of models starting from $q(\mathbf{I})$ to approximate the target distribution $f(\mathbf{I})$ governing training positives \mathcal{X}_+ . This corresponds to maximizing the log-likelihood $\log p(\mathbf{I})$ on \mathcal{X}_+ . Initially $p = q$, and the data matrix has a log-likelihood $L_0(\mathbf{R})$. After pursuing K blocks, the resulting image log-likelihood is

$$L(\mathbf{R}, \boldsymbol{\beta}, \mathbf{b}) = L_0(\mathbf{R}) + \sum_{k=1}^K \text{Score}(\mathcal{B}_k). \quad (4.36)$$

In the above equation, we have used the vector representation in Eq. (4.27) and (4.29). Here we denote the dictionary of HITs in a $K \times D$ real-valued sparse matrix $\boldsymbol{\beta}$. And we denote the structural configurations on all N images in a $K \times N$ binary sparse matrix \mathbf{b} . The k -th block can then be denoted as a pair of sparse vectors $(\mathbf{b}_{k,:}, \boldsymbol{\beta}_{k,:})$, where the non-zero items in $\boldsymbol{\beta}_{k,:}$ denotes columns of \mathcal{B}_k , and the non-zero items in $\mathbf{b}_{k,:}$ denote the rows of \mathcal{B}_k .

Block pursuit is a penalized maximum likelihood estimation problem, minimizing a two-term cost function:

$$-L(\mathbf{R}, \boldsymbol{\beta}, \mathbf{b}) + \text{penalty}(\boldsymbol{\beta}), \quad (4.37)$$

which measures how well the dictionary of templates $\Delta^{(2)}$ (encoded in $\boldsymbol{\beta}$) explain the data matrix through their activations \mathbf{b} . The penalty term is an l_0 penalty on $\boldsymbol{\beta}$:

$$\text{penalty}(\boldsymbol{\beta}) = \eta \cdot \sum_{j=1}^D \mathbf{1}_{\boldsymbol{\beta}_{j \neq 0}} \quad (4.38)$$

where $\mathbf{1}()$ is an indicator function. η controls the trade-off between the two terms, and we find $\eta = 0.1$ usually leads to good learning results.

In a fully expanded form, the optimization problem in (4.37) is:

$$\begin{aligned} \min_{\mathbf{b}, \beta} & -\frac{1}{N} \left(\sum_{i=1}^N \sum_k \mathbf{b}_{k,i} \sum_{j=1}^D \beta_{k,j} \mathbf{R}_{i,j} - \log Z_k \right) \\ & + 0.1 \cdot \sum_{k,j} \mathbf{1}_{\beta_{k,j} \neq 0} \end{aligned} \quad (4.39)$$

$$\text{where } Z_k = \frac{1}{|\mathcal{X}_-|} \sum_{\mathbf{I} \in \mathcal{X}_-} \exp \left\{ \sum_{j=1}^D \beta_{k,j} r_j(\mathbf{I}) \right\}$$

We also enforce that the coefficients $\beta_{k,:}$ of HIT_k is confined within a local region of the object window (*e.g.* the top-left subwindow in the 3 by 3 grid in Figure 4.19), and for each local region, exactly one block is activated for each image example so that the activated HITs do not overlap.

Due to the highly non-convex l_0 penalty, a naive global optimization algorithm would result in exponential complexity of $O(2^{K \times D})$. Inspired by the matching pursuit algorithm for signal reconstruction under sparse constraints, we propose a shared matching pursuit algorithm generalized from the one used in [170]. This algorithm greedily selects blocks with largest scores, and has a linear complexity $O(KD)$.

EM-type iterations. The rows of blocks (*i.e.* activations of parts \mathbf{b}) are not observed on training images, and we need to repeatedly infer them using the estimated AOT model. So the block pursuit algorithm as outlined below is an EM-type algorithm that alternate between model parameters β and latent variables \mathbf{b} :

The learned blocks can be ranked by the score (or information gain) in Eq. (4.33) and the blocks with small scores are discarded.

4.3.3.2 Deformable Block Pursuit

In the previous subsection, we assume the images are aligned so that the visual parts of different images appear in similar locations. To apply the block pursuit algorithm for non-aligned images or images with articulated objects, we expand the **E1** step in the baseline block pursuit algorithm by inferring both geometric and structural configurations. Now the input is a set of feature vectors $\mathbf{R} = \{\mathbf{R}_{1,:}, \dots, \mathbf{R}_{N,:}\}$ with different dimensions, but the coefficient matrix remains to be $K \times D$. Similar to Eq. (4.37), the objective is:

$$\min_{\beta, \mathbf{b}, \tau} -L(\mathbf{R}, \beta, \mathbf{b}, \tau) + \text{penalty}(\beta), \quad (4.40)$$

where

$$\begin{aligned} L(\mathbf{R}, \beta, \mathbf{b}, \tau) &= L_0(\mathbf{R}) + \\ & \sum_{i=1}^N \sum_k \mathbf{b}_{k,i} \sum_{j=1}^D (\beta_{k,j} \mathbf{R}_{i,\tau_i(j)} - \log z_{k,j}). \end{aligned} \quad (4.41)$$

In the E step, we not only solve for the best structural configuration \mathbf{b} but also infer the best geometric configuration τ_i on each positive example $\mathbf{R}_{i,:}$ using the inference algorithm in Section 4.3.4. τ_i denotes the localization (*i.e.* position x, y , rotation o , scale s) for the object bounding box, part bounding boxes as well as primitives on image \mathbf{I}_i . Part bounding boxes sit at canonical locations relative to the object center. Each part is subject to a random local perturbation $(\Delta x, \Delta y, \Delta o, \Delta s)$, which is independent of other parts.

Algorithm 2: Block Pursuit for Part Learning

Input: Data matrix \mathbf{R} , initialization of structural configuration $\mathbf{b}^{(0)}$, the number of blocks K .

Output: Coefficient matrix $\beta^{(T)}$ and structural configuration $\mathbf{b}^{(T)}$ after T iterations.

```
1  $t \leftarrow 1$ . Compute the correlation matrix  $Corr$  between each pair  $(j, j')$  of features.
2 For each candidate feature  $r_j$ , and for a grid of possible values  $\{\beta^{(m)}\}$  for the coefficient  $\beta$ ,
  compute the corresponding normalizing constant  $z_j^m$ , and the corresponding expectations
   $\{E[r_j; \beta^{(m)}]\}$  for all  $m$ :
   $E[r_j; \beta^{(m)}] = \frac{1}{|\mathcal{X}_-|} \sum_{\mathbf{I} \in \mathcal{X}_-} r_j(\mathbf{I}) \exp\{\beta^{(m)} r_j(\mathbf{I})\}$ 
3 repeat
4   for  $k = 1$  to  $K$  do
5     Compute the average response on activated positive examples:  $\bar{r}_{k,j}^+ = \frac{1}{N} \sum_{i=1}^N \mathbf{b}_{k,i} \mathbf{R}_{i,j}, \forall j$ 
6     Find the best  $\beta_{k,j}$  and  $z_{k,j}$  by  $\beta_{k,j}^* \leftarrow \beta^{(\mu)}, z_{k,j}^* \leftarrow z_j^{(\mu)},$ 
7      $\mu = \arg \min_m (E[r_j; \beta^{(m)}] - \bar{r}_{k,j}^+)^2$ 
8     Then compute the gain of feature  $j$  for the  $k$ -th block:
9     
$$\text{gain}_{k,j} = \begin{cases} \beta_{k,j}^* \bar{r}_{k,j}^+ - \log z_{k,j}^* & j \in S_{c(t)} \\ 0 & \text{otherwise} \end{cases}$$

10    end
11     $\beta \leftarrow \mathbf{0}$ 
12    for  $k = 1$  to  $K$  do
13      repeat
14        update  $\beta_{k,:}$ :
15        
$$j^* \leftarrow \arg \max_j \text{gain}_{k,j}$$

16        
$$\beta_{k,j}^{(t)} \leftarrow \beta_{k,j}^*, z_{k,j}^{(t)} \leftarrow z_{k,j}^*$$

17        
$$\text{gain}_{k,j'} \leftarrow 0, \forall j' \text{ s.t. } Corr(j, j') > \text{Thres} = 0.9$$

18      until  $\max_j \text{gain}_{k,j} < \gamma$ , or the maximum allowed number of selected features is reached;
19    end
20     $\mathbf{b} \leftarrow \mathbf{0}$ .
21    for  $i = 1$  to  $N$  do
22      repeat
23        
$$k^* \leftarrow \arg \max_k \sum_j \beta_{k,j}^{(t)} \mathbf{R}_{i,j} - \log z_{k,j}^{(t)}$$

24        
$$\mathbf{b}_{k^*,i}^{(t)} \leftarrow 1$$

25        set  $\mathbf{b}_{k,i}^{(t)} = 0$  if non-zero entries of  $\beta_{k,:}$  and  $\beta_{k^*,:}$  overlap.
26      until all values in  $\mathbf{b}$  are assigned;
27    end
28     $t \leftarrow t + 1$ 
29 until  $\mathbf{b}$  converges;
```

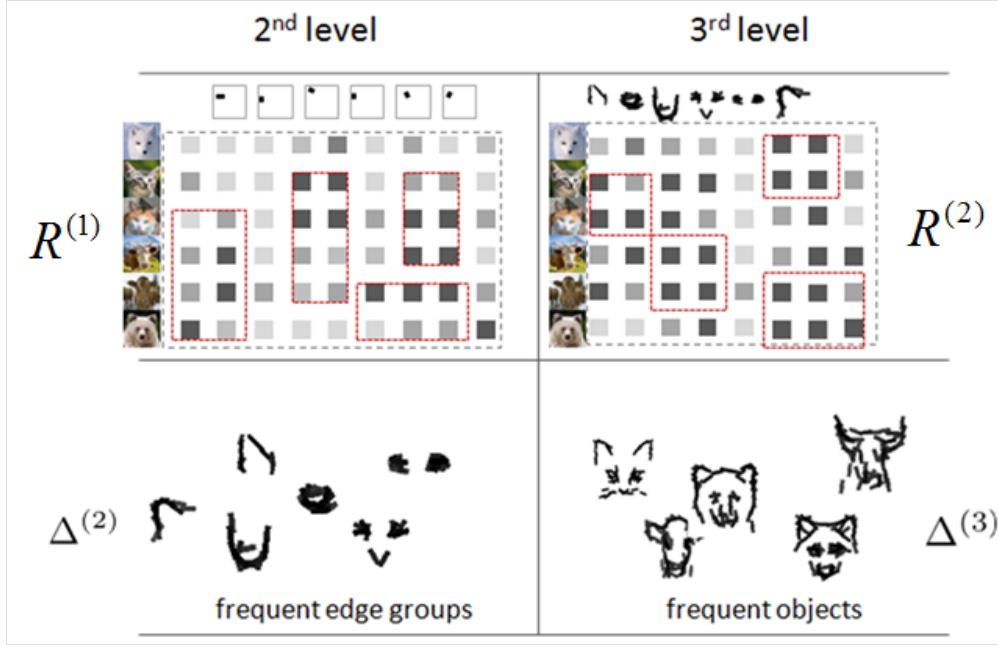


Figure 4.16: Block pursuit on images. Reprinted/modified, with permission, from (Zhang and Zhu, 2011) [141].

Similarly, primitives are subject to independent local perturbations around their canonical locations inside their parent part bounding box. The inference algorithm contains one round of bottom-up steps followed by one round of top-down steps. In the bottom-up steps, the local perturbations are accounted in local maximizations (Up-2, Up-4). The optimal localization of object bounding box is first inferred (Up-6); then the optimal localizations of parts are obtained by retrieving the arg-max perturbations of parts (Down-5). The optimal localizations of primitives are found similarly.

4.3.3.3 Recursive Block Pursuit

Now we have pursued K blocks or terminal nodes $\Delta^{(2)} = \{\text{HIT}_k, k = 1, \dots, K\}$. We then augment the data matrix by K new columns consisting of responses on the HITs. For clarity, we denote \mathbf{R} as $\mathbf{R}^{(1)}$ to indicate responses on $\Delta^{(1)}$. And we denote $\mathbf{R}^{(2)}$ as the newly computed responses on $\Delta^{(2)}$ for the N images. Each entry of $\mathbf{R}^{(2)}$ is a template matching score:

$$\mathbf{R}_{i,k}^{(2)} = \text{Score}(\text{HIT}_k, \mathbf{I}_i). \quad (4.42)$$

The block pursuit algorithm or its deformable version can be carried on recursively on $\mathbf{R}^{(2)}$. This leads to a compositional hierarchy. In our experiments, we find the simple three-level (object-part-primitive) hierarchy works well for detecting articulated objects in cluttered images.

4.3.3.4 Graph Compression

So far we are focused on pursuing blocks to identify meaningful parts from the training images and assume a trivial structural AOT which allows for any activation patterns of parts as long as they don't overlap. This can be problematic as the parts that co-appear in different locations are usually correlated, and certain structural

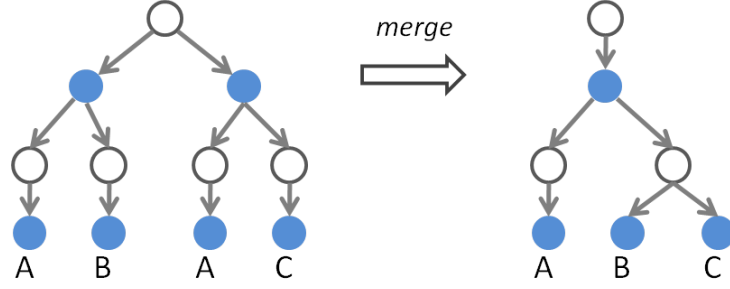


Figure 4.17: Graph compression by sharing. Reprinted/modified, with permission, from (Zhang and Zhu, 2011) [141].

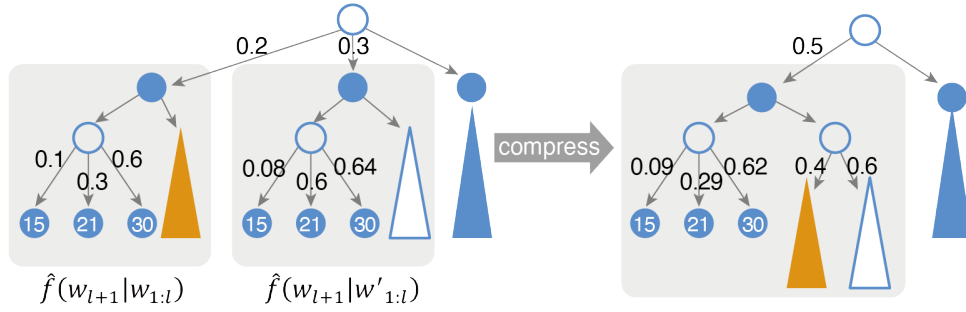


Figure 4.18: Graph compression by merging. Reprinted/modified, with permission, from (Zhang and Zhu, 2011) [141].

configurations of parts should be prohibited (*e.g.* bear’s left ear + cat’s right ear). In particular, such a flexible model may easily match cluttered background patches and cause a large number of false positives. To learn the structural AOT from examples, we extend the optimization problem of block pursuit by adding a prior model $p(\mathbf{b}; \text{AOT}^{\text{str}})$ for structural configuration \mathbf{b} and a hyper prior controlling the complexity of AOT^{str} . This results in an objective function with four terms:

$$\begin{aligned} \min_{\beta, \mathbf{b}, \tau, \text{AOT}^{\text{str}}} & -L(\mathbf{R}, \mathbf{b}, \beta, \tau) + \text{penalty}(\beta, \mathbf{b}) \\ & -\log p(\mathbf{b}; \text{AOT}^{\text{str}}) + \gamma|\text{AOT}^{\text{str}}| \end{aligned} \quad (4.43)$$

where $|\text{AOT}^{\text{str}}|$ denotes the total number of nodes in the AOT, and the first two terms are exactly the block pursuit cost function and penalty in Eq. (4.41). To solve (4.43), we adopt a coordinate descent method which alternate between (β, \mathbf{b}) and AOT^{str} :

- i) Given AOT^{str} , solve for (β, \mathbf{b}) using the block pursuit algorithm. Here we need to adapt the **E1** step, such that \mathbf{b} is inferred using dynamic programming on AOT^{str} .
- ii) Given (β, \mathbf{b}) , solve for AOT^{str} by minimizing $-\log p(\mathbf{b}; \text{AOT}^{\text{str}}) + \gamma|\text{AOT}^{\text{str}}|$.

Step ii) is a penalized maximum likelihood problem with log-likelihood term $L^{\text{str}} = \log p(\mathbf{b}; \text{AOT}^{\text{str}})$. It is solved by a graph compression procedure as follows. Initially, we form a giant AOT that has one root OR node branching over the object configurations in $\Delta^{(3)}$. For example:

$$\text{AOT}_0 = (1, 6, 20, 41) \cup (1, 6, 34, 49) \cup (6, 18, 27, 52) \dots$$

where \cup means OR, and each vector means non-zero entries of \mathbf{b} (*i.e.* activated parts) in one configuration from $\Delta^{(3)}$.

We call this AOT_0 a *memorization AOT* since it simply memorizes the observed structural configurations. Since the number of configurations is combinatorial, this AOT is huge and tends to overfit. Then we apply an iterative compression procedure that includes two operators:

- **Sharing.** This operator restructures the AOT by sharing parts. For example, $(1, 6, 20, 41) \cup (1, 6, 34, 49) \Rightarrow (1, 6, ((20, 41) \cup (34, 49)))$.
- **Merging.** This operator merges OR nodes with similar branching probabilities, and re-estimates the merged probabilities.

The two operators are illustrated in Figure 4.17 and 4.18. Each of the two operators results in a loss in log-likelihood $\Delta L^{\text{str}} \leq 0$ and a reduction in model complexity (number of nodes) $\Delta|\text{AOT}^{\text{str}}| < 0$. We decide to apply the merging or sharing operator if

$$\Delta L^{\text{str}} - \gamma \cdot \Delta|\text{AOT}^{\text{str}}| > 0 \quad (4.44)$$

i.e. the reduction in complexity outweighs the loss of log-likelihood.

Re-parameterization of the γ factor. Directly optimizing Eq.(4.44) requires fine tuning of the γ parameter, and the optimal value of $\gamma \in [0, +\infty)$ is very sensitive to the training data. We adopt a robust re-parameterization of γ using another parameter $\alpha \in [0, 1]$. Observing that Eq.(4.44) is essentially testing whether two distributions are the same, we propose to use the χ^2 test with significance level $1 - \alpha$ (where $\alpha \in [0, 1]$) to approximately implement the decision in Eq.(4.44). If the branching probabilities of the two OR nodes are (a_1, \dots, a_M) and (b_1, \dots, b_M) with $\sum_i a_i = 1, \sum_i b_i = 1, a_i > 0, b_i > 0, \forall i$, then the χ^2 test statistic is computed as $\chi^2 = \sum_i (a_i - b_i)^2 / a_i^2$. We compare this value to $F_{M-1, 1-\alpha}$ which can be looked up in the F-table. If $\chi^2 < F_{M-1, 1-\alpha}$, then we decide merge these two OR nodes. In the experiments, we use α as the control parameter for model complexity instead of γ .

4.3.4 Inference on AOTs

As the AOT is a tree-structured graphical model, so a dynamic programming procedure is in place for efficient inference. In particular, the dynamic programming takes the form of recursive SUM and MAX operations, which has been commonly used in hierarchical object models [38, 134, 170] with variations. The inference algorithm is an interleaved procedure between dynamic programming on AOT^{geo} and dynamic programming on AOT^{str} :

Algorithm 3: Inference by recursive SUM-MAX

Input: Testing image \mathbf{I} , AOT, β and $\{Z_k\}$.

Output: i) Geometric configuration τ : detected location, rotation and scaling for the whole object, activated parts and activated primitives. ii) Structural configuration \mathbf{b} : which parts are activated.

Up-1 Compute atomic feature response $r_j(\mathbf{I}) = \text{SUM1}(x_j, y_j, o_j, s_j)$ for all locations (x_j, y_j) , rotations o_j and scalings s_j of the atomic filter (*e.g.* image primitive).

Up-2 Perform local maximization on SUM1 maps over local translation, rotation and scaling to obtain $\text{MAX1}(x, y, o, s)$ and $\text{ARGMAX1}(x, y, o, s)$, where MAX1 stores local maximum responses and ARGMAX1 stores the local transformation of primitives that result in the local maxima.

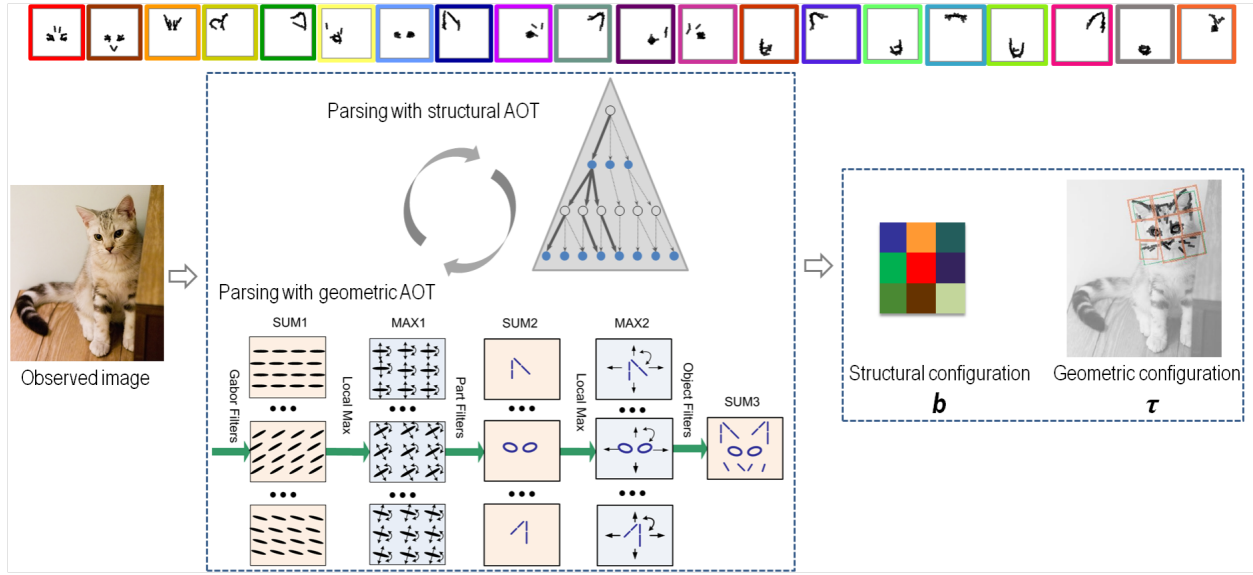


Figure 4.19: Inference of AOT on an observed image. **The top row:** the dictionary of parts displayed with different colors in the border. Showing top 20 of them. **The bottom row:** the parsing procedure with the structural AOT and geometric AOT. The structural AOT stores an AND-OR grammar that generates a set of admissible configurations (*i.e.* activation of parts) on an image. In this AOT, we divide the object template into a 3 by 3 grid, and within each of the 9 cells, only one part is allowed to be activated. A parse from the structural AOT results in a structural configuration (k_1, \dots, k_9) illustrated as a 3 by 3 color code. A parse from the geometric AOT results in the geometric configuration of activated primitives shown as black bars overlayed on the image. We also show the whole geometric parse tree by displaying the transformed object and part bounding boxes. Reprinted/modified, with permission, from (Zhang and Zhu, 2011) [141].

- Up-3** For each part k ($k = 1, \dots, K$), compute part score maps $\text{SUM2}_k(x, y, o, s)$ for all locations, orientations and scales by transforming the sparse part template (*i.e.* HIT template) $\beta_{k,:}$ by translation (x, y) , rotation o and scaling s and computing the dot product between $\beta_{k,:}$ and the portion of MAX1 map under the transformed part window:

$$\text{SUM2}_k(x, y, o, s) = \sum_{j=1}^D \beta_{k,j} \text{MAX1}(\tau_{x,y,o,s}(x_j, y_j, o_j, s_j)) - \log Z_k,$$

where $\tau_{x,y,o,s}(x_j, y_j, o_j, s_j)$ the destination transformation of the primitive (x_j, y_j, o_j, s_j) after it moves with the template, which itself is transformed by translation (x, y) , rotation o and scaling s .

- Up-4** Compute MAX2, ARGMAX2 maps for all parts by local maximization on SUM2 maps.

- Up-4.5** For each transformation (x, y, o, s) of the object, collect the all the part scores as a K dimensional vector $\mathbf{r}^{(2)} = (r_1^{(2)}, \dots, r_K^{(2)})$. Infer the best structural configuration \mathbf{b}^* using AOT^{str} based on $\mathbf{r}^{(2)}$.

- Up-5** For each transformation (x, y, o, s) of the object, compute the SUM3 score by applying the object filter specified by $\hat{\mathbf{b}}$:

$$\text{SUM3}(x, y, o, s) = \sum_{k=1}^K \mathbf{b}_k^* \cdot \mathbf{r}_k^{(2)},$$

where $\mathbf{r}^{(2)}$ is the collected vector of MAX2 scores of parts and $\hat{\mathbf{b}}$ is the best structural configuration both computed in **Up-4.5**.

- Up-6** Compute the global maximum of SUM3 score maps over all object transformations (x, y, o, s) .
- Down-6** Retrieve the detected (absolute) transformation $(\hat{x}, \hat{y}, \hat{o}, \hat{s})$ of the object from the maximization in **Up-6**.
- Down-5** Compute the temporary transformation $\{\tau_{\hat{x}, \hat{y}, \hat{o}, \hat{s}}(x_k, y_k)\}$ for each activated part k (such that $\mathbf{b}_k > 0$) on **I** by letting the part move with the object, where (x_k, y_k) is the canonical location of the part center relative to the object center.
- Down-4** For each activated part $k \in \{k : \mathbf{b}_k > 0\}$, retrieve the detected (absolute) transformation $(\hat{x}_k, \hat{y}_k, \hat{o}_k, \hat{s}_k)$ of the k -th part from ARGMAX2 maps computed in **Up-4**.
- Down-3** For each activated part k , for each of its activated primitive $j \in \{\beta_{k,j} > 0\}$, compute its temporary transformation $\{\tau_{\hat{x}_k, \hat{y}_k, \hat{o}_k, \hat{s}_k}(x_j, y_j, o_j)\}$ on **I** by letting the primitive move with the part, where (x_j, y_j, o_j) is the canonical location and rotation of the j -th primitive in the part template.
- Down-2** For each activated part k , for each of its activated primitive j , retrieve its detected (absolute) transformation $(\hat{x}_j, \hat{y}_j, \hat{o}_j, \hat{s}_j)$ on **I**.
-

In **Up-4.5**, the procedure to infer the best structural configuration \mathbf{b} also takes the form of recursive SUM-MAX. The input is a K dimensional vector $\mathbf{r}^{(2)} = (\mathbf{r}_1^{(2)}, \dots, \mathbf{r}_K^{(2)})$ of all candidate part scores. For notational convenience, we represent the sparse binary vector \mathbf{b} as its non-zero entries $(k_1, k_2, \dots) \subset \{1, \dots, K\}$. By induction, assume for all the OR nodes in level l , we have found the best parse tree that generates the sub-configurations (k_1, \dots, k_l) with exactly l parts being activated. We have also computed the scores of these OR nodes achieved by the best parse trees. Then for an OR node OR^{l+1} in level $l+1$ (the parent level), we find its best parse tree by the following two steps:

- SUM** Identify this OR node's children nodes, which are AND nodes. For each of the AND nodes AND_i^{l+1} , compute its score by summation of :

$$\text{Score}(\text{AND}_i^{l+1}) = \sum_{\text{OR}_j^l \in \text{Children}(\text{AND}_i^{l+1})} \text{Score}(\text{OR}_j^l)$$

- MAX** Find the best branch of this OR node by maximization:

$$i^* = \arg \max_i \text{Score}(\text{AND}_i^{l+1}).$$

Retrieve the best parse tree by concatenating the best sub-parse-trees in $\text{Children}(\text{AND}_{i^*}^{l+1})$. From this parse tree, we get the non-zero entries $(k_1^*, \dots, k_{l+1}^*)$ of the best sub-configuration \mathbf{b}^* . We then compute the best score of this OR node by:

$$\text{Score}(\text{OR}^{l+1}) = \sum_{k=1}^K \mathbf{b}_k^* \mathbf{r}_k^{(2)}$$

4.3.5 Example: The Synthesized 1D Text AOT Learning

We illustrate the AOT model in a supervised setting, where the ground-truth labels for training images and object bounding boxes are given. We measure the performance of classification and detection for articulated objects. We synthesize a 1D example where we know the underlying AOT as ground truth that generates the training data. Let AOT^* be the true AOT. Figure 4.20 shows AOT^* for sentences composed of three parts: subject + linking verb + adjective/present participle (*e.g.* winter is now leaving). The three parts are

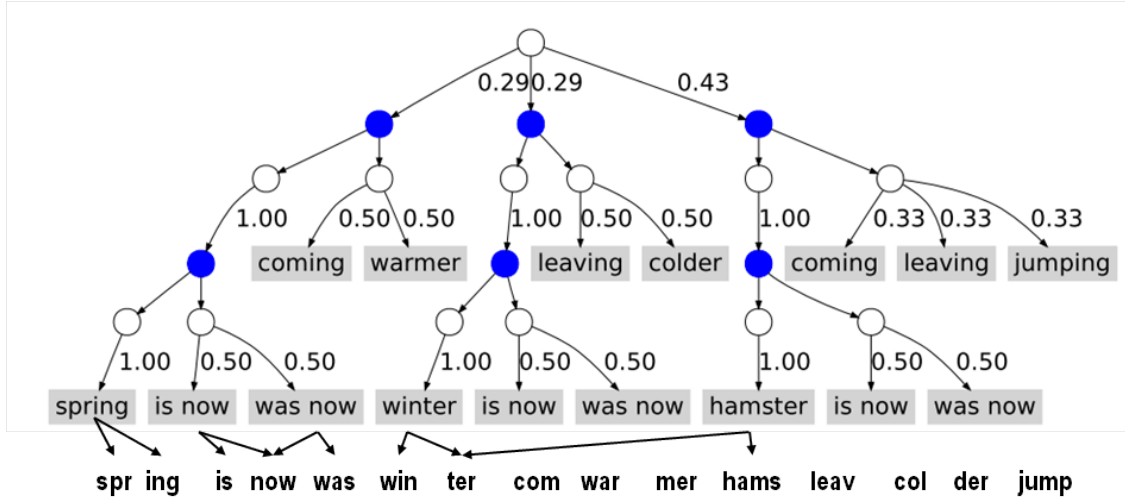


Figure 4.20: A stochastic AND-OR template for generating a sentence composed by three parts: subject + linking verb + adjective/present participle, such as “hamster is now jumping”. Shaded circles denote AND nodes. Empty circles represent OR nodes. Shaded rectangles are terminal nodes. Reprinted/modified, with permission, from (Zhang and Zhu, 2011) [141].

1 st level $\Delta^{(1)}$						2 nd level $\Delta^{(2)}$			3 rd level $\Delta^{(3)}$	
	frequency	gain		frequency	gain		frequency	gain		
now	0.0202	0.0806	ming	0.0092	0.0368	is now	0.012	0.083	<u>spring is now coming</u> <u>winter was now cold</u> <u>hamster is now jumping</u>	
s no	0.0200	0.0801	nter	0.0086	0.0345	was now	0.008	0.058		
ing	0.0206	0.0619	mste	0.0084	0.0334	hamster	0.007	0.052		
no	0.0203	0.0610	as	0.0110	0.0329	leaving	0.006	0.040		
s n	0.0201	0.0603	amst	0.0082	0.0328	winter	0.005	0.031		
is n	0.0133	0.0530	hams	0.0082	0.0327	spring	0.005	0.027		
as n	0.0119	0.0478	inte	0.0076	0.0302	coming	0.004	0.026		
is	0.0114	0.0456	wint	0.0073	0.0293	jumping	0.003	0.019		
ter	0.0151	0.0454	ping	0.0072	0.0288	warmer	0.002	0.014		
ster	0.0098	0.0392	int	0.0095	0.0284	colder	0.002	0.012		
was	0.0097	0.0388	avin	0.0071	0.0284	jumpin	0.000	0.001		
ving	0.0096	0.0383								
ring	0.0094	0.0375								

Figure 4.21: **Left:** The learned dictionary of terminal nodes $\Delta^{(1)}$ for three/four letter groupings (white space is included). We only show the top ones, together with their frequencies and information gains side by side, up to a constant multiple. **Middle:** The learned second level dictionary $\Delta^{(2)}$ for words composed by entries in the children dictionary $\Delta^{(1)}$. **Right:** The learned dictionary for sentences as combinations of $\Delta^{(2)}$ entries. Reprinted/modified, with permission, from (Zhang and Zhu, 2011) [141].

correlated, such that not all combinations of the three parts are admissible. For example, the combination *spring is now jumping* is not allowed. Each part is in turn composed of a prefix and a postfix. Each part, prefix/postfix and letter can be missing (*i.e.* occluded by random letters) with a small probability (0.01). Finally, random letters of varying lengths are inserted between the three parts of the sentence.

4.3.5.1 The Data and Data Matrix

Table 4.1 shows several example strings generated by this underlying AOT. Our goal is to learn an AOT from the sampled example strings, and compare the learned AOT with the underlying one in Figure 4.20 to

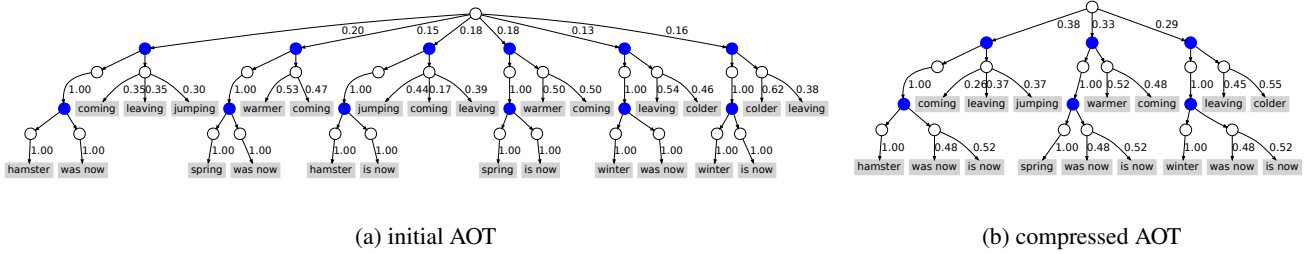


Figure 4.22: **Left:** the initial AOT naively constructed. **Right:** the compressed AOT. Both are obtained from the same 100 training sequences sampled from the underlying AOT in Figure 4.20. The model complexity parameter α is set to 0.05. The initial AOT has 13 free parameters. The compressed AOT has only 9 free parameters and successfully recovers the structure of the true underlying AOT. Reprinted/modified, with permission, from (Zhang and Zhu, 2011) [141].

study the effectiveness of the learning algorithm in identifying its parts and composite structures.

Table 4.1: String examples.

1. nkfnwkn spring yzxyxu was now jvzeaw warmert gprh
2. oqsdq bov hamster iwxw was now tdxtzby coming bjxp
3. lhtuwbcdzfb hamster aquo is now zgoclu jumping mmqrlu
4. jlmzrsrl winter vmqdle is now napla leaving douggekwh

4.3.5.2 Recursive Block Pursuit for Text Data

We first identify frequent substrings of length l ($l = 3$ or 4), such as “ing”, “ster”, as significant blocks in the data matrix. These blocks are selected into the first level dictionary $\Delta^{(1)}$ (Figure 4.21).

Once $\Delta^{(1)}$ is learned, the strings are re-encoded using the entries in $\Delta^{(1)}$. We then construct a new data matrix by collecting co-occurrences of $\Delta^{(1)}$ entries. As a result, frequent combinations such as “spr”+“ing” are identified as significant blocks and selected into the second level *word* dictionary $\Delta^{(2)}$. An entry in the word level dictionary covers 6 to 8 letters. The word dictionary contains many duplicate or overlapping entries, such as “hamster” and “amster”. The nuance entries like “amster” are pruned by a greedy procedure of finding best matching and local inhibition. In the end, only high frequency words remain in the top of $\Delta^{(2)}$ (Figure 4.21). Notice that compared to $\Delta^{(1)}$, $\Delta^{(2)}$ contains much less ambiguity. Finally the level 3 dictionary (sentences) $\Delta^{(3)} = \{\text{“spring is now coming”, } \dots\}$ is easily obtained by identifying frequent combinations of words.

4.3.5.3 Evaluation on the Text Example

In this experiment, we are interested in studying the following factors that influence model identifiability:

- n : training sample size.
- s : the average length of random letters inserted between two words (see Table 4.1) in the underlying AOT. When s is small, words are hard to separate, which results in large ambiguity. s implies the articulation and separation of parts in images.
- α : the parameter controlling model complexity.

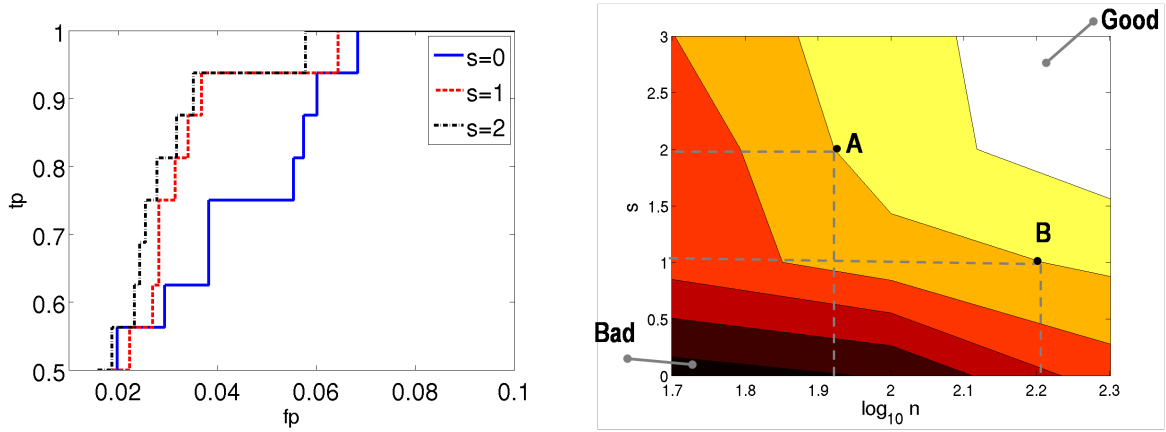


Figure 4.23: The effect of the separation parameter s and training sample size n on learned dictionary. **Left:** ROC curves for different separation parameters. **Right:** AUC as a function of separation s and sample size n . Reprinted/modified, with permission, from (Zhang and Zhu, 2011) [141].

n and s are parameters of training data, and α is a parameter of the learning algorithm.

We design a comprehensive experiment with about 10^5 combinations of parameters. And the results are summarized in Figure 4.23 and 4.24.

Evaluating the learned terminal nodes of AOT. To compare the underlying true terminal nodes Δ_{true} with the learned terminal nodes Δ . We use the ROC curve and AUC (area under ROC curve) to evaluate the difference between manually labeled ground-truth Δ_{true} and the learned Δ with entries ranked by information gain. Figure 4.23 (left figure) plots three ROC curves for three different values of s for sample size $n = 100$. After repeating this for different n , we obtain a series of ROC comparisons. To summarize this, Figure 4.23 (right figure) shows the isolines of AUC as a function of two variables: s the separation parameter, and n the training sample size. Take the two points A , B as an example, when the separation decreases by 1 from A to B , we need about twice ($10^{0.3}$) as many training examples to achieve the same AUC.

We find that the block pursuit algorithm can successfully identify the terminal nodes Δ for $n > 100$ and $s > 2$, up to an accuracy of $\text{AUC} = 0.99$.

Evaluating the graph topology and branching probabilities of the learned structural AOT. Another important factor is the parameter α which controls model complexity. We set α to different values and compress the sentence-level dictionary $\Delta^{(3)}$ into a series of AOTs with varying complexity. We then compute the distance between the learned AOT and the underlying true AOT* shown in Figure 4.20. We use the Kullback-Leibler divergence as the distance between AOTs, which is estimated by Monte-Carlo simulation on their samples. We first sample m configurations from AOT*:

$$\{\mathbf{b}_1, \dots, \mathbf{b}_m\} \sim p(\mathbf{b}; \text{AOT}^*).$$

Then we compute:

$$\mathcal{K}(\text{AOT}^* | \text{AOT}) \approx \sum_{i=1}^m \log \frac{p(\mathbf{b}_i; \text{AOT}^*)}{p(\mathbf{b}_i; \text{AOT})}$$

For each \mathbf{b}_i ($i = 1, \dots, m$) we compute $p(\mathbf{b}; \text{AOT})$ and $p(\mathbf{b}; \text{AOT}^*)$ by the product of branching probabilities along the paths of AOT that generates \mathbf{b} .

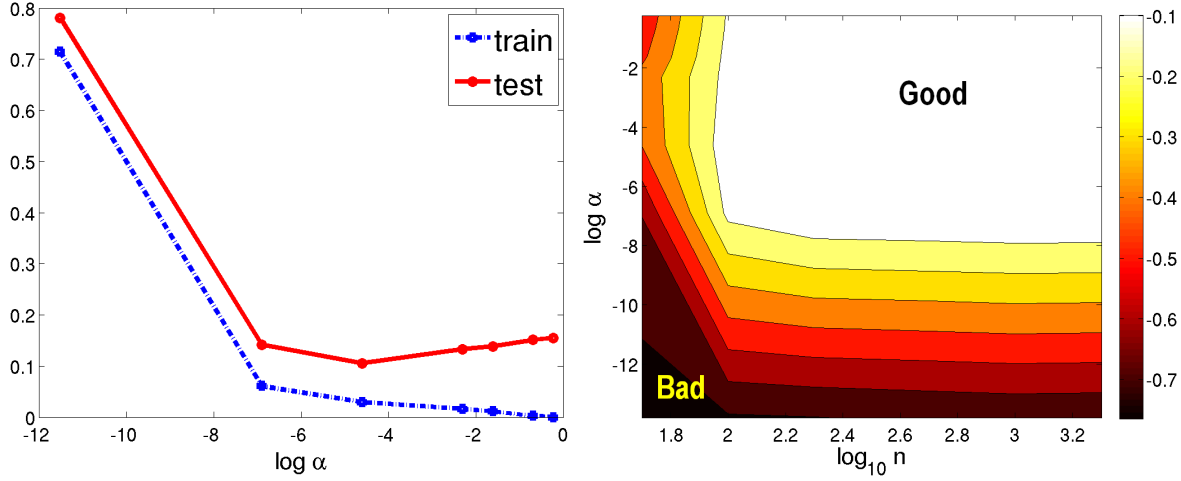


Figure 4.24: The effect of the model complexity α and training sample size n on model generalizability. **Left:** Error of learned model (KL divergence) as a function of model complexity α , plotted on training and testing data respectively. **Right:** KL divergence as a function of n and α . Reprinted/modified, with permission, from (Zhang and Zhu, 2011) [141].

We perform 10^5 repeated cross validations to compare training error and testing error. Different from the classification scenario, the *training error* for AOT is defined as the KL divergence $\mathcal{K}(\hat{f}_n^{\text{train}} \parallel \text{AOT})$ between the learned AOT and training data \hat{f}_n^{train} . Here \hat{f}_n^{train} denotes the empirical distribution on the training data $\{\mathbf{b}_1^{\text{train}}, \dots, \mathbf{b}_n^{\text{train}}\}$. The *testing error* for AOT is defined as the KL divergence $\mathcal{K}(\hat{f}_m^{\text{test}} \parallel \text{AOT})$, where $\hat{f}_m^{\text{test}} = \{\mathbf{b}_1^{\text{test}}, \dots, \mathbf{b}_m^{\text{test}}\}$ is another independent sample from AOT*. And $m \gg n$. In Figure 4.24 (left), the horizontal axis is the logarithm of α which is sampled at seven points (10^{-6} , 10^{-3} , 10^{-2} , 0.1, 0.2, 0.5, 0.8), and the vertical axis denotes the model error computed as KL divergence. Recall that large α results in larger more complexity. When the model becomes more complex, the training error always decreases, but the testing error would first decrease but then increase due to overfitting. Figure 4.24 (right) shows at what sample size and what α values can we successfully recover the generating grammar. The horizontal axis is the logarithm of training sample size $\log_{10} n$, and the vertical axis is $\log \alpha$. The color intensity in this 2D contour map denotes the testing error.

We find that the graph compression algorithm can successfully identify the AOT for $n > 100$ and $10^{-6} < \alpha < 10^{-2}$, up to a tolerance of 0.1 in KL divergence.

4.4 Structure Learning: Unsupervised Structure Learning

In unsupervised learning of stochastic And-Or grammars, we aim to learn a grammar from a set of unannotated i.i.d. data samples (e.g., natural language sentences, quantized images, action sequences). The objective function is the posterior probability of the grammar given the training data:

$$P(G|X) \propto P(G)P(X|G) = \frac{1}{Z} e^{-\alpha \|G\|} \prod_{x_i \in X} P(x_i|G)$$

where G is the grammar, $X = \{x_i\}$ is the set of training samples, Z is the normalization factor of the prior, α is a constant, and $\|G\|$ is the size of the grammar. By adopting a sparsity prior that penalizes the size of the grammar, we hope to learn a compact grammar with good generalizability. In order to ease the learning

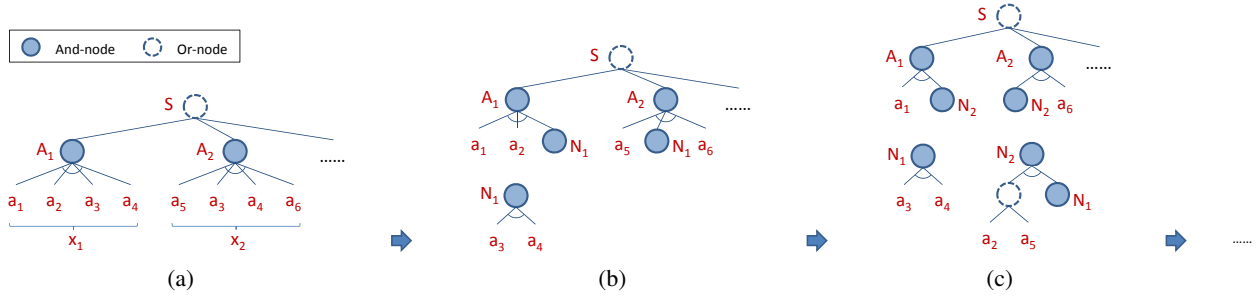


Figure 4.25: An illustration of the learning process. (a) The initial grammar. (b) Iteration 1: learning a grammar fragment rooted at N_1 . (c) Iteration 2: learning a grammar fragment rooted at N_2 . Reprinted/modified, with permission, from (Tu, Pavlovskaja and Zhu, 2013) [153].

process, during learning we approximate the likelihood $P(x_i|G)$ with the Viterbi likelihood (the probability of the best parse of the data sample x_i). Viterbi likelihood has been empirically shown to lead to better grammar learning results [114, 143] and can be interpreted as combining the standard likelihood with an unambiguity bias [151].

4.4.1 Algorithm Framework

We first define an initial grammar that generates the exact set of training samples. Specifically, for each training sample $x_i \in X$, there is an Or-rule $S \rightarrow A_i$ in the initial grammar where S is the start symbol and A_i is an And-node, and the probability of the rule is $\frac{1}{\|X\|}$ where $\|X\|$ is the number of training samples; for each x_i there is also an And-rule $A_i \rightarrow a_{i1}a_{i2} \dots a_{in}$ where a_{ij} ($j = 1 \dots n$) are the terminal nodes representing the set of atomic patterns contained in sample x_i , and a set of relations are specified between these terminal nodes such that they compose sample x_i . Figure 4.25a shows an example initial grammar. This initial grammar leads to the maximal likelihood on the training data but has a very small prior probability because of its large size.

Starting from the initial grammar, we introduce new intermediate nonterminal nodes between the terminal nodes and the top-level nonterminal nodes in an iterative bottom-up fashion to generalize the grammar and increase its posterior probability. At each iteration, we add a grammar fragment into the grammar that is rooted at a new nonterminal node and contains a set of grammar rules that specify how the new nonterminal node generates one or more configurations of existing terminal or nonterminal nodes; we also try to reduce each training sample using the new grammar rules and update the top-level And-rules accordingly. Figure 4.25 illustrates this learning process. There are typically multiple candidate grammar fragments that can be added at each iteration, and we employ greedy search or beam search to explore the search space and maximize the posterior probability of the grammar. We also restrict the types of grammar fragments that can be added in order to reduce the number of candidate grammar fragments, which will be discussed in the next subsection. The algorithm terminates when no more grammar fragment can be found that increases the posterior probability of the grammar.

4.4.2 And-Or Fragments

In each iteration of our learning algorithm framework, we search for a new grammar fragment and add it into the grammar. There are many different types of grammar fragments, the choice of which greatly influences the efficiency and accuracy of the learning algorithm. Two simplest types of grammar fragments

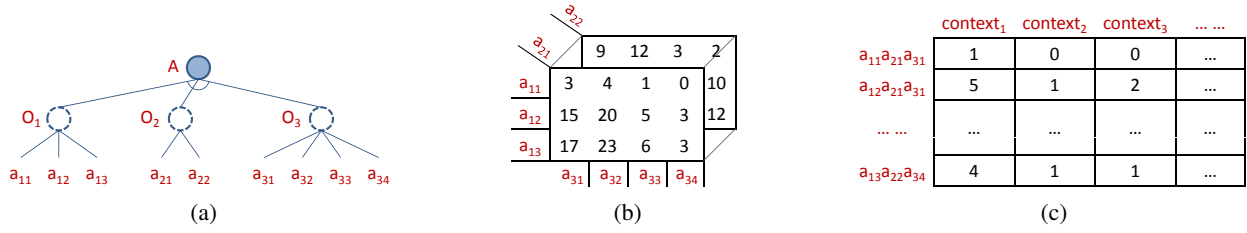


Figure 4.26: (a) An example And-Or fragment. (b) The n -gram tensor of the And-Or fragment based on the training data (here $n = 3$). (c) The context matrix of the And-Or fragment based on the training data. Reprinted/modified, with permission, from (Tu, Pavlovskaja and Zhu, 2013) [153].

are *And-fragments* and *Or-fragments*. An And-fragment contains a new And-node A and an And-rule $A \rightarrow a_1 a_2 \dots a_n$ specifying the generation from the And-node A to a configuration of existing nodes $a_1 a_2 \dots a_n$. An Or-fragment contains a new Or-node O and a set of Or-rules $O \rightarrow a_1 | a_2 | \dots | a_n$ each specifying the generation from the Or-node O to an existing node a_i . While these two types of fragments are simple and intuitive, they both have important disadvantages if they are searched for separately in the learning algorithm. For And-fragments, when the training data is scarce, many compositions modeled by the target grammar would be missing from the training data and hence cannot be learned by searching for And-fragments alone; besides, if the search for And-fragments is not properly coupled with the search for Or-fragments, the learned grammar would become large and redundant. For Or-fragments, it can be shown that in most cases adding an Or-fragment into the grammar decreases the posterior probability of the grammar even if the target grammar does contain the Or-fragment, so in order to learn Or-rules we need more expensive search techniques than greedy or beam search employed in our algorithm; in addition, the search for Or-fragments can be error-prone if different Or-rules can generate the same node in the target grammar.

Instead of And-fragments and Or-fragments, we propose to search for *And-Or fragments* in the learning algorithm. An And-Or fragment contains a new And-node A , a set of new Or-nodes O_1, O_2, \dots, O_n , an And-rule $A \rightarrow O_1 O_2 \dots O_n$, and a set of Or-rules $O_i \rightarrow a_{i1} | a_{i2} | \dots | a_{im_i}$ for each Or-node O_i (where $a_{i1}, a_{i2}, \dots, a_{im_i}$ are existing nodes of the grammar). Such an And-Or fragment can generate $\prod_{i=1}^n m_i$ number of configurations of existing nodes. Figure 4.26a shows an example And-Or fragment. It can be shown that by adding only And-Or fragments, our algorithm is still capable of constructing any context-free And-Or grammar. Using And-Or fragments can avoid or alleviate the problems associated with And-fragments and Or-fragments: since an And-Or fragment systematically covers multiple compositions, the data scarcity problem of And-fragments is alleviated; since And-rules and Or-rules are learned in a more unified manner, the resulting grammar is often more compact; reasonable And-Or fragments usually increase the posterior probability of the grammar, therefore easing the search procedure; finally, ambiguous Or-rules can be better distinguished since they are learned jointly with their sibling Or-nodes in the And-Or fragments.

To perform greedy search or beam search, in each iteration of our learning algorithm we need to find the And-Or fragments that lead to the highest gain in the posterior probability of the grammar. Computing the posterior gain by re-parsing the training samples can be very time-consuming if the training set or the grammar is large. Fortunately, we show that by assuming grammar unambiguity the posterior gain of adding an And-Or fragment can be formulated based on a set of sufficient statistics of the training data and is efficient to compute. Since the posterior probability is proportional to the product of the likelihood and the prior probability, the posterior gain is equal to the product of the likelihood gain and the prior gain, which we formulate separately below.

Likelihood Gain. Remember that in our learning algorithm when an And-Or fragment is added into the grammar, we try to reduce the training samples using the new grammar rules and update the top-level And-rules accordingly. Denote the set of reductions being made on the training samples by RD . Suppose in reduction $rd \in RD$, we replace a configuration e of nodes $a_{1j_1} a_{2j_2} \dots a_{nj_n}$ with the new And-node A , where a_{ij_i} ($i = 1 \dots n$) is an existing terminal or nonterminal node that can be generated by the new Or-node O_i in the And-Or fragment. With reduction rd , the Viterbi likelihood of the training sample x where rd occurs is changed by two factors. First, since the grammar now generates the And-node A first, which then generates $a_{1j_1} a_{2j_2} \dots a_{nj_n}$, the Viterbi likelihood of sample x is reduced by a factor of $P(A \rightarrow a_{1j_1} a_{2j_2} \dots a_{nj_n})$. Second, the reduction may make sample x identical to some other training samples, which increases the Viterbi likelihood of sample x by a factor equal to the ratio of the numbers of such identical samples after and before the reduction. To facilitate the computation of this factor, we can construct a *context matrix* CM where each row is a configuration of existing nodes covered by the And-Or fragment, each column is a context which is the surrounding patterns of a configuration, and each element is the number of times that the corresponding configuration and context co-occur in the training set. See Figure 4.26c for the context matrix of the example And-Or fragment. Putting these two types of changes to the likelihood together, we can formulate the likelihood gain of adding the And-Or fragment as follows (see the supplementary material for the full derivation).

$$\frac{P(X|G_{t+1})}{P(X|G_t)} = \frac{\prod_{i=1}^n \prod_{j=1}^{m_i} \|RD_i(a_{ij})\| \|RD_i(a_{ij})\|}{\|RD\| \|RD\|} \times \frac{\prod_c (\sum_e CM[e, c])^{\sum_e CM[e, c]}}{\prod_{e, c} CM[e, c]^{CM[e, c]}}$$

where G_t and G_{t+1} are the grammars before and after learning from the And-Or fragment, $RD_i(a_{ij})$ denotes the subset of reductions in RD in which the i -th node of the configuration being reduced is a_{ij} , e in the summation or product ranges over all the configurations covered by the And-Or fragment, and c in the product ranges over all the contexts that appear in CM .

It can be shown that the likelihood gain can be factorized as the product of two tensor/matrix coherence measures as defined in [93]. The first is the coherence of the n -gram tensor of the And-Or fragment (which tabulates the number of times each configuration covered by the And-Or fragment appears in the training samples, as illustrated in Figure 4.26b). The second is the coherence of the context matrix. These two factors provide a surrogate measure of how much the training data support the context-freeness within the And-Or fragment and the context-freeness of the And-Or fragment against its context respectively. See the supplementary material for the derivation and discussion.

The formulation of likelihood gain also entails the optimal probabilities of the Or-rules in the And-Or fragment.

$$\forall i, j \quad P(O_i \rightarrow a_{ij}) = \frac{\|RD_i(a_{ij})\|}{\sum_{j'=1}^{m_i} \|RD_i(a_{ij'})\|} = \frac{\|RD_i(a_{ij})\|}{\|RD\|}$$

Prior Gain. The prior probability of the grammar is determined by the grammar size. When the And-Or fragment is added into the grammar, the size of the grammar is changed in two aspects: first, the size of the grammar is increased by the size of the And-Or fragment; second, the size of the grammar is decreased because of the reductions from configurations of multiple nodes to the new And-node. Therefore, the prior gain of learning from the And-Or fragment is:

$$\frac{P(G_{t+1})}{P(G_t)} = e^{-\alpha(\|G_{t+1}\| - \|G_t\|)} = e^{-\alpha((ns_a + \sum_{i=1}^n m_i s_o) - \|RD\|(n-1)s_a)}$$

where s_a and s_o are the number of bits needed to encode each node on the right-hand side of an And-rule and Or-rule respectively. It can be seen that the prior gain penalizes And-Or fragments that have a large size but only cover a small number of configurations in the training data.

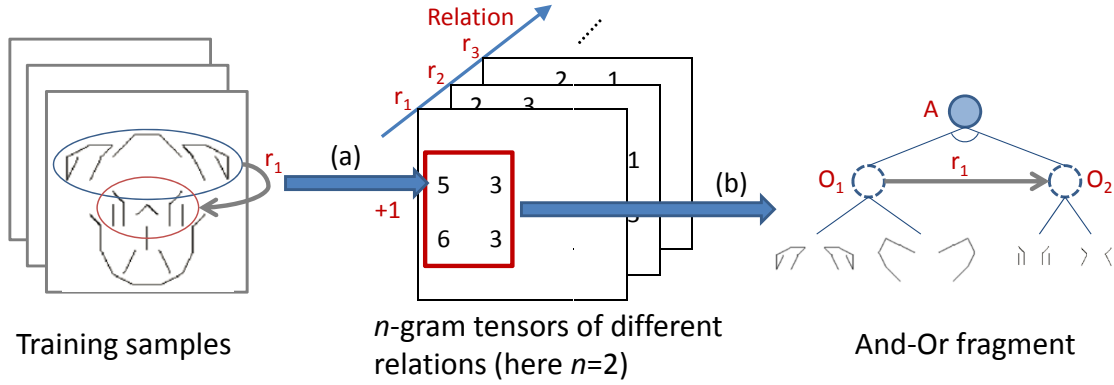


Figure 4.27: An illustration of the procedure of finding the best And-Or fragment. r_1, r_2, r_3 denote different relations between patterns. (a) Collecting statistics from the training samples to construct or update the n -gram tensors. (b) Finding one or more sub-tensors that lead to the highest posterior gain and constructing the corresponding And-Or fragments. Reprinted/modified, with permission, from (Tu, Pavlovskaja and Zhu, 2013) [153].

In order to find the And-Or fragments with the highest posterior gain, we could construct n -gram tensors from all the training samples for different values of n and different And-rule relations, and within these n -gram tensors we search for sub-tensors that correspond to And-Or fragments with the highest posterior gain. Figure 4.27 illustrates this procedure. In practice, we find it sufficient to use greedy search or beam search with random restarts in identifying good And-Or fragments. The algorithm runs reasonably fast: our prototype implementation can finish running within a few minutes on a desktop with 5000 training samples each containing more than 10 atomic patterns.

4.5 Structure Learning: Pruning from Full Graph

Learning the structure of a graphical model is generally hard. However, in certain cases, it is possible to convert the structure learning problem to a parameter learning problem. The learning-by-parsing framework is a learning framework that applies to situations where a complete and-or graph can be generated. One such case would be to generate all possible image tessellations, which we present in the later section as a case study.

In this section, we first present the general idea and framework of learning-by-parsing, then we discuss two case studies that used this learning framework.

4.5.1 General Framework

The main idea of learning-by-parsing is to prune the complete graph structure based on the estimated parameters. In this sense, learning the structure of a representation is just to learn its parameters.

Given a complete graph structure (*i.e.* the full grammar), the learning objective is to maximize the marginal log-likelihood of the given data based on this grammar. One standard way of achieving this goal when hidden variables are missing in the dataset is the expectation-maximization algorithm. EM-algorithm involves two steps for achieving maximum likelihood. The E-step, also known as expectation step, computes the inferred hidden variable based on the current model parameter. The M-step, a.k.a. maximization step, updates the model parameter based on the inferred variable from E-step. In the learning-by-parsing

Algorithm 3: Structure Learning of And-Or Grammars

Input: the training set X
Output: an And-Or grammar G

```
1  $G \leftarrow$  the initial grammar constructed from  $X$ 
2 while True do
3    $F \leftarrow \{\}$ 
4   repeat
5      $f \leftarrow$  an And-Or fragment with two Or-nodes and two leaf nodes constructed from a
       randomly selected bigram from  $X$ 
6     optimize the posterior gain of  $f$  using greedy or beam search via four operators:
       adding/removing Or-nodes, adding/removing leaf nodes
7     if  $f$  increases the posterior gain and  $f \notin F$  then
8       | add  $f$  into  $F$ 
9     end
10  until after a pre-specified number of iterations;
11  if  $F$  is empty then
12    | return  $G$ 
13  end
14   $f^* \leftarrow$  the fragment in  $F$  with the highest posterior gain
15  insert  $f^*$  into  $G$ 
16  reduce  $X$  using the grammar rules in  $f^*$  and update  $G$  accordingly
17 end
```

framework, the hidden variable is usually the parse graph of an And-Or graph. In practice, it is difficult to sum/integrate over all possible parse trees in the E-step, so various approximations exist to implement the standard EM-algorithm. A typical approximation used in this framework is Viterbi approximation, which gives the following form of EM-algorithm.

The whole framework is summarized in the following:

- Define a complete grammar for the given problem
- EM-algorithm for maximum likelihood estimation
 - E-step: Inference based on current parameter
 - M-step: Update parameter based on inferred variable
- Prune the complete grammar based on the estimated parameters

4.5.2 Example: Learning Image Tangram Model

4.5.2.1 Introduction

A typical scene category contains an enormous number of distinct scene configurations that are composed of objects and regions of varying shapes in different layouts. In image tangram model, a representation named Hierarchical Space Tiling (HST) is proposed to quantize the huge and continuous scene configuration space. Then, the HST is augmented with attributes (nouns and adjectives) to describe the semantics of the objects and regions inside a scene. The scene configurations and attributes are simultaneously learned following

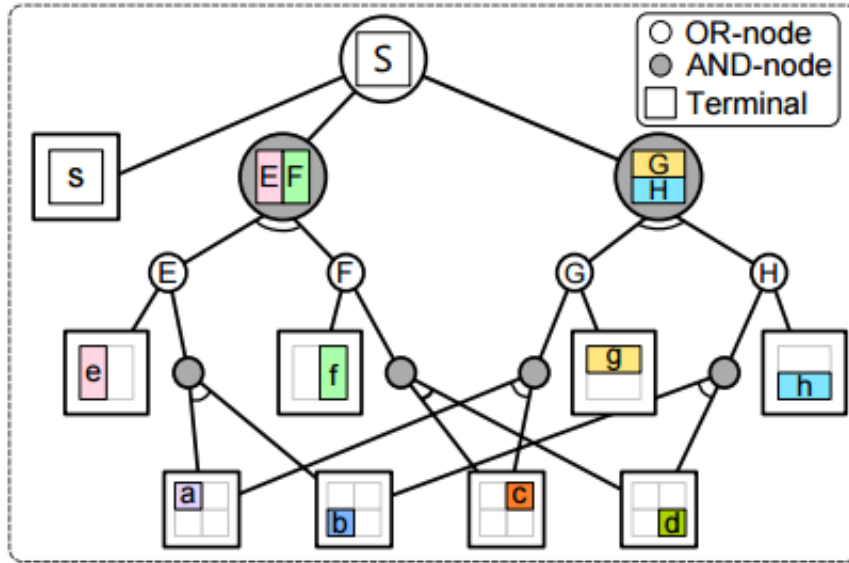
the learning-by-parsing framework from a collection of natural images associated with descriptive text. Given a test image, we compute the most probable parse tree with the associated attributes by dynamic programming. In applications, we apply this model to four tasks: scene classification, attribute recognition, attribute localization, and pixel-wise scene labeling.

4.5.2.2 Representation

The HST model consists of two components, HST-geo and HST-att respectively. HST-geo models the spatial configuration of a scene layout, while HST-att models the semantics of such a layout.

4.5.2.3 HST-geo

We divide the image lattice into an $n_w \times n_h$ grid and treat each cell as an atomic shape element, then organize these atomic shape elements in an And-Or Tree(AoT) structure. The following figure shows a HST-geo example with $n_w = n_h = 2$.



There are three types of nodes in the HST-geo:

- **or-nodes** V^{OR}

This type of nodes is shown as the hollow circles in the above figure, corresponding to the grammar rules like

$$r^{OR} : S \rightarrow s|EF|GH \quad (4.45)$$

which act as "switches" between the possible decompositions. The branching probabilities $p(s|S)$, $p(EF|S)$, $p(GH|S)$ account for the preference for each decomposition and can be learned.

- **and-nodes** V^{AND}

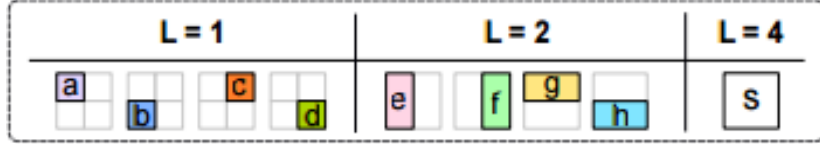
This type of nodes is shown as the solid circles in the above figure, corresponding to the grammar rules like

$$r^{AND} : E \rightarrow a \cdot b \quad (4.46)$$

which represent a fixed decomposition from a parent node E to its child nodes a and b. For simplicity, only two-way decompositions are considered in this model.

- **terminal nodes** V^T

This type of nodes is shown as the hollow squares in the above figure. The nodes in HST-geo can terminate at all levels to represent the visual concepts at multiple resolutions. We see a terminal node as a “scene part”, and the terminal nodes from all levels form a scene part dictionary $\Delta = V^T$ as the following.



The atomic shape elements are at the bottom of the hierarchy ($L = 1$). According to the grammar rules described above, a number of atomic shape elements compose higher-level nodes at different scales, locations and shapes. The “level” L here means the number of atomic shape elements being used. To avoid the combinatorial explosion, only regular shapes, i.e., squares and rectangles, are allowed. The HST-geo can also allow non-regular shape elements, such as triangles, parallelograms and trapezoids, which make the representation more flexible but complex.

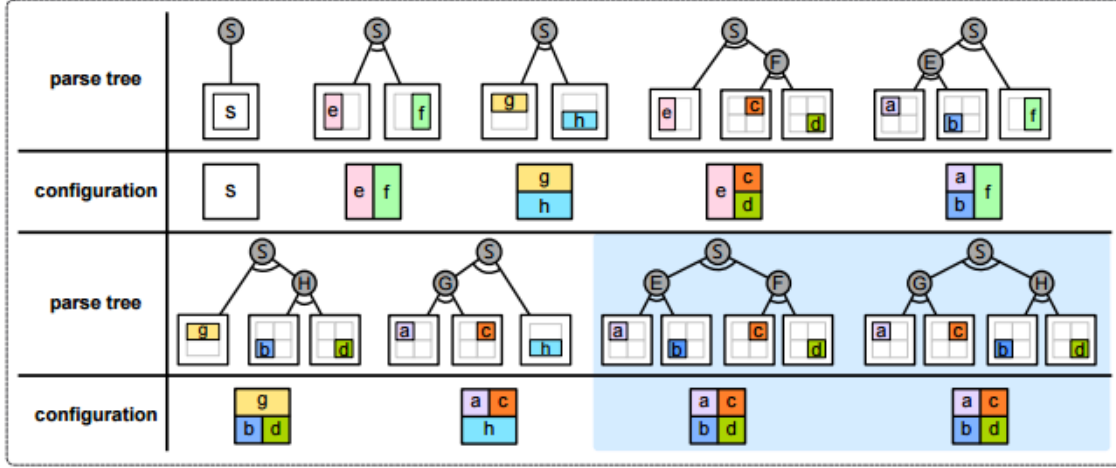
Formally, we define the HST-geo as a 4-tuple

$$HST - geo = (S, V^N, V^T; \theta) \quad (4.47)$$

where S is a start symbol at root and $V^N = V^{AND} \cup V^{OR}$ is a set of non-terminal nodes. Let v index the node and $Ch(v)$ denote its child node set. θ is a set of branching probabilities at Or-nodes.

$$\theta = \{\theta(v_i|v); v \in V^{OR}, v_i \in Ch(v)\} \quad s.t. \quad \sum_{i=1}^{|Ch(v)|} \theta(v_i|v) = 1; \forall v \in V^{OR} \quad (4.48)$$

The HST-geo is recursively defined with homogeneous structures. Starting from a root which is an Or-node, HST-geo generates alternating levels of And-nodes and Or-nodes, and stops at terminal nodes. The And-Or structure defines a full space of possible parsing with probabilistic context free grammar(PCFG), which we call a “full HST”. By selecting the branches at Or-nodes, a parse tree pt can be derived. Intuitively, when a parse tree collapses, it produces a planar configuration. We utilize this configuration to represent the layout/configuration of a scene. The following diagram enumerates all the parse trees and configurations generated from a 2×2 HST-geo using only squares and rectangles.



The parse trees highlighted in the blue panel in this diagram show the ambiguity in the HST-geo. The ambiguity arises from the shape elements shared by more than one parent node, which will admit two or more reasonable parse trees for one configuration. The ambiguity can be reduced during learning.

4.5.2.4 HST-att

Terminal nodes at HST-geo may have semantic labels, called scene attributes. Thus we extend the HST-geo by HST-att. There are two types of attributes.

- **Adjective attributes** A^{adj}

Those are adjectives, such as “green” and “cloudy”, that describe the characteristics of a scene.

- **Noun attributes** A^n

Those are nouns, such as “field” and “sky”, denote the objects and regions inside a scene. Each noun attribute, acting as an appearance-Or node, has a mixture of adjective attributes, *e.g.*, sky can be blue, cloudy or overcast.

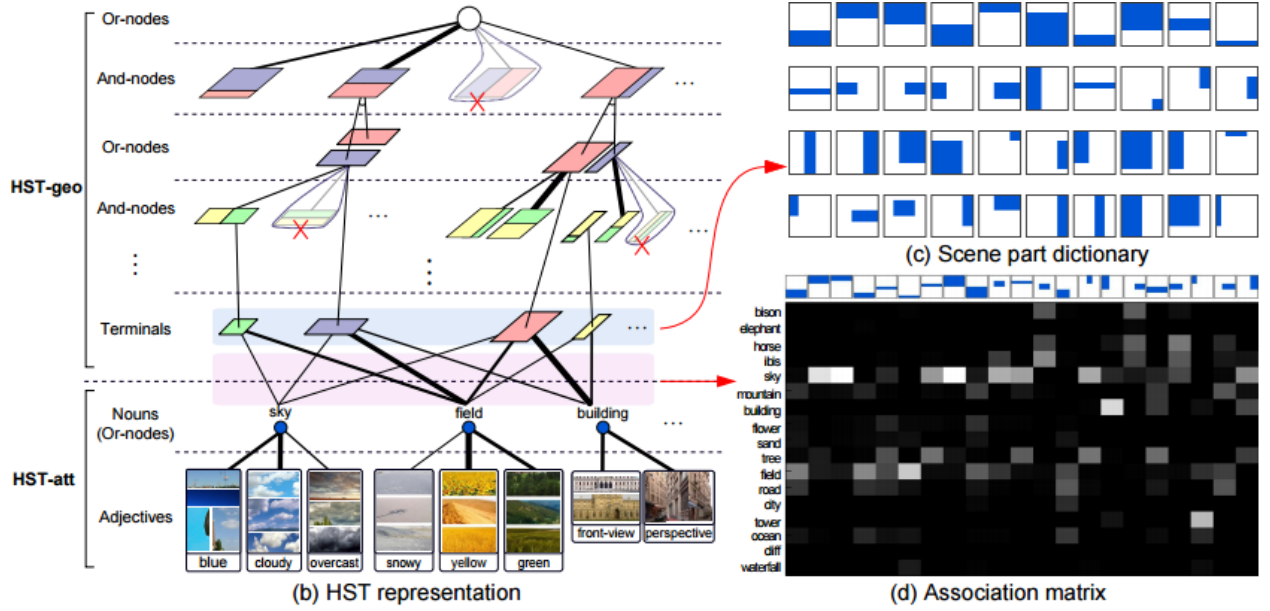
We link each terminal node $v \in \Delta$ in the HST-geo to a noun attribute $a^n \in A^n$, and further to an adjective attribute $a^{adj} \in A^{adj}$ according to an association matrix Φ .

$$\Phi : A^n \times \Delta \mapsto [0, 1], s.t. \sum_{a \in A^n} \Phi(a, v) = 1, \forall v \in \Delta \quad (4.49)$$

where the rows in Φ are the noun attributes and the columns are the scene parts, and we normalize the sum of each column to 1. Φ measures the probabilities of assigning noun attributes to certain scene parts, *e.g.*, “road” has high probability appearing at the bottom of an image. Formally, we define the HST-att as a 3-tuple

$$HST - att = (A^n, A^{adj}, \Phi). \quad (4.50)$$

The whole representation of HST is illustrated in the following figure.



4.5.2.5 Learning

As we stated in the introduction, the input of learning is a set of natural images $I = \{I_m\}_{m=1}^M$ and their text descriptions $A = \{A_m = (A_m^n, A_m^{adj})\}_{m=1}^M$, where $A_m^n \subset A^n$ and $A_m^{adj} \subset A^{adj}$ denote the noun and adjective attribute sets for the image I_m , respectively.

The hidden variables of HST are

$$\{pt_m^+ = (pt_m, \phi_m)\}_{m=1}^M \quad (4.51)$$

where pt_m is the inferred parse tree and ϕ_m is the attribute assignment.

Formally, ϕ_m is a mapping from the the inferred attribute set \hat{A}_m to the terminal node set of pt_m , i.e., $V^T(pt_m)$.

$$\phi_m : \hat{A}_m \times V^T(pt_m) \mapsto \{0, 1\} \quad (4.52)$$

where $\phi_m(a, v) = 1$ if an attribute a is assigned to a terminal node v and $\phi_m(a, v) = 0$ otherwise.

The objective of learning is to estimate the HST parameters, i.e., the branching probabilities θ and the association matrix Φ , by maximizing a log-likelihood.

$$(\theta^*, \Phi^*) = \arg \max_{\theta, \Phi} \log P(I, A; \theta, \Phi) \quad (4.53)$$

$$= \arg \max_{\theta, \Phi} \sum_{m=1}^M \log \sum_{pt_m^+} P(I_m, A_m, pt_m^+; \theta, \Phi). \quad (4.54)$$

We first separately learn the HST-geo and HST-att as an initial HST model. Then we jointly learn (θ, Φ) and infer the hidden parse trees $\{pt_m^+\}_{m=1}^M$. Starting from a full HST, we learn the branching probabilities and association matrix, prune the redundant branches, and finally get a compact model. Therefore, we transfer the structure learning of AoT to a tractable parameter learning problem. The complete learning algorithm is summarized as the following.

Algorithm 4: Learning Algorithm of HST

```

; // Initialization
1 Learn HST-geo with parameter  $\Theta^{(0)}$  based on the multi-scale segmentation.
2 Learn HST-att with parameter  $\Phi^{(0)}$  and train appearance models.
; // Iteratively learn HST
3 repeat
4   Jointly infer  $pt^+$  for each training sample.
5   Update  $\Theta^{(t+1)}$  in HST-geo.
6   Update  $\Phi^{(t+1)}$  in HST-att and retrain appearance models.
7 until convergence;

```

4.5.2.6 Initial Learning

The initial learning consists of learning HST-geo and learning HST-att individually. We first present the learning of HST-geo without text input. Since we do not have ground-truth scene configurations, we use multi-scale segmentations, corresponding to the coarse-to-fine scene configurations, to propose candidate terminal nodes or scene parts in the HST-geo. Given a training image I_m , we first adopt [39] to obtain a multi-scale segmentation by tuning $z \in \{300, 400, \dots, 5000\}$, where z controls the granularity of segmentation. Then we select six distinct segmented layers by comparing the adjacent layers in pixels, and compose a multi-scale segmentation set $C_m = \{(C_m^k, z_m^k)\}_{k=1}^6$, where C_m^k is one segmented layer with the control variable z_m^k . To learn the HST-geo, we estimate the branching probabilities Θ by maximizing a log-likelihood. We sum out the hidden variables C_m^k and pt_m .

$$\begin{aligned}
\Theta^* &= \arg \max_{\Theta} \log P(I; \Theta) \\
&\propto \arg \max_{\Theta} \sum_{m=1}^M \sum_{pt_m, k} P(I_m | C_m^k) P(C_m^k, pt_m; \Theta)
\end{aligned} \tag{4.55}$$

where $p(I|C^k)$ is the likelihood given one segmented layer and we omit the index m hereafter in the derivation for simplicity when there is no confusion.

$$\log P(I|C^k) \propto - \sum_{c \in \{R, G, B\}} \sum_{r \in C^k} \|I^c(r) - \bar{I}^c(r)\|^2 - z^k \tag{4.56}$$

where c is a color channel; r is a segmented region in C^k ; $I(r)$ is the image patch covered by r and $\hat{I}_c(r)$ is the average intensity of r at color channel c . This term measures the segmentation homogeneity of pixel intensity and penalizes the large z^k .

$p(C^k, pt; \Theta) \propto \exp\{-E(C^k, pt; \Theta)\}$, following the Gibbs distribution, is the joint probability with Θ being the parameters to be learned. Since the HST-geo embodies a PCFG, the contextual relations among the And-nodes are not considered. Thus the energy is defined on two potential terms corresponding to the Or-nodes and terminal nodes of a parse tree.

$$E(C^k, pt; \Theta) = \sum_{v \in V^{OR}(pt)} E^{OR}(v_i | v) + \lambda \sum_{v \in V^T(pt)} E^T(C^k(v) | v) \tag{4.57}$$

where λ is the parameter balancing the two terms ($\lambda = 0.25$ is set empirically through cross validation). $V^{OR}(pt)$ and $V^T(pt)$ denote the sets of Or nodes and terminal nodes in pt .

The energy of an Or-node is defined on its branching probability,

$$E^{OR}(v_i|v) = -\ln(v_i|v) = -\ln \frac{\#(v \rightarrow v_i)}{\sum_{i=1}^{|Ch(v)|} \#(v \rightarrow v_i)} \quad (4.58)$$

where $\#(v \rightarrow v_i)$ is the number of times that v selects the i -th node/branch $v_i \in Ch(v)$. We learn the branching probabilities in the following subsection. The energy for a terminal node is defined on the homogeneity of the terminal node in terms of the segmentation label, *i.e.*, how well the configuration of pt fits the segmented layer C^k .

$$E^T(C^k(v)|v) = -\ln \frac{\sum_{i \in C^k(v)} 1[l_i^k = l_v^k]}{|C^k(v)|} \quad (4.59)$$

where $1[\cdot]$ is the indicator function and $C^k(v)$ denotes the segmented patch covered by the terminal node v . In the k -th layer, l_i^k is the segmentation label of pixel i and l_v^k is the dominant label of terminal node v . In addition, the terminal nodes are allowed to be locally adjustable to fit the scene boundaries. We introduce 12 node activities including perturbations in location ($\delta(x) = [\pm 8, \pm 16]$), scale ($\delta(s) = [1 \pm \frac{1}{32}, 1 \pm \frac{1}{16}]$) and orientation ($\delta(a) = [\pm \frac{\pi}{48}, \pm \frac{\pi}{24}]$). Taking the multi-scale segmentations as input, we solve Eq.(4.55) by a learning-by-parsing method which is an EM-like strategy. The E-step infers the optimal parse trees pt^* which approximate the scene configurations with small error and low complexity by dynamic programming. The M-step estimates the parameters Θ by maximum likelihood estimation (MLE).

(i) E-step: parse tree inference. Keeping the current branching probabilities Θ fixed and assuming $p(I)$ is uniform, an optimal parse tree pt^* can be inferred from the HST-geo for each training sample.

$$\begin{aligned} pt^* &= \arg \max_{pt} \log P(pt|I; \Theta) \\ &= \arg \max_{pt} \log P(I, pt; \Theta) \\ &\approx \arg \max_{pt, k} \log P(I|C^k)P(C^k, pt; \Theta) \end{aligned} \quad (4.60)$$

Here, we use the best-segmented layer to approximate the summation of all layers. In practice, given a training image, we infer an optimal parse tree for each segmented layer by minimizing Eq.(4.57). Then we choose the best parse tree and segmented layer according to Eq.(4.60).

Because of the tree structure of HST-geo and the independence assumption in PCFG, the optimal parse tree can be obtained by Dynamic Programming (DP). Specifically, we start with calculating the data term (Eq.(4.59)) for each terminal node, then for the upper-level Or-node v , DP evaluates all its possible branches $v_i \in Ch(v)$ and selects the best one such that

$$v_i^* = \operatorname{argmin}_{v_i \in Ch(v)} (E^{OR}(v_i|v) + E(C^k(v_i, pt(v_i); \Theta))) \quad (4.61)$$

where $pt(v_i)$ is the sub-tree with v_i as the root and the sub-tree energy $E(C^k(v_i); pt(v_i); \Theta)$ is defined in Eq.(4.57).

(ii) M-step: update branching probabilities. We rewrite the objective function in Eq.(4.55) as

$$L(\Theta) = \sum_{m=1}^M \log \sum_{pt_m, k} P(I_m|C_m^k)P(C_m^k, pt_m; \Theta) + \sum_{v=1}^{|V^{OR}|} \alpha_v (1 - \sum_{i=1}^{|Ch(v)|} \theta(v_i|v)) \quad (4.62)$$

where α_v is the Lagrange multiplier for the branching probabilities at each Or-node to be normalized.

We estimate Θ by MLE, which takes the derivative of $L(\Theta)$ w.r.t. $\theta(v_i|v)$ and sets it to zero. We adopt the Viterbi algorithm [159], which is an approximated method using the optimal parse tree instead of all parse trees, and update the branching probabilities (see supplementary material for detailed derivation).

$$\theta^{(t+1)}(v_i|v) = \frac{1}{\alpha_v} \sum_{m=1}^M 1[v_i \in pt_m^*] \cdot P(pt_m^*|C_m^k; \Theta^t) P(I_m|C_m^k) \quad (4.63)$$

where pt_m^* and C_m^k are the parse tree and segmented layer inferred in the E-step, and $1[v_i \in pt_m^*]$ indicates if the branch v_i is selected in pt_m^* . Θ is set to be uniform as initialization. We repeat inferring parse trees (E-step) and updating parameters (M-step) until convergence. Finally, those branches whose probabilities are below a certain threshold (say 0.01) are pruned. We collect the terminal nodes from all levels, and they compose the scene part dictionary Δ .

Recall A^n is the noun attribute set, A^{adj} is the adjective attribute set and $\Phi : A^n \times \Delta \mapsto [0, 1]$ is the association matrix measuring the probabilities of assigning noun attributes to scene parts. For an image I_m , $\phi_m : A_m \times V^T(pt_m) \mapsto \{0, 1\}$ is the attribute assignment from the attribute set $A_m = (A_m^n; A_m^{adj})$ to the terminal node set $V^T(pt_m)$.

We compute Φ by counting the co-occurrence of nouns $a \in A^n$ and terminal nodes $v \in V^T(pt)$ in the training images.

$$\Phi(a, v) = \frac{\sum_{m=1}^M 1[a \in A_m^n] \cdot 1[v \in V^T(pt_m) \cdot \phi_m(a, v)]}{\sum_{a \in A^n} \Phi(a, v)} \quad (4.64)$$

where $\phi_m(a; v) \in \{0, 1\}$ indicates whether a noun attribute a is assigned to a terminal node v .

In learning the HST-geo stated before, an optimal parse tree pt^* can be inferred for each training sample. However, the correspondence between scene parts (terminal nodes in pt^*) and attributes is still unknown because the attributes are annotated at image level rather than on image regions. So we initialize ϕ_m by turning on all possible assignments, *i.e.*, $\phi_m(a; v) = 1, \forall (a; v) \in A_m^n \times V^T(pt_m^*), m = 1, \dots, M$.

Then, we learn the HST-att through an iterative procedure, including two steps.

(i) Update association matrix Φ through noun attribute localization. Given the current $\Phi^{(t)}$, we establish a bipartite graph $G(V^T(pt_m^*), A_m^n, \xi_m)$ for each training image, where $V^T(pt_m^*)$ denotes the terminal nodes in pt^*_m and A_m^n denotes the noun attributes from text. If $|V^T(pt_m^*)| \neq |A_m^n|$, add dummy nodes for balance. ξ_m is the edge set connecting $V^T(pt_m^*)$ and A_m^n , whose weight is defined as:

$$w(a, v) = \Phi^{(t)}(a, v) \cdot P(a|I(v)) \quad (4.65)$$

Let $F(I(v), a)$ be the score of classifying the image patch $I(v)$ as an attribute a , thus $p(a|I(v)) = \max_c \frac{\exp\{F^c(I(v), a)\}}{\sum_{a'} \exp\{F^c(I(v), a')\}}$. As defined in Section 4.5.2.4, one noun attribute has a mixture of adjectives, c is the number of adjectives and F^c denotes the adjective classifier. At initialization, $p(a|I(v))$ is set to be uniform, since the appearance models are empty.

We adopt the Hungarian algorithm [11] to solve the bipartite graph and get a one-to-one matching $\phi_m^{(t+1)}$, *i.e.*, for each image we localize the noun attributes to scene parts (terminal nodes). Then we update the association matrix $\Phi^{(t+1)}$ by recalculating Eq.(4.64).

(ii) Update attribute appearance models. For each noun attribute, as shown in Figure 4.28(b), we crop the image patches covered by the assigned terminal nodes and do clustering according to the given adjectives, such as “rocky mountain” and “snowy mountain”. Then for each cluster (*i.e.*, a noun and adjective pair), we train a kernel SVM classifier with one-versus-all mode as its appearance model. As in [161], we adopt bag-of-words (BoW) features on color histogram and SIFT, and utilize the histogram intersection kernel.

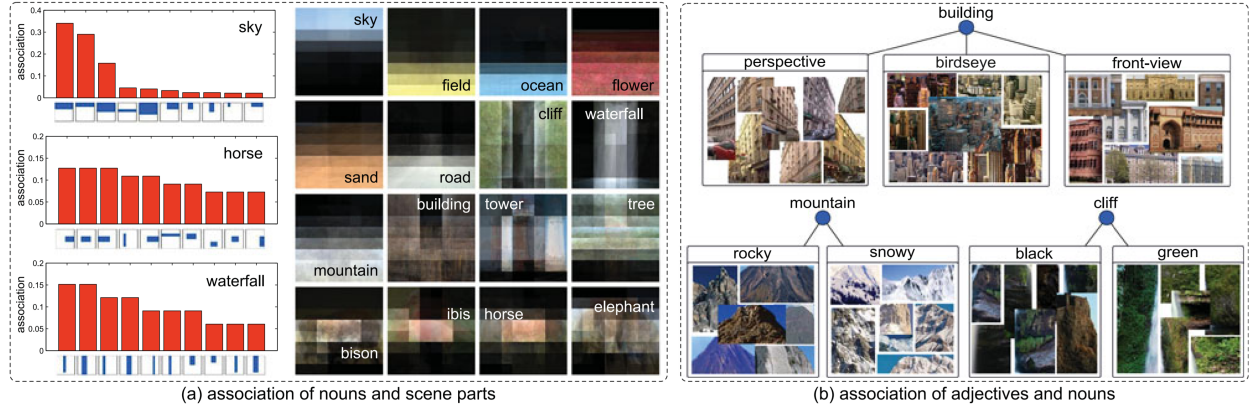


Figure 4.28: Analysis of HST-att learning. (a) The histogram on the left shows the association probability between a noun attribute and scene parts, where the horizontal axis indexes different scene parts and the vertical axis is the association probability from Φ . The right figure shows the learned spatial prior for each noun attribute, which is the average image of attribute patches. (b) The adjective clusters belonging to the noun attributes. Reproduced from Ref. [161] with permission of IEEE, © 2015.

Repeating the above steps till the change of $\{\phi_m\}_{m=1}^M$ below a threshold, finally, we get the association matrix Φ and the attribute appearance models.

Figure 4.28(a)(left) shows the association of noun attributes and scene parts/terminal nodes, where the horizontal axis indexes the scene parts and the vertical axis indexes the association probability from Φ . For example, “sky” has a higher probability to cover the top of an image and “horse” has a higher probability to cover the middle part. To qualitatively evaluate the association, for each noun attribute, we average the image patches assigned to it. Interestingly, as illustrated in Figure 4.28(a)(right), although learning in a weakly supervised way, our association shows the similar spatial priors of the object categories to [89] (see Figure.3 in [89]). Figure 4.28(b) shows that the image patches assigned to each noun are split into multiple clusters according to the given adjectives.

4.5.2.7 Joint Inference

Using the learned HST-geo and HST-att as an initial HST model, we can infer

$$pt_m^+ = (pt_m, \phi_m), \quad \phi_m : \hat{A}_m \times V^T(pt_m) \mapsto \{0, 1\}$$

directly from the image I_m and text description A_m , rather than multi-scale segmentation, by maximizing $p(I_m, A_m, pt_m^+; \Theta, \Phi) \propto \exp\{-E(I_m, A_m, pt_m^+; \Theta, \Phi)\}$. Let $\hat{a} = (\hat{a}^n, \hat{a}^{adj}) \in \hat{A}_m$ denote the inferred attribute. The energy is rewritten from Eq.(4.57).

$$\begin{aligned} E(I_m, A_m, pt_m^+; \Theta, \Phi) = & \sum_{v \in V^{OR}(pt_m)} E^{OR}(v_i|v) + \lambda_1 \sum_{v \in V^T(pt_m)} E^n(\hat{a}^n|v) \\ & + \lambda_2 \sum_{\hat{a}^n \in \hat{A}_m^n} E^{adj}(\hat{a}^{adj}|\hat{a}^n) + \lambda_3 \sum_{v \in V^T(pt_m)} E^T(\hat{a}|I(v)) \\ & + \sum_{\hat{a} \in \hat{A}_m} E^A(\hat{a}, A_m) \end{aligned} \quad (4.66)$$

where $(\lambda_1, \lambda_2, \lambda_3)$ are the parameters balancing the energy terms ($(\lambda_1, \lambda_2, \lambda_3) = (0.7, 0.1, 2)$ are set empirically through cross validation). The first term measures the scene configuration prior which is the same as Eq.(4.58). The second term measures the noun attribute association with the terminal node:

$$E^n(\hat{a}^n|v) = -\ln \Phi(\hat{a}^n, v) \quad (4.67)$$

The third term is designed to model the compatibility between a noun and an adjective

$$E^{adj}(\hat{a}^{adj}|\hat{a}^n) = -\ln P(\hat{a}^{adj}|\hat{a}^n) \quad (4.68)$$

where $p(a^{adj}|a^n) = \frac{\sum_{m=1}^M 1[a^n \in A_m^n] 1[a^{adj} \in A_m^{adj}]}{\sum_{m=1}^M 1[a^n \in A_m^n]}$ can be counted from the given text phrases. The fourth term is an attribute specific data term

$$E^T(\hat{a}|I(v)) = -\ln P(\hat{a}|I(v)) \quad (4.69)$$

where $I(v)$ is the image patch occupied by v and $p(\hat{a}|I(v)) = \frac{\exp\{F^c(I(v), \hat{a})\}}{\sum_{a'} \exp\{F^c(I(v), a')\}}$. $F(I(v), a)$ is the score of classifying $I(v)$ as attribute a . The last term $E^A(\hat{a}, A_m)$ assumes ∞ on attributes outside A_m and 0 otherwise, making sure $\hat{a} \in A_m$.

Because both the HST-geo and HST-att are tree structured, DP algorithm can be applied to jointly infer the parse tree pt_m and attribute assignment ϕ_m . In the joint inference, we start with calculating E^T for the terminal nodes. Then, for every terminal node, DP evaluates all possible attributes according to the sum of E^T , E^n and E^{adj} , and assigns a best attribute (a noun and adjective pair) to it. Next, DP iteratively proceeds to the upper-level Or-nodes and selects best branches until finds the optimal parse tree with associated attributes:

$$(pt_m^+)^* = \arg \max_{pt_m^+} P(I_m, A_m, pt_m^+; \Theta, \Phi) \quad (4.70)$$

In training, we use E^A to constrain the consistency between inferred attributes and given text descriptions. While it cannot be done for testing images as their attributes are unknown. Thus in testing, we simply abandon this energy term.

Initialization	
1	Learn HST-geo with parameter $\Theta^{(0)}$ based on the multi-scale segmentation.
2	Learn HST-att with parameter $\Phi^{(0)}$ and train appearance models.
Iteratively learn HST	
3	Jointly infer pt^+ for each training sample.
4	Update $\Theta^{(t+1)}$ in HST-geo.
5	Update $\Phi^{(t+1)}$ in HST-att and retrain appearance models.
6	Repeat 3 - 5 until convergence.

Figure 4.29: The learning algorithm. Reproduced/modified from Ref. [161] with permission of IEEE, © 2015.

Follow the learning-by-parsing framework introduced in Section 4.5.2.6, we can re-estimate the HST-geo and HST-att based on $\{pt_m^+\}_{m=1}^M$. We summarize the entire learning procedure in Figure 4.29, which

contains two steps. (i) Separately learn HST-geo parameters Θ and association matrix Φ based on the multi-scale segmentations as an initialization; and (ii) Iteratively update the HST-geo and the HST-att based on the joint inference.

5

Parsing Algorithms for Inference in And-Or Graphs

5.1 Classic Search and Parsing Algorithms

5.1.1 Heuristic Search in And-Graph, Or-Graph and And-Or-Graph

In this section, we begin by introducing a few conventional search algorithms in artificial intelligence. The researchers had studied these algorithms in the 1960s-80s and conventional search algorithms handle the hierarchic graph structures in a "top-down" manner with or without heuristic information. These techniques are very revealing. We should pay attention to their formulation, data structures, and ordering schemes, which are essential ingredients in designing more advanced algorithms in visual inference. Unfortunately, most of the vision literature has largely ignored these issues. Figure 5.1 gives a brief classification of the search algorithms.

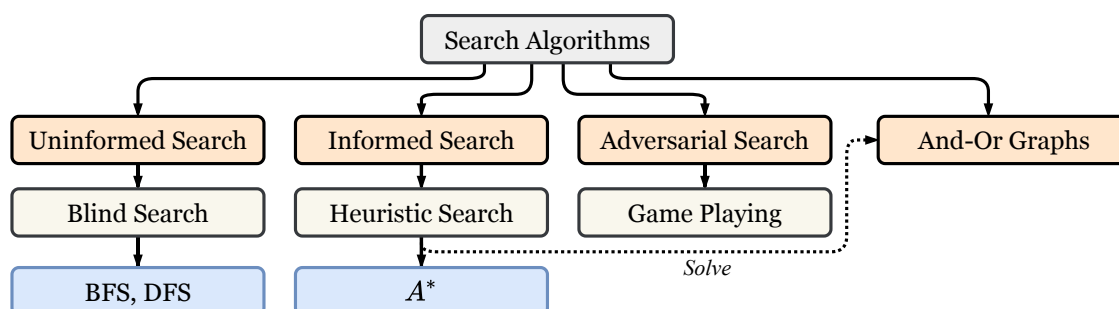


Figure 5.1: The four main types of search algorithms

Let's first look at a toy example.

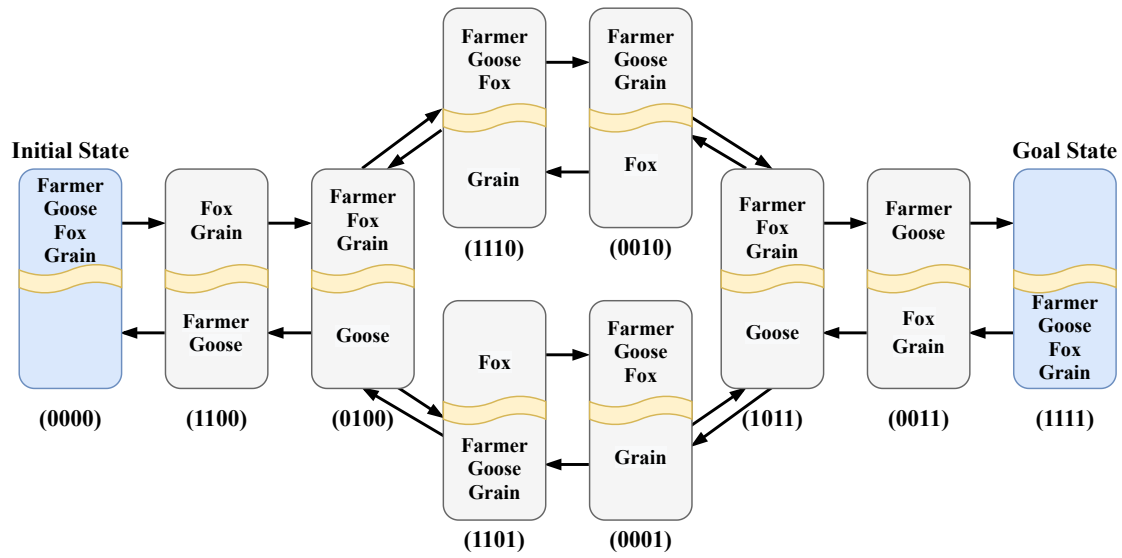


Figure 5.2: The state space of the river crossing puzzle and the transition among them. The yellow ribbon represents the river; farmer, goose, fox, and grain can be at either side of the river.

River Crossing Puzzle

A *farmer* wants to move himself, a silver *fox*, a fat *goose*, and some tasty *grain* across a river. Unfortunately, his *boat* is so tiny he can take only one of his possessions across on any trip. Worse yet, an unattended fox will eat a goose, and an unattended goose will eat Grain. How can he cross the river without losing his possessions?

To solve such a problem in a computer, we must formulate it properly. One solution is to enumerate all possible states and find the transitions among them. All possible states comprise the state space, and a state graph can be used to describe the transitions.

In Figure 5.2, each rectangle describes a valid state with each object at either side of the river. And each object in a state is assigned with a binary value to indicate whether it has crossed the river or not, *i.e.*, "0" represents the object is above the river and "1" for at opposite side, as shown under each rectangle in Figure 5.2. There should be $2^4 = 16$ states in the state graph, but 6 of them are invalid. The "boat" is a metaphor, it has limited capacity and thus the states are connected locally. The whole process can be seen as a task planning problem which will be introduced in next section. The final solution can be any path from initial state (0000) to the goal state (1111), as shown in Figure 5.2. Here, we provide the problem formulation by listing all key elements in solving the problem,

- **State Space:** there are constraints in the state space, *e.g.*, fox eats goose;
- **States:** configurations, objects;
- **Operators:** available actions are often limited, *e.g.*, the boat has limited capacity; the operator's actions also represent the transitions from current state to available neighboring nodes toward the goal state(s).

- **Initial State:** No parent state;
- **Goal State(s)** goal test;
- **Metrics** for multiple solutions, metrics measure their performance.

To design more effective algorithms, we need to design "big boats", which corresponds to large moves in the state space. In later formulation, the constraints on the state space will not be valid-vs-invalid, but probabilistic. Some states receive higher probabilities, and some receive low probabilities.

There are four criteria in designing a search algorithm,

- **Completeness:** is the algorithm guaranteed to find a solution if a solution exists?
- **Time Complexity:** this is often measured by the number of nodes visited by the algorithm before it reaches a goal node.
- **Space Complexity:** this is often measured by the maximum size of memory that the algorithm once used during the search.
- **Optimality:** is the algorithm guaranteed to find an optimal solution if there are many solutions? A solution is *optimal* in the sense of minimum cost.

We say an algorithm is **asymptotically optimal** in space (or time) if the amount of space (time) required by the algorithm is within some additive or multiplicative factor of the minimum amount of space (time) required for the task in a computer.

5.1.1.1 Search in a tree (Or-Tree)

Although the search is often performed in a graph in the state space, our study will focus on tree-structured graph for two reasons: 1) It is convenient and revealing to analyze algorithm performance on trees. We can do the analysis on trees, and then generalize it to graphs. 2) If a search algorithm does not visit nodes that were visited in previous steps, then its paths form a tree.

A tree has some outstanding properties which affect the efficiency of search algorithms:

- A tree is a connected graph with no loop: for a tree of n nodes, it always has $n - 1$ edges.
- In a tree there is only one root, each node other than root has one and only one parent, but they can have multiple children; the node that has no child is called terminal node/leaf, and non-leaf node is also called non-terminal/internal node.
- The **branching factor** b is the maximum number of successors of any node.
- The **depth**, d , of the shallowest goal node (*i.e.*, the number of steps along the path from the root).
- For a complete tree with depth d and branching factor b , there are b^d leaves and $1 + b + b^2 + \dots + b^{d-1} = \frac{b^d - 1}{b - 1} = O(b^{d-1})$ non-terminal nodes. Thus for $b > 1$, the leaves outnumber all the other nodes in the tree.

Uninformed Search

The minimum information for searching a graph is the goal test — a function that returns true when a node is a goal. If there is no other information available, the search algorithm is called **uninformed search**.

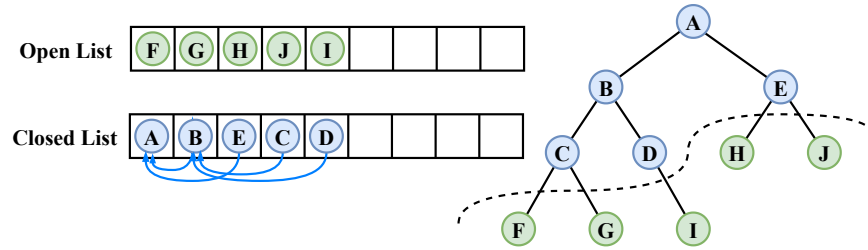


Figure 5.3: The data structure used in search algorithm: (1) *open list* — current visiting nodes; (2) *closed list* — contains untouched nodes

This mainly includes: (1) *Depth First Search* (DFS); (2) *Breadth First Search* (BFS); and (3) *Iterative Deepening Search*. There are other variants, such as, limited depth search, bi-directional search, which are not introduced in this textbook.

In general, the algorithms operate on two lists. An **open list** stores all the leaf nodes of the current search tree, as the green nodes in Figure 5.3. A **closed list** stored all the internal nodes that have been visited (expanded), those light blue nodes in Figure 5.3. In the closed list, each node has a pointer to its parent as the blue links of the list at the left hand side of Figure 5.3, thus the algorithm can trace the visiting path from a node to the root, at the final stage of extracting the solution. The nodes in the open list are sorted by some merits, *e.g.*, depth, breadth. So, these search algorithms are also categorized as **Best-First Search** since the merit applied is greedy to the minimum or maximum values. We summarize the general search algorithm with the following pseudo code:

Algorithm 5: General Search Algorithm

Input: *open list* $\leftarrow s_0$ (initial node) and *closed list* $\leftarrow \emptyset$

```

1 repeat
2   if open list is empty then
3     | exit with failure
4   end
5   take the first node s from the open list
6   if goal-test(s) = true then
7     | exit with success
8     | extract the path from s to s0
9   end
10  insert s in the closed list, s is said to be visited (or expanded)
11  insert the children of s in the open list in a certain order
12 until
```

We mention three remarks as below:

- *Remark I:* To avoid repeated visits, if a child of *s* appears in the closed/open lists, it will not be added.
- *Remark II:* A node is said to be "visited" after it enters the closed list, not the open list.
- *Remark III:* In uninformed search, the *space complexity* is the *maximum length* that the open list once reached. The *time complexity* is measured by the *length* of the closed list.

Both *Depth First Search* (DFS) and *Breath First Search* (BFS) follows the paradigm of Algorithm 5, their difference comes from the way they order nodes in the *open list*. DFS uses a **stack**, it always puts the

latest node on the top of the stack. While *BFS* uses a **queue**, it always puts the latest node at the rear of the queue.

5.1.1.2 Complexities of DFS and BFS

For ease of analysis, we assume the graph is a complete tree with finite depth d and branching factor b . Suppose the goal node is at depth d . For *DFS*, the *best case* time complexity is $O(d)$, which approves that the goal is at the leftmost branch. The *worst case* time complexity is $O(b^d)$ and the goal is at the rightmost branch. The space complexity of *DFS* is $O(db)$. For *BFS*, the *best* and *worst cases* time complexity are both $O(b^d)$. The space complexity of *BFS* is $O(b^d)$. But the *DFS* has a problem if the tree has a depth $D \gg d$. Even worse, if D can be infinite, then it may never find the goal even if d is small — not complete.

5.1.1.3 Iterative Deepening Search (IDS)

The *DFS* has advantage of less space/time complexity, but has a problem when $D \gg d$. As d is often unknown to us, we can adaptively search the tree with an incremental depth. This leads to *Iterative Deepening Search (IDS)* that combines the advantage of *DFS* and *BFS*. Algorithm 6 summarizes the *IDS* as pseudo code.

Algorithm 6: IDS Search Algorithm

Input: set $D_{max} \leftarrow 1$ and the goal node depth d is unknown

- 1 **repeat**
- 2 Do a *DFS* starting from the root for a fixed depth D_{max} .
- 3 Find a goal node, *i.e.*, $d \leq D_{max}$ then exit.
- 4 $D_{max} = D_{max} + \Delta$.
- 5 **until**

Now let us make an analysis of complexity of the *IDS*, the time complexity is $O(b^d)$ and is evaluated with below equation,

$$\frac{b^2 - 1}{b - 1} + \frac{b^3 - 1}{b - 1} + \cdots + \frac{b^{d+1} - 1}{b - 1} = \frac{b^{d+2} - b^2 - bd + d}{(b - 1)^2} \quad (5.1)$$

The space complexity of *IDS* is simply $O(db)$. We can draw conclusion that *IDS* is *asymptotically* optimal in both space and time. Apparently, it wastes a lot of time to repeatedly visit nodes at the shallow levels of the tree. But this does not matter because of the property 5 of tree.

Informed Search

In some applications, an agent may have information about its goals or tasks. By using such information, we expect it can improve performance in space and time complexity. Let us first consider some real-life examples. First, suppose that we process a graph of routing in a supermarket with a tree search strategy. At a moment, we are at node A , as in Figure 5.4, we are checking out from this supermarket. It has four lines B , C , D , and E . You can decide which line to stand based on heuristic information $h(X)$ of how long each line is. Similarly, when you are driving on a highway which has 4 lanes, which lane do you choose to drive at a given time period $t(X)$?

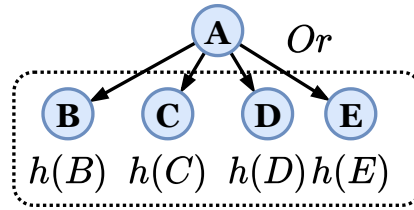


Figure 5.4: The *heuristic function* $h(\cdot)$ applied at each leaf node provides information to root node A , the *Or* operation is used to choose the satisfied one.

A poem quoted in Pearl [108]

"Heuristics, Patient rules of thumb,
So often scorned: Sloppy! Dumb!
Yet, Slowly, common sense come"

In the 1980s, the probabilistic models are not well known to the CS search literature, Pearl viewed heuristics as ways to

"inform the search algorithm with simplified models"

In our understanding today, the heuristics are discriminative methods which inform or drive the Markov chain search in DDMCMC. The Heuristics are "[computational knowledge](#)" in addition to the representational knowledge (models and dictionaries).

There are two new functions that an agent can explore, $g(s)$ and $h(s)$, respectively

- $g(s)$: this is a function that measures the "cost" incurred from the initial node so to the current node s . As it illustrates in Figure 5.5, the first half of the orange path starts from S_0 to S_3 , its cost is evaluated with function $g(s)$.
- $h(s)$: this is a function (or "budget") that estimates the forth-coming cost from s to a goal node s_g . This function is called a **heuristic function**. In Figure 5.5 the second half of the orange path from S_3 to goal node S_g is speculated with function $h(s)$.

In summary, for each node, there is an evaluation of the total cost for a path passing through this node then reaching the goal node by $f(s) = g(s) + h(s)$, as the orange path demonstrated in Figure 5.5, node S_3 is one of the intermediate nodes along the orange path. One more thing we need to mention is that it is not limited to have only one goal node.

Below gives the pseudo code for an *informed search*, sometime also call "*best*"-first-search (BFS) algo-

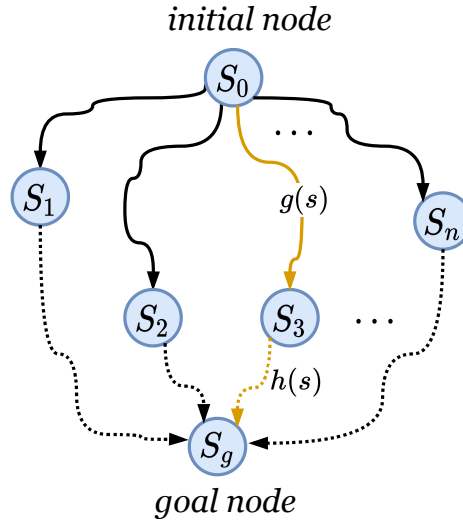


Figure 5.5: We evaluate the cost of a path from root through a intermediate node to the goal node by using heuristic function.

rithm,

Algorithm 7: Best-First-Search Algorithm

Input: *open list* $\leftarrow s_0$ and *closed list* $\leftarrow \emptyset$

```

1 repeat
2   if open list is empty then
3     | exit with failure
4   end
5   take the first node  $s$  from the open list
6   insert  $s$  into the closed list,  $s$  is said to be visited / expanded
7   if goal-test( $s$ ) = true then
8     | exit with success
9     | extract the path from  $s$  to  $s_0$  in the closed list
10  end
11  insert the children of  $s$  in the open list in an increasing order of a function  $x(s)$ 
12 until
```

Now, let us discuss how to choose the function $x(s)$ in Algorithm 7. The pseudo-code is called "*best-first*" search, because the algorithm always expands a node which it "think" is the "best" or most promising. There are different choices for the function $x(s)$:

- $x(s)$ can be $g(s)$. In case, each edge has equal cost, then $g(s) = d(s)$ is the depth. Then it reduces to a *breadth-first search*;
- $x(s)$ can be $h(s)$. This is called the *greedy search*. It is similar to the depth first search and may not be complete;
- $x(s)$ can be $f(s) = g(s) + h(s)$. This is called a *heuristic search*.

By using the function $f(s) = g(s) + h(s)$, a *heuristic search* algorithm can back-trace the most promising path. Two criteria can be taken into account to make the search effective: 1) in terms of a small search

tree, 2) in terms of an optimal solution. A concrete example in Figure 5.6 and 5.7 demonstrate the application of "best"-first search to solve a traditional 8-puzzle problem.

The 8-Puzzle

The objective of the 8-puzzle is to rearrange a given initial configuration of eight numbered tiles arranged on a 3×3 board into a given final configuration called the **goal state**. The rearrangement is allowed to proceed only by sliding one of the tiles onto the empty square from an orthogonally adjacent position.

As shown in Figure 5.6(a), before moving to the next step from the initial state S_0 , we need to evaluate which of the three alternatives A , B , or C appears most promising? Of course, the answer can be obtained by exhaustively searching subsequent moves in the puzzle to find out which of the three states leads to the shortest path to the goal. **Exhaustive search**, however, is utterly impractical when the number of states that must be examined is immense. This "combinatorial explosion" occurs in such puzzles when the path length from the initial state to goal state is large or when larger boards (e.g., 8×8) are involved. The decision to select the most promising alternative, therefore, cannot rely on exhaustive search, but rather it requires the use of **presearch judgement**.

One of the judgements that we might make in solving a path-finding problem is **estimating how close** a state, or configuration, is to the goal. In the 8-Puzzle there are two very common heuristics, or rules of thumb, which are used for estimating the proximity of one state to another. The first is the **number of mismatched tiles**, those by which the two states differ, which we will call heuristic function h_1 (used by Figure 5.6 and 5.7). The second is the **sum of the** (horizontal and vertical) **distances** of the mismatched tiles between the two states, which we will call heuristic function h_2 . This second heuristic function is also known as the **Manhattan or city-block distance**. To see how these heuristic functions work, below are the estimates for the proximities of the states A , B , and C from the goal state in Figure 5.6(a):

$$\begin{aligned} h_1(A) &= 5 & h_1(B) &= 3 & h_1(C) &= 5 \\ h_2(A) &= 6 & h_2(B) &= 4 & h_2(C) &= 7 \end{aligned} \tag{5.2}$$

Evidently, both heuristics proclaim that state B is the one closest to the goal and that it should, therefore, be explored before A and C .

The heuristic functions are intuitively appealing and readily computable, and may be used to prune the space of possibilities in such a way that only configurations lying close to the solution path will actually be explored. Search strategies that exhibit such pruning through the use of heuristic functions will be discussed later in this section.

Figure 5.6 and 5.7 show the 7 iterations for solving the 8-Puzzle with using heuristic function $h_1(s)$. The cost of each node is marked below the state node, the first number on the left hand side is the value of $g(s)$ and the second the value of heuristic $h_1(s)$.

When designing a heuristic function for the search algorithm, we also need to consider its admissibility. One question is raised: Can we design a heuristic function $h(s)$ so that the first found path is always the optimal (shortest or minimum cost) path?

Definition I: a heuristic function $h(s)$ is said to be **admissible** if it never over-estimates the true cost from s to a goal s_g — $C(s, s_g)$, i.e.,

$$h(s) \leq C(s, s_g) \tag{5.3}$$

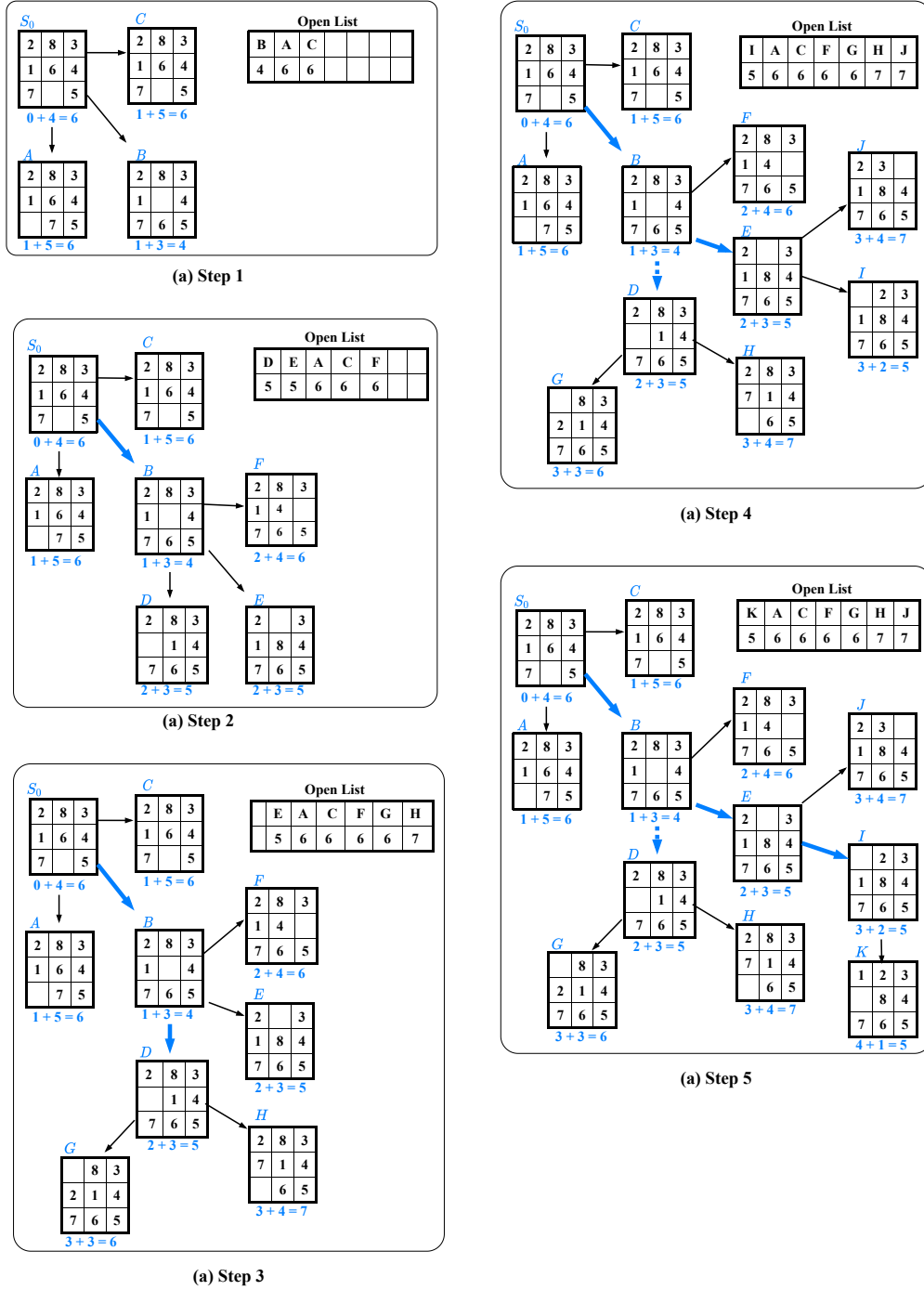


Figure 5.6: 8-Puzzle problem: step 1 to 5

Definition II: a heuristic search algorithm is called an *A**-algorithm if it uses an admissible heuristic function. *Definition III:* we say $h_2(s)$, which provides better heuristic, is more informed than $h_1(s)$ if

$$h_1(s) < h_2(s) \leq C(s, s_g) \quad (5.4)$$

With the above definitions, we give the following propositions conforming to them.

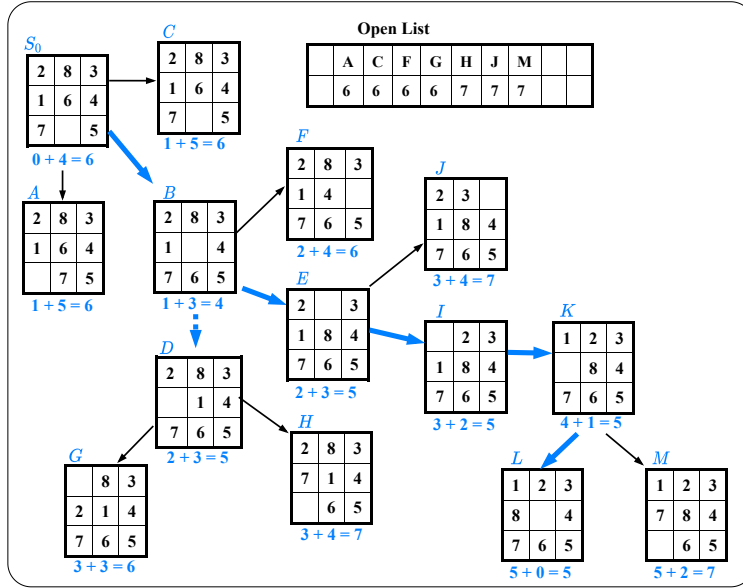
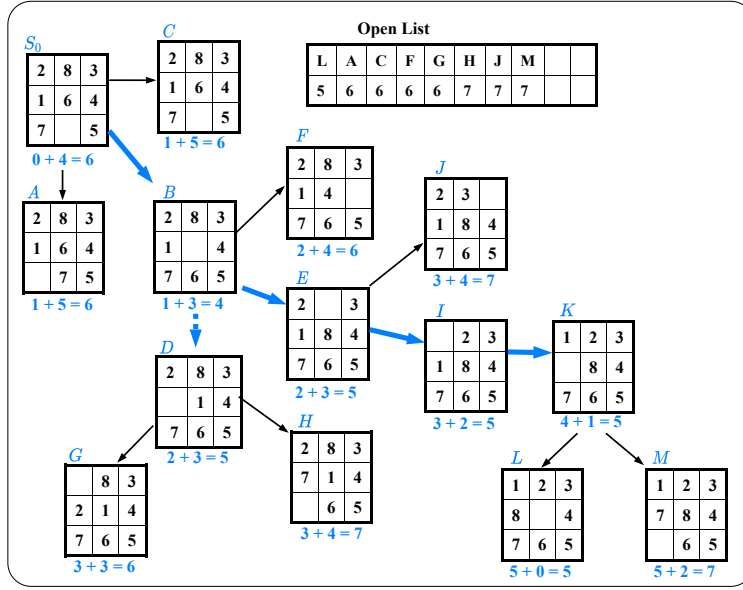


Figure 5.7: 8-Puzzle problem: step 6 and 7

Proposition I: An A^* -algorithm is always complete and optimal.

Proposition II: Let A_1^* and A_2^* be two algorithms using heuristic functions $h_1(s)$ and $h_2(s)$, respectively. Then the nodes searched by A_2^* is always a subset of those searched by A_1^* , if $h_2(s)$ is more informed than $h_1(s)$, i.e., $h_1(s) < h_2(s) \leq C(s, s_g)$. Figure 5.8 gives two A^* algorithms A_1^* and A_2^* with two heuristic functions h_1 and h_2 , respectively. A_1^* 's searching space is in light blue shaded area, and A_2^* 's searching space is in dark blue shaded area. It is obvious that A_2^* employs a better heuristic function than A_1^* , with which A_2^* makes less effort in fulfilling its goal.

We further prove the *optimality* of A^* -algorithm with below statements:

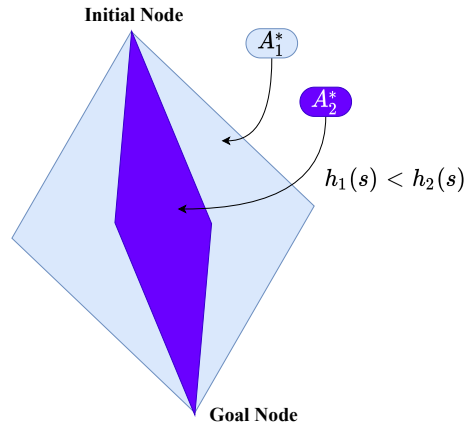


Figure 5.8: Two A^* -algorithms A_1^* and A_2^* with heuristic functions $h_1(s)$ and $h_2(s)$, respectively; it satisfies that $h_1(s) < h_2(s)$, then we conclude that h_2 is more admissible since it is informed better with smaller search space.

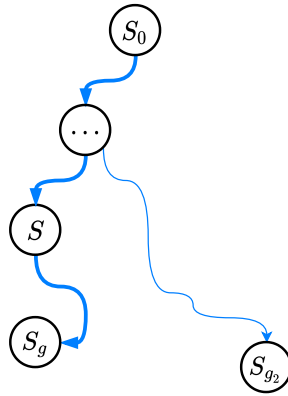


Figure 5.9: Illustration of the proof by refutation of A^* -algorithm

Proof by refutation

Let s_g be an optimal goal state with minimum cost $f^*(s_g)$. by refutation, suppose A^* algorithm find a path from an initial state to a goal s_{g_2} with cost $f(s_{g_2}) > f^*(s_g)$. It could be that $s_{g_2} = s_g$ and just the paths found by A^* has larger cost. Then there must be a node s on the optimal path which is not chosen by A^* . Since A^* has expanded before s is expanded. Thus it must be that $f(s) > f(s_{g_2}) > f^*(s_g)$. This implies $h(s) > h^*(s)$. It is contradictory to the assumption of A^* algorithm. Figure 5.9 demonstrates a scenario of such proof by refutation.

5.1.1.4 Iterative Deepening A^*

Obviously, the heuristic function reduces computational complexity. The extent to which the complexity decreases depends on how informed $h(s)$ is. For example, if $h(s) = 0$, the algorithm is clearly uninformed,

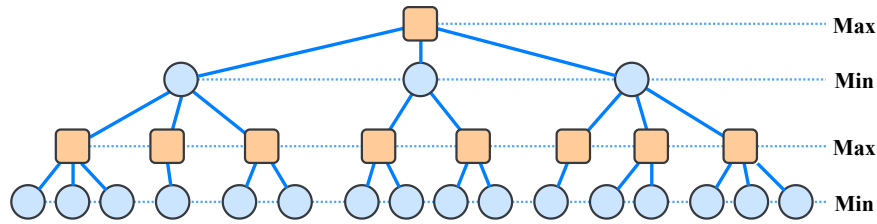


Figure 5.10: The Adversarial Search Tree

and the A^* algorithm degenerates to BFS . We learn that BFS is actually an A^* special case, and thus it is complete and optimal. But we know that BFS is notorious for space complexity, thus we doubt that A^* may suffer from the same problem. Our worry is often true in real situation. To reduce the space complexity, we again introduce an iterative deepening algorithm called **Iterative Deepening A^* (IDA^*)** this time. It employs a DFS algorithm with a bound for $f(s)$, and increases this bound incrementally.

Adversarial Search

In this section, we introduce a new search scenario: *game playing*. Games, like the puzzles discussed in previous sections, offer a perfect laboratory for studying complex problem-solving methodologies. The configuration of a game consists of:

- *Two adversarial players* — who alternate in making moves — each viewing the opponent's failure as his own success.
- *Zero-sum game* — it is (confusingly) defined as one where the total payoff to all players is the same for every instance of the game. And the game begins from a specified initial state and ends in a position that, using a simple criterion, can be declared with *win-lose*, *lose-win*, or *draw*. To be specified, take chess as example, chess is zero-sum because every game has payoff of either $0 + 1$, $1 + 0$, or $\frac{1}{2} + \frac{1}{2}$.
- *Game information* — most games played by computer programs, including chess, checkers, and GO, have the perfect accessibility to *game information* — unlike bridge in which the players' hands are hidden.
- *Deterministic rule* — no dice used, unlike *Back-Gammon* where the outcome of a die determines the available moves.

The problem is still formulated as a graph on a state space with each state being a possible configuration of the game. This graph of gaming is further explicitly represented in a tree form, which is called **game tree**. The root node is the initial position of the game, its successors are the positions that the first player can reach in one move, their successors are the positions resulting from the second player's move, the two players alternatively take turn in the successive moves. Terminal or leaf nodes are those representing *WIN*, *LOSS*, or *DRAW*. Each path from the root to a terminal node describes one case of complete play of the game, Figure 5.10 illustrates fictional scenarios.

We call the two players *MIN* and *MAX* respectively. Due to space/time complexity, an algorithm often can afford to search to a certain depth of the tree. Some time this search tree can also be called *Game Tree*. Then for a leaf s of the tree, we compute a *static evaluation function* $e(s)$ based on the configuration s , for example piece advantage, control of positions and lines *etc*. We call it static because it is based on a single node not by looking ahead of some steps. Then for non-terminal nodes, we compute the *backed-up values* propagated from the leaves.

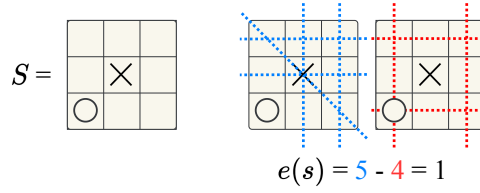


Figure 5.11: Heuristic function of a *Tic-Tac-Toe* game

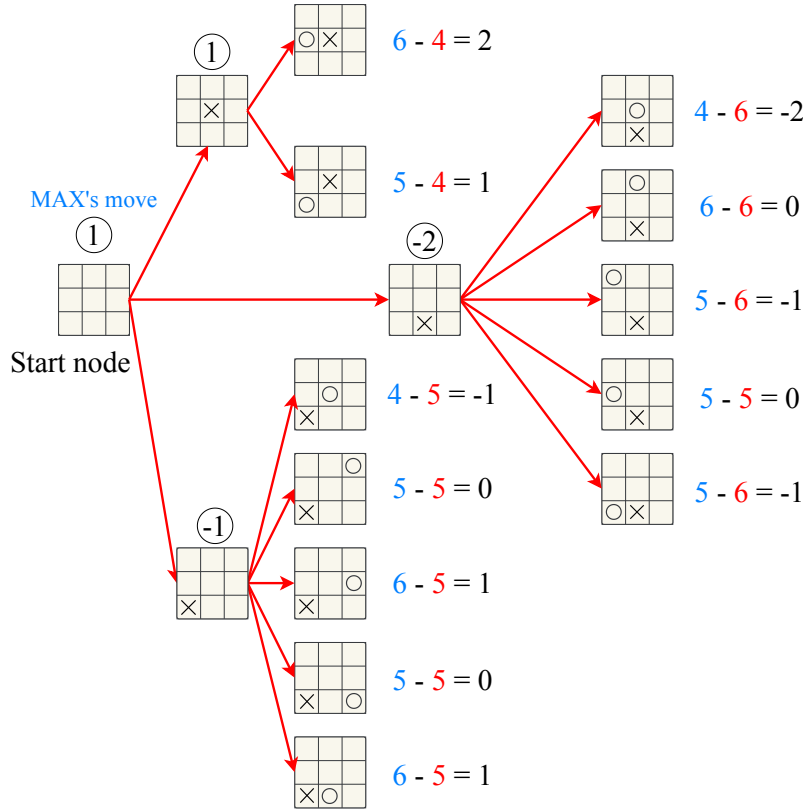


Figure 5.12: The *Game Tree* of puzzle *Tic-Tac-Toe*: the evaluation of first round

Take a *Tic-Tac-Toe* game as example, each player holding \bigcirc or \times selects the next move according the winning amount. At each move, players calculate the open lines as the heuristic function; then the player chooses the one he wins more or loses less. Figure 5.11 shows one state (or position) of a proceeding game with each player calculating his open lines (the line with available slot and non-blocking by opponent)

$$e(s) = \{\text{number of open lines of } MAX - \text{number of open lines of } MIN\} \quad (5.5)$$

Figure 5.12, 5.13, and 5.14 demonstrate the whole process of the game with its adversarial search tree at each round.

5.1.1.5 Alpha-Beta Search

The problem with *min-max* search is that the number of game states it has to examine is exponential in the depth of the tree. Unfortunately, we cannot eliminate the exponent, but it turns out we can effectively prune

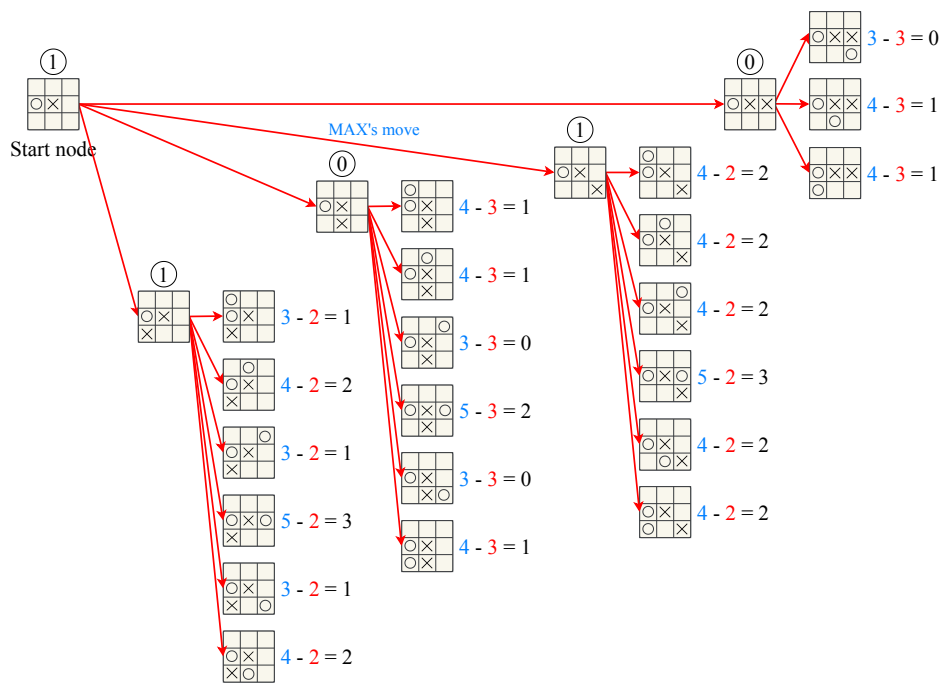


Figure 5.13: The *Game Tree* of puzzle *Tic-Tac-Toe*: the evaluation of second round

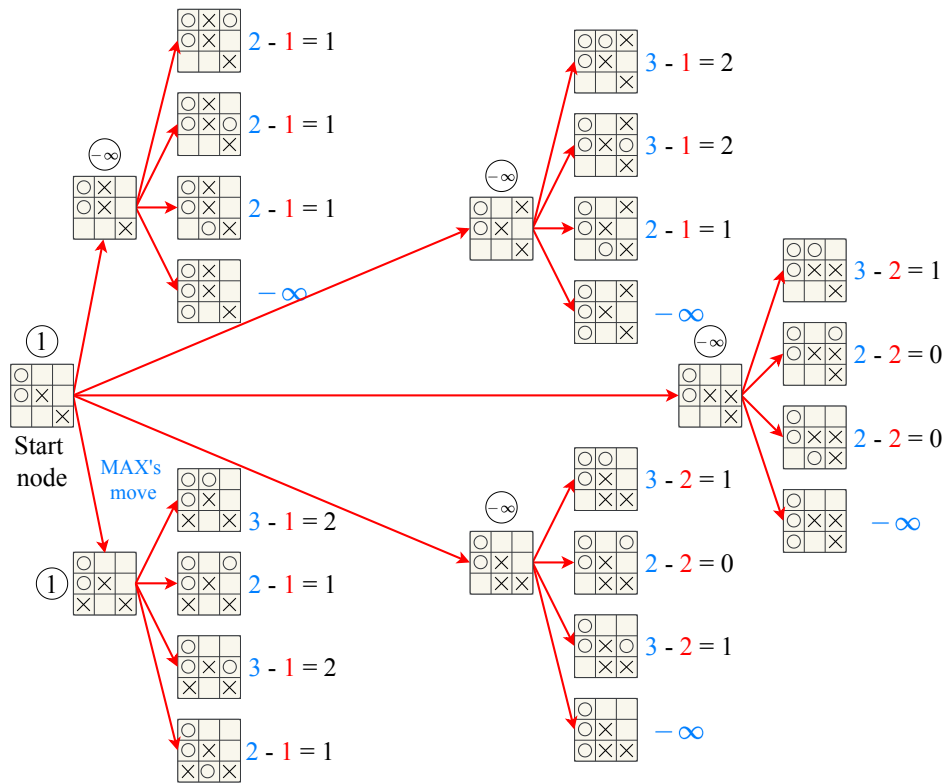


Figure 5.14: The *Game Tree* of puzzle *Tic-Tac-Toe*: the evaluation of third round

the search tree to some extent without changing the final search results.

Now for each non-terminal nodes, we define an α and a β values: A **MIN** node updates its β value as the minimum among all children who are evaluated. So the β value decreases monotonically as more children nodes are evaluated. Its α value is passed from its parent as a threshold. A **MAX** node updates its α value as the maximum among all children who are evaluated. So the α value increases monotonically as more children nodes are evaluated. Its β value is passed from its parent as a threshold.

To make it more clear, since the $e(s)$ of each state denotes the value from player **MAX**'s view, it's sure that **MAX** nodes tend to select the largest value among the coming states, *i.e.*, their child nodes. And regarding the zero-sum game property, **MIN** nodes consequently tend to select the smallest value among the coming states, *i.e.*, their child nodes, which will in turn maximize player **MIN**'s rewards.

We start the *alpha-beta* search with $AB(root, \alpha = -\infty, \beta = +\infty)$, then do a *depth-first* search which updates (α, β) along the way, we terminate a node whenever the following cut-off condition is satisfied.

$$\beta \leq \alpha; \quad (5.6)$$

A recursive function for the *alpha-beta* search with depth bound D_{max} updates the values of α and β as it goes along and prunes the remaining branches at a node as soon as the value of the current node is known to be worse than the current α or β value for MAX and MIN, respectively. The complete algorithm (Algorithm 8) is given below,

Algorithm 8: The *alpha-beta* search Algorithm

```

1 Function AB (s,  $\alpha$ ,  $\beta$ ) :
2   if depth(s) ==  $D_{max}$  then
3     | return e(s)                                /* leaf node */
4   end
5   for k = 1 to b(s) do
6     |  $s' \leftarrow k^{th}$  child node of s;
7     | if s is a MAX node then
8       | |  $\alpha \leftarrow \max(\alpha, AB(s', \alpha, \beta))$           /* update  $\alpha$  for max node */
9       | | if  $\beta \leq \alpha$  then
10      | | | return  $\alpha$                                        /*  $\beta$ -cut */
11      | | end
12      | | else
13      | | |  $\beta \leftarrow \min(\beta, AB(s', \alpha, \beta))$         /* update  $\beta$  for min node */
14      | | | if  $\beta \leq \alpha$  then
15      | | | | return  $\beta$                                        /*  $\alpha$ -cut */
16      | | | end
17      | | end
18   end
19   if s is a MAX node then                                /* all children expanded */
20
21   | return  $\alpha$ 
22   else
23   | return  $\beta$ 
24   end

```

Now let us examine again the *tic-tac-toe* puzzle with *alpha-beta* pruning algorithm on the *min-max* game tree, Figure 5.15 shows the whole process, the interested reader can try to construct this by himself,

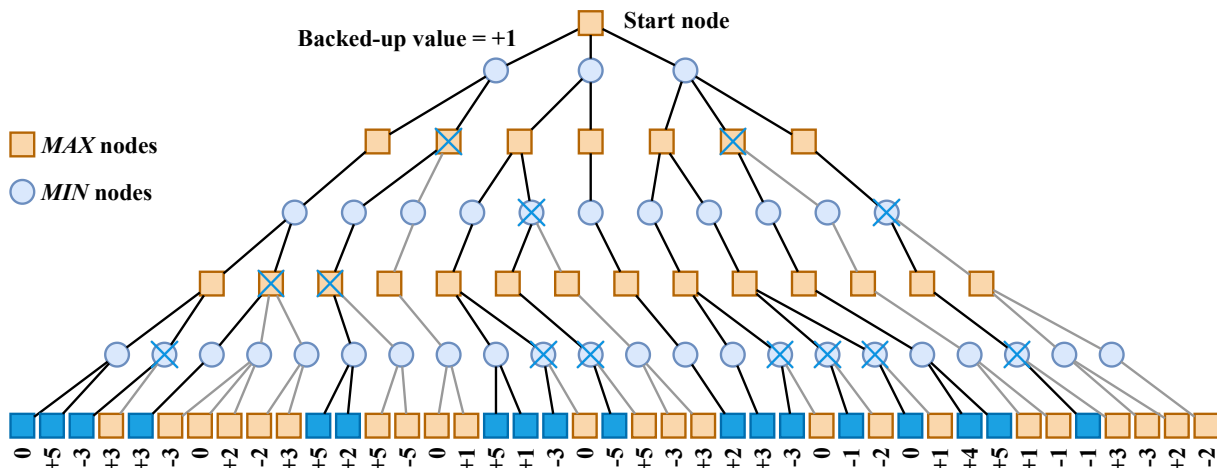


Figure 5.15: The *Minimax Adversarial Search Tree* of puzzle *Tic-Tac-Toe* solved with *alpha-beta* pruning algorithms

Search in an And-Or Graphs

This section introduces a very powerful data structure used to describe the composition of sub-problems and problem reduction representations.

The Counterfeit Coin Problem

We are given 12 coins, one of which is known to be heavier or lighter than the rest. Using a two-pan scale, we must find the counterfeit coin and determine whether it is light or heavy in no more than three tests. What is the best weighing strategy that guarantees that, regardless of the tests' outcomes, the counterfeit coin can be identified and characterized in at most three weighings?

If we are lucky and the balance tips one way or another, we are left with the easy problem of finding which of the two suspects is the counterfeit coin, knowing that the remaining ten coins are honest. This simple problem can, of course, be solved in a single step, weighing one of the suspects against one of the honest coins. The thing to notice, though, is that this two-suspect problem can be solved in isolation, independently of the problems that may ensue in case the balance remains neutral. Moreover, the same simple problem will appear frequently at the end of many elaborate strategies, and so, if we recognize it as an individual entity deserving an individual code, the solution to this subprogram can be found once, remembered, and shared among all strategies that lead to it.

This suggests that we treat subproblems as individual nodes in some graphs, even though none of these subproblems alone can constitute a complete solution to our original problem. In other words, since every action in our example introduces a triplet of subproblems, all of which must be solved, we prefer to represent the situation resulting from an action not as a single node containing a list of subproblems but rather as a triplet of individual nodes, each residing atop its own specialized set of candidate solution strategies. Figure 5.16 illustrates this problem-reduction representation.

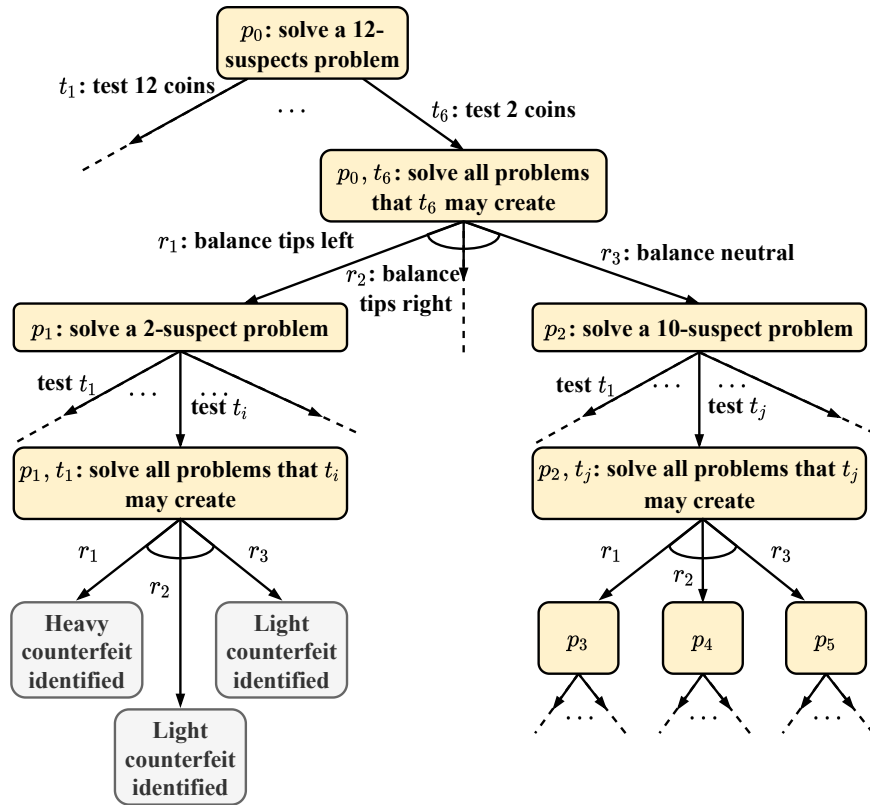


Figure 5.16: Problem-reduction representation for the Counterfeit Coin problem. Each node stands for one subproblem, and the curved arcs indicate which sets of subproblems must be solved conjunctively to make up a complete solution. Modified with permission, from (Pearl, 1984 [108])

Note that although the individual subproblems are assigned separate nodes, they are still bound by the requirement that all must be solved before the parent problem is considered solved. The purpose of the curved arcs in Figure 5.16 is to remind us of this fact. Thus the search-space graph in this representation consists of two types of nodes:

- Nodes that represent alternative ways for solving a sub-task from which they emanate.
- Nodes that are connected to a parent problem node which composes of the individual subproblems. This kind of nodes describes a decomposition of a task into a number of sub-tasks.

The former are identical to the nodes in state-space graphs and are called **OR nodes**, an *OR*-node is solvable if at least one of its children nodes is solvable. The latter are unique to problem-reduction and are called **AND nodes**, an *AND*-node is solvable only when all of its children nodes are solvable. An *AND/OR* graph may not be fully explicated at the beginning and is often expanded along the search process. A *solution graph* is a sub-graph of the *AND/OR* graph. It starts from the root node and makes choice at *OR*-nodes and all of its leaf nodes are solvable. A solution graph is the **parse graph** in our terminology. Further, a *solution base graph* is a partial solution graph containing all the open nodes to be explored, or partial parse graphs.

In Figure 5.16, *OR*-nodes represent the strategist's choices of tests, *AND*-nodes represent Nature's choice of test outcomes. In cases where a node gives rise to a mixture of *OR*-nodes and *AND*-nodes, we can still

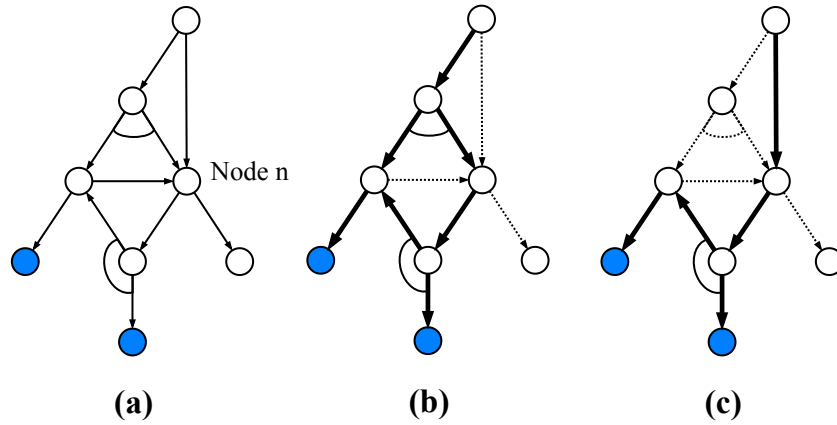


Figure 5.17: An AND/OR graph (a) and two of its solution graphs (b) and (c). Terminal nodes are marked as dark blue dots. Modified with permission, from (Pearl, 1984 [108])

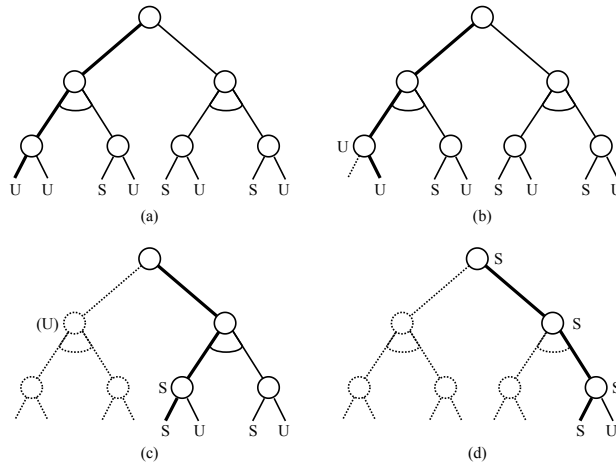


Figure 5.18: Typical steps in the execution of backtracking search of an *And/Or* tree. The heavy line represents the traversal path, whereas the broken lines represent portions of the tree that can be purged from memory. Modified with permission, from (Pearl, 1984 [108])

maintain the purity of node types by imagining that every bundle of *AND* nodes emanates from a separate (dummy) *AND* node and that these dummy *AND*-nodes are *OR* linked to the parent node.

In general, an *AND*-node and an *OR*-node may simultaneously link to the same node in an *AND/OR* graph, such as the node marked (*n*) in Figure 5.17. In this figure, however, each node issues either *OR*-nodes or *AND*-nodes, but not both.

During the search, we need to maintain the open lists at two levels:

1. a list of the *solution bases*,
2. lists of *open nodes* for each solution bases.

The score $f(s) = g(s) + h(s)$ for each node s is updated. A node s may be reachable from the root from multiple paths. Therefore, its score will have to be updated, and [rollback the updates](#) to its ancestor nodes.

Figure 5.18 illustrates the steps taken by backtracking in a left-to-right traversal of a binary *AND/OR* tree of depth 3. The symbols *S* and *U* stand for the "solve" and "unsolvable" labels that propagate up the traversal

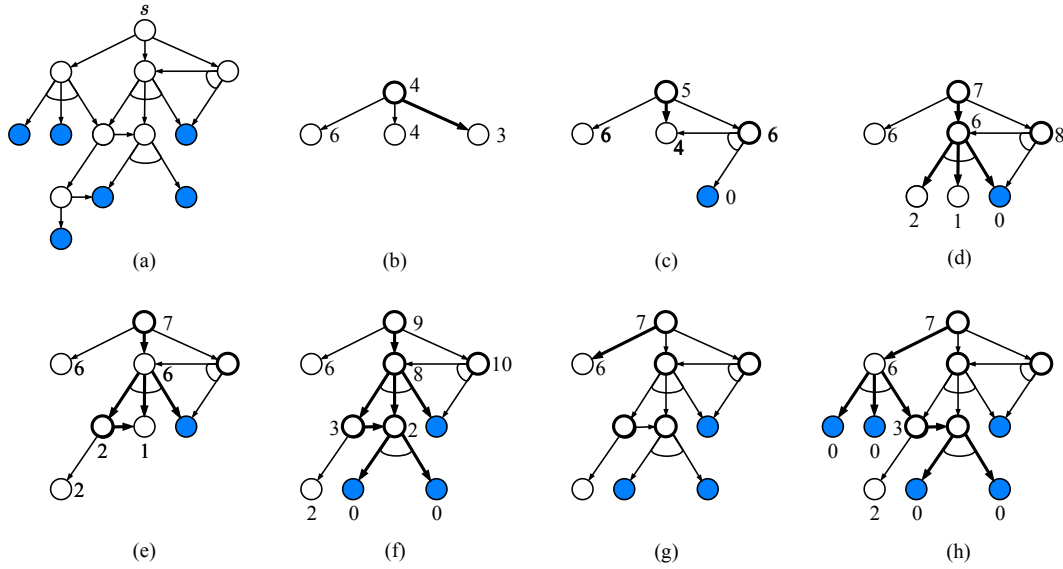


Figure 5.19: Successive steps in the execution of *general-best-first* (GBF) search on the implicit *And/Or* graph of part (a). Solid circles represent solved nodes, heavy hollow circles nodes in CLOSED, and thin circles nodes in OPEN. The heavy lines stand, at each stage, for the current most promising solution base. Modified with permission, from (Pearl, 1984 [108])

path. Note that labelling any intermediate node permits us to remove that node from storage and shift the traversal path one step to the right. This removal, however, although not affecting the correct labelling of the start node, prevents us from reconstructing the actual solution graph once s is labeled as solved. Therefore, if in addition to testing whether a problem can be solved we also wish to exhibit the final solution tree, additional bookkeeping is needed. Prior to its removal, each "solved" node should transmit to its father a code for the solution graph residing below it. These codes are then combined by the various *AND*-nodes and passed upward until the start node, if labelled "solved" receives a code for the entire solution graph beneath it. For example, in Figure 5.18(c) the intermediate node labelled S will transmit to its father the code $(1, 1)$, indicating that a solution can be obtained by going down two levels following leftmost successors. At stage (d) the *AND*-node assembles the tree code $(1, 1) (2, 1)$, and finally the start node receives the solution tree $2, ((1, 1)(2, 1))$.

The *general best-first search* (GBF), namely the commitment to explore the best among all available candidates, can be applied in searching *AND/OR* graphs. Let us look an example of applying *GBF* on *AND/OR* graphs. Assume that we wish to find a solution graph in the (implicit) *AND/OR* graph G of Figure 5.19(a). Expanding s in Figure 5.19(b), we unravel three solution bases (*i.e.*, three disjoint subsets of potential solution graphs represented by the three arcs emanating from s). To apply the best-first principle we need to use some criterion (f_1) for assessing the promise offered by these solution bases. Let us assume that we decide to pursue our search within whichever subset appears to contain the solution graph with the smallest number of edges. Although the problem itself is not posed as an optimization task (any solution graph will do), we can reasonably argue that the pursuit of the smallest graph automatically guides us toward an early termination.

Now, to rank our three candidates by this criterion, we need to obtain some information regarding the class of solution graphs stemming from each of the three given arcs. If the entire G graph were accessible, we could have ranked out candidates perfectly; all we would need to know is the size of the smallest solution

graph beneath each one of the three nodes currently on OPEN list. Not having access to G itself, we must try to estimate the needed information from the descriptions of the nodes themselves, and from our general knowledge of the problem domain.

Assuming that such estimates were computed and gave the values 6, 4, and 3 to the three nodes on OPEN list, we now make an additional assumption that these estimates be taken at face value for the purpose of computing the merit of each candidate solution base. Recalling that perfect estimates, if available, should be increased by one to give the required size assessment of our candidates, we apply the identical procedure to the imperfect estimates and obtain the values 7, 5, and 4. The best candidate appears to be the rightmost, with an overall estimate of 4, and this estimate can now be stored at s for later use.

This one simple step applied to the three nodes in Figure 5.19(b) involves two fundamental principles in heuristic programming. The choice of an auxiliary minimization criterion as a means to hasten the search invokes what we call the **small-is-quick** principle. That choice determines the kind of heuristic estimates h ; h should contain estimates of a set of parameters sufficient for the computation of the minimization criterion chosen. Substituting these imperfect estimates for the actual values of the parameters constitutes the second principle and would be referred to as the **face-value** principle.

In Figure 5.19(b), the rightmost candidate solution base is declared most promising, and its (only) OPEN node is expanded. This allows re-evaluation of the candidate using the estimates assigned to its newly generated nodes. In our example (Figure 5.19(c)), the rightmost candidate was reevaluated to 7, rendering it inferior to the middle candidate, valued at 5. Therefore that latter candidate is declared most promising and draws future attention.

The graphs in Figure 5.19 illustrates successive developments of the search graph G' (in thin lines) and the shifting of its most promising solution base G_0 (in heavy lines). Figure 5.19(d) illustrates a situation where the node selection function is invoked. The reason for choosing the leftmost tip node of G_0 for expansion stems from the following argument: If the current G_0 is in fact a base of a legitimate solution, all its tip nodes must eventually be expanded, and the order of expansion is immaterial. If it is not, however, the most sensible node to expand would be the one that reveals the error the earliest.

We repeat applying the above analysis and the two principles of general *best-first* search to successive steps in Figure 5.19(e) - (h).

5.1.2 Bottom-Up Chart Parsing

In this section, we give an introduction to the bottom-up chart parsing methods. A chart parser is a type of parser which adopts dynamic programming to generate partial hypothesized results. This type of methods got its name from the data structure (chart) used for storing intermediate results along the parsing process. With its practical value and nice properties, a lot of well-known methods share insights on the chart-parsing idea, including the shift/reduce algorithm, Cocke-Younger-Kasami (CYK) algorithm [21, 74, 178] and also the Earley algorithm [30]. In the following section, we provide illustrations on both the motivations and details of bottom-up chart parsing methods.

5.1.2.1 Parsing Inefficiency

The main motivation of bottom-up chart parsing originates from the inefficiency of backtracking (search-based) parsers. With no intermediate results stored, a backtracking parser may perform many redundant actions, for example (i) rediscovering phrases it has already successfully found, and (ii) re-exploring hypotheses that have already failed. In a simple example from [97], with a grammar containing the following

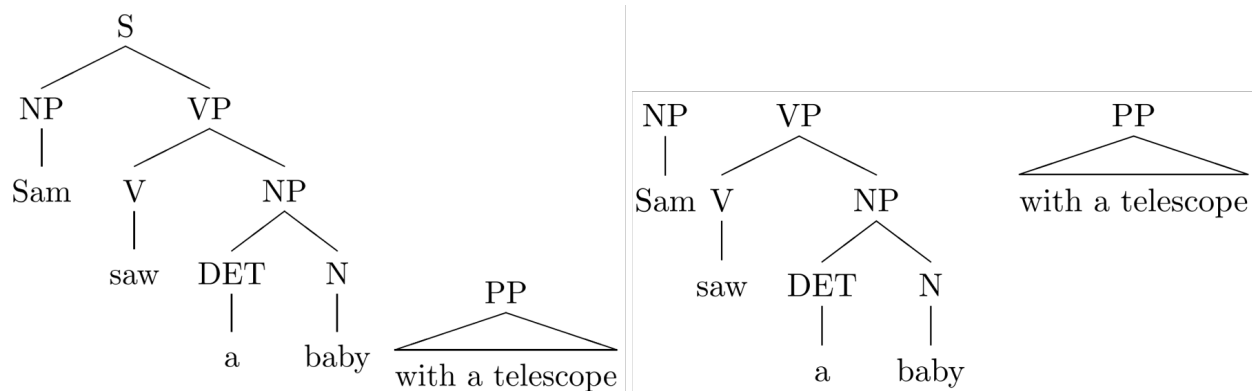


Figure 5.20: Tree inadequacy: on the left subfigure, when the prepositional phrase may attach to both NP and VP, we need two separate trees as we can not summarize the common parts of previously parsed constituents in parse trees; on the right subfigure, we can not use parse trees to represent incomplete structures, especially in a bottom-up fashion, since there is no root.

the two verb phrase expansions:

$$VP \rightarrow V \text{ NP PP}$$

$$VP \rightarrow V \text{ NP NP}$$

and a sentence that we are trying to parse:

"I sent the very pleasant double-glazing salesman that I met on holiday in Marbella last year a postcard."

The first rule will be attempted in the search and the long object noun phrase will be parsed. Then when looking for a prepositional phrase, it will naturally fail and backtrack to try the second rule. In this case, the parsing constituents before the final noun phrase will be repeated and finally *"a postcard"* will be parsed as the final noun phrase. This undesired behaviour of redundant parsing operations is a natural outcome of searching without memorization. A backtrack search parsing method may, in the worst case, need exponential number of steps (with respect to the input) to try all alternatives of the given context-free grammar (CFG) in order to produce all possible parse trees. In this case, the space-time complexity will both be exponential with respect to the input string length. As a grammar parsing problem could be easily decomposed into sub-problems, an intuitive solution to this issue is to adopt a dynamic-programming-based approach, store intermediate results and avoid redundant parsing operations.

Another problem of using search-based methods is over the data structures used. In fact, parse trees do not provide a way of representing local ambiguity. A new constituent may attach to many possible predecessor nodes, and representing each possibility requires a separate tree. However, there exists common information among these possible parse trees when solving local ambiguities. Moreover, parse trees do not provide a way of representing incomplete structures, this might cause problems when the input cannot be completely parsed. A visualization of these two problems could be seen in Fig.5.20. These deficiencies urge us to have a better data structure to provide (i) a compact representation for local ambiguity and (ii) a representation for partial parses. The chart data structure is therefore designed to provide a solution to these issues and we leave the detailed descriptions to the following sections.

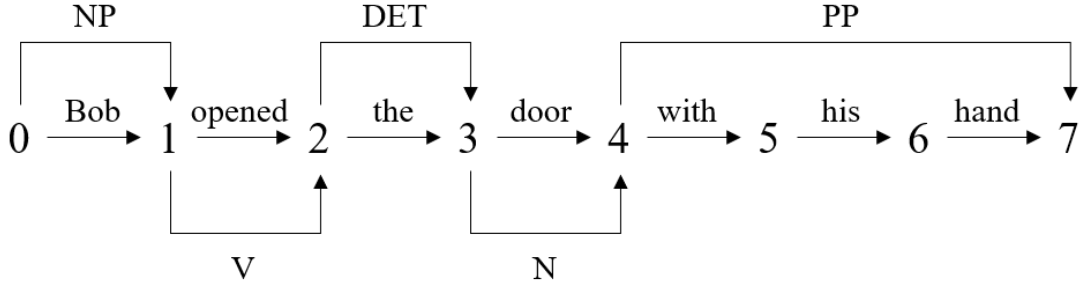


Figure 5.21: An example chart for the sentence "Bob opened the door with his hand.". Vertices are connected by edges containing grammatical information. Each edge covers a span of words from the starting vertex to the ending vertex indicating what words are covered/parsed.

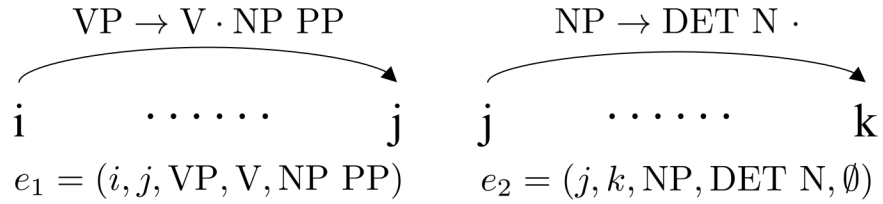


Figure 5.22: Active edge and passive edge: the left subfigure shows an example active edge where the dot is not at the end of the production rule indicating remaining constituents to be parsed; the right subfigure shows an passive edge.

5.1.2.2 Charts

A chart is an abstract data structure used to store intermediate results during parsing. It consists of a collection of vertices, one between each word of the input, connected by edges, labelled with grammatical information. The main goal of charts is to store partial results of the parsing process such that they can be reused.

Formally, charts are represented by directed graphs $\mathcal{G} = \langle V, E \rangle$. For an input sentence with n words, $V = \{0, 1, \dots, n\}$, and the i -th word is marked by two nodes in V , *i.e.*, node $i - 1$ and node i . An example is shown in Fig.5.21.

Each edge $e \in E$ characterizes a completed or partial constituent spanning a group of words. Along with the parsing process, edges we generate to connect vertices correspond to the derivation of production rules, and are therefore equipped with parsing information. An edge is represented by a 5-tuple

$$e = (start, finish, category, found, remainder),$$

where *category* is a non-terminal node in the grammar, indicating the left hand side of a production rule; *found* is parsed parts of the right hand side of *category* which explains word from span $(start, finish)$; *remainder* is the remainder of the right hand side of non-terminal *category* left to be parsed. We further define **active edges** as edges that have non-empty *remainder* set and **passive edges** as edges that have empty *remainder* set. An example is shown in Fig.5.22. The dotted rules shown in Fig.5.22 are used to represent stages in the parsing process, where constituents before the dot is parsed and the others are remained to be parsed. In this sense, active edges corresponds to incomplete parses.

Returning to the tree inadequacies we discussed previously, using a chart representation will easily solve the local ambiguity and partial parsing problem. For the example we showed earlier, we can use a char to

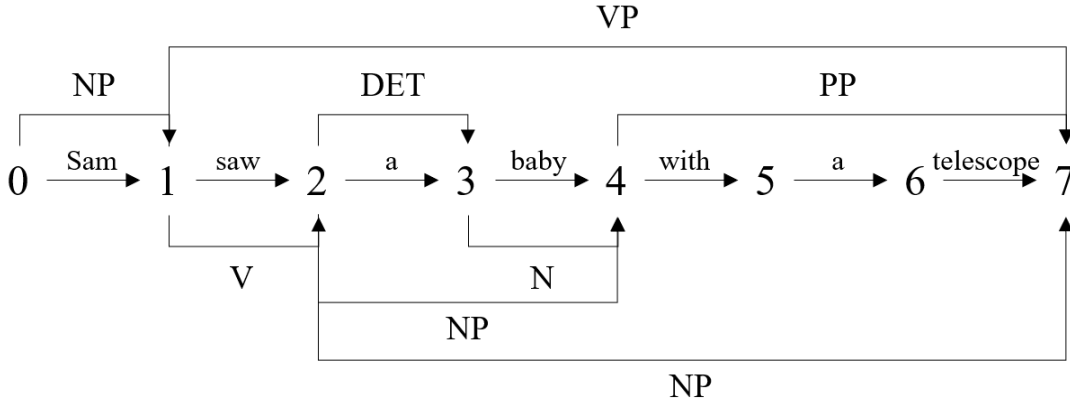


Figure 5.23: The equivalent chart containing multiple possible parses for subsets of words.

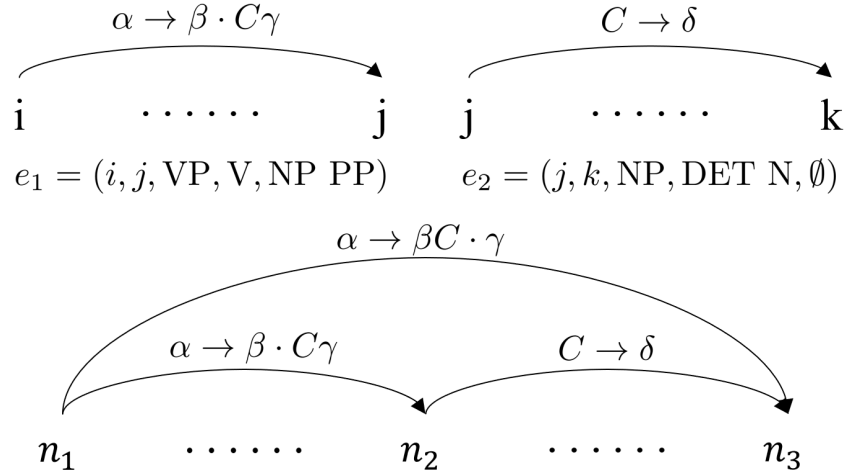


Figure 5.24: Illustration of the fundamental rule: With active edge on the top-left and passive edge on the top-right, we can combine these edges to a new edge shown at the bottom that aggregates the partial parses.

solve the problem as shown in Fig.5.23. As each edge in the chart is only associated with the span of words and grammatical information, we can use edges to summarize common information when facing local ambiguity. Partial parses are also inherently solvable since in chart, we do not necessarily restrict edges to form a full parse tree.

With the definition of active and passive edges, we can combine partial parses into a complete parse in the chart notations. This is done by combining active edges with passive edges to make new edges (either passive or active) following **the fundamental rule**. More specifically, suppose the active edge goes from node n_1 to node n_2 is $e_1 = (n_1, n_2, \alpha, \beta, C \gamma)$ and the passive edge goes from node n_2 to n_3 is $e_2 = (n_2, n_3, C, \delta, \emptyset)$, then we can combine this two edges and create a new edge $e_3 = (n_1, n_3, \alpha, \beta C, \gamma)$ in the chart as shown in Fig.5.24.

Before going into the details of chart parsing algorithms, we give the definition of **agenda**. An agenda is a data structure that keeps track of the things that we still have to do. More intuitively, it can be treated as a list of edges to be added to the chart and create new edges according to the fundamental rule. Newly created edges are then put into the agenda such that we can look at them to see whether they can be combined with

$$\begin{aligned}
S &\rightarrow NP VP \\
NP &\rightarrow DET N \\
VP &\rightarrow V NP \\
DET &\rightarrow \text{"a"} \mid \text{"the"} \\
N &\rightarrow \text{"boy"} \mid \text{"bicycle"} \\
V &\rightarrow \text{"rode"}
\end{aligned}$$

Figure 5.25: A simple grammar example.

other edges in any way. The main reason for having this concept is to decide in what order edges should be added to the chart, which further affects parsing efficiency. Two common agendas are:

- Stack agenda: every time an edge is added, it is placed on the front of the agenda, corresponding to depth-first-search.
- Queue agenda: every time an edge is added, it is placed on the end of the agenda, corresponding to breadth-first-search.

Other common choices include using a priority queue agenda taking the probability of edges as priority values.

5.1.2.3 Bottom-up Chart Parsing Algorithms

With the definition of charts, edges, and agenda, the goal of chart parsing is to produce a complete edge that spans from vertex 0 to n and is of *category* S where S is the start symbol of the given CFG. The general idea of chart parsing follows from edge combination we mentioned earlier, as we can enumerate all possible edges and try to combine them to see if the final edge could be obtained. Depending on whether an agenda is used, chart parsing algorithms can be categorized into **passive** and **active** chart parsing algorithms. To better illustrate the algorithms, we use a simple CFG shown in Fig.5.25 as an example. We use the sentence "*The boy rode a bicycle*" together with this CFG throughout this section to illustrate the behaviors of different parsing algorithms.

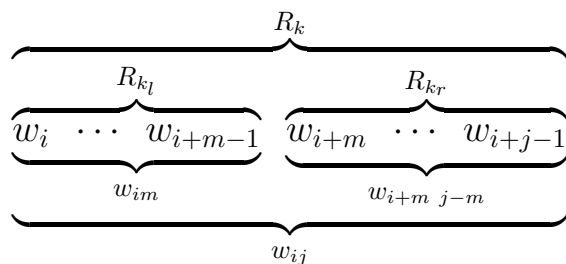
Bottom-Up Passive Chart Parsing We use the Cocke-Younger-Kasami (CYK) algorithm as an example to illustrate the basic steps of a bottom-up passive chart parser. From a bottom-up perspective, the CYK algorithm concentrates on substrings of the input sentence, shortest substrings first and works its way up. Following the notations defined previously, the CYK algorithm starts from the unit substrings (of length 1) and attempt to match them with passive edges. Then, it tries to see if passive edges can compose new passive edges for substrings of length from 2 to n (the length of the string).

More formally, let us suppose that a sentence $X = w_1 \cdots w_n$ and the grammar \mathcal{G} with S being the start symbol. Let $w_{ij} = w_i w_{i+1} \cdots w_{i+j-1}$ denote the substring of X of length j starting with w_i , then $X = w_{1n}$. Let $R \in \mathcal{G}$ denote the production rules in the given CFG. For storing the parsing information of each substring w_{ij} , we create a CYK table V with $V[i, j]$, $1 \leq i \leq n$, $1 \leq j \leq n - i + 1$ to store the rules (passive edges) which can explain substring w_{ij} . In a probabilistic parsing setting, we can use a set of 4-tuples

$$v_k = (k, \text{probability}, \text{pointerLeft}, \text{pointerRight})$$

to store all partial information we need in the CYK table. Here k corresponds to a production rule R_k in the grammar, *probability* is the probability of parsing the current substring with rule R_k , *pointerLeft* and *pointerRight* are two pointers pointing to table entries of the decomposed substrings of the current string.

Then when we gradually increase the length of the substring in a dynamic programming manner, we should have sub-problems merging into a full problem. A substring w_{ij} of length j could be decomposed to two substrings w_{im} with length m and $w_{i+m, j-m}$ with length $j - m$.



As we should have already had the parse (R_{k_l} and R_{k_r}) for these shorter length substrings, we can have the iterating function:

$$V[i, j] = \{v_k = (k, P(R_k|w_{ij}), v_{k_l}, v_{k_r}) \mid R_k \rightarrow R_{k_l} R_{k_r}, R_{k_l} \Rightarrow w_{im}, R_{k_r} \Rightarrow w_{i+m, j-m}, R_{k_l}, R_{k_r} \in G\}$$

The full algorithm is therefore as shown in Alg.9. The algorithm has three nested loops each of which has the range at most 1 to n . With each loop, the algorithm checks all production rules in the grammar \mathcal{G} , therefore the worst case of running time is $O(n^3|G|)$. We also show an example here for the ride bicycle example we mentioned earlier and the corresponding chart in Fig.5.26. In fact, each table entry $V[i, j]$ is storing all valid passive edges for parsing substring w_{ij} with v_k having the same semantics to the edges e we discussed. We used a slightly different notation for passive edges as in this case we don't need to distinguish between active and passive edges, however, to see the connection, v_k can be seen as a passive edge $e_k = (i, i + j, R_k, R_{k_l} R_{k_r}, \emptyset)$ with auxiliary information pointing to the related table entries of decomposed substrings

Algorithm 9: CYK algorithm

Input : sentence $X = w_1 \cdots w_n$, grammar \mathcal{G} with production rules $R \in \mathcal{G}$ and start symbol S .

Output: verify whether $S \Rightarrow X$

```

1 Construct a  $n \times n$  CYK table  $V$ 
2 for  $i = 1 \rightarrow n$  do
3    $V[i, 1] = \{v_k = (k, P(R_k|w_{i1}), \text{NULL}, \text{NULL}) \mid R_k \Rightarrow w_{i1}, R_k \in \mathcal{G}\}$ 
4 for  $j = 2 \rightarrow n$  do
5   for  $i = 1 \rightarrow n - j + 1$  do
6     for  $m = 1 \rightarrow j - 1$  do
7        $V[i, j] = \{v_k = (k, P(R_k|w_{ij}), v_{k_l}, v_{k_r}) \mid R_k \rightarrow R_{k_l} R_{k_r}, R_{k_l} \Rightarrow w_{im}, R_{k_r} \Rightarrow w_{i+m, j-m}, R_{k_l}, R_{k_r} \in G\}$ 
8 return  $V[1, n]$ 
```

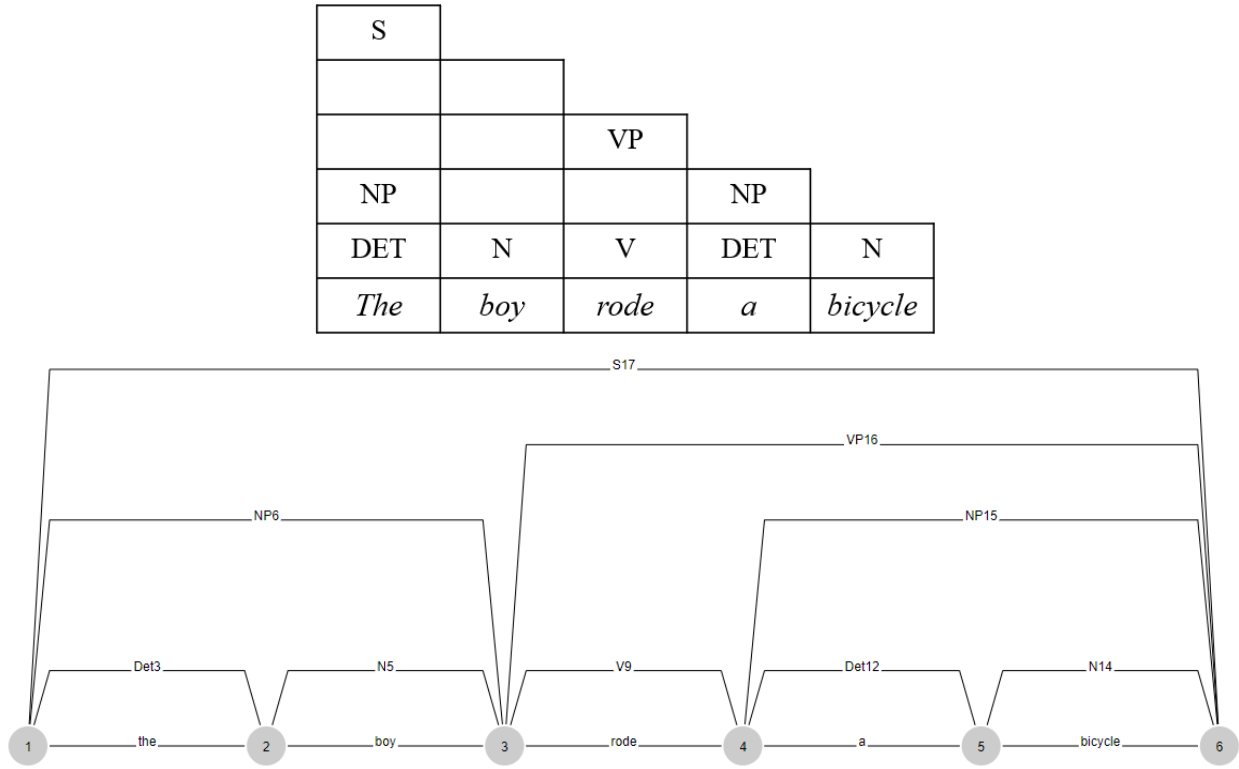


Figure 5.26: CYK parsing example: CYK table entries generated after parsing (top), the edges added to the chart (bottom).

and parsing probabilities. One thing worth mentioning, as we can see in Fig.5.26, is that the CYK table is always triangular. We can easily adopt this nice property and optimize the memory efficiency in Alg.9, this is left as a design choice and will not be discussed in detail.

Bottom-up Active Chart Parsing One minor drawback of the passive chart-parsing algorithm is that we can not use a strategy to make the parsing process more efficient. This reflects agendas we discussed in the previous sections. In fact, if we consider active edges, instead of enumerating all passive combinations, we can follow a certain scheme to combine active/passive edges into new edges. This allows better parsing efficiency with prior information on the statistics of the grammar being parsed.

To start with, we need to clarify how to initialize agendas. With the start symbol S , we can add inactive edges by finding every rule that has the category of the edge as its left corner. For example, using the sample grammar we provided in Fig.5.25, we can have edge $e = (0, 1, \text{DET}, \emptyset, \emptyset)$. Given this edge e , we add edges

$$\begin{aligned}
 e_1 &= (0, 0, \text{NP}, \emptyset, \text{DET N}) & \text{NP} &\rightarrow \cdot \text{DET N} \\
 e_2 &= (0, 0, S, \emptyset, \text{NP VP}) & S &\rightarrow \cdot \text{NP VP}
 \end{aligned}$$

into the agenda as e_1 has DET as first symbol on its RHS and e_2 has NP as its first symbol. Such a process is also known as bottom-up predict. More specifically, we show the details of the steps below:

Algorithm 10: Bottom-Up Active Chart Parsing Algorithm

Input : sentence $X = w_1 \cdots w_n$, grammar \mathcal{G} with production rules $R \in \mathcal{G}$ and start symbol S .

Output: verify whether $S \Rightarrow X$

```
1 Initialize chart  $C = \emptyset$  and agenda
    $A = \{\text{all passive edges valid from words}\} \cup \{\text{active edges predicted}\}$ 
2 while  $A \neq \emptyset$  do
3   select edge  $e = (\text{start}, \text{finish}, \text{category}, \text{found}, \text{remainder})$  from agenda
4   add  $e$  to  $C$  in position  $(\text{start}, \text{finish})$  if  $e \notin C$ 
5   if  $e$  active then
6     for  $e' \in C, \text{finish}_{e'} = \text{start}$  do
7       Combine  $e$  and  $e'$  into new edge  $\bar{e}$ 
8       if  $\bar{e} \neq \text{NULL}$  then                                     // can be combined
9         add  $\bar{e}$  and  $\bar{e}' \in \{\bar{e}' \mid \text{active edges predicted from } \bar{e}\}$  to A
10 Succeed if there is a passive edge  $e = (0, n, S, \text{found}, \emptyset) \in C$ 
```

Bottom-Up Predict

- For each passive edge e with category B in the agenda
 - find every rule $r : \alpha \rightarrow B\gamma$ that has B as its first RHS symbol.
 - add an empty active edge $e = (0, 0, \alpha, \emptyset, B\gamma)$ to the agenda
- Repeat this process using the categories of the active edges added.

With predicting operation and passive edges we start with from the unit substrings as we had in CYK, we are able to initialize the agenda with both active and inactive edges. During parsing, we need to follow the same scheme when passive edges are combined from active/passive edges.

The example of passive/active edges generated while parsing the ride bike example is shown in Fig.5.27.

5.1.2.4 Limitation

One of the drawbacks of a bottom-up strategy is that all constituents are licensed by the grammar built, regardless of whether they could be incorporated into a complete parse, that is, the algorithm is insufficiently goal-driven. For example, suppose our sentence was “the search for Spock was successful”. The phrase “search for Spock” would be found not only as part of a noun phrase, but also a present-tensed verb phrase. But a verb phrase can never directly follow a determiner such as “the”, so adding grammar context from a top-down perspective would allow us to prune the search tree.

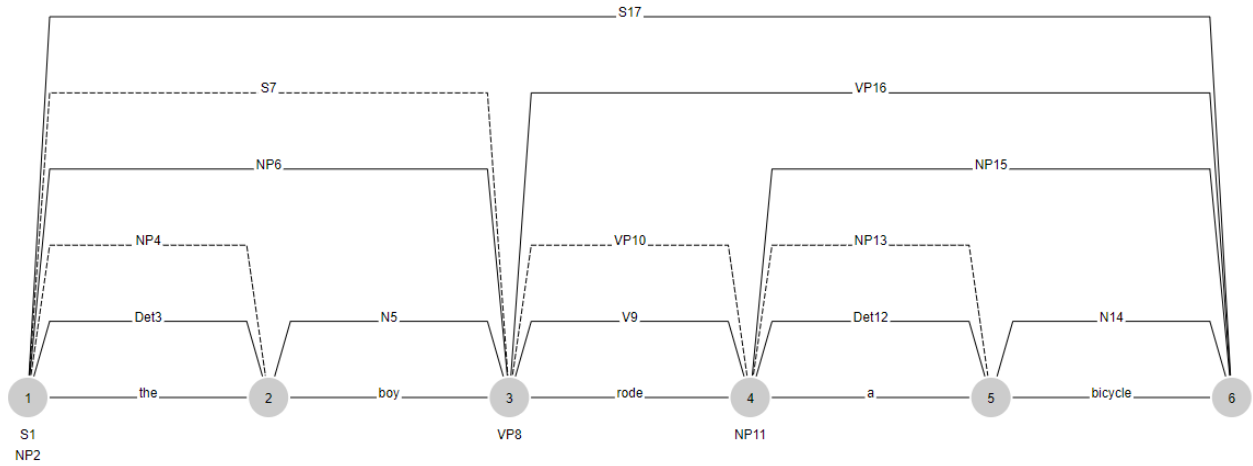


Figure 5.27: Chart visualization for bottom-up active parsing

5.1.3 Top-Down Earley Parser and Generalization

As mentioned in the previous section, bottom-up parsing algorithms ignore top-down knowledge from the grammar. All possible edges in the chart are generated in a bottom-up fashion, not considering whether these edges or the new edges could actually be derived from the starting symbol. This adds a lot of redundant constituents which can not be incorporated into a complete parse. Therefore, one intuitive idea to solve this issue is to make predictions, generate edges, in a top-down manner. Earley parser [30] is an exemplar top-down parsing algorithm and we will treat it as the main focus of this section to illustrate the ideas.

Recall the bottom-up active parsing algorithm we discussed previously, the bottom-up predict operation generates new edges from the smallest units of the sentence and checks if there are production rules taking these constituents as their RHS. A dotted rule (active edge) is then added to the agenda for later use. In a top-down parser, we generate active edges in a similar fashion, however, from the starting symbol of grammar S instead of all words. When initializing the agenda, the **top-down prediction** operation is illustrated as follows:

Top-Down Predict

- For an edge with category A
 - for every rule that has the goal category A as its LHS, $r : A \rightarrow \beta$, add an active edge to the agenda of the form $e = (0, 0, A, \emptyset, \beta)$
- Repeat this process using the categories of the active edges added until no edge can be added.

Using the example grammar we discussed, we begin with the start symbol S and production rule $S \rightarrow$

NP VP, the following edges are added to the agenda in top-down prediction:

$$\begin{aligned}
 S &\rightarrow \cdot \text{NP VP} & e &= (0, 0, S, \emptyset, \text{NP VP}) \\
 \text{NP} &\rightarrow \cdot \text{DET N} & e &= (0, 0, \text{NP}, \emptyset, \text{DET N}) \\
 \text{DET} &\rightarrow \cdot \text{"a"} & e &= (0, 0, \text{DET}, \emptyset, \text{"a"}) \\
 \text{DET} &\rightarrow \cdot \text{"the"} & e &= (0, 0, \text{DET}, \emptyset, \text{"the"})
 \end{aligned}$$

You might have noticed that all edges created are active edges in top-down prediction. This allows us to add the grammar constraints into chart parsing so that only grammar rules that can be legally applied will be put on the chart. This guaranteed that no redundant constituents which can not be incorporated into a complete parse will be generated, since every edge is generated following a path from the start symbol S .

The Earley parser proposed in the 1970s is one of the top-down chart parsing algorithms that implement the above ideas. We follow the notations used in most of existing literature when describing Earley parser and connect them to concepts we discussed in chart parsing.

We start with the definition of **states**. A state is a tuple $(A \rightarrow \alpha \cdot \beta, i, j)$ consisting of (i) the dotted production rule currently being matched $(A \rightarrow \alpha \cdot \beta)$, and (ii) the position i in the input at which the matching of this production began and the end position j . Using our previous definition of edges, each Earley state is equivalent to edge $e = (i, j, A, \alpha, \beta)$. Seeded with production rules of start symbol S and initial states $\{(S \rightarrow \cdot \alpha, 0, 0)\}$ (initialized chart), the parser repeatedly executes three operations: **predict**, **scan** and **complete**:

Earley Parser Operations

- **Predict:** for each state in the chart of the form $(A \rightarrow \alpha \cdot B\beta, i, j)$, add $(B \rightarrow \cdot \gamma, j, j)$ to the chart for every production rule with B on the left-hand side ($r : B \rightarrow \gamma$).
- **Scan:** for each state in the chart of the form $(A \rightarrow \alpha \cdot a\beta, i, j)$, where a is a word token, add $(A \rightarrow \alpha a \cdot \beta, i, j + 1)$ to the chart.
- **Complete:** for each of the form $(B \rightarrow \gamma \cdot, j, k)$, find states of the form $(A \rightarrow \alpha \cdot B\beta, i, j)$ and add $(A \rightarrow \alpha B \cdot \beta, i, k)$ to the chart.

The three operations match with our previous discussion on chart parsing. Predict and complete operations are identical to top-down predicting and combining edges according to the fundamental rule in chart parsing. And Scan operation is to ensure that all passive edges generated could be incorporated into a complete parse given the grammar.

During implementation, the *finish* field can be substituted by the definition of state set $S(m)$. Formally, state set $S(m)$ contains all Earley states that has the span from start to m . We use the alternative notation $(A \rightarrow \alpha \cdot B\gamma, i)$ in $S(j)$ to denote edge $e = (i, j, A, \alpha, B\gamma)$ without *finish* field. The detailed algorithm is illustrated in Alg.11. PREDICT, SCAN and COMPLETE corresponds to the operations we discussed earlier but under the state set convention. It is obvious that these two forms of operation are equivalent. Notice that in Alg.11, we use an additional symbol Γ to summarize all productions with start symbol S . This sets a unique pseudo root where top-down prediction starts from, even if there exist multiple production rules that take the start symbol S as LHS.

Algorithm 11: Top-Down Earley Parsing Algorithm

Input : sentence $X = w_1 \cdots w_n$, grammar \mathcal{G} with production rules $R \in \mathcal{G}$ and start symbol S .

Output: verify whether $S \Rightarrow X$

```
1 Initialize  $n + 1$  empty state sets  $S(0), \dots, S(n)$ 
2 Add state  $(\Gamma \rightarrow \cdot S, 0)$  to state set  $S(0)$ 
3 for  $k = 0 \rightarrow n$  do
4   for state  $s \in S(k)$  do
5     if  $s$  passive then COMPLETE( $s, k$ )
6     if  $s$  active, next of  $s$  is terminal then SCAN( $s, k, X$ )
7     if  $s$  active, next of  $s$  is non-terminal then PREDICT( $s, k, \mathcal{G}$ )
8 Succeed if there is a state  $s = (\Gamma \rightarrow S\cdot, 0) \in S(n)$ 
9 def PREDICT( $s = (A \rightarrow \alpha \cdot B\gamma), k, \mathcal{G}$ ):
10   for  $r : B \rightarrow \gamma \in \mathcal{G}$  do
11      $\lfloor$  Add state  $s' = (B \rightarrow \cdot\gamma, k)$  to state set  $S(k)$ 
12 def SCAN( $s = (A \rightarrow \alpha \cdot a\beta, i), k, X$ ):
13   if  $a = w_k$  then
14      $\lfloor$  Add state  $s' = (A \rightarrow \alpha a \cdot \beta, i)$  to state set  $S(k + 1)$ 
15 def COMPLETE( $s = (B \rightarrow \gamma\cdot, j), k$ ):
16   for  $s = (A \rightarrow \alpha \cdot B\gamma, i) \in S(j)$  do
17      $\lfloor$  Add state  $s' = (A \rightarrow \alpha B \cdot \beta, i)$  to state set  $S(k)$ .
```

For visualization purposes, we choose a less space-consuming arithmetic grammar to illustrate the process of Earley parser. The state sets are shown in Fig.5.28. As there is only one production rule attached to the start symbol, here we do not add the pseudo root Γ and check if there is state $s = (S \rightarrow R\cdot, 0)$ in state set $S(3)$ for string “0+1”. For example, in state set $S(0)$, the second state $(R \rightarrow \cdot R + R, 0)$ is generated by top-down prediction from the initial state $(S \rightarrow \cdot R, 0)$ through PREDICT operation. In state set in $S(1)$, the second state $(R \rightarrow R \cdot + R, 0)$ is generated by combining state $(R \rightarrow 0\cdot, 0)$ (passive edge) with state $(R \rightarrow \cdot R + R, 0)$ (active edge) in $S(0)$ through COMPLETE operation. Other states are generated in a similar fashion following the three parser operations. We comment on the fourth column in Fig.5.28 for each states’ origin. In this case, the state $(S \rightarrow R\cdot, 0)$ indicates that the string $X = “0+1”$ can be successfully parsed by the given grammar.

As we can see from the previous illustrations, there are a lot of similarities between top-down and bottom-up chart parsing methods. In fact, the active chart parsing algorithm is basically the same as Earley’s algorithm, although it does not necessarily have to parse from left to right. As for complexity, the time complexity for both algorithms is $O(n^3)$. However, with top-down restrictions that eliminate inappropriate partial parses, the space complexity of Earley parser is $O(n)$, instead of $O(n^2)$ as in bottom-up passive parser (CYK parser). With its nice property of combining grammar context constraints with current sequence inputs, the Earley parser is also used for tasks outside the natural language domain, including video

Sample grammar: $S \rightarrow R; R \rightarrow R + R; R \rightarrow "0" | "1"$
Input string: 0 + 1

State: | state # | rule | origin | comment |

$S(0)$

(1)	$S \rightarrow \cdot R$	0	start rule
(2)	$R \rightarrow \cdot R + R$	0	predict: (1)
(3)	$R \rightarrow \cdot 0$	0	predict: (1)
(4)	$R \rightarrow \cdot 1$	0	predict: (1)

$S(1)$

(1)	$R \rightarrow 0 \cdot$	0	scan: $S(0)(3)$
(2)	$R \rightarrow R \cdot + R$	0	complete: (1) and $S(0)(2)$
(3)	$S \rightarrow R \cdot$	0	complete: (2) and $S(0)(1)$

$S(2)$

(1)	$R \rightarrow R + \cdot R$	0	scan: $S(1)(2)$
(2)	$R \rightarrow \cdot R + R$	2	predict: (1)
(3)	$R \rightarrow \cdot 0$	2	predict: (1)
(4)	$R \rightarrow \cdot 1$	2	predict: (1)

$S(3)$

(1)	$R \rightarrow 1 \cdot$	2	scan: $S(2)(4)$
(2)	$R \rightarrow R + R \cdot$	0	complete: (1) and $S(2)(1)$
(3)	$R \rightarrow R \cdot + R$	0	complete: (1) and $S(2)(2)$
(4)	$S \rightarrow R \cdot$	0	complete: (2) and $S(0)(1)$

Figure 5.28: An example of the Earley parser for parsing “0+1” with a simple arithmetic grammar. From Ref. [30].

understanding and future prediction. We leave the details to Chapter 7 for further illustrations.

5.1.4 Inside-Outside Algorithm for Parsing and Learning

In this section, we introduce the probabilistic context-free grammars and describe the famous Inside-Outside algorithm for its inference and learning. The key idea in probabilistic context-free grammars is to extend context-free grammars to define a probability distribution over possible parses. Why is this probability distribution useful? A crucial idea is that once we have this probability distribution, we can rank possible parses for any sentence in the order of probability and deal with ambiguity in the parsing process, in which a sentence might have multiple possible parses.

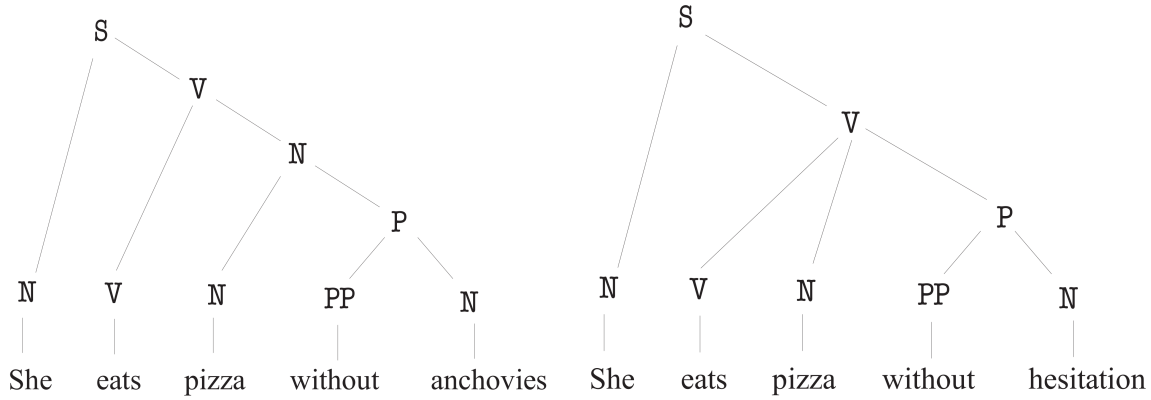


Figure 5.29: Two similar sentences and their syntactic parse trees.

5.1.4.1 Probabilistic Context-free Grammars

Probabilistic Context-Free Grammars (PCFGs) augments CFGs by associating each production rule with a probability. Formally, it is defined by a 5-tuple $G = (V, \Sigma, R, S, P)$, where P is the set of probabilities on production rules R . Since P defines a probability distribution, it has following constraints: for any non-terminal node $\alpha \in V$,

$$\phi(\alpha \rightarrow \beta) \geq 0 \quad (5.7)$$

$$\sum_{\beta} \phi(\alpha \rightarrow \beta) = 1.0 \quad (5.8)$$

Given a parse tree t using a set of rules $\{\alpha_0 \rightarrow \beta_0, \alpha_1 \rightarrow \beta_1, \dots, \alpha_n \rightarrow \beta_n\}$, its probability is defined as:

$$p(t) = \prod_{i=1}^n \phi(\alpha_i \rightarrow \beta_i) \quad (5.9)$$

5.1.4.2 Inside-Outside Algorithm

To make a grammar probabilistic, we need to assign a probability to each production rule. How should these probabilities be determined? It is natural to expect that these probabilities depend on the domain that is being parsed and we would like to tune the rule probabilities to fit our data. According to the statistical principle of maximum likelihood, we should choose the rule probabilities so as to maximize the likelihood of the observed data. Next, we will describe how this principle is realized in the Inside-Outside algorithm for parameter estimation of context-free grammars.

Consider two similar sentences and their syntactic parses in Figure 5.29. These two sentences only have one different word. In the absence of other constraints, both parses are valid for each sentence. That is, one could, at least syntactically, speak of a type of “pizza without hesitation”, though this would certainly be semantic gibberish, if not completely without taste. One of the goals of probabilistic training and parsing is to enable the statistics to correctly distinguish between such structures for a given sentence. This is indeed a formidable problem. Throughout this chapter, we will refer to the above sentences and grammar to demonstrate how the training algorithm is actually carried out. While certainly a toy example, it will serve to illustrate the general algorithm, as well as to demonstrate the strengths and weaknesses of the approach.

The Inside-Outside algorithm starts from some initial settings of the probabilities, and iteratively adjusts them to increase the likelihood of the training corpus. Denote the two sentences in Figure 5.29 as W_1, W_2 and their parses as T_1, T_2 . Then their probabilities can be written as:

$$\begin{aligned} P_\phi(W_1, T_1) = & \phi(S \rightarrow N V) \phi(V \rightarrow V N) \phi(N \rightarrow N P) \times \\ & \times \phi(P \rightarrow PP N) \phi(N \rightarrow \text{She}) \phi(V \rightarrow \text{eats}) \times \\ & \times \phi(N \rightarrow \text{pizza}) \phi(PP \rightarrow \text{without}) \phi(N \rightarrow \text{anchovies}) \end{aligned} \quad (5.10)$$

$$\begin{aligned} P_\phi(W_2, T_2) = & \phi(S \rightarrow N V) \phi(V \rightarrow V N P) \phi(P \rightarrow P PP) \times \\ & \times \phi(N \rightarrow \text{She}) \phi(V \rightarrow \text{eats}) \phi(N \rightarrow \text{pizza}) \times \\ & \times \phi(PP \rightarrow \text{without}) \phi(N \rightarrow \text{hesitation}) \end{aligned} \quad (5.11)$$

In the absence of other restrictions, each sentence can have both parses. Thus we also have:

$$\begin{aligned} P_\phi(W_1, T_2) = & \phi(S \rightarrow N V) \phi(V \rightarrow V N P) \phi(P \rightarrow P PP) \times \\ & \times \phi(N \rightarrow \text{She}) \phi(V \rightarrow \text{eats}) \phi(N \rightarrow \text{pizza}) \times \\ & \times \phi(PP \rightarrow \text{without}) \phi(N \rightarrow \text{anchovies}) \end{aligned} \quad (5.12)$$

$$\begin{aligned} P_\phi(W_2, T_1) = & \phi(S \rightarrow N V) \phi(V \rightarrow V N) \phi(N \rightarrow N P) \times \\ & \times \phi(P \rightarrow PP N) \phi(N \rightarrow \text{She}) \phi(V \rightarrow \text{eats}) \times \\ & \times \phi(N \rightarrow \text{pizza}) \phi(PP \rightarrow \text{without}) \phi(N \rightarrow \text{hesitation}) \end{aligned} \quad (5.13)$$

The likelihood of our corpus with respect to the parameters ϕ is thus

$$L(\phi) = (P_\phi(W_1, T_1) + P_\phi(W_1, T_2)) (P_\phi(W_2, T_1) + P_\phi(W_2, T_2)). \quad (5.14)$$

Starting at some initial parameters ϕ , the inside-algorithm re-estimates the parameters to obtain new parameters ϕ' for which $L(\phi') > L(\phi)$. This process is repeated until the likelihood doesn't increase.

To derive how the Inside-Outside algorithm updates the parameters, we assume that the grammar is in Chomsky normal form. Given the initial probabilities ϕ and a training corpus $\{W_1, W_2, \dots, W_N\}$, the parameters are re-estimated to obtain new parameters ϕ' as follows:

$$\begin{aligned} \phi'(A \rightarrow BC) &= \frac{\text{count}(A \rightarrow BC)}{\sum_\alpha \text{count}(A \rightarrow \alpha)} \\ \phi'(A \rightarrow w) &= \frac{\text{count}(A \rightarrow w)}{\sum_\alpha \text{count}(A \rightarrow \alpha)} \\ \text{count}(A \rightarrow BC) &= \sum_{i=1}^N c_\phi(A \rightarrow BC, W_i) \\ \text{count}(A \rightarrow w) &= \sum_{i=1}^N c_\phi(A \rightarrow w, W_i) \end{aligned} \quad (5.15)$$

The number $c_\phi(A \rightarrow \alpha, W_i)$ is the expected number of times that the production rule $A \rightarrow \alpha$ is used in generating the sentence W_i when the rule probabilities are given by ϕ . To calculate these expected counts, we need to introduce two key ingredients: *inside* and *outside* probabilities.

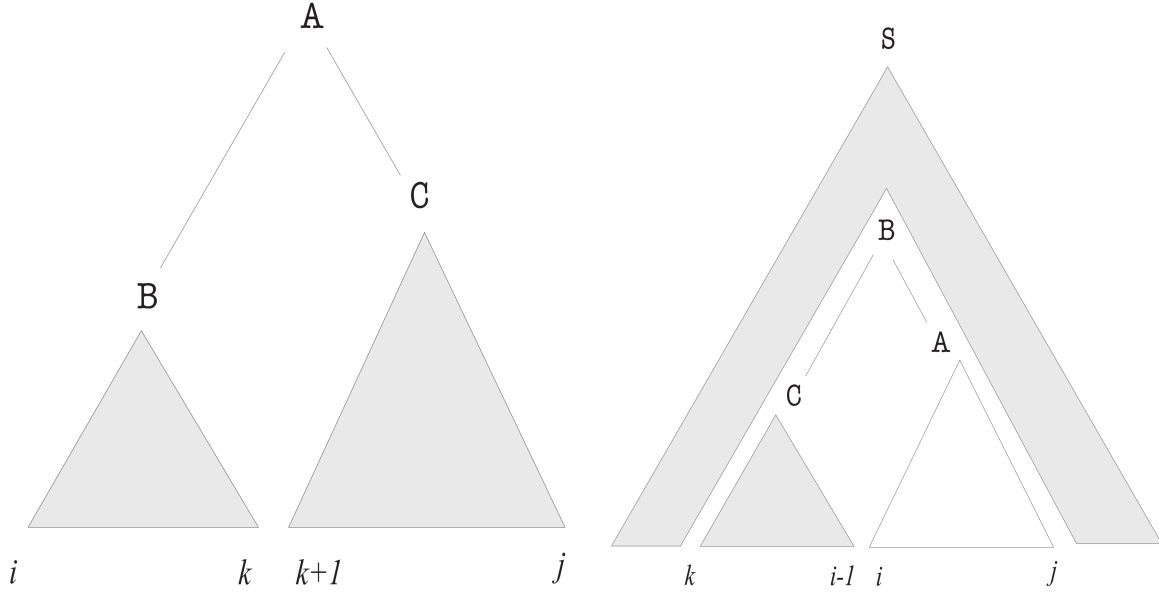


Figure 5.30: The *inside* [left] and *outside* [right] probabilities.

Consider a sentence $W = w_1 w_2 \dots w_n$. The *inside* probability of a non-terminal node A over the range $[i, j]$ is defined as the probability that A derives the string of words $w_i \dots w_j$ in the sentence W :

$$\alpha_{ij}(A) = P_\phi \left(A \xrightarrow{*} w_i \dots w_j \right). \quad (5.16)$$

The *outside* probability of A is defined as the probability that the start symbol S derives the string $w_1 \dots w_{i-1} A w_{j+1} \dots w_n$:

$$\beta_{ij}(A) = P_\phi \left(S \xrightarrow{*} w_1 \dots w_{i-1} A w_{j+1} \dots w_n \right) \quad (5.17)$$

Having these *inside* and *outside* probabilities, we can define the expected counts for a production rule $A \rightarrow BC$ used in the derivation of the sentence W as:

$$c_\phi(A \rightarrow BC, W) = \frac{\phi(A \rightarrow BC)}{P_\phi(W)} \sum_{1 \leq i \leq j \leq k \leq n} \beta_{ik}(A) \alpha_{ij}(B) \alpha_{j+1,k}(C) \quad (5.18)$$

$$c_\phi(A \rightarrow w, W) = \frac{\phi(A \rightarrow w)}{P_\phi(W)} \sum_{1 \leq i \leq n} \beta_{ii}(A) \quad (5.19)$$

To efficiently calculate the inside and outside probabilities, we leverage following facts:

$$\alpha_{ij}(A) = \sum_{B,C} \sum_{i \leq k \leq j} \phi(A \rightarrow BC) \alpha_{ik}(B) \alpha_{k+1,j}(C) \quad (5.20)$$

$$\begin{aligned} \beta_{ij}(A) = & \sum_{B,C} \sum_{1 \leq k < i} \phi(B \rightarrow C A) \alpha_{k,i-1}(C) \beta_{kj}(B) \\ & + \sum_{B,C} \sum_{n \geq k > j} \phi(B \rightarrow A C) \alpha_{j+1,k}(C) \beta_{ik}(B) \end{aligned} \quad (5.21)$$

Actually, we can use compute these probabilities by dynamic programming approaches, like the CYK algorithm.

To summarize, the Inside-Outside algorithm consists of the following steps:

1. initialize the parameters ϕ and set all of the counts to zero;
2. compute the inside and outside probabilities for each sentence in the training corpus;
3. compute the expected number of times that each rule is used in generating each sentence and add to the total count;
4. update the parameters by:

$$\phi'(A \rightarrow \alpha) = \frac{\text{count}(A \rightarrow \alpha)}{\sum_{\gamma} \text{count}(A \rightarrow \gamma)} \quad (5.22)$$

5. repeat the process until it converges.

The Inside-Outside algorithm is guaranteed not to decrease the log likelihood. It can be stopped when the change in log likelihood is sufficiently small.

5.1.5 Figure of Merit Parsing

Chart parsing is a commonly used algorithm for parsing natural language texts. A **chart** is the data structure containing all constituents from which the algorithm generates subtrees, that is, constituents for which a derivation has been found and may appear in some complete parse of the sentence. The **agenda** is a structure that stores a list of constituents, in which the algorithm has not yet combined them with other constituents, but we can find a derivation. Initially, the agenda contains the terminal symbols from the sentence ready for processing. In the processing, the agenda removes a constituent and adds it to the chart, and the system considers how it uses this constituent to extend its current structural hypothesis by combining it with other constituents in the chart according to the grammar rules. (We will often refer to these expansions of rules as "edges".) In general, this can lead to the creation of new, more encompassing constituents, and the agenda then stores them. Once the algorithm has processed one constituent, it chooses a new one to remove from the agenda, and so on. Traditionally, the agenda is represented as a stack, so that the last item added to the agenda is the next one removed.

Best-first probabilistic chart parsing is a variation of chart parsing that attempts to find the most likely parses first, by adding constituents to the chart in order of the likelihood that they will appear in a correct parse, rather than merely popping constituents off of a stack. Some probabilistic figures of merit are assigned to the constituents on the agenda, and the constituent maximizing this value is the next to be added to the chart.

The purpose of applying chart parsing is to compare how well several figures of merit select constituents to be moved from the agenda to the chart. Ideally, the objective is to pick the constituent that maximizes the conditional probability:

$$p(N_{j,k}^i | t_{0,n}) \quad (5.23)$$

where $t_{0,n}$ is the sequence of the n tags, or parts of speech, in the sentence (numbered t_0, \dots, t_{n-1}), and $N_{j,k}^i$ is a nonterminal of type i covering terms $t_j \dots t_{k-1}$. (See Figure 5.31.) The agenda uses a *priority queue* to keep track of partial derivations. The key to best-first chart parsing is how to estimate it, then the order of constituents in agenda is calculated based on some figures of merit of constituents which measure the *likelihood* that the constituents will appear in a correct parse, rather than simply popping constituents off of a stack (which simulates *DFS*) or a queue (which simulates *BFS*).

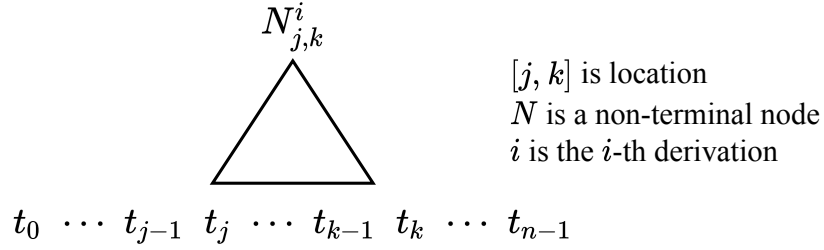


Figure 5.31: Constituent $N_{j,k}^i$ in a sentence $t_{0,n}$. Reprinted, with permission, from (Sharon, *et al.*, 1998 [15])

5.1.5.1 Ideal Figure of Merit for a Node $N_{j,k}^i$

Our "ideal" figure of merit can be rewritten as:

$$\begin{aligned}
 p(N_{j,k}^i | t_{0,n}) &= \frac{p(N_{j,k}^i, t_{0,n})}{p(t_{0,n})} \\
 &= \frac{p(N_{j,k}^i, t_{0,j}, t_{j,k}, t_{k,n})}{p(t_{0,n})} \\
 &= \frac{p(t_{0,j}, N_{j,k}^i, t_{k,n}) p(t_{i,k} | t_{0,j}, N_{j,k}^i, t_{k,n})}{p(t_{0,n})}
 \end{aligned} \tag{5.24}$$

We apply the usual independence assumption that given a nonterminal, the tag sequence it generates depends only on that nonterminal, giving:

$$\begin{aligned}
 p(N_{j,k}^i | t_{0,n}) &\approx \frac{p(t_{0,j}, N_{j,k}^i, t_{k,n}) p(t_{i,k} | N_{j,k}^i)}{p(t_{0,n})} \\
 &= \frac{p^{(out)}(N_{j,k}^i) p^{(in)}(N_{j,k}^i)}{p(t_{0,n})}
 \end{aligned} \tag{5.25}$$

where **inside probability** of the constituent $N_{j,k}^i$ is defined as $p^{(in)}(N_{j,k}^i) = p(t_{j,k} | N_{j,k}^i)$, which is the *likelihood* rooted at $N_{j,k}^i$ and is the probability of the words in the constituent given that the constituent is dominated by a particular nonterminal symbol, as shown in Figure 5.32; and **outside probability** of a constituent $N_{j,k}^i$ is defined as $p^{(out)}(N_{j,k}^i) = p(t_{0,j}, N_{j,k}^i, t_{k,n})$, which is *context* for $N_{j,k}^i$ and also means the probability of that constituent and the rest of the words in the sentence, see Figure 5.33 for the illustration; and the denominator $p(t_{0,n})$ is a constant.

When the figures of merit are simple designs, we can apply heuristic to save time. The *straight inside probability* describes the side effect of omitting the outside probability $p^{(out)}(N_{j,k}^i)$ and $p(t_{0,n})$ term, this leads to the preference of shorter constituents to longer ones, further results in a "thrashing" effect. To avoid thrashing, some techniques are used to normalize the inside probability for use as a figure of merit. One approach is to take the geometric mean of the inside probability, to obtain a per-word inside probability. The per-word inside probability of the constituent $N_{j,k}^i$ is calculated as $\sqrt[k-j]{p^{(in)}(N_{j,k}^i)}$. In computer vision terminology, this is also seen as the *score* per-word for the vision grammar.

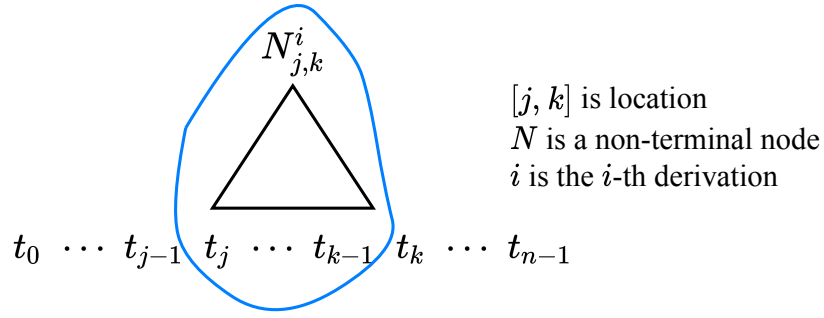


Figure 5.32: Inside probability includes only words within the constituent. Reprinted, with permission, from (Sharon, *et al.*, 1998 [15])

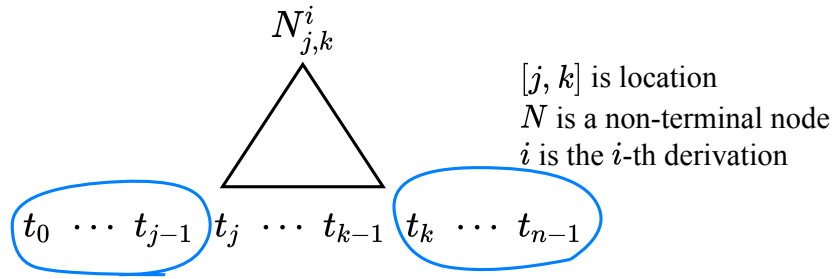


Figure 5.33: Outside probability includes the entire context of the constituent. Reprinted, with permission, from (Sharon, *et al.*, 1998 [15])

An alternative way to rewrite the "ideal" figure of merit is as follows:

$$\begin{aligned}
 p(N_{j,k}^i | t_{0,n}) &= \frac{p(N_{j,k}^i, t_{0,n})}{p(t_{0,n})} \\
 &= \frac{p(t_{0,j}, t_{k,n}) p(N_{j,k}^i | t_{0,j}, t_{k,n}) p(t_{j,k} | N_{j,k}^i, t_{0,j}, t_{k,n})}{p(t_{0,j}, t_{k,n}) p(t_{j,k} | t_{0,j}, t_{k,n})}
 \end{aligned} \tag{5.26}$$

Once again applying the usual independence assumption that given a nonterminal, the tag sequence it generates depends only on that nonterminal, we can rewrite the figure of merit as follows:

$$p(N_{j,k}^i | t_{0,n}) \approx \frac{p(N_{j,k}^i | t_{0,j}, t_{k,n}) p^{(in)}(N_{j,k}^i)}{p(t_{j,k} | t_{0,j}, t_{k,n})} \tag{5.27}$$

Assume $p(N_{j,k}^i | t_{0,j}, t_{k,n}) \approx p(N_{j,k}^i) = p(N^i)$ and use a **trigram model** for

$$\begin{aligned}
 p(t_{j,k} | t_{0,j}, t_{k,n}) &\approx p(t_{j,k} | t_{j-2}, t_{j-1}) \\
 &= \prod_{a=j}^{k-1} p(t_a | t_{a-2}, t_{a-1})
 \end{aligned} \tag{5.28}$$

then we have

$$p(N_{j,k}^i | t_{0,n}) = \frac{p(N^i) p^{(in)}(N_{j,k}^i)}{p(t_{j,k} | t_{j-2}, t_{j-1})} \tag{5.29}$$

which is also known as **trigram statistics**. When solving the probability distribution $p(N_{j,k}^i | t_{0,n})$ we can apply an A^* algorithm, we need to design a heuristic function $h(s)$.

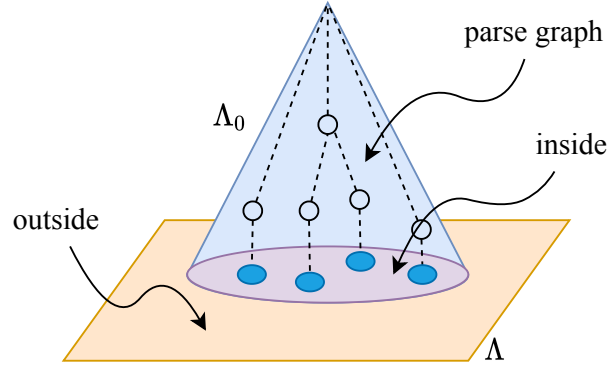


Figure 5.34: inside and outside probability of a parse graph on an image



Figure 5.35: 3 Scene examples. From Ref. [61].

Suppose we have an image I_Λ with its universe Λ , and it contains several objects, for instance, human face (say Λ_0 in example), dog, car, *etc.*, as shown in Figure 5.34. The log likelihood of the face is calculated by $\log P(A_{\Lambda_0}|I_\Lambda)$, further we define the fitness inside probability of Λ_0 as $P(I_{\Lambda_0}|A)$, and context fitness outside probability of Λ_0 as $P(A, I_{\Lambda, \Lambda_0})$. Therefore, the log likelihood can be described as sum of two scores

$$\log P(A_{\Lambda_0}|I_\Lambda) = \underbrace{\text{score}^{(in)}(A)}_{\text{bottom up}} + \underbrace{\text{score}^{(out)}(A)}_{\text{top down}} \quad (\text{joint parsing}) \quad (5.30)$$

From the above equation, we see that there's a bottom-up pass for calculating the inside probability, and a top-down pass for the outside probability. Further, the bottom-up process and top-down process are seen as the current cost function $g(s)$ and heuristic function (predicated score) $h(s)$ of an A^* algorithm, respectively, that is, $f(A) = g(A) + h(A)$.

5.1.5.2 Figures of Merit Example: Scene Parsing

In this section we discuss the scene parsing application with the analysis by figures of merit.

Given some scene examples in Figure 5.35, we first define a projection of planar rectangles in 3-space into the image plane. As illustrated in Figure 5.36, it has two pairs of parallel line segments in 3D which intersect at two vanishing points p_1 and p_2 in the image plane. Each intersection of a line pair has a orientation, *e.g.*, θ_1, θ_2 for p_1 and θ_3, θ_4 for p_2 , respectively. This projection is a form of representation and we see them as terminal nodes in graph analysis of the scene understanding. Therefore, the set of terminal nodes is defined as

$$V_T = \{(a, x(a)) : x(a) \in \Omega_a\} \quad (5.31)$$

where Ω_a is the universe, a is the terminal node, and its attributes $x(a)$, in our example, as shown in Figure

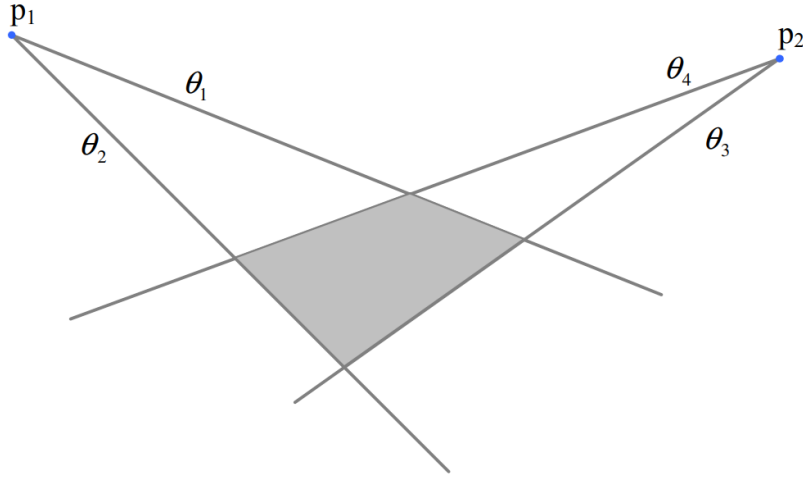


Figure 5.36: A planar rectangle (shaded) is described by several variables. The two vanishing points p_1 and p_2 and four directions $\theta_1, \theta_2, \theta_3, \theta_4$ at the two vanishing points. From Ref. [61] with permission.

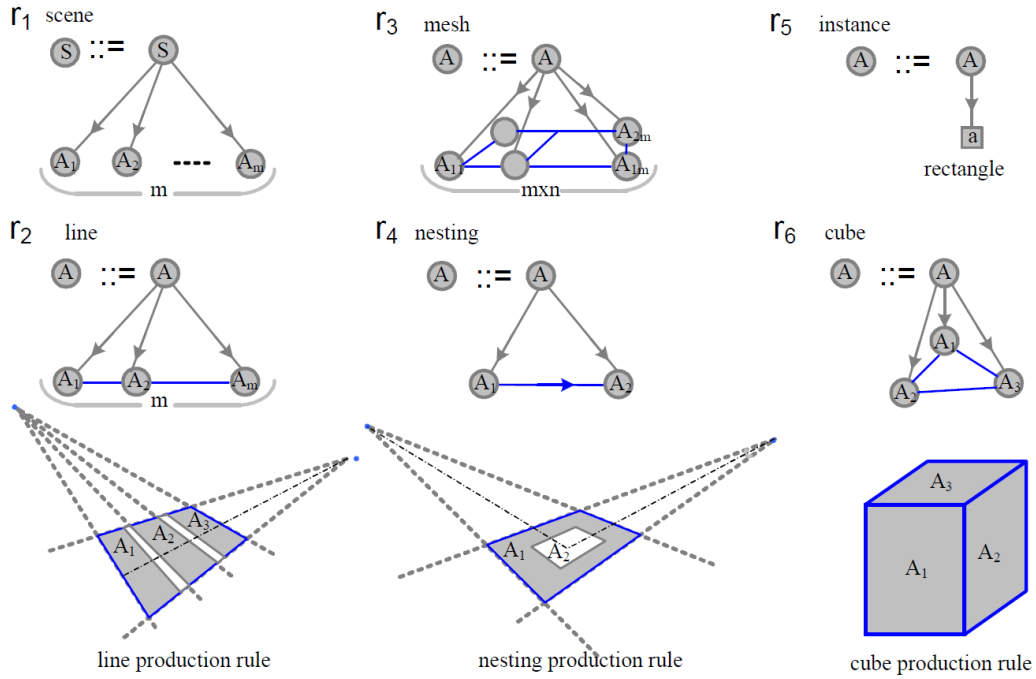


Figure 5.37: Six attribute grammar rules. Attributes will be passed between a node to its children and the horizontal lines show constraints on attributes. Reprinted from Ref. [61] with permission.

5.36, two vanishing points have coordinates $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, respectively. Then we get $x(a) = (x_1, y_1, x_2, y_2, \theta_1, \theta_2, \theta_3, \theta_4)$.

In addition to a representation, we also need an interpretation of the geometric relationship. Hence, we further define a grammar for image interpretation. Our representation has the root node S for the scene and non-terminal nodes A for objects and surfaces.

$$V_N = \{(S, X(S)), (A, X(A)) : X(S) = n, X(A) \in \Omega_A\} \quad (5.32)$$

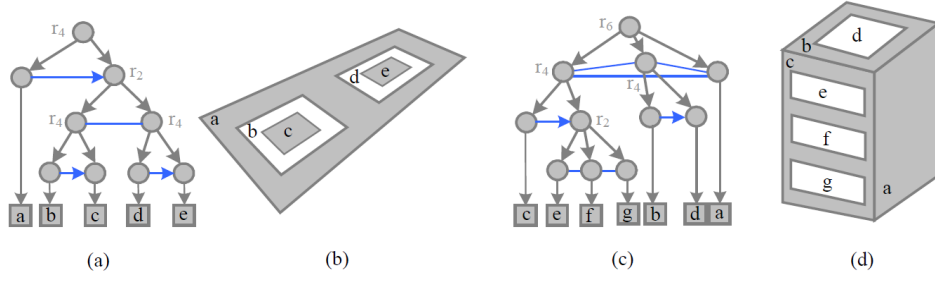


Figure 5.38: Two examples of rectangle object configurations (b) and (d) and their corresponding parse graphs (a) and (c). The production rules are shown as nonterminal nodes. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

The scene node S generates n independent objects. The object node A can be instantiated (assigned) to a rectangle (rule r_5), or be used recursively by the other four production rules: r_2 – the line production rule, r_3 – the mesh production rule, r_4 – the nesting production rule, and r_6 – the cube production rule. The six production rules are summarized in Figure 5.37.

This simple grammar can generate a language with a huge number of configurations for generic objects and scenes. Figure 5.38 shows two typical configurations — a floor pattern and a toolbox pattern, and their corresponding parse graphs.

The attribute $X(A) = (\ell(A), n(A), X_o(A))$ includes a label ℓ for the type of object (structure) represented by A , $\ell(A) \in \Omega_\ell = \{line, mesh, nest, rect, cube\}$, $n(A)$ for the number of children nodes in A , and $X_o(A)$ for its geometric properties and appearance. The variables in $X_o(A)$ depend on object type of A , therefore we denote the attribute space of A as the union of the five different subspaces.

$$\Omega_A = \Omega_A^{line} \cup \Omega_A^{mesh} \cup \Omega_A^{nest} \cup \Omega_A^{rect} \cup \Omega_A^{cube} \quad (5.33)$$

In the following, we briefly explain the constraint equations associated with the rules. In most cases, the constraint equations are straightforward but tedious to enumerate. Therefore we choose to introduce the typical examples. The simplest rule is r_5 for instantiation. It assigns a rectangle and the associated attributes to a non-terminal node A . Therefore the constraint equation is simply an assignment for the attribute variables.

$$r_5 : A \rightarrow \alpha; X_o(A) = x(a); \quad (5.34)$$

This assignment may go in either direction in the computation. For the line production rule r_2 , we choose $m = 3$ for simplicity.

$$\begin{aligned} r_2 : A &\rightarrow (A_1, A_2, A_3); \\ g_i(X_o(A)) &= f_i(X_o(A_1), X_o(A_2), X_o(A_3)), i = 1, 2, \dots, k. \end{aligned} \quad (5.35)$$

A is the bounding rectangle for A_1, A_2, A_3 and shares with them the two vanishing points and orientations. Given $X_o(A)$, the three rectangles A_1, A_2, A_3 have only 4 degrees of freedom for the two intervals, all the other $3 \times 8 - 4 = 20$ attributes are decided by the above attribute equations. One can derive the constraint equations for the other rules in a similar way.

Our objective is to compute a parse graph G by maximizing the posterior probability. The algorithm should achieve two difficult goals: (i) Constructing the parse graph, whose structure is not predetermined but constructed "on-the-fly" from the input image and primal sketch representation. (ii) Estimating and passing the attributes in the parse graph. Our algorithm consists of three phases. In **phase I**, we compute

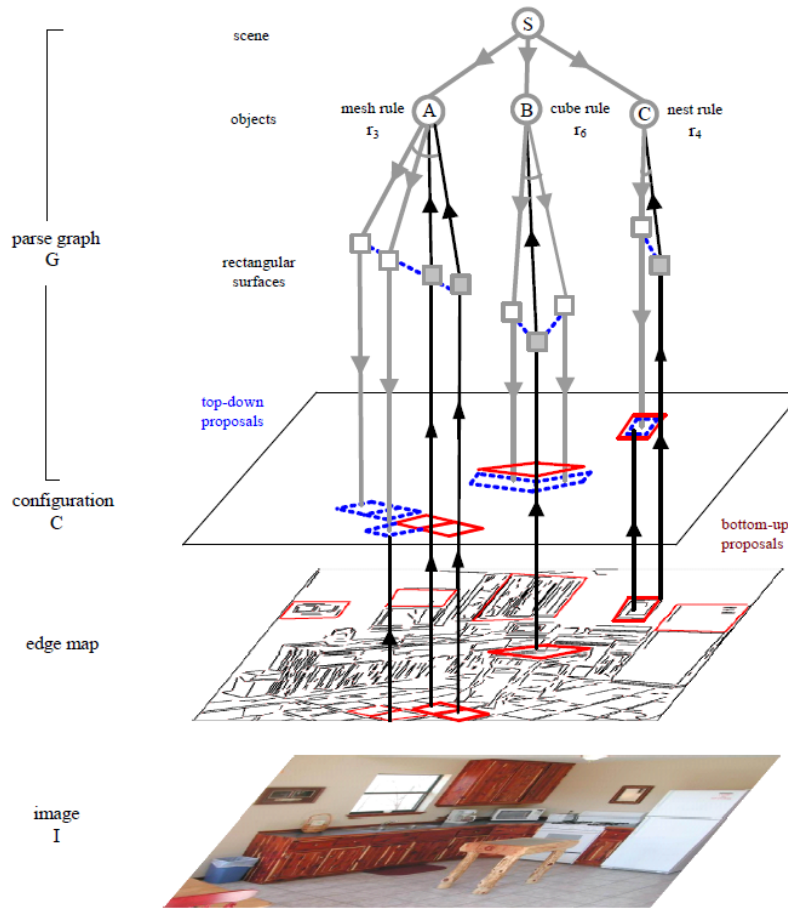


Figure 5.39: The hierarchic parse graph is a generative representation and it produces a configuration in the image plane. This configuration generates the image in a primal sketch model. The parse graph is constructed by an iterative top-down/bottom-up algorithm. The rectangular primitives detected in bottom-up step activate the grammar rules to predict missing or occluded components in top-down process. Reprinted/modified from Ref. [61] with permission.

a primal sketch representation, and initialize the configuration to the free sketches. Then a number of rectangle proposals are generated from the sketch by a bottom-up detection algorithm. In **phase II**, we adopt a simplified generative model by assuming independent rectangles (only r_5 and r_1 are considered). Thus we recognize a number of rectangles proposed in phase I to initialize rule r_5 in the parse graph. The algorithm in phase II is very much like matching pursuit [94]. Finally, **phase III** constructs the parse graph with bottom-up/top-down mechanisms. The building up of the parse graph is shown as Figure 5.39, the red rectangles are in chart and blue rectangles are in agenda. The rectangles in chart are closed/accepted proposals in parse graph, while the rectangles in agenda are still open proposals needed to be explored.

We start *Phase I* with edge detection and edge tracing to get a number of long contours. Then we compute a primal sketch representation, C_{sk} , using the likelihood model. We segment each long contour into a number of n straight line segments by polygon-approximation. In man-made scenes, the majority of line segments are aligned with one of three principal directions and each group of parallel lines intersects at a vanishing point due to perspective projection. We define all lines ending at a vanishing point to be a parallel line group. A rectangle has two pairs of parallel lines which belong to two separate parallel line

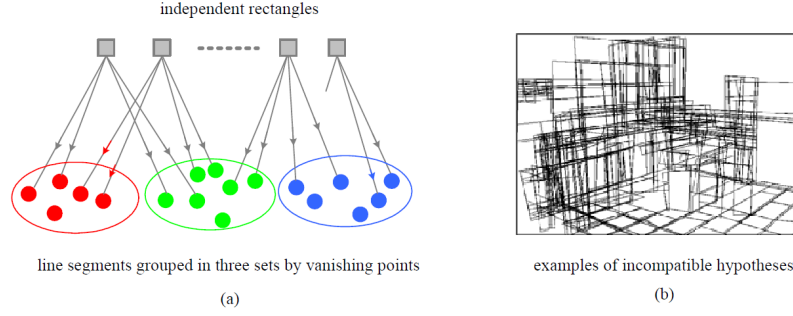


Figure 5.40: Bottom-up rectangle detection. The n line segments are grouped into three sets according to their vanishing points. Each rectangle consists of 2 pairs of nearly parallel line segments (represented by a small circle). Reprinted/modified from Ref. [61] with permission.

groups. We run the vanishing point estimation algorithm [78] to group all the line segments into three groups corresponding to the principal directions. With these three line groups, we generate the rectangle hypotheses as in RANSAC [47].

We exhaustively choose two lines candidates from each set as shown in Figure 5.40(a), and run some simple compatibility tests on their positions to see whether two pairs of lines delineate a valid rectangle. For example, the two pairs of line segments should not intersect each other as shown in Figure 5.40(b). This will eliminate some obviously inadequate hypotheses. This yields an excessive number of bottom-up rectangle candidates denoted by

$$\Phi = \{\pi_1, \dots, \pi_L\} \quad (5.36)$$

These candidates may conflict with each other. For example, two candidate rectangles may share two or more edge segments and only one of them should appear. We mark this conflicting relation among all the candidates. Thus if one candidate is accepted in the later stage, those conflicting candidates will be downgraded or eliminated. In *Phase II*, we adopt a simplified model which uses only two rules r_1 and r_5 . This model assumes the scene consists of a number of independent rectangles selected from Φ which explain away some line segments and the remaining lines are free sketches. A similar model has been used on signal decomposition with wavelets and sparse coding, thus our method for selecting the rectangles is similar to the matching pursuit algorithm [94]. *Phase III* aims to utilize the algorithm to construct the parse graph, it adapts a similar greedy method as in *Phase II*. In this phase, we include the four other production rules r_2, r_3, r_4, r_6 and use the top-down mechanism for computing rectangles which may have been missed in bottom-up detection. Use the kitchen scene in Figure 5.39 as example, the four rectangles (in red) are detected and accepted in the bottom-up phases I-II. They generate a number of candidates for larger groups using the production rules, and three of these candidates are shown as non-terminal nodes A, B, and C respectively. We denote each candidate by

$$\Pi = (r_\Pi, A_{(1)}, \dots, A_{(n_\Pi)}, B_{(1)}, \dots, B_{(k_\Pi)}) \quad (5.37)$$

In the above notation, r_Π is the production rule for the group. It represents a type of spatial layout or relationship of its components. For example, A, B, C in Figure 5.39 use the mesh r_3 , cube r_6 , and nesting r_4 rules respectively. In Π , $A_{(i)}, i = 1, 2, \dots, n_\Pi$ are the existing non-terminal nodes in G which satisfy the constraint equations of rule r . $A_{(i)}$ can be either a non-terminal rectangle accepted by rule r_5 in phase II or the bounding box of a non-terminal node with three rules r_2, r_3, r_4 . The cube object does not have a natural bounding box. We call $A_{(i)}, i = 1, 2, \dots, n_\Pi$ the bottom-up nodes for Π and they are illustrated by the upward arrows in Figure 5.39. In contrast, $B_{(j)}, j = 1, 2, \dots, k_\Pi$ are the top-down non-terminal nodes

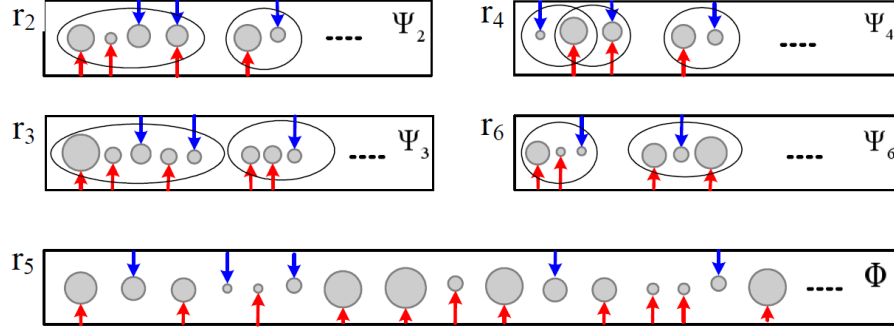


Figure 5.41: Four sets of proposed candidates $\Psi_2, \Psi_3, \Psi_4, \Psi_6$ for production rules r_2, r_3, r_4, r_6 respectively and the candidate set Φ for the instantiation rule r_5 . Each circle represents a rectangle π or a bounding box of a non-terminal node. The size of the circle represents its weight $\omega(\pi)$. Each ellipse in $\Psi_2, \Psi_3, \Psi_4, \Psi_6$ stands for a candidate Π which consists of a few circles. A circle may participate in more than one candidate. Reprinted from Ref. [62] with permission.

predicted by rule r_Π , and they are shown by the blue rectangles in Figure 5.39 with downward arrows. Some of the top-down rectangles may have already existed in the candidate set Φ but have not been accepted in Phase II or simply do not participate in the bottom-up proposal of Π . Such nodes bear both upward and downward arrows.

Figure 5.41 shows the five candidate sets for the five rules. Each grammar rule r_i can be seen as an assembly line and maintains an open list and closed list of particles, a particle is a production rule partially matched, its probability measures an approximated posterior probability ratio. Ψ_i is the candidate set of rule r_i for $i = 2, 3, 4, 6$ respectively. Each candidate $\Pi \in \Psi_i$ is shown by an ellipse containing a number of circles $A_{(i)}, i = 1, \dots, n_\Pi$ (with red upward arrows) and $B_{(j)}, j = 1, \dots, k_\Pi$ (with blue downward arrows). These candidates are weighted in a similar way as the rectangles in Φ by the log-posterior probability ratio.

$$\Psi_i = \{(\Pi_j, \omega_j) : i = 1, 2, \dots, N_i\}, i = 2, 3, 4, 6 \quad (5.38)$$

$\Phi = \{(\pi_i, \omega_i) : i = 1, 2, \dots, M\}$ for rule r_5 has been discussed in Phase II. Now Φ also contains top-down candidates shown by the circles with downward arrows. They are generated by other rules. A non-terminal node A in graph G may participate in more than one group of candidate Π 's, just as a line segment may be part of multiple rectangle candidate π 's. This creates overlaps between the candidates and needs to be resolved in a generative model.

At each step, the parsing algorithm will choose the candidate with the largest weight from the five candidate sets and add a new non-terminal node to the parse graph. If the candidate is $\pi \in \Phi$, it means accepting a new rectangle. Otherwise the candidate is a larger structure Π , and the algorithm creates a non-terminal node of type r by grouping the existing nodes $A_{(i)}, i = 1, 2, \dots, n_\Pi$ and inserts the top-down rectangles $B_{(j)}, j = 1, \dots, k_\Pi$ into the candidate set Φ . The key part of the algorithm is to generate proposals for π 's and Π 's and maintain the five weighted candidate sets $\Phi, \Psi_i, i = 2, 3, 4, 6$ at each step. We

summarize the algorithm as follows:

Algorithm 12: Constructing the parse graph \mathcal{G}

Input: $\mathcal{G} = \{A_1, \dots, A_m\}$ from phase II and $\Phi = \{(\pi_i, \omega_i) : i = 1, \dots, M - m\}$ from phase I

```

1 for rule  $r_i, i = 2, 3, 4, 6$  do
2   create candidate set  $\Psi_i = \text{Proposal}(\mathcal{G}, r_i)$ 
3   compute the weight  $\omega(\Pi)$  for  $\Pi \in \Psi_i$ 
4 end
5 while  $\omega_+ > \delta_1$  do
6   /* select a candidate with the heaviest weight, create a node
    $A_+$  with bounding box. */
7    $\omega_+(A_+) = \max\{\omega(A) : A \in \Phi \cup \Psi_2 \cup \Psi_3 \cup \Psi_6 \cup \Psi_6\}$ 
8   /* insert  $A_+$  to the parse graph  $\mathcal{G}$  */
9    $\mathcal{G} \leftarrow \mathcal{G} \cup \{A_+\}$ 
10  set the parent node of  $A_+$  to the non-terminal node which proposed  $A_+$  in the top-down phase
    or to the root  $S$  if  $A_+$  was not proposed in top-down.
11  if  $A_+ = \pi \in \Phi$  is a single rectangle then
12    /* Add the rectangle to the configuration */
13     $C(\mathcal{G}) \leftarrow C(\mathcal{G}) \cup \{\pi_+\}$ 
14  else
15    if  $A_+ = \Pi = (r_\Pi | A_{(1)}, \dots, A_{(n_\Pi)}, B_{(1)}, \dots, B_{(k_\Pi)})$  then
16      set  $A_+$  as the parent node of  $A_{(1)}, \dots, A_{(n_\Pi)}$ 
17      insert top-down candidates  $B_{(1)}, \dots, B_{(k_\Pi)}$  in  $\Phi$  with parent nodes  $A_+$ 
18    end
19  end
20  augment the candidate sets  $\Psi_i, i = 2, 3, 4, 6$  with the new node  $A_+$ 
21  compute weights for the new candidates and update  $\omega(\Pi)$  if  $\Pi$  overlaps with  $A_+$ 
22 end
23 return parse graph  $\mathcal{G}$ 

```

Figure 5.42 shows a snapshot of one iteration of the algorithm on the kitchen scene. Figure 5.42(b) is a subset of rectangle candidates Φ detected in phase I. We show a subset for clarity. At the end of phase II, we obtain a parse graph $\mathcal{G} = \{A_1, A_2, \dots, A_{21}\}$ whose configuration $C(\mathcal{G})$ is shown in (c). By calling the function $\text{Proposal}(\mathcal{G}, r_i)$, we obtain the candidate sets $\Psi_i, i = 2, 3, 4, 6$. The candidate sets are shown in (d-f). For each candidate $\Pi = (r_\Pi, A_{(1)}, \dots, A_{(n_\Pi)}, B_{(1)}, \dots, B_{(k_\Pi)})$, $A_{(i)}, i = 1, 2, \dots, n_\Pi$ are shown in red and $B_{(j)}, j = 1, 2, \dots, k_\Pi$ are shown in blue.

The function $\text{Proposal}(\mathcal{G}, r_i)$ for generating candidates from the current nodes $\mathcal{G} = \{A_i : i = 1, 2, \dots, m\}$ using r_i is not so hard, because the set $|\mathcal{G}|$ is relatively small ($m < 50$) in almost all examples. Each A_i has a bounding box (except the cubes) with 8 parameters for the two vanishing points and 4 orientations. We can simply test any two nodes A_i, A_j by the constraint equations of r_i . It is worth mentioning that each $A \in \mathcal{G}$ alone creates a candidate Π for each rule r_2, r_3, r_4, r_6 with $n(\Pi) = 1$. In such cases, the top-down proposals $B_{(j)}, j = 1, \dots, k_\Pi$ are created using both the constraint equations of r_i and the edge maps. For example, based on one rectangle A_8 , the top of the kitchen table in Figure 5.42(c), it proposes two rectangles by the cube rule r_6 in Figure 5.42(f). The parameters of those two rectangles are decided by the constraint equations of r_6 and the edges in the images.

We test our algorithm on a number of scenes with rectangle structures and show both qualitative results through image reconstruction (or synthesis) using the generative model and quantitative results through an

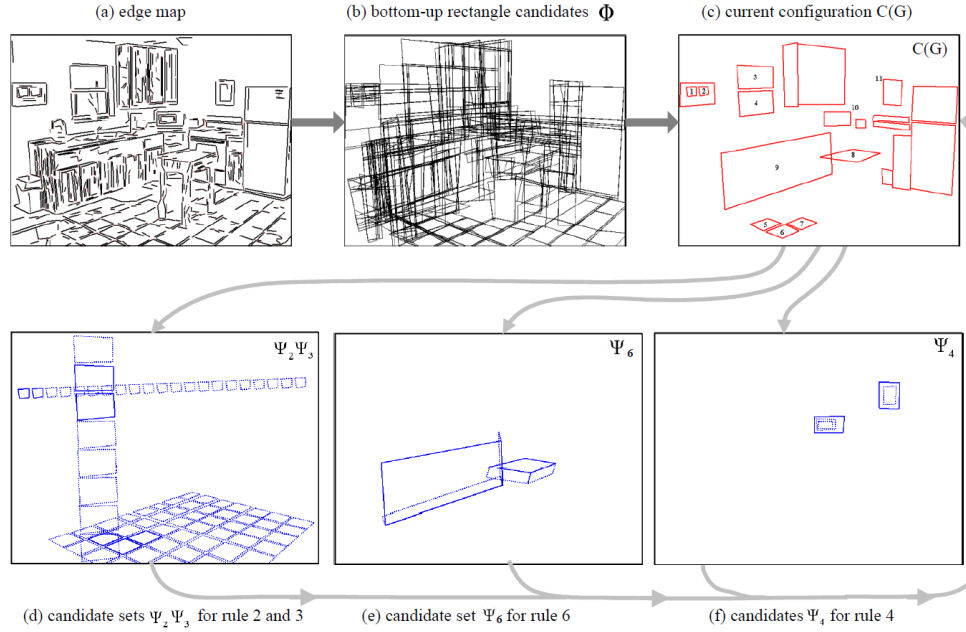


Figure 5.42: A kitchen scene as running example. (a) is the edge map, (b) is a subset of Φ for rectangle candidates detected in phase *I*. We show a subset for clarity. (c) is the configuration $C(\mathcal{G})$ with a number of accepted rectangles in phase *II*. (d-f) are candidates in $\Psi_2, \Psi_3, \Psi_4, \Psi_6$ respectively. They are proposed based on the current node in \mathcal{G} (*i.e.* shown in (b)). Reprinted from Ref. [61] with permission.

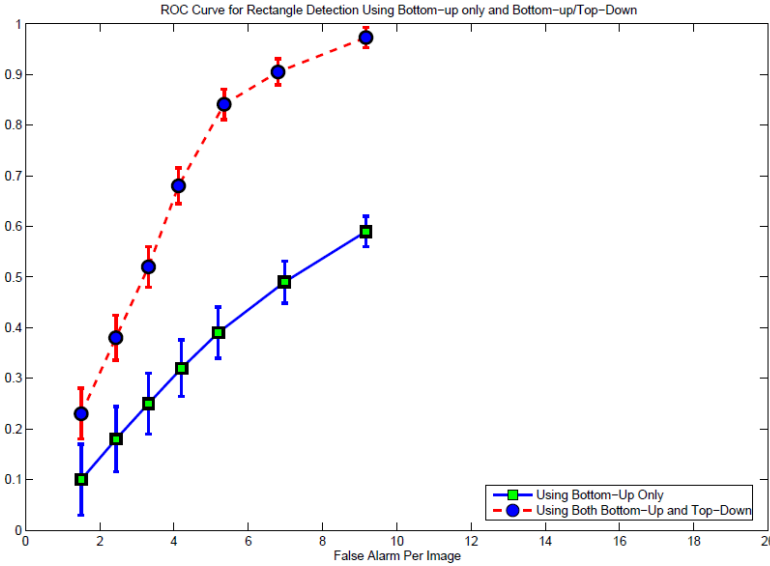


Figure 5.43: ROC curves for the rectangle detection results by using bottom-up only and, using both bottom-up and top-down. Reprinted from Ref. [62] with permission.

ROC curve comparing the performance of two approaches: (i) pure bottom-up rectangle detection, and (ii) our methods.

We show three results of the computed configurations and synthesized images in Figures 5.44 and 5.45.



Figure 5.44: Some experimental results. (a) Input image. (b) Edge map. (c) Computed rectangle configurations and missing rectangles compared with the ground truth: true positive rectangles are shown with solid lines, false positive rectangles are shown in dotted lines and missing rectangles are shown in dashed lines. Reprinted from Ref. [62] with permission.



Figure 5.45: (d) Reconstructed image from the primal sketch model using the rectangle configurations only. (e) Reconstructed images after adding some background sketches to the configurations. Reprinted from Ref. [62] with permission.

In Figure 5.44, the first row shows the input images, the second row shows the edge detection results, the third row shows the detected and grouped rectangles in the final configurations and missing rectangles compared with the ground truth (with true positive, false positives and missing rectangles being shown in

different line styles). In Figure 5.45 the first row are the reconstructed images based on the rectangle results in the third row of Figure 5.44. We can see that the reconstructed images miss some structures. Then we add the generic sketches (curves) in the edges, and final reconstructions are shown in the second row of Figure 5.45.

5.2 Scheduling Top-down and Bottom-up Processes for Object Parsing

The concept of top-down/bottom-up process has long been used by biology, psychology and computer vision communities to describe human vision understanding. When Hubel and Wiesel first discovered the Gabor filter-like neuron in cat's V1 area, several feed-forward model theories appear. These models view our brain as a hierarchical classifier that extracts features layer by layer. Poggio's papers "A feedforward architecture accounts for rapid categorization" and "Hierarchical models of object recognition in cortex" are good examples. These kinds of structures are called discriminative models. Although this new type of model helped the community leap forward one step, it doesn't solve the problem. Part of the reason is that there are ambiguities if you are only viewing part of the image locally and a feed-forward-only structure can't achieve global consistency. Therefore the idea that some kind of feedback model has to exist gradually emerged. Some of the early works in the computer science community had first came up with models that rely on feedback, such as Geoffrey Hinton's Boltzman Machine invented back in the 1980s which developed into the so-called deep learning around late 2000. Around the same time, Lamme in the neuroscience community started a series of research on the recurrent process in the vision system. His paper "The distinct modes of vision offered by feedforward and recurrent processing" published in 2000 addressed why recurrent (feedback) processing might be associated with conscious vision (recognizing object). However, it was only around early 2000 had David Mumford clearly addressed the importance of feedback in the paper "Hierarchical Bayesian inference in the visual cortex". Around the same time Wu, Zhu and others had also combined feedback and feedforward models successfully on textures in the paper "Visual learning by integrating descriptive and generative methods". Since then the computer vision community has partly embraced the idea that the brain is more like a generative model which in addition to categorizing inputs is capable of generating images.

In psychology, visual routine is a means of extracting information from a visual scene by Shimon Ullman. He proposed that the human visual system's task of perceiving shape properties and spatial relations is split into two successive stages: an early "bottom-up" state during which base representations are generated from the visual input, and a later "top-down" stage during which high-level primitives dubbed "visual routines" extract the desired information from the base representations. In humans, the base representations generated during the bottom-up stage correspond to retinotopic maps (more than 15 of which exist in the cortex) for properties like color, edge orientation, speed of motion, and direction of motion. These base representations rely on fixed operations performed uniformly over the entire field of visual input, and do not make use of object-specific knowledge, task-specific knowledge, or other higher-level information. In 2000, the paper "Competition for consciousness among visual events: The psychophysics of reentrant visual processes" also addressed the reentrant (feedback) visual process and proposed a model where conscious vision is associated with the reentrant visual process.

In hierarchical models, bottom-up and top-down are two basic computing mechanisms and are often used with three strategies:

- (i) pure bottom-up inference which passes messages in a feed-forward manner in the hierarchy, starting from data-driven features [5, 127, 134].
- (ii) pure top-down inference which passes messages in a feed-back manner in the hierarchy, starting from



Figure 5.46: Motivation of the α , β and γ inference processes using human face detection as an example. There are three cases of human faces appearing in the top image, each of which entails a different inference process, termed the α (face), β (face) and γ (face) processes respectively, as illustrated in the bottom panel. General identifications of the three processes are illustrated in Fig.5.47 and formal definitions of the three processes are introduced in Sec.5.2.1. Reproduced from Ref. [168] with permission of Springer, © 2011.

template matching [28, 146].

(iii) one pass of bottom-up inference followed by one phase of top-down inference [9, 29, 33, 85, 154].

In the recent vision literature, it is well acknowledged that both bottom-up and top-down inference processes contribute to object detection, recognition and parsing, and they should be combined [71, 83]. In hierarchical models, we observe that certain nodes, such as the human face, are often interpreted in a top-down fashion. One does that because it is much more effective to detect the full human face than individual facial components. In contrast, some other nodes such as junctions and handwriting digits are more effectively computed through bottom-up binding. For example, it is very difficult to detect rectangles directly. Instead, we can detect parallel lines or L-junctions first and then bind those compatible parallel lines or compatible L-junctions under some constraints. Furthermore, if we take scale and occlusion into account, one may have to adapt different computing strategies for different object instances. Fig.5.46 shows three cases in detecting human faces: the first case is a normal situation in which human faces are at middle resolution without occlusion, the second contains human faces at higher resolution but with occlusion and the third contains human faces at extremely low resolution. Intuitively, these three cases entail three different inference processes as illustrated at the bottom of Fig.5.46: human faces in the first case can be detected directly based on image data features, but those features work for the first case would fail in the second and the third cases due to occlusion and low resolution respectively. Human faces in the second case can be computed by binding those detectable facial components such as eyes and mouth, *etc.*, and those in the third

case can be predicted from their detectable surrounding contexts such as the head-shoulders. It is natural to ask the following three questions.

- (i) What inference processes, bottom-up and top-down, can be identified for nodes in hierarchical models?
- (ii) How much information does each of them contribute for different nodes?
- (iii) How should they be integrated to improve detection performance?

5.2.1 Integrating α - β - γ Processes in Inference

In this section, we present a *numerical study* of the bottom-up and top-down inference processes in hierarchical models — the And-Or graph. Three inference processes are identified for each node A in a recursively defined And-Or graph: the $\alpha(A)$ process detects node A directly based on image features, the $\beta(A)$ process computes node A by binding its child node(s) bottom-up and the $\gamma(A)$ process predicts node A top-down from its parent node(s). All the three processes contribute to computing node A from images in complementary ways. The objective of the numerical study is to explore how much information each process contributes and how these processes should be integrated to improve performance.

The α , β and γ processes in AoG. Fig.5.47 shows a portion of an AoG using the face example discussed in Fig.5.46 where node A represents human face, node P represents head-shoulder and node C_i 's represent facial components ($i = 1, 2, 3$). As an AoG is recursively defined, we can consider the α , β and γ processes of And-node A in Fig.5.47 without loss of generality.

Definition 1: (the α process). The $\alpha(A)$ process handles situations in which node A is at middle resolution without occlusion. Node A can be **detected directly** (based on its compact image data) and **alone** (without taking advantage of surrounding context) while its children or parts are not recognizable alone in cropped patches. An example of $\alpha(\text{face})$ process is shown in the left-bottom panel of Fig.5.46. Most of the sliding window detection methods in computer vision literature belong to this process. It can be viewed as either bottom-up or top-down. By bottom-up, it means that discriminative models are used to train the α process, such as the Adaboost classifiers [158]. By top-down, it means that generative models are used, such as the active basis model [170].

Definition 2: (the β process). When node A is at high resolution, it is more likely to be occluded in a scene. Node A itself is not detectable in terms of the $\alpha(A)$ process due to occlusion. A subset of node A 's child nodes can be detected in cropped patches (say, their α processes are activated). Then, the $\beta(A)$ process computes node A by **binding** the detected child nodes **bottom-up** under some compatibility constraints. An example of $\beta(\text{face})$ process is illustrated in the middle-bottom panel of Fig.5.46. Most of component [8,64], fragment [157] or part [2,132] based methods, the constellation models [36,43] and the pictorial models [40] belong to this process.

Definition 3: (the γ process). The $\gamma(A)$ process handles situations in which node A is at very low resolution. Node A can not be detected alone in isolation based on $\alpha(A)$, and neither can its parts. Then, the $\beta(A)$ process also fails. An example of $\gamma(\text{face})$ process is illustrated in the right-bottom panel of Fig.5.46. So, information outside of the local window must be incorporated. The $\gamma(A)$ process **predicts** node A **top-down** from a parent node whose α process is activated. We let the parent node pass contextual information, such as information from some sibling nodes or other spatial context. Most of the context-based methods [44,66,148] belong to this process.

For node A , all the three inference processes, $\alpha(A)$, $\beta(A)$ and $\gamma(A)$, contribute to computing it from images in complementary ways. The effectiveness of each process depends on the scale and occlusion

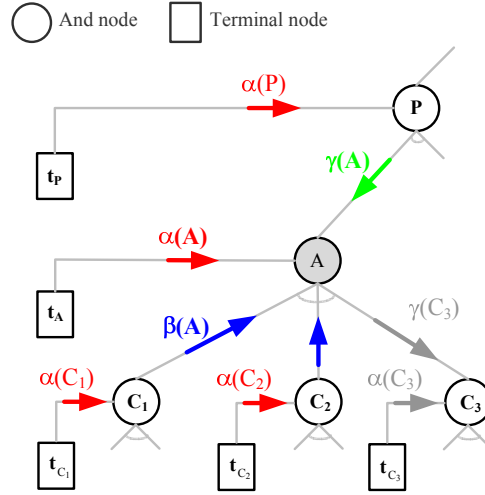


Figure 5.47: Illustration of identifying the $\alpha(A)$, $\beta(A)$ and $\gamma(A)$ inference processes for each And-node A in an AoG (see texts in Sec.5.2.1 for detail definitions). The $\alpha(A)$ process is directly based on the compact image data of node A (either bottom-up or top-down), the $\beta(A)$ process generates hypotheses of node A by bottom-up binding the α processes of some child node(s) (for example, $\{\alpha(C_1), \alpha(C_2)\} \rightarrow \beta(A)$), and the $\gamma(A)$ process predicts hypotheses of node A from the α processes of some parent node(s) (for example, $\alpha(P) \rightarrow \gamma(A)$ or $\beta(A) \rightarrow \gamma(C_3)$ in a top-down fashion). In computing, each process has two states: “on” or “off”, for example, $\alpha(C_3)$ process is off and we show it in grey. As an AoG is defined recursively, each And-node has its own α , β and γ processes (except that the root node’s γ processes and the β -processes of leaf nodes are always off). Reproduced from Ref. [168] with permission of Springer, © 2011.

conditions. As shown in Fig.5.48, the three cases of human faces shown in Fig.5.46 can be handled by the $\alpha(\text{face})$, $\beta(\text{face})$ and $\gamma(\text{face})$ respectively. Intuitively, for robust inference we should integrate them. As an AoG is a recursive structure, the three inference processes are also defined recursively and each And-node has its own α , β and γ inference processes (except that the γ process of the root node and the β processes of leaf nodes are always disabled).

Motivation for training the α , β and γ processes separately. As in [168], we train the three processes separately based on their respective isolated training data. We introduce the isolation method in Sec.5.2.2.1. Here, we propose the motivation and necessity to do that. Suppose we want to learn a human face classifier (*i.e.* the $\alpha(\text{face})$ process). There are two choices in selecting positive examples: (i) only face examples like those pointed by the $\alpha(\text{face})$ arrow in Fig.5.48, or (ii) a set of human face examples mixing all those shown in the right middle box in Fig.5.48. In the literature, people often get positive examples by cropping image patches only based on the labelled bounding boxes. When labelling the bounding box for an object instance, however, one often already takes advantage of all the information coming from the α , β and γ processes. Most of the existing work often trained a classifier based on a set of mixed positive examples (especially, mixing the α case and the γ case). Then, the learned classifier could be contaminated depending on the mixing rate implicitly [4, 44, 149]. We can explain the contamination. Generally, whether a feature is selected into a classifier depends on how different the feature responses of positive examples and those of negative examples are. The feature responses of a set of mixing positive examples do not reflect the true discriminative power of the feature, however.

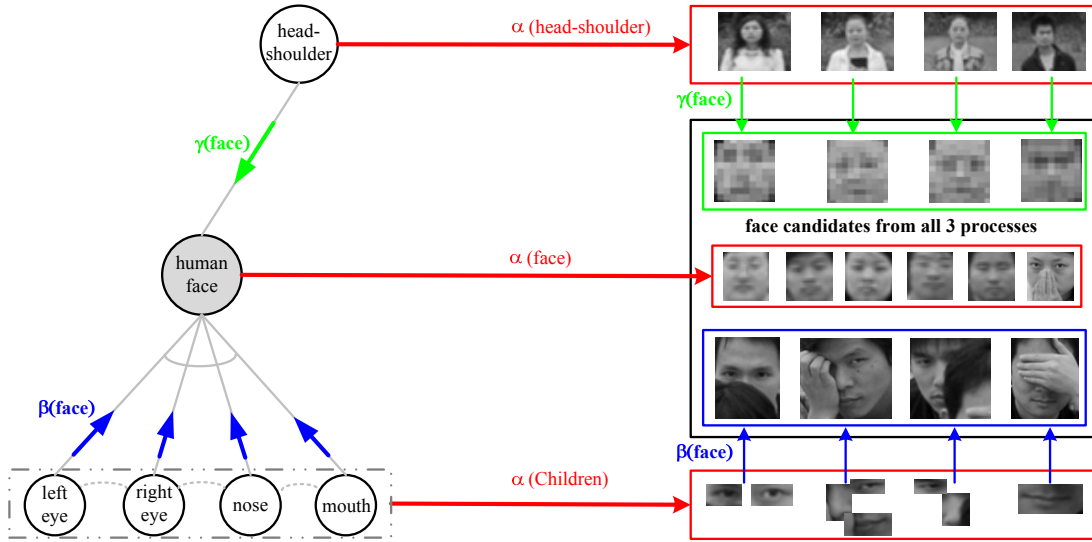


Figure 5.48: Illustration of integrating the $\alpha(\text{face})$, $\beta(\text{face})$ and $\gamma(\text{face})$ in the human face AoG for face detection. The three inference processes are effective in complementary ways relatively depending on the scale and occlusion conditions. The typical situations shown here are common to other object categories. Reproduced from Ref. [168] with permission of Springer, © 2011.

5.2.1.1 Problem Formulation

In this section, we introduce the AoG for object representation [185] and formulate object parsing using AoG under the Bayesian framework. Then, we derive the α , β and γ processes in the Bayesian formula.

5.2.1.2 The AOG Representation

If the AoG is specified by a quadruple,

$$\mathcal{G} = (V_N, V_T, E, \mathcal{P}) \quad (5.39)$$

$V_N = V_{and} \cup V_{or}$ is a set of nonterminal nodes with an And-node set V_{and} representing decompositions (shown by solid circles in Fig.5.49) and an Or-node set V_{or} representing alternative structures (shown by dash circles in Fig.5.49). A nonterminal node is denoted by capital letters, for example, $P, A, C_1 \in V_{and}$, $O \in V_{or}$.

V_T is a set of terminal nodes (shown by rectangles in Fig.5.49). In an AOG, each And-node can directly terminate to image data through a terminal node when it is at low resolution. In traditional hierarchical models, however, only leaf nodes can link to image data. A terminal node is denoted by lowercase t with the subscript letter of the corresponding nonterminal node, for example, $t_A, t_{C_1} \in V_T$.

$E = E_{or} \cup E_{dec} \cup E_t \cup E_{rel}$ is a set of edges including four types:

- (i) $E_{or} = \{ \langle O, A \rangle : O \in V_{or}, A \in V_{and} \}$ is a set of vertical switching edges which link Or-nodes to corresponding And-nodes as alternatives.

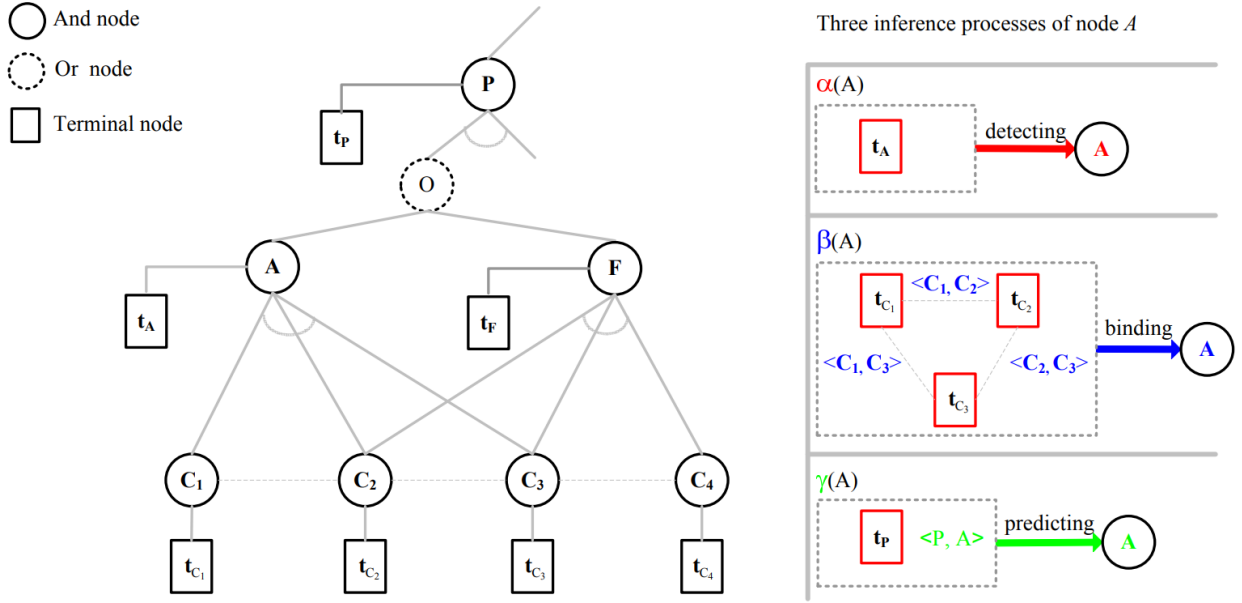


Figure 5.49: Illustration of the And-Or graph (AoG) representation. There are three types of nodes: And-nodes for decompositions, Or-nodes for alternative structures and terminal nodes for image data links. In an AoG, all And-nodes can directly terminate to image data through a terminate node when it is at low resolution. In traditional hierarchical models, only leaf nodes can link to image data. The α , β and γ processes are specified for each And-node A in an AoG as illustrated in the right panel. Reproduced from Ref. [168] with permission of Springer, © 2011.

- (ii) $E_{dec} = \{ \langle A, C \rangle : A \in V_{and}, C \in ch(A), ch(A) \neq \emptyset \}$ is a set of vertical decomposition edges which connect And-nodes to their child And-nodes. $ch(A) \subset V_{and}$ denotes the set of child nodes of node A .
- (iii) $E_t = \{ \langle A, t_A \rangle : A \in V_{and}, t_A \in V_T \}$ is a set of vertical terminating edges which connect And-nodes to their corresponding terminal nodes.
- (iv) $E_{rel} = \{ \langle A, F \rangle : A, F \in V_{and}, prt(A) \cap prt(F) \neq \emptyset \}$ is a set of horizontal relation edges which connect among And-nodes at the same layer, often pairwise. $prt(A), prt(F) \subset V_{and}$ denotes the sets of parent nodes of nodes A and F respectively.

\mathcal{P} is the probability defined over the space of all valid parse graphs which are defined below.

In an AoG \mathcal{G} , each Or-node $O \in V_{or}$ has a switching variable indicating the occurring frequency of its branches, denoted by $p(A|O)$ ($A \in V_N$). Both And-nodes $A \in V_{and}$ and terminal nodes $t \in V_T$ have a vector of attributes denoted by $X(A)$ and $x(t)$ respectively. For a subset $\mathbf{v} \subset V_{and}$, we denote $X(\mathbf{v})$ as the concatenation of attributes for And-nodes in the subset \mathbf{v} . The attributes include location, scale, orientation, etc. As illustrated in the right panel of Fig.5.49, the attributes of an And-node can be passed from attributes of other nodes in three ways: (i) the corresponding terminal node directly, (ii) child And-node(s) during binding process or (iii) parent node(s) during prediction process.

Parse graph. A parse graph, pg , is one instance of the AoG by selecting variables at the Or-nodes and specifying the attributes for And-nodes through the three ways stated above. We have,

$$pg = (V_N^{pg}, V_T^{pg}, E^{pg}, p(pg)) \quad (5.40)$$

where $V_N^{pg} = V_{and}^{pg} \cup V_{or}^{pg}$ ($V_N^{pg} \subset V_N$) is the nonterminal node set of the parse graph, $V_T^{pg} \subset V_T$ is the terminal node set and $E^{pg} = E_{or}^{pg} \cup E_{dec}^{pg} \cup E_t^{pg} \cup E_{rel}^{pg}$ ($E^{pg} \subset E$) is the edge set. We have a parse tree if we omit the horizontal relation edges $E_{rel}^{pg} \subset E^{pg}$ in a parse graph.

$p(pg)$ is the prior probability of parse graph pg , measuring the occurring probability of each switching edge $\langle O, A \rangle \in E_{or}^{pg}$ and the compatibility probabilities among the attributes of And-nodes (pairwise used in [168]) in V_{and}^{pg} with vertical decomposition edge $\langle P, A \rangle \in E_{dec}^{pg}$ and horizontal relation edge $\langle C_i, C_j \rangle \in E_{rel}^{pg}$. So, we have,

$$p(pg) = \frac{1}{Z} \exp\{-\mathcal{E}(pg)\} \quad (5.41)$$

where $Z = \sum_{pg} \exp\{-\mathcal{E}(pg)\}$ is the partition function and $\mathcal{E}(pg)$ is the total energy,

$$\begin{aligned} \mathcal{E}(pg) = & - \sum_{\langle O, A \rangle \in E_{or}^{pg}} \log p(A|O) \\ & - \sum_{\langle P, A \rangle \in E_{dec}^{pg}} \log p(X(A)|X(P)) \\ & - \sum_{\langle C_i, C_j \rangle \in E_{rel}^{pg}} \log p(X(C_i), X(C_j)) \end{aligned} \quad (5.42)$$

and $p(A|O)$ is the switching probability estimated by the occurring frequency in training data [185], and the second term $p(X(A)|X(P))$ captures the top-down prediction model, and $p(X(C_i), X(C_j))$ captures the compatibilities in the bottom-up binding model. They will be specified in the learning algorithm in Sec.5.2.2.

Given an input image I with domain defined on lattice Λ , the inference of AoG is to construct a parse graph for each object instance and its structure is not predefined but inferred on the fly.

Configuration. A configuration \mathcal{C} is the set of all terminal nodes in a valid parse graph pg , flattened in an image lattice.

$$\mathcal{C}(pg) = \{(t, x(t)) : t \in V_T^{pg}\} \quad (5.43)$$

The image data likelihood of a parse graph pg , $p(I|pg)$, is measured based on the terminal nodes in V_T^{pg} (since they link to image data). Further, if there was no occlusion between different terminal nodes (which is true for roughly rigid object categories such as the human face), we can factorize the likelihood as,

$$p(I|pg) = p(I|\mathcal{C}(pg)) = \prod_{t \in V_T^{pg}} p(I_{\Lambda_t}|t) \quad (5.44)$$

where $\Lambda_t \in \Lambda$ is the image domain occupied by the terminal node t .

In inference, we do not need to compute $p(I|pg)$ exactly, instead we measure the likelihood ratio between $p(I|pg)$ and a reference background model which is made implicitly in our derivation.

5.2.1.3 Bayesian Formulation of Object Parsing Using AOG

An AoG represents the object grammar of an object category. Given an input image I_Λ , it contains an unknown number K object instances at different scales. Some object instances may be occluded. Each object instance is represented by a parse graph pg_k ($k = 1, \dots, K$). For the human face parsing, Fig.5.46 shows a typical testing image and the left-bottom panel in Fig.5.50 shows a number of inferred parse trees of human face instances.

The goal of object parsing using AoG is to construct a parse graph for each object instance in I_Λ on the fly. We seek a world representation W for image I_Λ ,

$$W = (K, \{pg_k\}_{k=1}^K) \quad (5.45)$$

Under the Bayesian framework, we infer W by maximizing a posterior probability,

$$W^* = \arg \max_{W \in \Omega} p(W|I_\Lambda) = \arg \max_{W \in \Omega} p(W)p(I_\Lambda|W) \quad (5.46)$$

where Ω is the solution space.

The prior probability $p(W)$ is,

$$p(W) = p(K) \prod_{k=1}^K p(pg_k) \quad (5.47)$$

where $p(K)$ is the prior distribution for the number of object instances (for example, an exponential model $p(K) \propto \exp\{-\lambda_0 K\}$) and $p(pg_k)$ is the prior model of a parse graph already addressed in Eqn.5.41 in Sec.5.2.1.2.

The likelihood $p(I_\Lambda|W)$. Let Λ_{pg_k} be the image lattice occupied by the parse graph pg_k ($1 \leq k \leq K$). Denote $\Lambda_{fg} = \cup_{k=1}^K \Lambda_{pg_k}$ as the foreground lattice and $\Lambda_{bg} = \Lambda \setminus \Lambda_{fg}$ as the remaining background lattice. $I_\Lambda = (I_{\Lambda_{fg}}, I_{\Lambda_{bg}})$. Let $q(I)$ be the generic background model which will be made implicit in our derivation. The likelihood $p(I_\Lambda|W)$ is,

$$\begin{aligned} p(I_\Lambda|W) &= p(I_{\Lambda_{fg}}|W)q(I_{\Lambda_{bg}}) \\ &= \frac{p(I_{\Lambda_{fg}}|W)q(I_{\Lambda_{bg}})q(I_{\Lambda_{fg}})}{q(I_{\Lambda_{fg}})} \\ &= q(I_\Lambda) \frac{p(I_{\Lambda_{fg}}|W)}{q(I_{\Lambda_{fg}})} \\ &= q(I_\Lambda) \prod_{k=1}^K \frac{p(I_{\Lambda_{pg_k}}|pg_k)}{q(I_{\Lambda_{pg_k}})} \end{aligned} \quad (5.48)$$

where $p(I_{\Lambda_{pg_k}}|pg_k)$ means that the domain Λ_{pg_k} is explained away by the parse graph pg_k and conversely, $q(I_{\Lambda_{pg_k}})$ explains the domain Λ_{pg_k} as background. These models compete with each other to perform parsing.

So, Eqn.5.46 can be reproduced as,

$$\begin{aligned} W^* &= \arg \max_{W \in \Omega} p(K)q(I_\Lambda) \prod_{k=1}^K [p(pg_k) \frac{p(I_{\Lambda_{pg_k}}|pg_k)}{q(I_{\Lambda_{pg_k}})}] \\ &= \arg \max_{W \in \Omega} p(K) \prod_{k=1}^K [p(pg_k) \frac{p(I_{\Lambda_{pg_k}}|pg_k)}{q(I_{\Lambda_{pg_k}})}] \end{aligned} \quad (5.49)$$

5.2.1.4 Object Parsing in a Greedy Pursuit Manner

In the literature, there are several ways to infer W^* in Eqn.5.49, such as the data-driven Markov chain Monte Carlo (DDMCMC) method used in [155]. Here our goal is to pursue object instances appearing in an input image and construct corresponding parse graphs. Often, when the number of object instances K is typically not too large, we can adopt the best-first-search algorithm [26] directly to pursue parse graphs sequentially by maximizing Eqn.5.49 in a greedy manner. Our pursuit inference algorithm integrates the α β and γ processes and includes two aspects: (i) generating proposals (hypotheses) for possible parse graphs and (ii) verifying parse graph proposals in a greedy pursuit manner.

5.2.1.5 Connecting the α , β and γ processes with Bayesian inference

We pursue parse graphs sequentially based on Eqn.5.49 starting from an empty $W_0 = \emptyset$,

$$W_0 = \emptyset \rightarrow W_1 \rightarrow \dots \rightarrow W_k \rightarrow \dots \rightarrow W_K = W^*$$

At each step we pursue a parse graph and at the step k (≥ 1) of pursuit, let $\Lambda_k = \Lambda \setminus \cup_{i=1}^{k-1} \Lambda_{pg_i}$. We pursue the k -th parse graph pg_k by,

$$pg^* = \arg \max_{pg \in \Omega_{pg}} p(pg) p(I_{\Lambda_{pg}} | pg) \quad (5.50)$$

where Ω_{pg} is the proposal space of parse graphs and we omit k in pg_k hereafter in the derivation for simplicity when there is no confusion.

Similar to derive Eqn.5.48, we have,

$$p(I_{\Lambda_{pg}} | pg) = q(I_{\Lambda_k}) \frac{p(I_{\Lambda_{pg}} | pg)}{q(I_{\Lambda_{pg}})} \quad (5.51)$$

So, Eqn.5.50 can be rewritten as,

$$\begin{aligned} pg^* &= \arg \max_{pg \in \Omega_{pg}} p(pg) \frac{p(I_{\Lambda_{pg}} | pg)}{q(I_{\Lambda_{pg}})} \\ &= \arg \max_{pg \in \Omega_{pg}} [\log p(pg) + \log \frac{p(I_{\Lambda_{pg}} | pg)}{q(I_{\Lambda_{pg}})}] \end{aligned} \quad (5.52)$$

which is consistent with Eqn.5.49.

Recall that the prior probability $p(pg)$ is defined in Eqn.5.41 in general. For object categories with roughly rigid configuration such as the human face, we can assume that there are no occlusion among different nodes at the same layer in a parse graph so that we can factorize the likelihood ratio $\frac{p(I_{\Lambda_{pg}} | pg)}{q(I_{\Lambda_{pg}})}$ with respect to Eqn.5.44,

$$\log \frac{p(I_{\Lambda_{pg}} | pg)}{q(I_{\Lambda_{pg}})} = \sum_{t \in V_T^{pg}} \log \frac{p(I_{\Lambda_t} | t)}{q(I_{\Lambda_t})} \quad (5.53)$$

Without loss of generality, we consider the AoG illustrated in Fig.5.49. Node A represents the object of interest such as the human face. $V_{and} = \{P, A, C_1, C_2, C_3\}$. Further, we consider a parse graph pg in which $V_{and}^{pg} = \{P, A, C_1, C_2\}$ and $V_T^{pg} = \{t_P, t_A, t_{C_1}, t_{C_2}\}$. In terms of Eqn.5.41, we have,

$$\log p(pg) = \log p(A|O) + \log p(X(A)|X(P))$$

$$\begin{aligned}
& + \sum_{i=1}^2 \log p(X(C_i)|X(A)) \\
& + \log p(X(C_1), X(C_2)) + \log Z
\end{aligned} \tag{5.54}$$

By combining Eqn.5.53 and Eqn.5.54, Eqn.5.52 can be rewritten as,

$$\begin{aligned}
pg^* = \arg \max_{pg} \{ & \log p(A|O) + \sum_{i=1}^2 \log p(X(C_i)|X(A)) \\
& + \underbrace{\log \frac{p(I_{\Lambda_{t_A}}|t_A)}{q(I_{\Lambda_{t_A}})}}_{\alpha(A) \text{ process}} \\
& + \underbrace{\left[\sum_{i=1}^2 \log \frac{p(I_{\Lambda_{t_{C_i}}}|t_{C_i})}{q(I_{\Lambda_{t_{C_i}})}} + \log p(X(C_1), X(C_2)) \right]}_{\beta(A) \text{ process}} \\
& + \underbrace{\left[\log \frac{p(I_{\Lambda_{t_P}}|t_P)}{q(I_{\Lambda_{t_P}})} + \log p(X(A)|X(P)) \right]}_{\gamma(A) \text{ process}} \}
\end{aligned} \tag{5.55}$$

where $p(X(C_i)|X(A))$ is the prediction model for the child node C_i of node A . From Eqn.5.55, we can see that our pursuit algorithm integrates the α , β and γ processes explicitly.

Define w_A^α , $w_A^{\beta(c)}$ and $w_A^{\gamma(P)}$ as the weights computed from the α , β and γ processes respectively,

$$w_A^\alpha = \log \frac{p(I_{\Lambda_{t_A}}|t_A)}{q(I_{\Lambda_{t_A}})} \tag{5.56}$$

$$w_A^{\beta(c)} = \sum_{i=1}^2 \log \frac{p(I_{\Lambda_{t_{C_i}}}|t_{C_i})}{q(I_{\Lambda_{t_{C_i}})}} + \log p(X(C_1), X(C_2)) \tag{5.57}$$

$$w_A^{\gamma(P)} = \log \frac{p(I_{\Lambda_{t_P}}|t_P)}{q(I_{\Lambda_{t_P}})} + \log p(X(A)|X(P)) \tag{5.58}$$

which will be specified by the learning algorithm in Sec.5.2.2. Then, we can rewrite Eqn.5.55 as,

$$\begin{aligned}
pg^* = \arg \max_{pg \in \Omega_{pg}} \{ & \log p(A|O) + \sum_{i=1}^2 \log p(X(C_i)|X(A)) \\
& + \underbrace{w_A^\alpha}_{\alpha(A) \text{ process}} + \underbrace{w_A^{\beta(c)}}_{\beta(A) \text{ process}} + \underbrace{w_A^{\gamma(P)}}_{\gamma(A) \text{ process}} \}
\end{aligned} \tag{5.59}$$

Now, we can introduce the formal specifications of the α , β and γ processes by connecting the general identifications of the three processes in Sec.5.2.1 with the Bayesian inference in terms of Eqn.5.55.

I. The α process detects node A by applying a log-likelihood ratio test, $\log \frac{p(I_{\Lambda t_A} | t_A)}{q(I_{\Lambda t_A})}$ (Eqn.5.56), directly based on image features when node A is at middle resolution without occlusion. The α process can be viewed as either bottom-up (feature classifiers such as the Adaboost method) or top-down (template matching such as the active basis model) inference process. For each And-node $A \in V_{and}$, the α process, denoted by $\alpha(A; \theta)$, is instantiated by a corresponding terminal node $t_A \in V_T$, where θ is a set of parameters. For example, in Fig.5.49, we have,

$$\alpha(A; \theta) : t_A \rightarrow A \text{ and } x(t_A) \Rightarrow X(A) \quad (5.60)$$

where $t_A \rightarrow A$ is calculated by w_A^α in Eqn.5.56 and will be specified in Eqn.5.70, and $x(t_A) \Rightarrow X(A)$ is used to activate the γ processes of node A 's child nodes, $p(X(C_i) | X(A))$, in Eqn.5.55 and the β process of node A 's parent node, $p(X(P) | X(A))$.

II. The β process computes node A by applying a bottom-up binding test, for example, in Eqn.5.57 $\log p(X(C_1), X(C_2))$, of its child nodes $\mathbf{c} = (C_1, C_2)$ which have been detected in a given step based on the log-likelihood ratio tests of their own α processes, $w_{C_i}^\alpha = \log \frac{p(I_{\Lambda t_{C_i}} | t_{C_i})}{q(I_{\Lambda t_{C_i}})}$. The β process handles the situation in which node A is at high resolution but with occlusion (the occlusion disable the $\alpha(A)$ process). Let $V_{and}^{ch} \subset V_{and}$ be the set of And-nodes which have children. For each node $A \in V_{and}^{ch}$, the β process of node A can be defined, denoted by $\beta(A | \mathbf{c}; \phi)$ where $\mathbf{c} \subseteq ch(A)$ and ϕ is a set of parameters. Given different \mathbf{c} 's, we obtain different β processes for node A . Consider $\mathbf{c} = (C_1, C_2)$ in Fig.5.49, we have,

$$\begin{aligned} \beta(A | \mathbf{c}; \phi) : t_{C_1} &\rightarrow C_1 \text{ and } x(t_{C_1}) \Rightarrow X(C_1) \\ t_{C_2} &\rightarrow C_2 \text{ and } x(t_{C_2}) \Rightarrow X(C_2) \\ (C_1, C_2) &\rightarrow A \text{ and } (X(C_1), X(C_2)) \Rightarrow X(A) \end{aligned} \quad (5.61)$$

where $t_{C_i} \rightarrow C_i$ are calculated by $w_{C_i}^\alpha$ ($i = 1, 2$), and $(X(C_1), X(C_2)) \Rightarrow X(A)$ will activate the β binding process of node A . Then, we calculate $w_A^{\beta(\mathbf{c})}$ which will be specified by Eqn.5.83. The $\beta(A | \mathbf{c}; \phi)$ will, in turn, activate the γ processes of the other child nodes of node A and the β process of node A 's parent node. Actually, this procedure is activated recursively in testing.

III. The γ process computes node A by applying a top-down prediction test, $\log p(X(A) | X(P))$ (in Eqn.5.58), from its parent node P which has been already detected in a given step based on the log-likelihood ratio test of the α process of node P , $w_P^\alpha = \log \frac{p(I_{\Lambda t_P} | t_P)}{q(I_{\Lambda t_P})}$. The γ process handles the situation in which node A is under very low resolution so both $\alpha(A)$ and $\beta(A)$ are disabled. Let $V_{and}^{prt} \subset V_{and}$ be the set of And-nodes which have parent node(s). For each node $A \in V_{and}^{prt}$, we can define its γ process, denoted by $\gamma(A | P; \varphi)$ where $P \in prt(A)$ is a parent node and φ is a set of parameters. Similarly, in Fig.5.49 we have,

$$\begin{aligned} \gamma(A | P; \varphi) : t_P &\rightarrow P \text{ and } x(t_P) \Rightarrow X(P) \\ P &\rightarrow A \text{ and } X(P) \Rightarrow X(A) \end{aligned} \quad (5.62)$$

where similarly, $t_P \rightarrow P$ is calculated by w_P^α and $X(P) \Rightarrow X(A)$ will activate the γ process of node A . Then, we calculate $w_A^{\gamma(P)}$ which will be specified by Eqn.5.90, and then we run the inference process recursively.

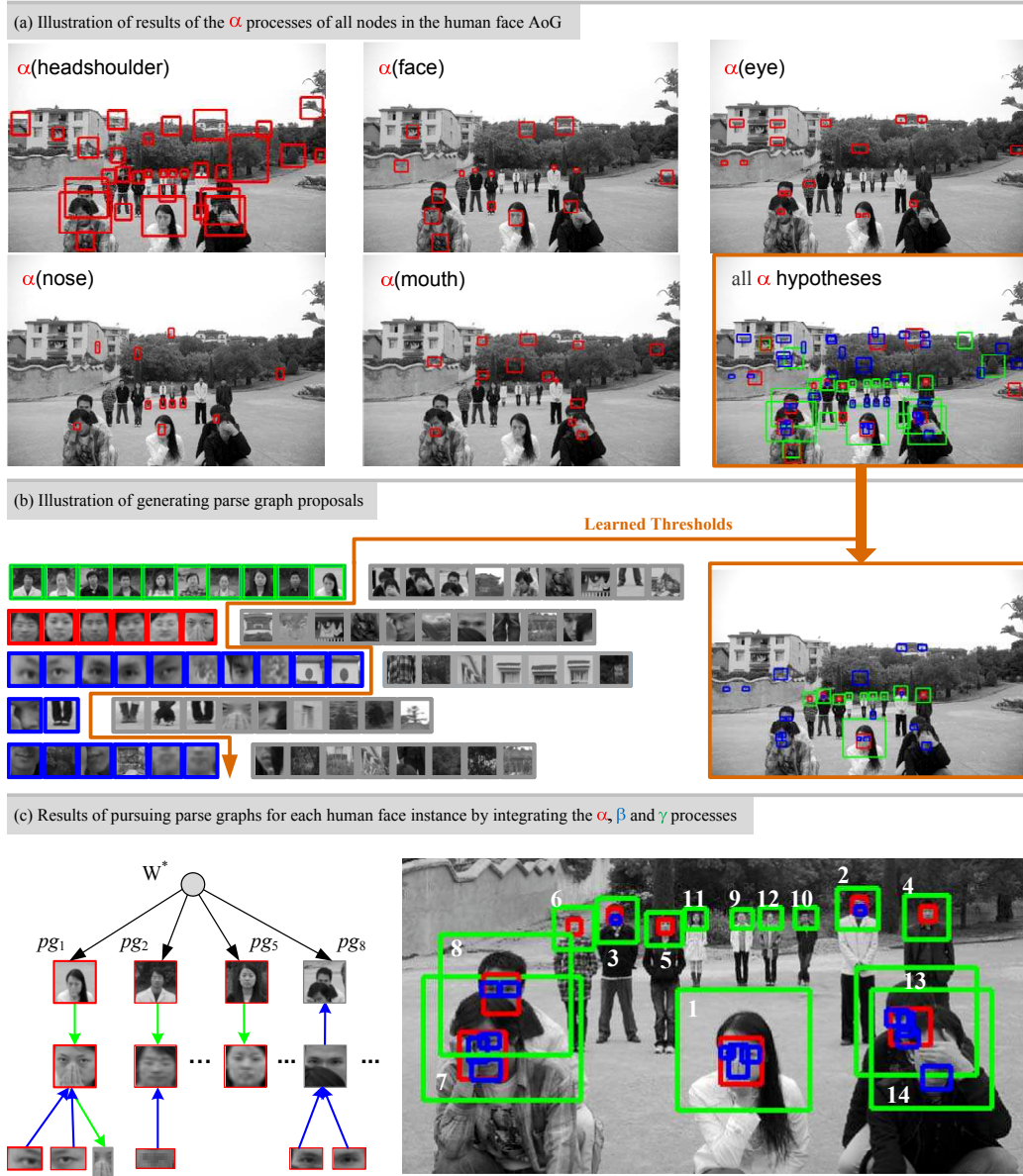


Figure 5.50: A running example of pursuing human faces and constructing corresponding parse graphs in a typical image by integrating the α , β and γ processes. The learning algorithms of the three processes are specified in Sec.5.2.2. In this figure, (a) shows the results of running all the α processes with a number of candidates of each node. (b) illustrates how we generate parse graph proposals. After applying the learned thresholds, we get promising candidates of each node and then run all the β and γ processes to propose possible parse graphs. (c) shows the results of pursuing object instances and constructing their parse graphs on the fly by integrating the α , β and γ processes in the proposed algorithm. In a greedy manner, we can get the pursuit indexes of all the object instances. For each object instance, we know the construction of the parse graph explicitly, which goes beyond only a bounding box for each detected object instance in traditional object detection (Best viewed in color). Reproduced from Ref. [168] with permission of Springer, © 2011.

Object parsing by integrating the α , β and γ processes

Input: an image I_Λ and an AoG \mathcal{G} .

Output: parse graphs pg_i ($i = 1, \dots, K$).

1. **α map generation:** $I_{\alpha(U)}, \forall U \in V_{and}$
run $\alpha(U; \theta_U)$ and compute the weight w_U^α .
2. **α hypotheses generation:** open list $OP(U)$ from $I_{\alpha(U)}$
apply thresholds $Th(w_U^\alpha)$ and local inhibitions.
3. **β bindings and merging.**
 - (1) run $\beta(A|c; \phi)$ and compute the weight $w_A^{\beta(c)}$
 - (2) apply $Th(w_A^{\beta(c)})$ to generate $\beta(A)$ hypotheses
and insert them into $OP(A)$ decreasingly;
 - (3) merge with compatible $\alpha(A)$ hypotheses
and compute weight $w_A^\alpha + w_A^{\beta(c)}$.
4. **γ predictions and merging.**
 - (1) run $\gamma(A|P; \varphi)$ and compute the weight $w_A^{\gamma(P)}$
 - (2) apply $Th(w_A^{\gamma(P)})$ to generate $\gamma(A)$ hypotheses
and insert them into $OP(A)$ decreasingly;
 - (3) merge with compatible $\alpha(A)$, $(\alpha + \beta)(A)$ hypotheses
and compute weight $w_A^\alpha + w_A^{\gamma(P)}$
or $w_A^\alpha + w_A^{\beta(c)} + w_A^{\gamma(P)}$.
5. **Object pursuing and parsing.**
In $OP(A)$, pursue node A according to w_A ,
construct parse graphs by retrieving all the α hypotheses.
Stop pursuing based on $Th(w_A)$.

Figure 5.51: The greedy pursuit algorithm for object parsing using AoG by integrating the α , β and γ processes. Reproduced from Ref. [168] with permission of Springer, © 2011.

5.2.1.6 The Algorithm

Our greedy pursuit algorithm is straightforward based on Eqn.5.55. Fig.5.50 shows a running example of human face parsing by the proposed algorithm. On the whole, the algorithm first runs all α processes (see the top panel in Fig.5.50) and applies thresholds to obtain candidates for each node. Then, to pursue object instances of node A , the algorithm recursively runs all β processes and γ processes to do bottom-up binding and top-down prediction and then generate the parse graph proposals (see the middle panel in Fig.5.50). The

pursuit is based on the pursuit index in Eqn.5.63. The parse graph for each pursued instance of node A is constructed by retrieving all the used α candidates (see the bottom panel in Fig.5.50). We summarize the algorithm in Fig.5.51.

Proposal Generation Ω_{pg} . In testing, in order to find the object instances which appear at different locations, scales and orientations, we need to search these three spaces to find possible proposals. For searching the scale space, we create the testing image pyramid for I_Λ by a certain down-sampling factor (we use 0.9 in our experiments) until the image size is smaller than the minimum of sizes of the learned active basis templates. We have the pyramid with L layers $\{I_{\Lambda_0} = I_\Lambda, I_{\Lambda_1}, \dots, I_{\Lambda_L}\}$. For different orientations, we rotate the learned active basis templates. We use the sliding window method to search all locations in the testing pyramid for all active basis templates. That is, we run all the α processes first and by applying the thresholds we get a list of candidates for each node in the AoG (which could be empty). Based on the list, we can generate parse graph proposals Ω_{pg} . Since we used fixed relative scales in training the $\alpha(A)$, $\beta(A)$ and $\gamma(A)$ processes (as stated in Sec.5.2.2.2), for example, given a hypothesis of node A (such as the human face), we roughly know the locations, scales and orientations of other nodes in terms of the learned β binding models $p(X(C_i)|X(A))$ and γ prediction models $p(X(A)|X(P))$. Then, we put all proposals in an open list which could be complete or partial parse graphs. Further, different parse graphs could also overlap to compete to explain the corresponding image domain. The open list would be explored to do the proposal verification and the ambiguity is solved based on the pursue index addressed below.

Pursuit Index. For each proposal, we compute its total weight,

$$w_A = w_A^\alpha + w_A^{\beta(c)} + w_A^{\gamma(P)} \quad (5.63)$$

which is the pursuit index we are seeking for proposal verification. Similarly, we can estimate the threshold $\text{Th}(w_A)$ in an evaluation dataset.

Performance Comparisons. According to Eqn.5.63, we can explicitly compare the performance of different integrations for each node A , for example, by plotting ROCs based on w_A^α , $w_A^\alpha + w_A^{\beta(c)}$, $w_A^\alpha + w_A^{\gamma(P)}$ and w_A respectively. Some comparison results are shown in Fig.5.62 for junctions and rectangles and Fig.5.63 for human faces. From these ROCs, we can see that how and how much the integration improves performance.

5.2.2 Learning the α , β and γ Processes

In this section, we introduce the learning algorithm under the MLE framework for the α , β and γ processes to specify w_A^α in Eqn.5.56, $w_A^{\beta(c)}$ in Eqn.5.57 and $w_A^{\gamma(P)}$ in Eqn.5.58 respectively. We train the three processes separately due to the observation that the effectiveness of the three processes depends on the scale and occlusion conditions as illustrated in Fig.5.46 and Fig.5.48, and for the purpose of evaluating the information contribution of each process individually. To that end, we propose an isolation method to block one process from the other two processes. The isolation is based on the manually labeled parse graphs which are available in the LHI image database [176].

5.2.2.1 Isolating the α , β and γ Processes

Scale and occlusion are the two main causes entailing the α , β and γ processes. So, each of the three processes of node A can be blocked through *scaling* and/or *masking* image patches of node A in terms

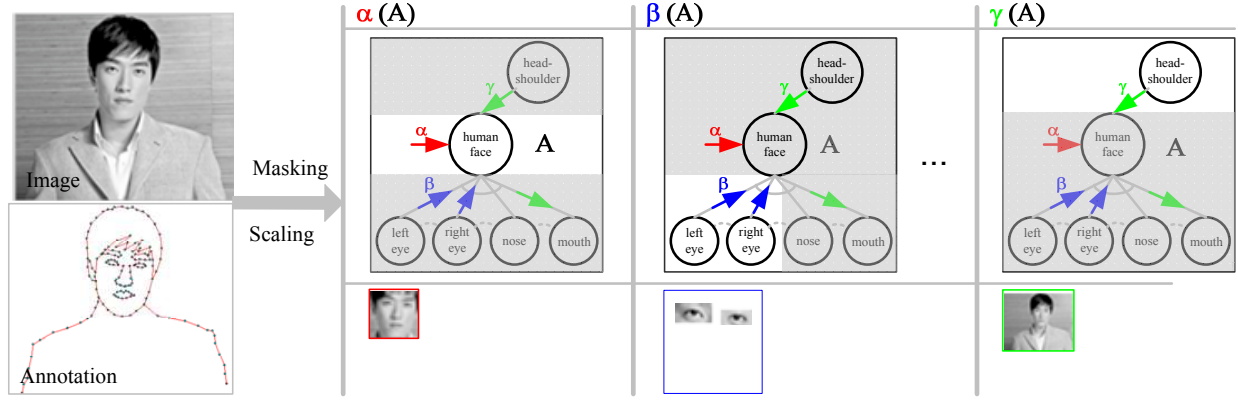


Figure 5.52: Illustration of isolating the α , β and γ processes of node A in an AoG. Consider the human face example in this figure. The isolation is performed by *scaling* and *masking* the image patch (the left-top panel) in terms of its annotation (the left-bottom panel). The annotation used in this section is the manually labelled parse graph. Details are specified in Sec.5.2.2.1. Based on the isolation, we generate training and evaluation data for each process in Sec.5.2.2.2. Reproduced from Ref. [168] with permission of Springer, © 2011.

of the labeled parse graphs. Then, we isolate one process by blocking the other two processes. Fig.5.52 illustrates the procedures with an example for human face (node A).

I. Isolating the $\alpha(A)$ process. Blocking both the $\beta(A)$ and $\gamma(A)$ processes will isolate the α process. First, we crop only the compact image patches of node A out of its context in terms of the annotations. Then, the image patches are down-sampled up to a certain scale at which the parts can not be recognized if cropped in isolation.

II. Isolating the $\beta(A)$ process. We have different $\beta(A)$ processes depending on the given subset of node A 's child nodes, $c \subseteq ch(A)$. Then, to isolate the $\beta(A|c)$ process is to block both the $\alpha(A)$ and $\gamma(A)$ processes meanwhile keeping the α processes of child nodes in c on. We first crop only the compact image patches of node A but just keep those patches whose resolutions are above a predefined value. Then, we scale all the image patches to the same size (also above the predefined value) and mask those portions of all the image patches with respect to the child node(s) not in c .

III. Isolating the $\gamma(A)$ process. We may have different γ processes given different parent nodes P . To isolate the $\gamma(A|P)$ process is to block the $\alpha(A)$ and $\beta(A)$ while keeping the $\alpha(P)$ process on. So, it is equivalent to isolate the $\alpha(P)$ process. First, we crop the compact image patches of the parent node P . Then, we down-sample the image patches up to a certain scale at which the node A itself can not be recognized if cropped in isolation.

By changing the testing image dataset of an object category with these isolating methods, we can cause most of the existing object detection or recognition methods to fail. To achieve robust performance, we train each process separately first and then integrate them explicitly (see Eqn.5.55) for our numerical study.

Next, we generate training and evaluation data for each process in terms of the isolating procedures.

5.2.2.2 Training Data for the α , β and γ Processes

Suppose we have a set of m positive images for an object of interest, $D^+ = \{(I_1, pg_1), \dots, (I_m, pg_m)\}$, where pg_i is the annotated parse graph for image I_i . Based on the parse graph, we can generate training and testing datasets for the three processes. For simplicity of notations, we assume that each node of interest

appears in each image I_i with good resolution.

I. The α process training dataset. Let $D_\alpha^+(A)$ denote the positive training dataset for the $\alpha(A)$ process of node A . Through the isolation method of the α process, for each $I_i \in D^+$, we obtain the α image patch of node A , denoted by $I_i^{(A)}$. So, we have,

$$D_\alpha^+(A) = \{I_i^{(A)} : i = 1, 2, \dots, m\}$$

II. The β process training dataset. Let $D_\beta^+(A|\mathbf{c})$ denote the positive training dataset for the $\beta(A|\mathbf{c})$ process. Through the isolation method of the β process, for each $I_i \in D^+$, we obtain the β image patch of node A given child node(s) in \mathbf{c} , denoted by $I_i^{(\mathbf{c})}$. Then, we get,

$$D_\beta^+(A|\mathbf{c}) = \{(I_i^{(\mathbf{c})}, X(\mathbf{c}|I_i^{(\mathbf{c})})) : i = 1, 2, \dots, m\}$$

where $X(\mathbf{c}|I_i^{(\mathbf{c})}) = \{X(C_j) : C_j \in \mathbf{c}, j = 1, \dots, |\mathbf{c}|\}$ is the concatenation of attributes for child node(s) in \mathbf{c} measured in $I_i^{(\mathbf{c})}$, which will be used to learn the bottom-up β binding model $p(X(C_j), X(C_k))$ for node A given child nodes in \mathbf{c} ($j \neq k, j, k = 1, \dots, |\mathbf{c}|$).

III. The γ process training dataset. Let $D_\gamma^+(A|P)$ denote the positive training dataset for the $\gamma(A|P)$ process. Through the isolation method of the γ process, for each $I_i \in D^+$, we can get the γ image patch of node A given the parent node P , denoted by $I_i^{(P)}$. So, we have,

$$D_\gamma^+(A|P) = \{(I_i^{(P)}, X(A|I_i^{(P)})) : i = 1, 2, \dots, m\}$$

where $X(A|I_i^{(P)})$ is the attributes of node A measured in $I_i^{(P)}$, which will be used to learn the top-down γ prediction model $p(X(A)|X(P))$ of node A given the parent node P .

When node P only has one child node A , we can transform the γ prediction model of node A given the parent node P $p(X(A)|X(P))$ into the β binding model of node P given the child node A equivalently.

In the same way, we can get the attributes $X(C_j|I_i^{(A)})$ to learn top-down γ prediction model for child nodes C_j 's of node A , $p(X(C_j)|X(A))$, where $C_j \in ch(A)$, $I_i^{(A)} \in D_\alpha^+(A)$.

Correspondingly, we collect negative datasets $D_\alpha^-(A)$, $D_\beta^-(A|\mathbf{c})$ and $D_\gamma^-(A|P)$ by randomly cropping image patches from generic background images.

Scale specifications in experiments. In the experiments for evaluating the information contributions in Sec.5.3.1, we prepare the data for multiple scales to observe how the information contributions change with scales. In the experiments for object parsing by integrating the α , β and γ processes of node A in Sec.5.3.2, we use fixed relative scales for the three processes. Consider the scales of the compact image patches of node A in these three processes, denoted by $s_{\alpha(A)}$, $s_{\beta(A)}$ and $s_{\gamma(A)}$ respectively. We set $s_{\alpha(A)} = b \times s_{\gamma(A)} = \frac{1}{b} \times s_{\beta(A)}$ ($b = 2$ used in [168]).

Given the data, we specify the training procedure under the MLE framework in the next section.

5.2.2.3 Learning the α process

Learning the α process involves selecting a modeling scheme for $\alpha(A; \theta)$ and estimating the parameters θ by maximizing the data likelihood on $D_\alpha^+(A)$. For example, in discriminative boosting methods, θ_A is the

learned strong classifier which consists of a set of boosted weak classifiers and the corresponding weights [158], and in generative model-based methods such as the active basis model, θ is the set of parameters specifying the learned deformable template [170]. Given $D_\alpha^+(A)$, we have,

$$\begin{aligned}\alpha(A; \theta^*) &= \arg \max_{\theta} p(D_\alpha^+(A) | A; \theta) \\ &= \arg \max_{\theta} \sum_{i=1}^m \log p(I_i^{(A)} | A; \theta)\end{aligned}\quad (5.64)$$

Solving $\alpha(A; \theta^*)$ depends on choosing a specific modeling scheme for $p(I | A; \theta)$. We use the active basis model which is briefly introduced here for the contents to be self-contained.

Active basis model. The active basis model is a deformable model which consists of a small number of Gabor wavelet elements (as visual primitives for modeling object category) at selected locations and orientations. These Gabor wavelet elements can slightly perturb their locations and orientations before they are linearly combined to generate the observed image. Let Λ be the domain of the image patch I and $\{B_{x,y,s,o}\}$ the dictionary of Gabor wavelet elements. The (x, y, s, o) are densely sampled: $(x, y) \in \Lambda$, s is a fixed size (often about 1/10 of the length of Λ) and $o \in \{i\pi/N, i = 0, \dots, N-1\}$ (e.g. $N = 15$). The dictionary forms an over-complete dictionary for modeling I_Λ . Then we obtain the sparse coding scheme $I = \sum_{i=1}^n a_i B_i + U$ where n is the number of selected bases, $B_i = B_{x_i, y_i, s, 0_i}$, a_i 's are the coefficients and U is the unexplained residual image. In the matrix form, we have $I = B\mathbf{a} + U$ (where $B = (B_1, \dots, B_n)$ and $\mathbf{a} = (a_1, \dots, a_n)'$). In terms of linear decomposition, we know that U resides in the remaining subspace orthogonal to B and we can write $U = \bar{B}\bar{\mathbf{a}}$ (where columns in \bar{B} are orthogonal to columns in B and both \bar{B} and $\bar{\mathbf{a}}$ would be made implicit in the active basis model). So, we have $I = B\mathbf{a} + \bar{B}\bar{\mathbf{a}}$. Then, we can specify the distribution of I given B as

$$p(I|B) = p(\mathbf{a}, \bar{\mathbf{a}}) \det(J) = p(\mathbf{a}) p(\bar{\mathbf{a}}|\mathbf{a}) \det(J) \quad (5.65)$$

where J is the Jacobi matrix of the linear transform from I to $(\mathbf{a}, \bar{\mathbf{a}})$ and $\det(J)$ the determinant of J .

On the other hand, let $q(I)$ be a reference distribution (which has a few choices discussed in [170]), and similarly, we can have

$$q(I) = q(\mathbf{a}, \bar{\mathbf{a}}) \det(J) = q(\mathbf{a}) q(\bar{\mathbf{a}}|\mathbf{a}) \det(J) \quad (5.66)$$

In the active basis model, we want to construct $p(I|B)$ by modifying $q(I)$ and assume $q(\bar{\mathbf{a}}|\mathbf{a}) = p(\bar{\mathbf{a}}|\mathbf{a})$, so we have

$$p(I|B) = q(I) \frac{p(\mathbf{a})}{q(\mathbf{a})} = q(I) \frac{p(a_1, \dots, a_n)}{q(a_1, \dots, a_n)} \quad (5.67)$$

Further, by applying the local inhibition principle, we can treat the selected Gabor wavelet elements independently, that is,

$$p(I|B) = q(I) \prod_{i=1}^n \frac{p(a_i)}{q(a_i)} \quad (5.68)$$

where $p(a_i)$ is parameterized as an exponential family model $p(a_i; \lambda_i) = \frac{1}{Z(\lambda_i)} \exp\{\lambda_i h(r_i)\} q(a_i)$ ($r_i = |< I, B_i >|^2$ is the local energy of Gabor filter response and $h(r_i) = \text{sigmoid}(r_i) = \zeta[\frac{2}{1+e^{-2r_i/\zeta}} - 1]$ is a sigmoid transformation function with ζ being the saturation level such as $\zeta = 6$), and $q(a_i)$ is pooled from generic background images at an off-line stage. The resulting model is

$$p(I|B) = q(I) \prod_{i=1}^n \frac{1}{Z(\lambda_i)} \exp\{\lambda_i h(r_i)\} \quad (5.69)$$

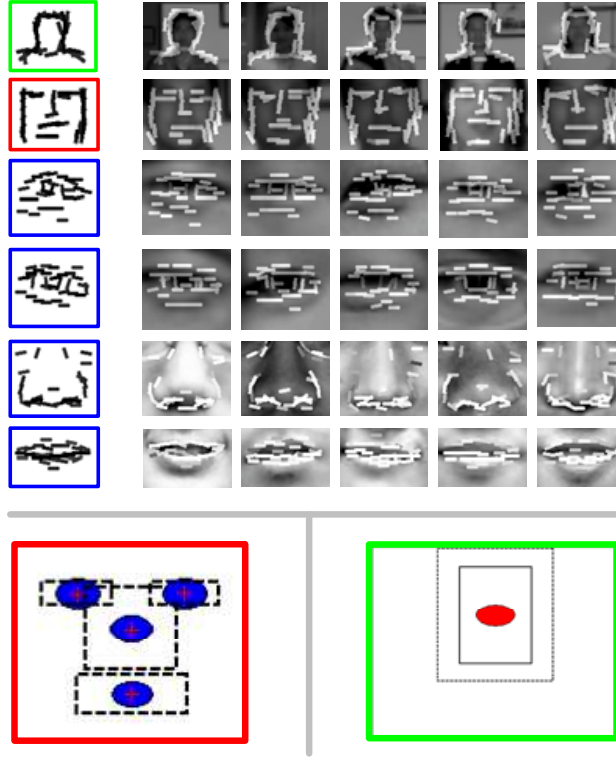


Figure 5.53: Illustration of learned α , β binding and γ prediction models for human face. The top panel shows the learned active basis model for the α process of each terminal node. The left-bottom panel illustrates the binding model for the β process in which the outside red box is the bounding box of face and the inside dash boxes are for the parts and the ellipses represent the location following a Gaussian distribution. The right-bottom panel shows the prediction model for the γ process in which the outside green box is the bounding box of head-shoulder and the inside solid and dash boxes represent the changeable size of the bounding box of face and the red ellipse represent the location of face following a Gaussian distribution. Reproduced from Ref. [168] with permission of Springer, © 2011.

In testing, the matching score (the α weight of a hypothesis of node A) is the log-likelihood ratio,

$$w_A^\alpha = \log \frac{p(I|B)}{q(I)} = \sum_{i=1}^n [\lambda_i h(r_i) - \log Z(\lambda_i)] \quad (5.70)$$

Active basis model can be also used to learn mixed image template modeling both shape and texture [138, 170].

The threshold $\text{Th}(w_A^\alpha)$ of the α process $\alpha(A; \theta)$ can be estimated in a validation α dataset.

The top panel of Fig.5.53 shows the learned active basis for each terminal node of a face AoG.

5.2.2.4 Learning the β process

Learning the β process involves specifying $\beta(A|\mathbf{c}; \phi)$ and estimating the parameters ϕ by maximizing the data likelihood on $D_\beta^+(A|\mathbf{c})$. $\beta(A|\mathbf{c}; \phi)$ composes (or binds) a (complete or partial) set of child nodes in \mathbf{c} , to generate hypotheses of node A . In the literature, component-based [8, 64], fragment-based [157] and

other part-based methods [2, 167] can be treated as this kind of process. Given $D_\beta^+(A|\mathbf{c})$, we obtain,

$$\begin{aligned}\beta(A|\mathbf{c}; \phi^*) &= \arg \max_{\phi} p(D_\beta^+(A|\mathbf{c})|\mathbf{c}; \phi) \\ &= \arg \max_{\phi} \sum_{i=1}^m \log p(I_i^{(\mathbf{c})}, X(\mathbf{c}|I_i^{(\mathbf{c})})|\mathbf{c}; \phi)\end{aligned}\quad (5.71)$$

The $\beta(A|\mathbf{c}; \phi)$ process includes two components, one is the α process of each child node in \mathbf{c} with parameters $\theta_{\mathbf{c}}$ and the other is the binding model for the given children \mathbf{c} with parameters Δ . So, $\phi = (\theta_{\mathbf{c}}, \Delta)$ and we have,

$$\begin{aligned}p(I^{(\mathbf{c})}, X(\mathbf{c}|I^{(\mathbf{c})})|\mathbf{c}; \phi) &= p(I^{(\mathbf{c})}|\mathbf{c}; \theta_{\mathbf{c}}) \times p(X(\mathbf{c}|I^{(\mathbf{c})}); \Delta)\end{aligned}\quad (5.72)$$

where for notation simplicity we use $I^{(\mathbf{c})}$ to represent $I_i^{(\mathbf{c})}$ generally.

We consider three types of attributes for binding, that is, the location $L_{\mathbf{c}}$, scale $S_{\mathbf{c}}$ and orientation $O_{\mathbf{c}}$ respectively. So, $\Delta = (\Delta_L, \Delta_S, \Delta_O)$. And, we model them in a pairwise manner. Consider $\mathbf{c} = (C_i, C_j)$ ($C_i, C_j \in ch(A)$), we have,

$$X(\mathbf{c}|I^{(\mathbf{c})}) = \{X(C_i), X(C_j)\} = (L_{\mathbf{c}}, S_{\mathbf{c}}, O_{\mathbf{c}}|I^{(\mathbf{c})})$$

and

$$L_{\mathbf{c}} = (L_{C_i}, L_{C_j}); S_{\mathbf{c}} = (S_{C_i}, S_{C_j}); O_{\mathbf{c}} = (O_{C_i}, O_{C_j})$$

For the α processes of child nodes in \mathbf{c} , we have,

$$\begin{aligned}\log p(I^{(\mathbf{c})}|\mathbf{c}) &= \log p(I^{(C_i, C_j)}|C_i, C_j) \\ &= \log p(I^{(C_i)}|C_i; \theta_{C_i}) + \log p(I^{(C_j)}|C_j; \theta_{C_j}) \\ &= \alpha(C_i; \theta_{C_i}) + \alpha(C_j; \theta_{C_j})\end{aligned}\quad (5.73)$$

For binding child nodes in \mathbf{c} , we obtain,

$$\begin{aligned}p(X(\mathbf{c}|I^{(\mathbf{c})}); \Delta) &= P(X(C_i), X(C_j); \Delta) \\ &= p(L_{\mathbf{c}}, S_{\mathbf{c}}, O_{\mathbf{c}}|I^{(\mathbf{c})}; \Delta) \\ &= p(L_{\mathbf{c}}|I^{(\mathbf{c})}; \Delta_L) \times p(S_{\mathbf{c}}|I^{(\mathbf{c})}; \Delta_S) \times p(O_{\mathbf{c}}|I^{(\mathbf{c})}; \Delta_O)\end{aligned}\quad (5.74)$$

Solving $\beta(A|\mathbf{c}; \phi^*)$ depends on choosing a specific modeling scheme for $p(L_{\mathbf{c}}|I^{(\mathbf{c})}; \Delta_L)$, $p(S_{\mathbf{c}}|I^{(\mathbf{c})}; \Delta_S)$ and $p(O_{\mathbf{c}}|I^{(\mathbf{c})}; \Delta_O)$. The three terms are modeled as Gaussian distributions in their respective transformed spaces [40].

Thus, each pairwise binding $\mathbf{c} = (C_i, C_j)$ is characterized by the expected relative location $\mu_{L_{\mathbf{c}}}$, scale $\mu_{S_{\mathbf{c}}}$ and orientation $\mu_{O_{\mathbf{c}}}$ and corresponding full covariance matrices $\Sigma_{L_{\mathbf{c}}}$, $\Sigma_{S_{\mathbf{c}}}$ and $\Sigma_{O_{\mathbf{c}}}$. So, we have $\Delta_L = (\mu_{L_{\mathbf{c}}}, \Sigma_{L_{\mathbf{c}}})$, $\Delta_S = (\mu_{S_{\mathbf{c}}}, \Sigma_{S_{\mathbf{c}}})$ and $\Delta_O = (\mu_{O_{\mathbf{c}}}, \Sigma_{O_{\mathbf{c}}})$ which can be estimated from the dataset $D_\beta^+(A|\mathbf{c})$. Then, we have,

$$\begin{aligned}p(L_{\mathbf{c}}|I^{(\mathbf{c})}; \Delta_L) &= p(L_{C_i}, L_{C_j}; \mu_{L_{\mathbf{c}}}, \Sigma_{L_{\mathbf{c}}}) \\ &= \mathcal{N}(L_{C_i} - L_{C_j}; \mu_{L_{\mathbf{c}}}, \Sigma_{L_{\mathbf{c}}})\end{aligned}\quad (5.75)$$

$$\begin{aligned}
p(S_{\mathbf{c}}|I^{(\mathbf{c})}; \triangle_S) &= p(S_{C_i}, L_{C_j}; \mu_{S_{\mathbf{c}}}, \Sigma_{S_{\mathbf{c}}}) \\
&= \mathcal{N}(S_{C_i} - S_{C_j}; \mu_{S_{\mathbf{c}}}, \Sigma_{S_{\mathbf{c}}})
\end{aligned} \tag{5.76}$$

and

$$\begin{aligned}
p(O_{\mathbf{c}}|I^{(\mathbf{c})}; \triangle_O) &= p(O_{C_i}, O_{C_j}; \mu_{O_{\mathbf{c}}}, \Sigma_{O_{\mathbf{c}}}) \\
&= \mathcal{N}(O_{C_i} - O_{C_j}; \mu_{O_{\mathbf{c}}}, \Sigma_{O_{\mathbf{c}}})
\end{aligned} \tag{5.77}$$

Further, we specify the three Gaussian distribution above in a transformed space to have zero means and diagonal covariances. To that end, we first compute the singular value decompositions of the three covariance matrices and then define the transformations. For example, for the location (the same is for the scale and orientation), we have,

$$\Sigma_{L_{\mathbf{c}}} = U_{L_{\mathbf{c}}} D_{L_{\mathbf{c}}} U_{L_{\mathbf{c}}}^T \tag{5.78}$$

$$T_{ij}(L_{C_i}) = U_{L_{\mathbf{c}}}^T (L_{C_i} - \mu_{L_{\mathbf{c}}}) \tag{5.79}$$

$$T_{ji}(L_{C_j}) = U_{L_{\mathbf{c}}}^T (L_{C_j}) \tag{5.80}$$

So, we can rewrite Eqn.5.75 as

$$\begin{aligned}
p(L_{\mathbf{c}}|I^{(\mathbf{c})}; \triangle_L) &= \mathcal{N}(L_{C_i} - L_{C_j}; \mu_{L_{\mathbf{c}}}, \Sigma_{L_{\mathbf{c}}}) \\
&= \mathcal{N}(T_{ij}(L_{C_i}) - T_{ji}(L_{C_j}); 0, D_{L_{\mathbf{c}}})
\end{aligned} \tag{5.81}$$

So, we calculate the binding score of nodes in \mathbf{c} as,

$$w_{\mathbf{c}}^{\text{bind}} = \log p(X(\mathbf{c}|I^{(\mathbf{c})}); \triangle) \tag{5.82}$$

Then, by combining Eqn.5.74 and Eqn.5.82, we obtain the weight of a β hypothesis of node A given the children \mathbf{c} ,

$$\begin{aligned}
w_A^{\beta(\mathbf{c})} &= \log p(I^{(\mathbf{c})}, X(\mathbf{c}|I^{(\mathbf{c})})|\mathbf{c}; \phi) \\
&= \log p(I^{(\mathbf{c})}|\mathbf{c}) + \log p(X(\mathbf{c}|I^{(\mathbf{c})}); \triangle) \\
&= w_{\mathbf{c}}^{\text{bind}} + \sum_{C_i \in \mathbf{c}} w_{C_i}^{\alpha}
\end{aligned} \tag{5.83}$$

The threshold $\text{Th}(w_A^{\beta(\mathbf{c})})$ of the β process $\beta(A|\mathbf{c}; \phi)$ can be estimated in a validation β dataset.

5.2.2.5 Learning the γ process

Learning the γ process involves specifying $\gamma(A|P, \varphi)$ and estimating the parameters φ by maximizing the data likelihood on $D_{\gamma}^+(A|P)$. $\gamma(A|P; \varphi)$ predicts hypothesis of node A from the α process of its parent node P . In the literature, context-based methods [66, 148] can be looked as γ processes. Given $D_{\gamma}^+(A|P)$, we have,

$$\begin{aligned}
\gamma(A|P; \varphi^*) &= \arg \max_{\varphi} p(D_{\gamma}^+(A|P)|P; \varphi) \\
&= \arg \max_{\varphi} \sum_{i=1}^m \log p(I_i^{(P)}, X(A|I_i^{(P)})|P; \varphi)
\end{aligned} \tag{5.84}$$

Also, $\gamma(A|P; \varphi)$ consists of two components, one is the α process of the parent node P with the parameters θ_P and the other is the predicting model from parent node P to node A itself with parameters ∇ . So, we have $\varphi = (\theta_P, \nabla)$ and obtain,

$$\begin{aligned} p(I^{(P)}, X(A|I^{(P)})|P; \varphi) \\ = p(I^{(P)}|P; \theta_P) \times p(X(A|I^{(P)}); \nabla) \end{aligned} \quad (5.85)$$

where we also use $I^{(P)}$ to represent $I_i^{(P)}$ in general.

In the γ process, we want to predict the location L_A , scale S_A and orientation O_A of node A from the parent node P . So, we have $\nabla = (\nabla_L, \nabla_S, \nabla_O)$ and

$$X(A|I^{(P)}) = X(A)|X(P) = (L_A, S_A, O_A|I^{(P)})$$

For the α process of the parent node P , we have,

$$\log p(I^{(P)}|P; \theta_P) = \alpha(P; \theta_P) \quad (5.86)$$

In order to predict a hypothesis of node A , we have,

$$\begin{aligned} p(X(A|I^{(P)}); \nabla) &= p(X(A)|X(P); \nabla) \\ &= p(L_A, S_A, O_A|I^{(P)}; \nabla) \\ &= p(L_A|I^{(P)}; \nabla_L) \times p(S_A|I^{(P)}; \nabla_S) \times p(O_A|I^{(P)}; \nabla_O) \end{aligned} \quad (5.87)$$

Then, solving $\gamma(A|P, \varphi^*)$ depends on how we model $p(L_A|I^{(P)}; \nabla_L)$, $p(S_A|I^{(P)}; \nabla_S)$ and $p(O_A|I^{(P)}; \nabla_O)$. The three terms are also treated as Gaussian distribution. So, $\nabla_L = (\mu_{L_A}, \Sigma_{L_A})$, $\nabla_S = (\mu_{S_A}, \Sigma_{S_A})$ and $\nabla_O = (\mu_{O_A}, \Sigma_{O_A})$. For example, we have,

$$\begin{aligned} p(L_A|I^{(P)}; \nabla_L) \\ &= p(L_A|I^{(P)}; \mu_{L_A}, \Sigma_{L_A}) \\ &= \mathcal{N}(L_A; \mu_{L_A}, \Sigma_{L_A}) \end{aligned} \quad (5.88)$$

where μ_{L_A} is the mean and Σ_{L_A} is the covariance, estimated by the statistics in $D_\gamma^+(A|P)$.

So, we compute the prediction score for node A from its parent node P as,

$$w_P^{\text{predict}} = \log p(X(A)|X(P); \nabla) \quad (5.89)$$

Then, the weight of a γ hypothesis of node A is,

$$w_A^{\gamma(P)} = w_P^{\text{predict}} + w_P^\alpha \quad (5.90)$$

Similarly, the threshold $\text{Th}(w_A^{\gamma(P)})$ of the γ process $\gamma(A|P; \varphi)$ can be estimated in a validation γ dataset.

The bottom panel of Fig.5.53 illustrates the learned Gaussian distributions in the β binding and γ prediction process for human face.

5.3 Example I: Integrating the α , β and γ for Image Parsing

In the experiments of our numerical study, we choose two hierarchical case studies, one is junctions and rectangles in low-to-middle-level vision and the other is human faces in high-level vision. And, we do two series of experiments, one is to evaluate the individual information contribution of the α , β and γ processes and the other is object parsing by integrating the three processes with performance comparisons.

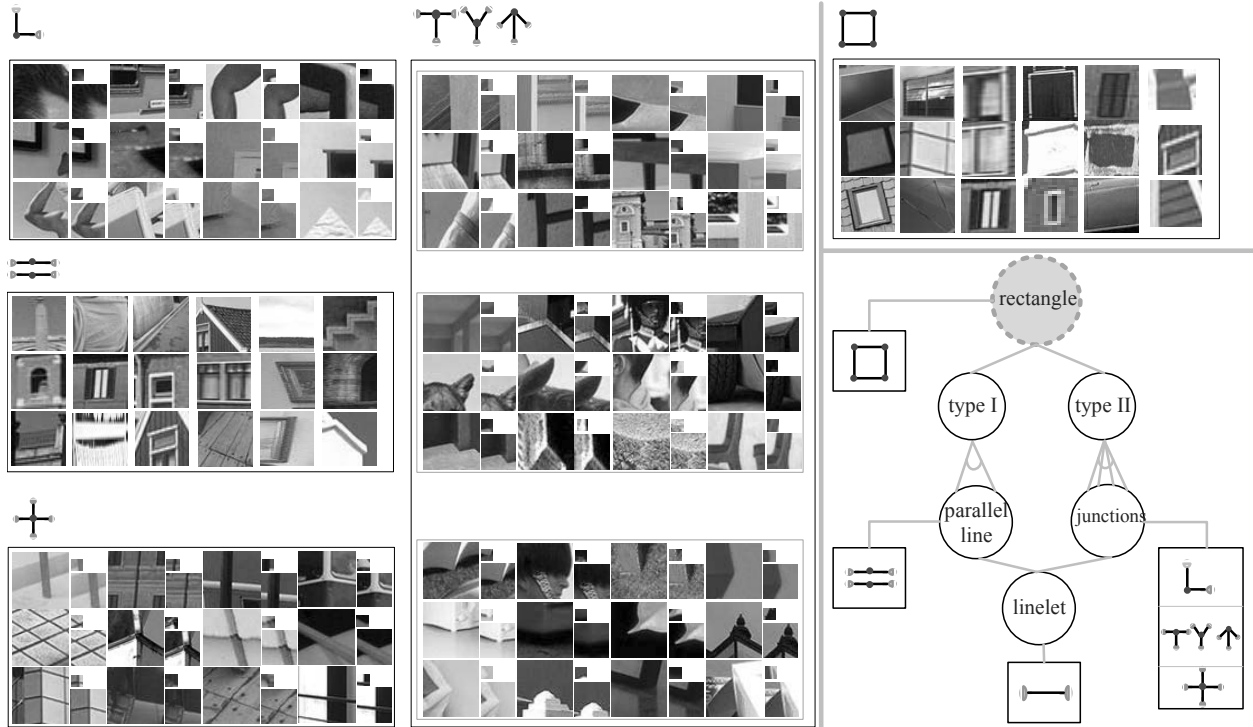


Figure 5.54: Illustration of the AoG for junctions and rectangles. The left panel shows some positive examples of L junction, cross junction, parallel line, T/Y/arrow junction. Each sample of L, cross and T/Y/arrow junctions is shown under three different scales (10×10 pixels, 20×20 pixels and 30×30 pixels) from which we can intuitively see that the α process would be very weak. The right-top panel shows some samples of rectangle in which we can also know that the α process would not work well due to the variabilities. The right-bottom panel shows the AoG for rectangle. Reproduced from Ref. [168] with permission of Springer, © 2011.

5.3.1 Experiment I: Evaluating Information Contributions of the α , β and γ Processes Individually

Junctions and rectangles. We consider five types of hierarchical image structures in low-to-middle-level vision including L-junction, cross junction, parallel line, T/Y/arrow junction and rectangle. In our experiments, we treat T/Y/arrow junction as the same type currently due to the similarity. As illustrated in the right-bottom panel of Fig.5.54, the rectangle node is an Or-node which has two types of decompositions, one is decomposed into two groups of parallel lines and the other is decomposed into four junctions such as four L-junctions.

The data. A set of 200 natural images from the LHI image database [176] is used in which the sketches are manually labeled. Based on the manually labeled sketches, we extract positive examples for the five types of hierarchical image structures and a common set of negative examples. Some positive examples are shown in Fig.5.54.

Training and testing. For the α process, we use first and second derivative Gaussian filters, LoG (Laplacian of Gaussian) filters, DoG (difference of Gaussian) and elongated DooG (difference of offset Gaussian) filters, all with 3 scales (10×10 , 20×20 and 30×30 pixels). The α process of line segment uses the primal sketch model [57] similar to the implicit testing used in our previous compositional boosting work [167].

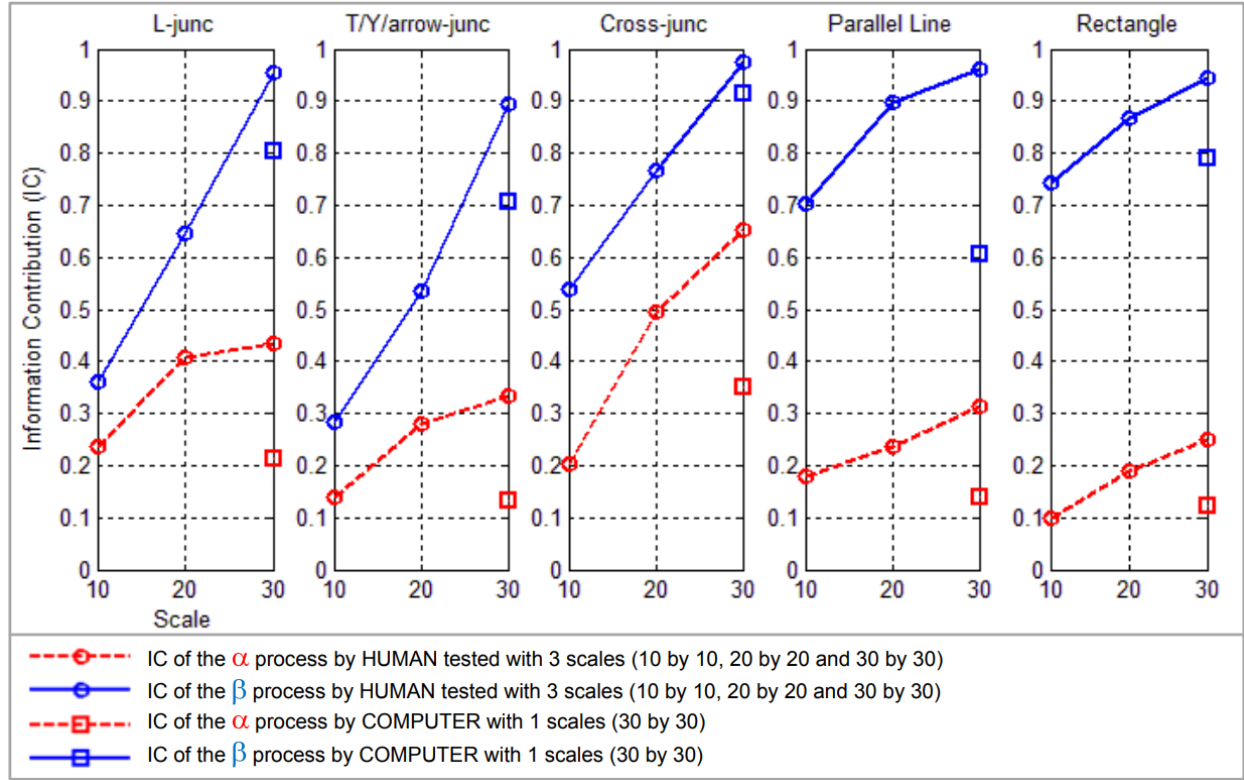


Figure 5.55: The information contributions of the α and β processes of junctions and rectangles. We test three scales, 10×10 , 20×20 and 30×30 pixels. We can observe that the β processes of junctions and rectangles are much better than their α processes. Reproduced from Ref. [168] with permission of Springer, © 2011.

The α processes of the five types of hierarchical image structures use the patch-based active basis model for both shape and texture. In testing, we search different 15 orientations in order to handle the rotation. For the β process, the five types of hierarchical image structures are computed by binding line segments in terms of the explicit testing on their relative locations, angles and distances between their endpoints. Rectangles are computed in two alternative ways, one is by binding two groups of parallel lines in terms of their relative locations and angles, and the other is by binding a set of incomplete (two or three) or complete (four) junctions in terms of their relative locations, angles and distances between the endpoints.

The observation: Fig.5.55 shows the information contributions of the α (red lines) and β (blue lines) processes of junctions and rectangles from the human study experiments at three scales (10×10 , 20×20 and 30×30 pixels). The results of computer experiments are shown by those small rectangles (the red ones for the α and the blue ones for the β process and for clarity only the results tested with the scale 30×30 pixels are shown). We observe that the β binding inference process dominates in low-to-middle-level vision.

Human faces. We consider the AoG of the human face which consists of six nodes, head-shoulder, face, left eye, right eye, nose and mouth, as shown in Fig.5.53. In our experiments, we treat the left and right eye nodes as the same type of node due to the similarity.

The data. A set of 1000 images from the LHI database is used in which all the six nodes are at good resolution and the parse graphs are manually labeled (see an example in Fig.5.52). We generate the training

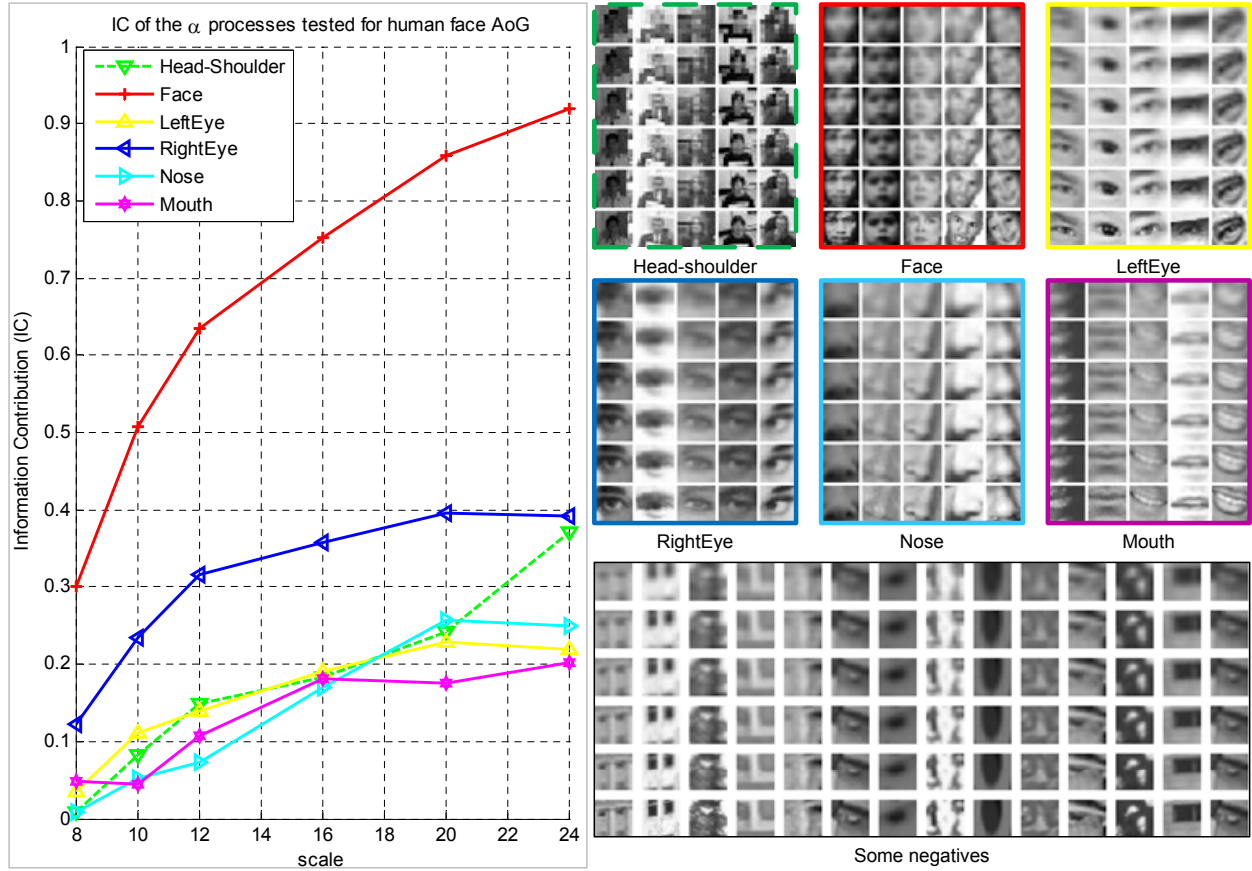


Figure 5.56: The left panel shows the information contributions of the α processes of nodes (*i.e.* head-shoulder, face, left eye, right eye, nose and mouth) in human face AoG in the human study. We test five scales (8×8 , 10×10 , 12×12 , 16×16 , 20×20 and 24×24 pixels). Some positive examples for each node and some negative examples are shown in the right panel. We can observe that the $\alpha(\text{face})$ process is stronger than the α processes of other nodes in the human face AoG. Reproduced from Ref. [168] with permission of Springer, © 2011.

data based on the parse graph.

Training and testing. For the α process, we use the Gabor filter. The learned α , β and γ processes are shown in Fig.5.53. Here, we test five scales for the α process (8×8 , 10×10 , 12×12 , 16×16 , 20×20 and 24×24 pixels), five scales for β process (38×38 , 50×50 , 60×60 , 80×80 and 100×100) and one scale for the γ process of human face (32×32).

The observation: We observe that the α process of the human face node is stronger than those of the other node in the human face AoG. The information contributions are shown in Fig.5.56.

5.3.2 Experiment II: Object Parsing in a Greedy Pursuit Manner by Integrating the α , β and γ Processes

Rectangles. We test a set of 50 images including 30 city scene images and 20 office scene images. A running example is shown in Fig.5.61 and more examples are shown in Fig.5.62. From the ROC comparisons in Fig.5.62, we can see that the β processes of junctions and rectangles in low-to-middle-level vision dominate

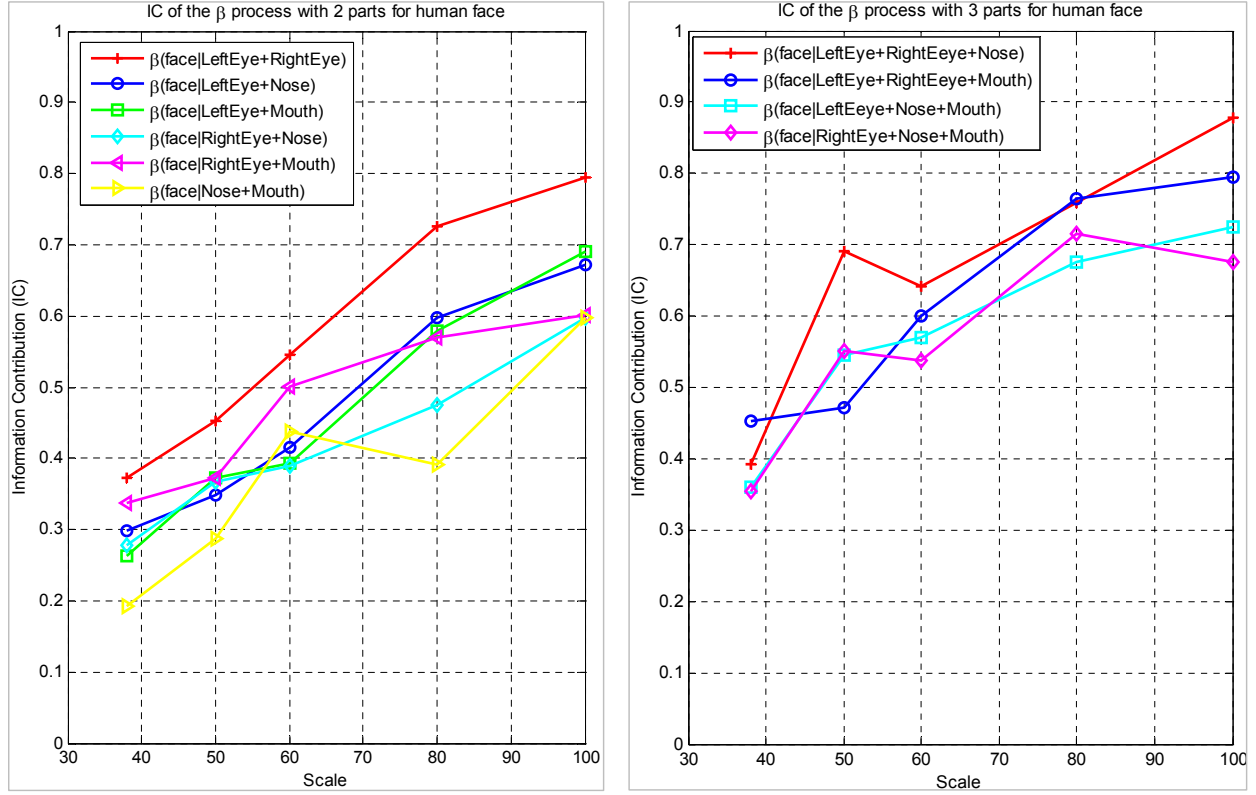


Figure 5.57: The information contributions of the β processes of the human face. *The left panel* show the information contributions of the β processes with 2 facial components. *The right panel* is for the β processes with 3 facial components. We test five scales, 38×38 , 50×50 , 60×60 , 80×80 and 100×100 pixels. Some examples are shown in Fig.5.60. In the β process, we can observe that the left eye and right eye are more informative than other facial components. Reproduced from Ref. [168] with permission of Springer, © 2011.

with much performance improved against the α processes.

Human faces. We test a set of 500 images in which more than half of human face instances are with occlusion or at very low resolution. A running example of the human face pursuit is shown in Fig.5.50 and more examples are shown in Fig.5.63. From the ROC comparisons in Fig.5.63, we can see that for human face, its α process works better than those of its child nodes such as eyes and nose and its parent node such as head-shoulder.

The ROC comparisons are consistent with the evaluated information contributions in experiment I.

5.4 Example II: Recognition on Object Categories

The second example, taken from [87], applies the top-down/bottom-up inference to five object categories – clock, bike, computer (screen and keyboard), cup/bowl, and teapot. The five categories are selected from a large-scale ground truth database from the Lotus Hill Institute. The database includes more than 500,000 objects over 200 categories parsed in And-Or graphs [176]. The probabilistic models are learned for these

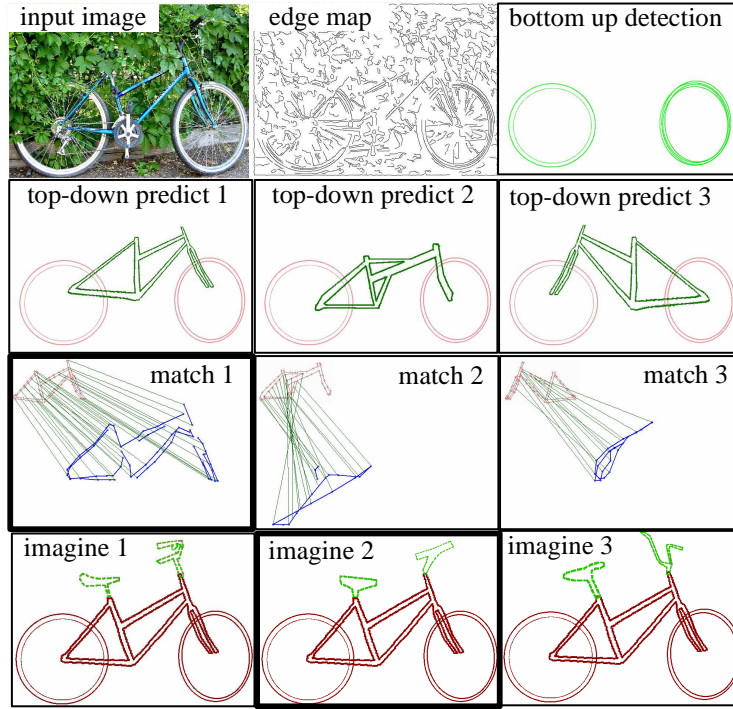


Figure 5.58: The top-down influence in inferring a partially occluded bike from clutter. Reproduced from Ref. [87] with permission of Elsevier, © 2009.

and-Or graphs using the MLE learning presented in the previous section. The clock and bike sampling results were shown in Figures 4.5 and 4.7.

The And-Or graphs together with their probabilistic models represent the prior knowledge above the five categories for top-down inference. Figure 5.58 shows an example of inferring a partially occluded bicycle from clutter.

In Figure 5.58. The first row shows the input image, an edge map, and bottom-up detection of the two wheels using Hough transform. The Hough transform method is adopted to detect parts like circles, ellipse and triangles. The second row shows some top-down predictions of bike frame based on the two wheels. The transform parameters of the bike frame are sampled from the learned MRF model. As we can not tell the front wheel from the rear at this moment, the frames are sampled for both directions. We only show three samples for clarity. The third row shows the template matching process that matching the predicted frames (in red) to the edges (in blue) in the image. The one with minimum matching cost is selected. The fourth row shows the top-down hallucinations (imaginings) for the seat and handlebar (in green). As these two parts are occluded. The three sets of hallucinated parts are randomly sampled from the And-Or graph model, in the same way as random sampling of the whole bike.

Finally, we show a few recognition examples in Figure 5.59 for the five categories. For each input image, the image on its right side shows the recognized parts from the image in different colors. It should be mentioned that the recognition algorithm is distinct from most of the classification algorithms in the literature. It interprets the image by a parse graph which includes the classification of categories and parts on the Or-nodes, and matches the leaf templates to images, and hallucinates occluded parts.

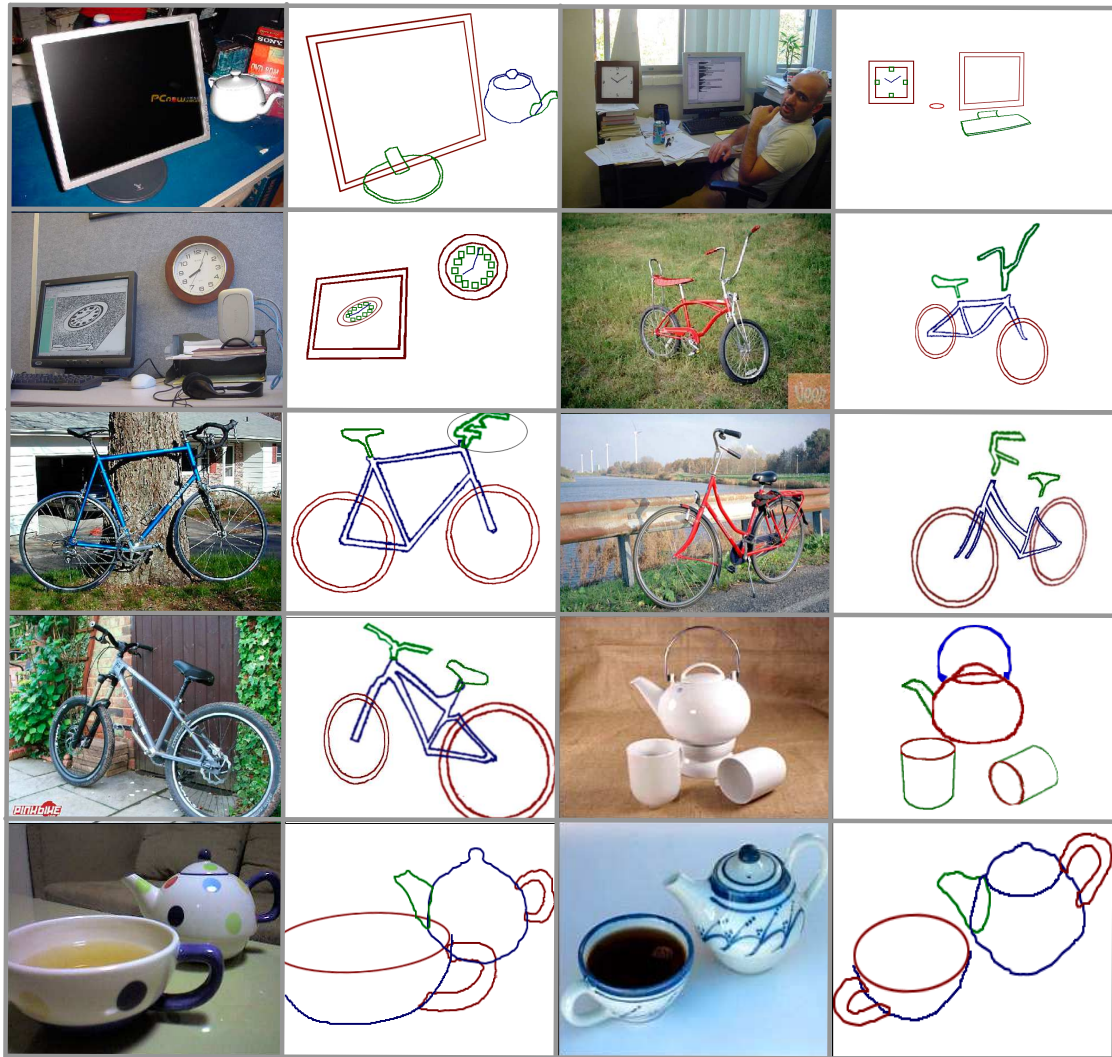


Figure 5.59: Recognition experiments on 5 object categories. Reproduced from Ref. [87] with permission of Elsevier, © 2009.

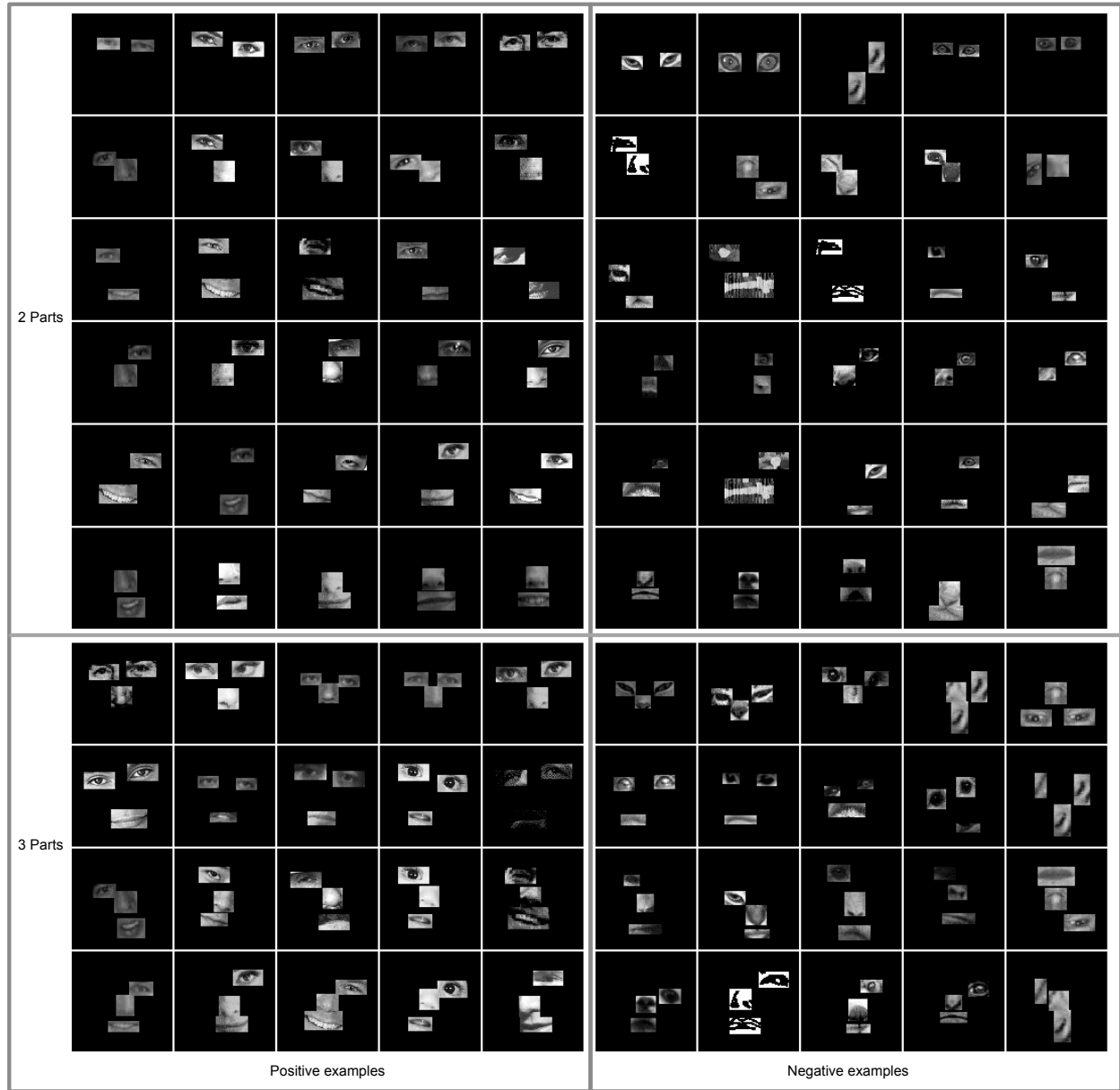


Figure 5.60: Some examples used in the human study of evaluating the information contribution of the β processes of human face with 2 and 3 facial components respectively. The left panel shows some positive examples and the right panel shows some negative examples. The examples are at 100×100 pixels for illustration. Reproduced from Ref. [168] with permission of Springer, © 2011.



Figure 5.61: A running example of pursuing junctions and rectangles in a typical image by integrating the α , β and γ processes. The left-top is the original input image. The middle-top is the edge probability map and the right-top shows the detected corners. The images in the second and the third row show the detected results of different kinds of junctions with the type name shown in the left-top in each image. The left-bottom shows the detected rectangles. The middle-bottom shows the final sketch by merging all detected results. Compared with the canny results shown in the right-bottom image, we can see that the final sketch obtained by the proposal algorithm is better. Reproduced from Ref. [168] with permission of Springer, © 2011.

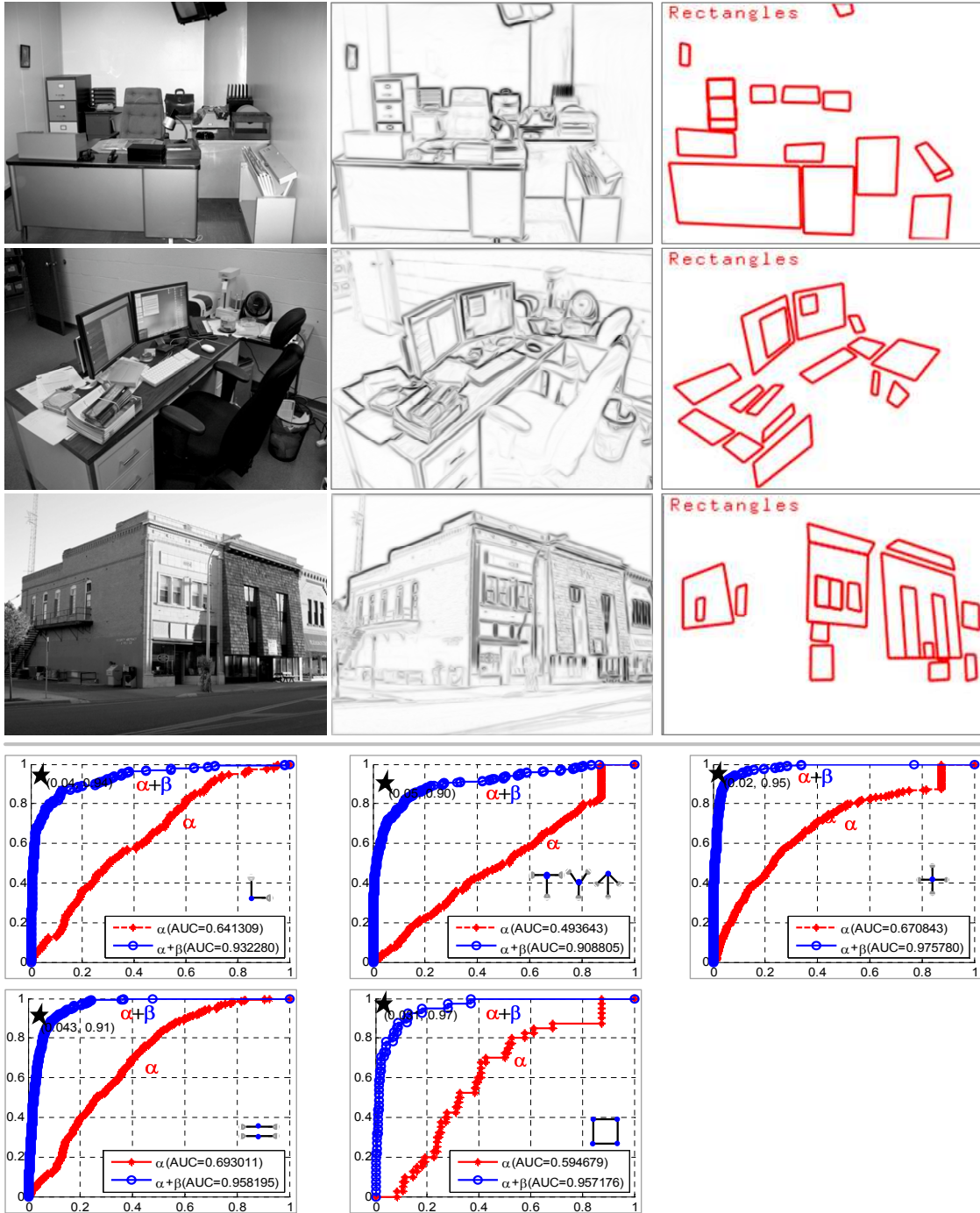


Figure 5.62: The top panel shows more results of rectangle pursuing. The bottom panel shows the ROC comparisons of the α process and the integration of the α , β processes for junctions and rectangle. Those black pentagrams show the performance by humans. Reproduced from Ref. [168] with permission of Springer, © 2011.

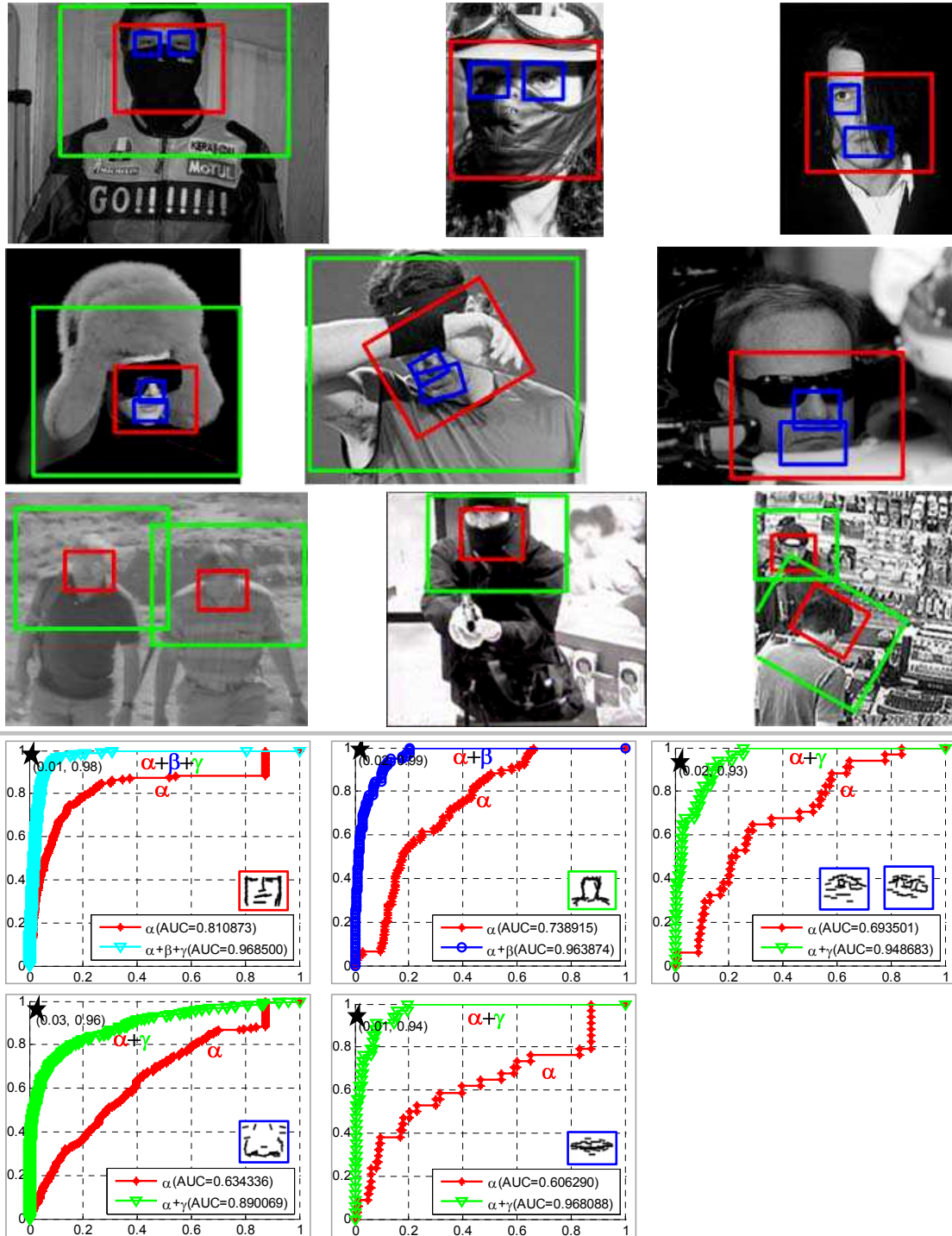


Figure 5.63: The top panel shows more results of human face pursuing. The bottom panel shows the ROC comparisons of the α process and different integrations of the α , β and γ processes. Those black pentagrams show the performance by humans. Reproduced from Ref. [168] with permission of Springer, © 2011.

6

Attributed And-Or Graph

In the recent decade, visual recognition has become the main focus of computer vision tasks and shows tremendous progress. The goal has been to recognize the object category. However, there are also many other qualities apart from its category to describe an object. These attributes could be semantic attributes like parts, shape, materials, colors, and physical properties, they could also be discriminative attributes since semantic attributes may not be enough to distinguish all the categories of objects, for example, comparisons among the objects.

Modeling these attributes can complement category-level recognition and improve the understanding of visual objects for the machines. Besides, attributes are usually shared among different object categories, thus modeling and learning the visual attributes explicitly helps to transfer knowledge to a novel category and achieve across category generalization, leading to a deeper understanding beyond the recognition.

In this chapter, we describe how to better model the images and scenes with an attributed grammar model. Unlike the stochastic image grammar in the last chapter, the attributed grammar augments each node (or production rule) with a set of attribute variables that impose constraints between a node and its off-spring in the parse graph. The attributed grammar is more general and can be applied to various applications where visual attributes are used to describe the visual entities. We introduce several examples of the attributed grammar, showing its representation power, and the modeling capability on a variety of vision tasks such as parsing, reconstruction, and synthesis.

6.1 Introduction of Attribute Grammar

Attribute grammars were introduced in the late 1960s by Knuth for specifying and implementing the (static) semantic aspects of programming languages [77]. Today most compiler-generators use Attribute Grammars to generate the components for the semantics analysis phase out of a user's specification automatically. An *Attribute Grammar* consists of three components, a context-free grammar \mathcal{G} , a finite set of attributes \mathcal{A} , and a finite set of semantic rules \mathcal{AR} :

$$\mathcal{AG} = (\mathcal{G}, \mathcal{A}, \mathcal{AR}) \quad (6.1)$$

The context-free grammar $\mathcal{G} = (N, T, P, S)$, where N is the set of non-terminals, T is the set of terminals, and P is the set of productions. Each production is of the form $A ::= \alpha$, where $A \in N$ and $\alpha \in (N \cup T)^*$. $S \in N$ is the start symbol. \mathcal{A} associates each grammar symbol $X \in N \cup T$ with a set of attributes, and \mathcal{AG} associates each production $\mathcal{R} \in P$ with a set of attribute computation rules [105]. $\mathcal{A}(X)$, where $X \in (N \cup T)$, can be further partitioned into two sets: synthesized attributes $S(X)$ and inherited attributes $I(X)$. $\mathcal{AR}(\mathcal{R})$, where $\mathcal{R} \in P$, contains rules for computing inherited and synthesized attributes associated with

Table 6.1: One Example of Attribute Grammar

Production	Semantic Rules
$\text{Expr1} \longrightarrow \text{Expr2} + \text{Term}$	$\text{Expr1.value} = \text{Expr2.value} + \text{Term.value}$
$\text{Expr} \longrightarrow \text{Term}$	$\text{Expr.value} = \text{Term.value}$
$\text{Term1} \longrightarrow \text{Term2} * \text{Factor}$	$\text{Term1.value} = \text{Term2.value} * \text{Factor.value}$
$\text{Term} \longrightarrow \text{Factor}$	$\text{Term.value} = \text{Factor.value}$
$\text{Factor} \longrightarrow (\text{Expr})$	$\text{Factor.value} = \text{Expr.value}$
$\text{Factor} \longrightarrow \text{integer}$	$\text{Factor.value} = \text{integer.value}$

the symbols in the production \mathcal{R} . By the definition, synthesized attributes are output to the left-hand-side symbols of productions, and inherited attributes are output to the right-hand-side symbols. (The synthesized attributes of terminal symbols are assumed to be externally defined.)

For example, the attribute grammar shown in Table 6.1 can be used to calculate the result of an expression written in the grammar. Note that this grammar only uses synthesized values, and therefore it's also called an S-attributed grammar.

While earlier chapters have shown that stochastic grammar models have great success in explicit representing and inferring complex compositional structures, pose estimation, scene parsing, and event prediction, these models do not incorporate any notions of visual attributes. In the meantime typical attribute models for objects and scenes are neither compositional nor part-based, and typically employ a large number of independent classifiers to decide which attribute labels to assign to a given bounding box. Such approaches do not explicitly capture the relationship between attribute labels, or localize the part regions described by the attributes. We propose a unified model to strengthen the strength and make up for the weakness of both stochastic grammar model and attributed model.

6.2 Attributed Graph Grammar Model

Following the previous definition in Section 3.5.1, an attributed And-Or graph is specified by a 5-tuple $G_{\text{and-or}} = \langle S, V_N, V_T, \mathcal{R}, \mathcal{P} \rangle$, where V_N and V_T are the sets of non-terminal and terminal nodes respectively. S is the root node for a scene or object category. \mathcal{R} is a set of production rules for relationships and \mathcal{P} is the probability for the grammar.

A non-terminal node is denoted by capital letters $A, A_1, A_2 \in V_N$, and a terminal node is denoted by lower case letters $a, b, c, a_1, a_2 \in V_T$. Both non-terminal and terminal nodes have a vector of attributes denoted by $X(A)$ and $x(a)$ respectively. $\mathcal{R} = \{r_1, r_2, \dots, r_m\}$ is a set of production rules expanding a non-terminal node into a number of nodes in $V_N \cup V_T$. Each rule is associated with a number of constraint equations. For example, the following is a rule that expands one node A into two nodes $A_1, A_2 \in V_N$.

$$r : A \rightarrow (A_1, A_2) \quad (6.2)$$

The associated equations are constraints on the attributes.

$$g_i(X(A)) = f_i(X(A_1), X(A_2)), i = 1, 2, \dots, n(r) \quad (6.3)$$

$g_i()$ and $f_i()$ are usually projection functions that take some elements from the attribute vectors. For instance, let $X(A) = (X_1, X_2, X_3)$ and $X(A_1) = (X_{11}, X_{12})$, then an equation could simply be an equivalence constraint (or assignment) for passing the information between nodes A and A_1 in either directions,

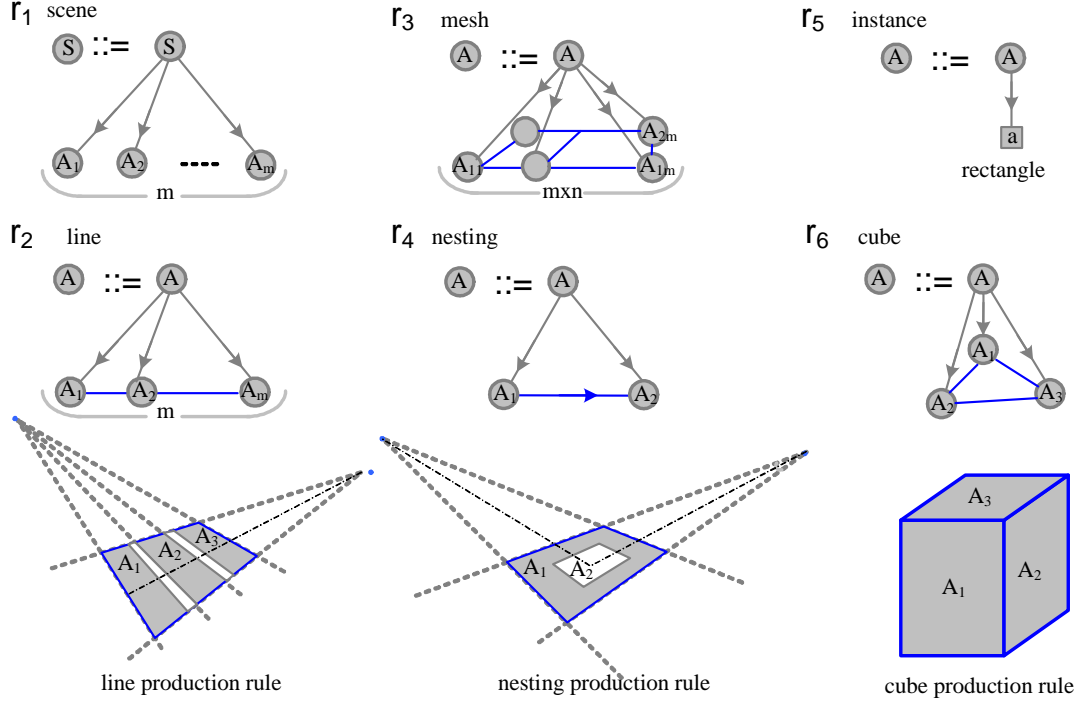


Figure 6.1: Six attribute grammar rules for generic man-made world scenes. This grammar features a single class of primitives – rectangle and four generic organizations – line, mesh, cube, and nesting. Attributes will be passed between a node to its children and the horizontal lines show constraints on attributes. Reprinted from Ref. [61] with permission.

$X_1 = X_{11}$. In the parsing process, sometimes we know the attributes of a child node X_{11} and then pass it to X_1 in rule r . This is called “bottom-up message passing”. Then X_1 may be passed to another child node’s attribute X_2 with $X_{21} = X_1$. This is called “top-down message passing”. A production rule may instantiate a non-terminal node to a terminal node

$$r : A \rightarrow a, \quad (6.4)$$

with constraints

$$g_i(X(A)) = f_i(x(a)), i = 1, 2, \dots, n(r) \quad (6.5)$$

Our proposed A-AOG relaxes the hard constraints [62,91] to soft energy terms and encodes three types of contextual information.

i) Consistency between the same attribute in parent and children nodes. For example, if the root node has gender attribute as female, then its parts are likely also female.

ii) Co-occurrence between attributes, *e.g.* a female is more likely to have long hair and wear a skirt.

iii) Correlations between the assignment of an attribute to the image feature of a node and its alternative choices, *e.g.* long hair and short hair will have preferences on the choices of image templates.

Next, we will show two examples for parsing and estimation with Attribute AoG.

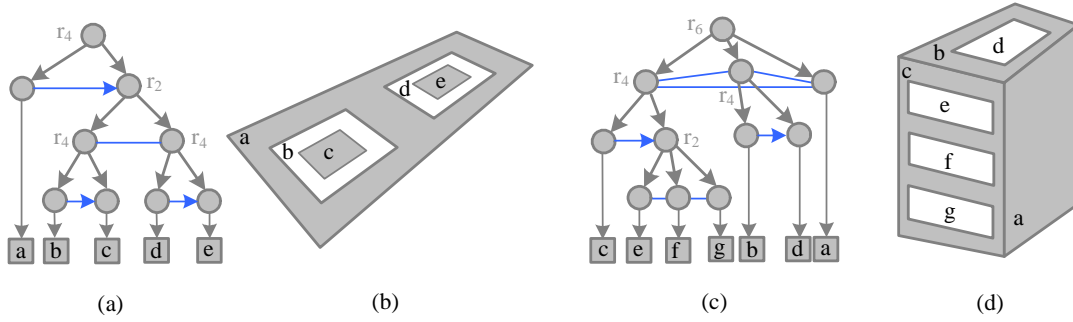


Figure 6.2: Two examples of rectangle object configurations (b) and (d) and their corresponding parse graphs (a) and (c). The production rules are shown as non-terminal nodes. Reproduced from Ref. [185] with permission of Now Publishers Inc, © 2007.

6.3 Example I: Parsing the Perspective Man-made World

In this case taken from [62], the grammar has one class of primitives as the terminal nodes (*i.e.* V_T), which are 3D planar rectangles projected on images. Obviously, rectangles are the most common elements in man-made scenes, such as buildings, hallways, kitchens, living rooms, *etc.* Each rectangle $a \in V_T$ is made of two pairs of parallel line segments in 3D space, which may intersect at two vanishing points through projection. The grammar has only two types of non-terminal nodes (*i.e.* V_N) – the root node S for the scene and a node A for any composite objects. The grammar has six production rules as shown in Fig. 6.1. The scene node S generates m independent objects (rule r_1). An object node A can be instantiated (assigned) to a rectangle (rule r_5), or be used recursively by the other four production rules: r_2 – the line production rule that aligns a number of rectangles in one row, r_3 – the mesh production rule that arranges a number of rectangles in a matrix, r_4 – the nesting production rule that has one rectangle containing the other, and r_6 – the cube production rule that aligns three rectangles into a solid shape. The unknown numbers m and n can be represented by the Or-nodes for different combinations.

Each production rule is associated with a number of equations that constrain the attributes of a parent node and those of its children. These rules can be used recursively to generate a large set of complex configurations. Fig. 6.2 shows two typical parsing configurations – (b) a floor pattern and (d) a toolbox pattern, and their corresponding parse graphs in (a) and (c) respectively.

The parsing algorithm adopts a greedy manner. For each of the 5 rules r_2, \dots, r_6 , it maintains an Open list and a Closed list. In an initial phase, it detects an excessive number of rectangles by a bottom-up rectangle detection process and thus fills the Open list for rule r_5 . Each particle consists of two pairs of parallel line segments.

The top-down and bottom-up computation is illustrated in Fig. 6.3 for a kitchen scene. Fig. 6.3 shows a parse graph under construction at a time step, the four rectangles (in red) are the accepted rectangles in the Closed list for r_5 . They activated a number of candidates for larger groups using the production rules r_3, r_4, r_6 respectively, and three of these candidates are then accepted as non-terminal nodes A, B, and C respectively. The solid upward arrows show the bottom-up binding, while the downward arrows show the top-down prediction.

Fig. 6.4 shows the five Open lists for the candidate sets of the five rules. At each step, the parsing algorithm will choose the candidate with the largest weight from the five particle sets and add a new non-terminal node to the parse graph. If the particle is in the r_5 Open list, it means accepting a new rectangle. Otherwise the algorithm creates a non-terminal node and inserts the missing children in this particle into

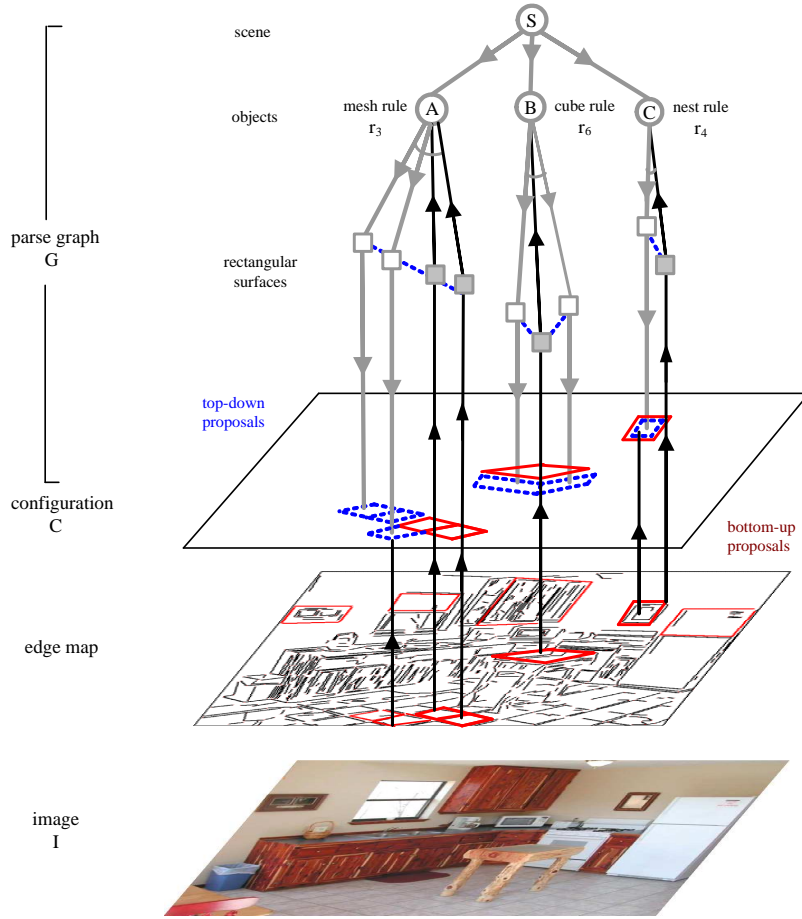


Figure 6.3: Illustrating the recursive bottom-up / top-down computation procedure in image parsing. The detection of rectangles instantiates some non-terminal nodes shown as upward arrows. They in turn activate graph grammar rules for grouping larger structures in nodes A, B, C respectively. These rules generate top-down prediction of dashed rectangles. The predictions are validated from the image under the Bayesian posterior probability. Reprinted/modified from Ref. [61] with permission.

their respective Open lists for future tests.

Fig. 6.5 shows two ROC curves for performance comparison in detecting the rectangles in 25 images against human-annotated ground truth. One curve shows the detection rate (vertical axis) over the number of false alarms per image (horizontal axis) for pure bottom-up method. The other curve is for the methods integrating bottom-up and top-down. From these ROC curves, we can clearly see the dramatic improvement by using top-down mechanism over the traditionally bottom-up mechanism only. Intuitively, some rectangles are nearly impossible to detect using the bottom-up methods and can only be recovered through the context information using the grammar rules.

6.4 Example II: Single-View 3D Scene Reconstruction and Parsing

In this example, taken from [91], Attribute AoG is used for solving two vision tasks jointly: i) parsing a 2D image into semantic regions in a hierarchical parse graph representation, *i.e.*, the recognition and ii)

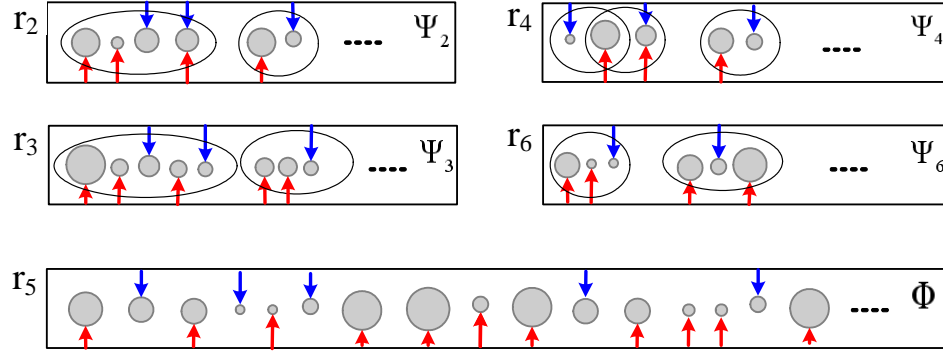


Figure 6.4: Illustration for the Open lists of the five rules. Reprinted from Ref. [62] with permission of IEEE, © 2008.

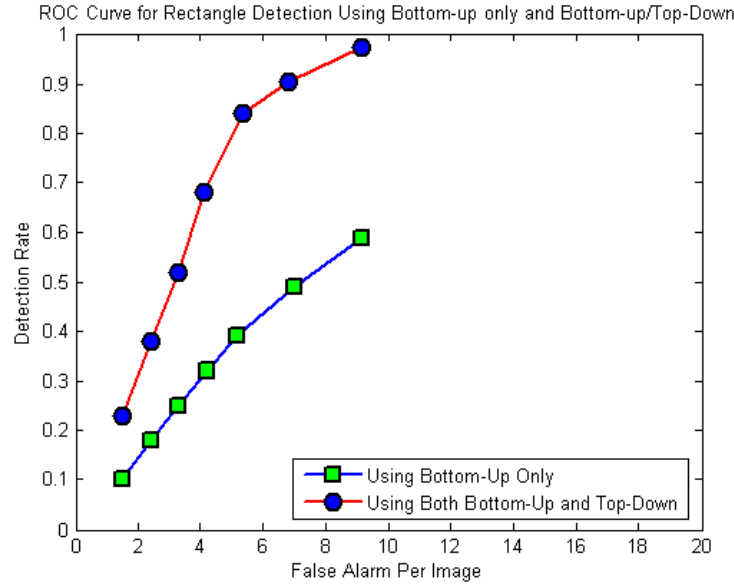


Figure 6.5: ROC curves for the rectangle detection results by using bottom-up only and, using both bottom-up and top-down. Reproduced from [62] with permission of IEEE, © 2008.

recovering the 3D geometric scene structures for nodes in the parse graph, *i.e.*, the reconstruction. The attributed grammar consists of a set of production rules, each describing some spatial relationship between planar surfaces in 3D scenes. By augmenting levels of geometric attributes (camera parameters, vanish points, surface normal *etc.*) to the nodes in the parse graph, the recognition and reconstruction are solved in a joint parsing process simultaneously.

We consider outdoor urban scenes that may contain multiple local Manhattan worlds (LMW) or “mixture Manhattan world” [144], where, for example, buildings are composed of multiple planar surfaces and touch the ground on contact lines. In contrast to the widely used Manhattan world assumption [22], here we consider a more general scenario that, the adjacent surfaces of a building may not be orthogonal to each

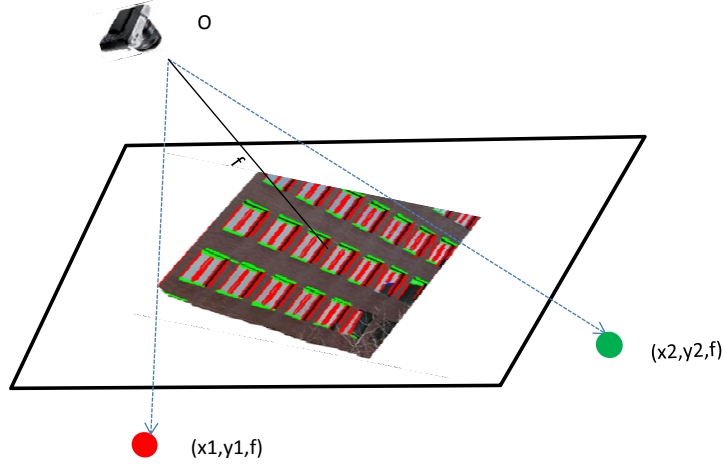


Figure 6.7: Calculation of surface normal. A planar region often contains two sets of orthogonal parallel lines converging at two vanishing points (x_1, y_1) and (x_2, y_2) , respectively. f is the camera focal length. Thus the surface normal is the cross-product of the two Manhattan axes (x_1, y_1, f) and (x_2, y_2, f) in the camera coordinate (taking camera position O as origin). Reprinted from Ref. [91] with permission.

6.4.1.1 Geometry Attributes from Edge Statistics

In man-made scenes, texture gradients and edges are not arbitrarily oriented, but reflect camera orientations with respect to the scene and surface layout in 3D space. Hence, we can extract the geometric attributes from edge statistics.

Attributes of Edges and Parallel Lines

In the pinhole camera model, a family of parallel lines, *i.e.* sharing the same 3D direction, in the 3D space project to straight edges that all point to the same point on the image plane, *i.e.* the vanishing point. Thus each line segment in the image has two geometric attributes:

- A *vanishing point* (x_i, y_i) in the image plane to which an edge points to. This can be directly obtained by clustering oriented edges based on their directions in the 2D image plane.
- A *3D direction* $\theta = (x_i, y_i, f)$ of edges or parallel lines in the 3D scene space where f is the camera focal length. As Fig. 6.7 illustrates, it follows from perspective geometry the ray from the camera position O to (x_i, y_i) is parallel to the families of parallel lines as well. Therefore, its direction is the unit vector by normalizing the triple vector (x_i, y_i, f) .

Attributes of Local Manhattan World

The outdoor urban scene often contains a mixture of local Manhattan worlds [22]. Each local Manhattan world is a block of well-aligned buildings with three sets of orthogonal parallel lines. Each set of parallel lines has a vanishing point (x_i, y_i) and 3D direction $\theta = (x_i, y_i, f)$. We refer to the rays from camera origin O to the vanishing points as the *Manhattan axes*. Thus each local Manhattan world has the following geometric attributes:

- A *Manhattan frame* with three orthogonal Manhattan axes $\{(x_1, y_1, f), (x_2, y_2, f), (x_3, y_3, f)\}$.
- An estimated focal length f following [14]

$$f^2 = -(x_i, y_i) \cdot (x_j, y_j), \quad i \neq j \in \{1, 2, 3\}. \quad (6.6)$$



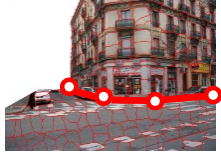
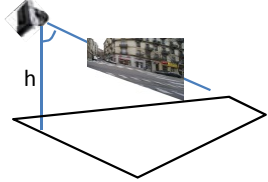
<i>Representation</i>	<i>Examples</i>	<i>Geometric Attributes</i>
Surface Fragment		Surface Normal; VPs;
Planar Surface		Surface Normal; Contact Line; Manhattan Axes; 2 VPs;
Composite Surface		Surface Normal ; Contact Spline; Manhattan Axes; K VPs;
Scene		Focal Length ; Vanishing Points; Manhattan Frames;

Figure 6.8: Illustration of hierarchical entities in the attribute planar representation and their associated geometric attributes. Each representation entity is also entitled with a semantic attribute, *i.e.* the object categories (*e.g.* building, sky *etc.*) it corresponds to. Reprinted from Ref. [91] with permission.

This follows the orthogonal condition that

$$(x_i, y_i, f) \cdot (x_j, y_j, f) = 0. \quad (6.7)$$

It is worth noting that this estimated focal length will be propagated to the scene node in the attribute parse graph. The equation $(x_i, y_i) \cdot (x_j, y_j) = (x_k, y_k) \cdot (x_j, y_j)$ poses consistency conditions among the attributes of Manhattan axes.

Attribute Planar Representation

In parallel to the edges, lines/parallel lines, and Manhattan structures, the region-based hierarchy comprises three representations: surface fragments, planar surfaces, and composite surfaces. Fig. 6.8 summarizes the attribute planar representation.

We augment every hierarchical entity with both semantic attributes and geometric attributes. The semantic attribute of an entity, *e.g.* planar surface, is simply its semantic category. In this work, we consider a few semantic categories for outdoor scenes, including “building”, “tree”, “ground”, “sky” and “other”. A composite surface might include two or more than two categories. Geometric attributes are used to describe the spatial properties of the hierarchy, which will be introduced in the rest of this section.

Geometric Attributes of Surface Fragment

We assume that each *super-pixel* in images is the projection of a *surface fragment* in space. A super-pixel is a small region of pixels that are connected and share similar appearance features, and often have

the same semantic label. Since these super-pixels often correspond to regions in buildings or marked road/highways/ground, which have edges or texture gradients, from which we can extract short edges and estimate which vanishing points they belong to.

As Fig. 6.7 illustrates, each super-pixel has two geometric attributes:

- Two vanishing points: $\{(x_1, y_1), (x_2, y_2)\}$ and thus two Manhattan axes $\{\mathbf{v}_1 = (x_1, y_1, f), \mathbf{v}_2 = (x_2, y_2, f)\}$.
- A *surface normal* direction which is the cross-product of the two Manhattan axes $\mathbf{n} = (x_1, y_1, f) \times (x_2, y_2, f)$.

For each superpixel, we extract its vanishing points and surface normal from local edge statistics, which might not be necessarily accurate. To improve robustness against noises, these statistics will be pooled together in bottom-up process and propagated to other nodes in the attribute parse graph. For a super-pixel that does not contain a sufficient number of edges, its surface normal will be inferred from surrounding scene context, *i.e.* top-down process in the parse graph.

Geometric Attributes of Planar Surface

We group spatially connected super-pixels into planar surfaces based on two types of features. i) Appearance features. We extract color and texture features to train a supervised classifier and assign a region to a few categories, *e.g.* 'building', 'tree', *etc.* ii) Geometry features. Superpixels in the same planar region should share the same surface normal. Both features are used in the iterative parsing process to form planar surfaces.

Each planar surface has three geometric attributes

- Two vanishing points: $\{(x_1, y_1), (x_2, y_2)\}$ and thus two Manhattan axes $\{(x_1, y_1, f), (x_2, y_2, f)\}$.
- Normal direction. As aforementioned, surface normal is simply the cross-product of the two Manhattan axes.
- A *contact line* and thus its 3D relative depth. The surface plane will intersect with other planes and form the contact lines. For example, Fig. 6.8 shows three planar surfaces of the building and their ground contact boundaries which can be approximated by straight lines respectively.

The contact lines may be occluded (*e.g.* between a building façade and the ground) or blurred (between two surfaces of the building). Fortunately, this can be solved by calculating the intersection line between adjacent surface planes, which usually points to one of the Manhattan axes associated with the surface planes. These geometric attributes are sufficient to reconstruct a planar-wise 3D scene model [66].

Geometric Attributes of Composite Surface

A composite surface consists of several planar surfaces that are physically connected. These surfaces might not belong to the same Manhattan frame. A composite surface has a set of geometric attributes that pose consistency constraints between its children nodes in the parse graph. Its geometric attributes include:

- All vanishing points and surface normal of its planar surfaces.
- Contact lines between adjacent surfaces.
- A linear spline fit of the contact lines with the ground.

As planar surfaces, *e.g.* building facade, are usually occluded by foreground objects, *e.g.* vehicles and trees, and their boundaries to the ground plane are often partially visible. In Section 6.4.4 we shall introduce a robust method for estimating contact splines under these severe occlusions.

Geometric Attributes of Scene

The whole scene will pool over the geometric attributes from its components. As it is shown in Fig. 6.6, the root node S has the following geometric attributes.

- Camera parameters are shared by all nodes in the parse graph. Note that our model can be extended to reason other camera parameters, including skew, and optical center *etc.*
- m Manhattan frames $\{(x_{ij}, y_{ij}, f), i = 1, 2, \dots, m, j = 1, 2, 3.\}$ for each local Manhattan world.

These global geometric attributes are used to constrain the geometric attributes of the entities in the parse graph. For example, the number of possible normal directions for planar surfaces is determined by the number of Manhattan axes detected for the global scene. In contrast, the past methods [65, 66] usually fix the number of surface normal orientations during inference.

6.4.2 Attribute Scene Grammar

Attribute grammar was firstly proposed by Han *et al.* in [62]. We extend it to model hierarchical scene representations in both 2D images and 3D scene space.

An attribute grammar is specified by a 5-tuple: $G = (V_N, V_T, S, \mathcal{R}, \mathcal{P})$, where V_N is a set of non-terminal nodes, V_T is a set of terminal nodes, S is the root node for the whole scene, \mathcal{R} is a set of production rules for spatial relationships, and \mathcal{P} is a probability for the grammar.

These production rules can be recursively applied to generate a hierarchical representation of the input scene, namely *Parse Graph*. A parse graph is a valid interpretation of the input 2D image and the desired 3D scene. A grammar generates a large set of valid parse graphs for one given image of the scene.

Terminal Nodes. We partition the input image into a set of superpixels and use them as terminal nodes. Each superpixel is the projection of a surface fragment in space. We denote all terminal nodes as $V_T = \{a, X(a)\}$, where $X(a)$ denotes a set of attribute variables.

Non-Terminal Nodes are sequentially produced by merging terminal nodes or other non-terminals with grammar rules. Each node represents a planar surface or composite surface in space. There is one root node for the whole scene, *i.e.* S , and five production rules.

Every non-terminal node in parse graph can be decomposed into children nodes or grouped with other nodes to form parent nodes by applying the above grammar rules.

We denote all non-terminal nodes as $V_N = \{(S, X(S)), (A, X(A))\}$ where S denotes the root node for the whole scene, A non-terminal node and $X(A)$ the attributes of A . Fig. 6.9 illustrates these five rules and Fig. 6.6 shows one parse graph that is capable of generating the input image.

Global and Local Attributes. Each node is associated with a number of attributes, which are either globally or locally defined.

Global attributes are defined for the root node S and inherited by all graph nodes. $X(S)$ includes i) a list of possible categories (*e.g.*, ‘building’) that appear in the input image, denoted as C ; ii) geometric attributes, including the camera focal length f and Manhattan frames detected in the input image. Formally, we have $X(S) = (f, m, \{M_i\}, C), i = 1, \dots, m$. As aforementioned, each Manhattan frame M_i contains three orthogonal axes.

Local attributes are defined over properties of intermediate nodes, *e.g.* surface normal. These attributes are usually inherited from the global attributes and thus should be consistently assigned. Fig. 6.6 illustrates global geometric attributes in the left panel and local geometric attributes in the right panel. Semantic attributes are not included in the figure. Both global or local attributes are used to impose constraints to obtain valid parse graphs.

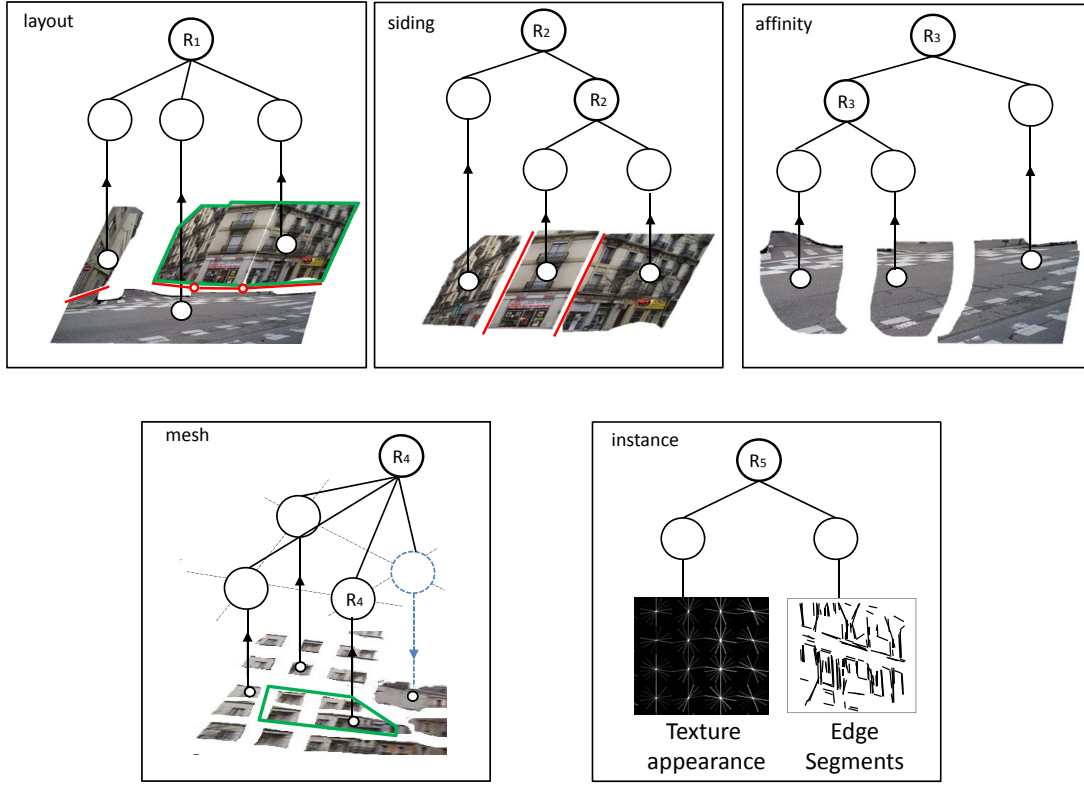


Figure 6.9: Illustration of the five grammar rules each of which is associated with a set of geometric attributes that imposes constraints over graph nodes and their offsprings. **Layout** rule: a children planar surface is supporting other n children entities; **siding** rule: two children planar surfaces of the same label are spatially connected; **affinity** rule: two children planar surfaces have the same appearance; **mesh** rule: multiple children surfaces appear in a 2D mesh structure; **instance** rule: links a children terminal node and its image representation. Reprinted from Ref. [91] with permission.

6.4.3 Probabilistic Formulation for 3D Scene Parsing

We utilize a hierarchical parse graph to explicitly encode the attribute hierarchy (introduced in Section 6.4.1) for joint recognition and reconstruction purposes. In particular, terminal nodes are able to form planar configuration in imaging plane or surface normal map of the input image; the parse graph with geometric attributes can be used to derive a full 3D scene model for reconstruction purposes.

Formally, let \mathbf{G} denote the parse graph to solve, \mathcal{A} all attributes in \mathbf{G} . Given an input image \mathbf{I} , we compute a world scene interpretation W in a joint solution space

$$W = (\mathcal{A}, \mathbf{G}) \quad (6.8)$$

The optimal solution W^* can be obtained by maximizing a posterior probability (MAP):

$$P(W|\mathbf{I}) \propto \exp\{-|\mathbf{V}_N| - \lambda^{\text{gra}} E(\mathbf{I}, \mathbf{G}, \mathcal{A})\} \quad (6.9)$$

where $|\mathbf{V}_N|$ indicates the number of non-terminal nodes. We use the first item to encourage compact parse graphs. λ^{gra} is a weight constant.

The energy $E(\mathbf{I}, \mathbf{G}, \mathcal{A})$ is defined over the hierarchy of \mathbf{G} , indicating how well \mathbf{G} can generate the input image \mathbf{I} . Let $r(A)$ indicates the grammar rule used at A . We have,

$$E(\mathbf{I}, \mathbf{G}, \mathcal{A}) = \sum_{A \in V_N} \beta_{r(A)} E^t(\mathbf{I}, X(A) | r(A)) \quad (6.10)$$

where $r(A) \in \{R_1, R_2, \dots, R_5\}$ indicates the grammar rule associated with A , $\beta_{r(A)}$ is a weight constant that is dependent on $r(A)$. The energy term $E^t(\mathbf{I}, X(A) | r(A))$ is associated with the nonterminal node A and conditioned on the corresponding grammar rule $r(A)$.

Table 6.2: Definitions of Grammar rules and their geometric attributes.

Rules	Notations	Geometric Attributes
R_1 : layout	$A \rightarrow (A_0, A_1, \dots, A_m)$	$X(A) = (f, m, \mathbf{M}_i, \theta_0, \theta_{ij}, \vec{l}_k, C)$
R_2 : Siding	$A \rightarrow (A_1, A_2)$	$X(A) = (\theta_i, \mathbf{M}_i, \vec{l}_k, c)$
R_3 : Affinity	$A \rightarrow (A_1, A_2)$	$X(A) = (\theta, \mathbf{M}, c)$
R_4 : Mesh	$A \rightarrow (A_1, A_2, \dots)$	$X(A) = (\theta, \mathbf{M}, \mathbf{v}_1, \mathbf{v}_2, c)$
R_5 : Instance	$A \rightarrow a$	$X(A) = (\theta, \mathbf{M})$

Table 1 summarizes the definitions, *e.g.*, geometric attributes, of all grammar rules. In the rest of this subsection, we introduce the definitions of five grammar rules.

Grammar Rule R_1 : Layout

The **Layout** rule $R_1 : A \rightarrow (A_0, A_1, \dots, A_m)$ states that a planar surface A_0 is supporting m entities. In this subsection, we assume that all stuff (objects, building, *etc.*) in the scene is standing on the ground. A_0 indicates the ground region in images (*e.g.* grass, road, sidewalk *etc.*), and A_1, \dots, A_m indicates the m children surfaces or composite surfaces produced by other grammar rules. Fig. 6.9 illustrates the use of R_1 , which merges two building blocks/surfaces and the ground. The rule R_1 is used to generate the root node S .

The geometric attributes of S include both global attributes and local attributes defined over its children nodes. The former includes, a list of possible categories, camera focal length f and m Manhattan frames. Each Manhattan frame includes three axes in space that are orthogonal to each other. The latter includes the normal directions of children surfaces, *e.g.*, θ_0 for A_0 , and the contact lines between A_0 and each of the m entities, denoted as \vec{l}_k . Formally, we have $X(S) = (f, m, \mathbf{M}_i, \theta_0, \theta_{ij}, \vec{l}_k, C)$, $i, k = 1, \dots, m$, where θ_{ij} represents one of the normal orientations in the i^{th} children node, C a list of category labels.

We use continuous splines \vec{l}_k to represent contact boundaries between A_0 and $\{A_k\}$, which are assumed to be piece-wise linear. Fig. 6.10 illustrates four typical scenes where contact splines are highlighted in red. A piece-wise linear spline consists of several control points and straight lines between them. Each straight line corresponds to the contact boundary of a planar region. In urban images, a contact line is usually parallel to one of the parallels families falling in the support region. This gives rise to a useful observation: if we can detect local edges in the given planar region and cluster these edges to parallel families, the direction of a contact line can be simply determined. With this observation, we will develop an effective search algorithm for discovering contact splines in Section 6.4.4.

We define the energy function for R_1 from two aspects. Firstly, the normal direction of the surface A_0 and other children surfaces should be as distinct as possible. This is different from the previous works [58, 65] which assume orthogonality between connected surfaces. Secondly, contact lines are likely to go through

VPs that have edges falling in A_k . Thus, we have,

$$E^t(\mathbf{I}, X(A)|R_1) = \sum_{i,j} D^{\cos}(\theta_0, \theta_{ij}) + \lambda^{\text{lay}} \sum_{l \in \vec{l}_k} \min D^{\cos}(l, \mathbf{v})$$

$$\forall \mathbf{v} \in \mathbf{M}, \mathbf{M} \in X(A_k) \quad (6.11)$$

where l indexes the line segment in the spline \vec{l}_k , \mathbf{M} the Manhattan world in $X(A_k)$, D^{\cos} the cosine distance between two directions or two straight lines in 3D space. Note that \mathbf{v} indicates one of the Manhattan axes in the Manhattan world associated with A_k .

Grammar Rule R_2 : Siding

The **siding rule** $R_2 : A \rightarrow (A_1, A_2)$ states that two planar surfaces or composite surfaces of the same label are spatially connected in the scene. The parent node A is a composite surface and the children nodes A_1, A_2 could be planar surfaces or composite surfaces. It requires that children surfaces share the same semantic label (*e.g.* building) but have different normal orientations. These surfaces are usually, but not necessarily, orthogonal with each other.

The attributes of R_2 include $X(A) = \{(\theta_i, \mathbf{M}_i), \vec{l}_k, c\}$, where θ_i is normal direction of the children surface A_i , $i = 1, 2$, \mathbf{M}_i the Manhattan frame associated with A_i , \vec{l}_k the contact line between children surfaces, and c_i the semantic label.

The energy function for R_2 is derived from two aspects. Firstly, two siding surfaces should have as distinct normal as possible, which is the case in most of the urban images.

Secondly, the contact line of A is likely to point to the vertical VP, denoted as \mathbf{v}_0 , as illustrates Fig. 6.9 illustrates. Formally, we have,

$$E^t(\mathbf{I}, X(A)|R_2) = \sum_{i \neq j} D^{\cos}(\theta_i, \theta_j) + \lambda^{\text{sid}} \sum_k D^{\cos}(\vec{l}_k, \mathbf{v}_0) \quad (6.12)$$

where λ^{sid} is a weight constant. Taking the production rule $R_2 : A \rightarrow (A_1, A_2)$ as example, the energy in Eq. (6.12) shall be minimized if A_1 and A_2 are orthogonal and they are split by a ray in image starting from the vertical VP.

Note that the semantic attributes c are used as hard constraints: a graph node of R_2 is only valid when the two children surfaces A_1 and A_2 share the same label.

Grammar Rule R_3 : Affinity

The **affinity rule** $R_3 : A \rightarrow (A_1, A_2)$ states that two planar surfaces have similar appearance and thus should belong to the same planar surface. The children surfaces A_1 and A_2 should be spatially connected in 3D scene. In practice, since they could be disjoint in image due to occlusions, we allow the grouping of disjoint regions by this rule if they have a high affinity in appearance. The attributes of A are defined as $X(A) = (\theta, \mathbf{M}, c)$ where θ is the normal direction, \mathbf{M} the related Manhattan frame, and c the semantic label, which are shared by the two children surfaces.

The grammar rule R_3 requires that the children surfaces A_1 and A_2 should have the same surface normal. Thus, the geometric attributes serve as hard constraints and we only utilize the appearance information to define the energy function $E^t(\mathbf{I}, X(A)|R_3)$.

The energy function for R_3 includes both unary terms and pair-wise terms, all of which are defined over superpixel partition of the parent surface A . Let s and t index two neighbor superpixels, c_s the semantic label of superpixel s . We have,

$$E^t(\mathbf{I}, X(A)|R_3) = \sum_s \phi_s(c_s) + \lambda^{\text{aff}} \sum_{s,t} \mathbf{1}(c_s = c_t) \quad (6.13)$$

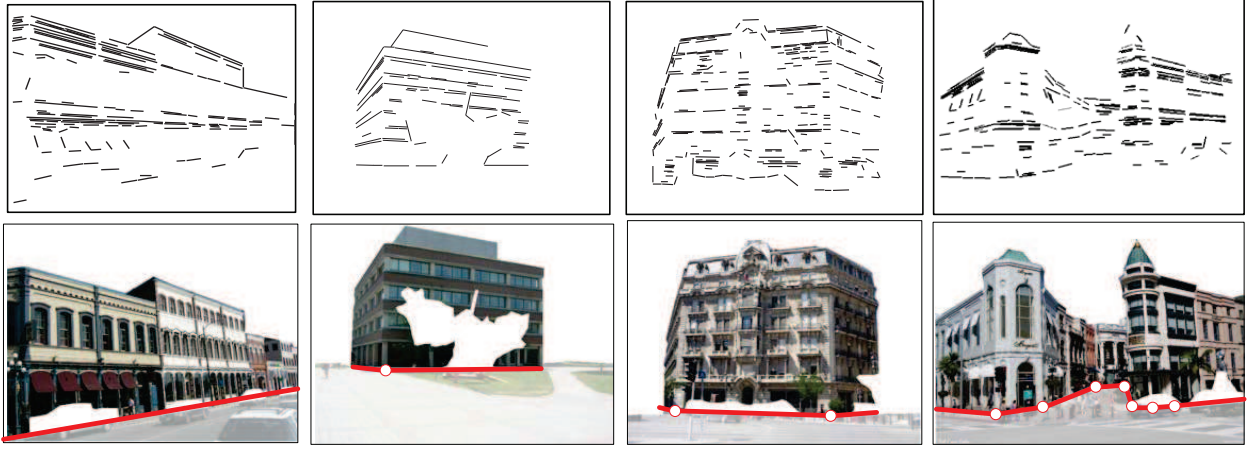


Figure 6.10: Illustration of piece-wise linear spline model for the contact boundaries of composite surfaces that comprise ground plane and buildings. Each spline consists of several control points and straight lines between these points. Note that each straight line correlates with one planar region in the composite surface. Reprinted from Ref. [91] with permission.

where $\phi_s(c_s)$ returns the negative class likelihood ratio, and $\mathbf{1}()$ is an indicator function. Like [145], we estimate $\phi_s(c_s)$ by applying a non-parametric nearest neighbor estimator over training data. The second term is defined as a Potts/Ising model to encourage homogeneousness of labelling.

We estimate surface normal based on edge statistics, as introduced in Section 6.4.1. However, if an image region does not contain any local edges, there is no cue to tell its normal direction directly, and we need to infer its normal from the scene context. In Section 6.4.4, we shall introduce a robust inference method to deal with these uncertainties.

Grammar Rule R_4 : Mesh

The **mesh rule** $R_4 : A \rightarrow (A_1, A_2, A_3, \dots)$ states that multiple surfaces are arranged in a mesh structure. Children surfaces should be spatially connected to each other and share the same normal direction. In perspective geometry, a mesh structure in image plane can be described by two orthogonal VPs. Formally, the attributes of A include: $X(A) = (\theta, \mathbf{M}, \mathbf{v}_1, \mathbf{v}_2, c)$, where $\mathbf{v}_1 = (x_1, y_1, f)$, $\mathbf{v}_2 = (x_2, y_2, f)$ are the coordinates of two VPs, $\theta = \mathbf{v}_1 \times \mathbf{v}_2$ is the normal direction of A , c the semantic label. The children surfaces share the same normal direction θ with A .

The energy function for R_4 is defined over edge statistics. Straight edges in a mesh region usually merge at two VPs. Let $\mathcal{E}(A)$ denote the set of local edges in A , and $\mathbf{l}_j = (x_j, y_j, \vec{d}_j) \in \mathcal{E}$ an edge at the position (x_j, y_j) with the orientation \vec{d}_j . Let v_i denote the image coordinate of the VP \mathbf{v}_i . If an edge \mathbf{l}_j points to \mathbf{v}_i , we have $(x_j, y_j) + \lambda_j^{\text{mes}} \vec{d}_j = v_i$. Thus, we define $E^t(\mathbf{l}, X(A)|R_5)$ as:

$$E^t(\mathbf{l}, X(A)|R_4) = \sum_{\mathbf{l}_j \in \mathcal{E}(A)} \min_{i, \lambda_j^{\text{mes}}} \|v_i - (x_j, y_j) - \lambda_j^{\text{mes}} \vec{d}_j\|^2 \quad (6.14)$$

where $i = 1, 2$. This least square energy term is minimized while all edges in the mesh region exactly point to one of the two VPs, *i.e.* \mathbf{v}_1 or \mathbf{v}_2 .

Grammar Rule R_5 : Instance

An instance rule $R_5 : A \rightarrow a$ instantiates a terminal node, *i.e.* a superpixel or a surface fragment, to image representations, including both texture appearances and edge segments. Fig. 6.9 illustrates how the

grammar rule R_5 links a non-terminal node to two image representations: histogram of oriented gradient (HoG) and straight edge map.

The potential $E^t(\mathbf{I}, X(A)|R_5)$ is defined over two aspects: i) the appearance of individual pixels in the region of A should be homogeneous; ii) the directions of local straight edges should be consistent with the Manhattan frame assigned or inherited from the parent nodes of A . Let \mathbf{I}_i and \mathbf{I}_k denote two neighbor pixels in region A , we have:

$$E^t(\mathbf{I}, X(A)|R_5) = \sum_{i,k} g(\mathbf{I}_i, \mathbf{I}_k) + \lambda^{\text{ins}} \sum_{l \in \mathcal{E}(A)} \min D^{\text{cos}}(l, \mathbf{v})$$

$$\forall \mathbf{v} \in \mathbf{M}, \mathbf{M} \in X(A) \quad (6.15)$$

where $g(\mathbf{I}_i, \mathbf{I}_k)$ returns the negative confidences of two pixels being homogeneous, λ^{ins} is a weight constant. The model $g(\mathbf{I}_i, \mathbf{I}_k)$ is directly estimated by the superpixel partition method [126] with both HoG features and edge features. The second term is used to encourage that all edges in A should be parallel to one of the Manhattan axes in $X(A)$.

Fig. 6.6 shows an exemplar parse graph generated by the proposed grammar. Each grammar rule describes a kind of spatial relationship, *e.g.*, R_1 for supporting, R_2 for being co-block, R_3 and R_4 for being co-planar. These simple rules are capable of producing a large number of tree-structure representations whereas only a portion of them are valid. It is worth noting that the tree structures are augmented to be parse graphs by linking nodes in the same layer that are spatially connected. This graph representation encodes both 2D appearance and 3D geometric properties of the hierarchical scene entities (as introduced in Section 6.4.1).

6.4.4 Inference

The inference algorithm aims to construct an optimal parse graph by sequentially applying the grammar rules to maximize a posterior $P(W|I)$. This task is challenging because: a) the parse graph does not have a pre-defined structure; b) the attribute constraints are essentially high-order.

We adopt a divide and conquer strategy to solve the optimal parse graph. It contains three stages. Firstly, we introduce an efficient algorithm for camera calibration to optimize the term $E(I, \mathcal{A})$ and fix the parameters \mathcal{A} throughout inference. Secondly, we solve the region labeling subproblem to optimization. In particular, we split the nonterminal nodes in a parse graph pg into two subset $V^1 = \{V \in V^N; r(V) = R_4\}$ that contains all the nodes of affinity rules, and $V^2 = \{V \in V^N; r(V) \neq R_4\}$ that contains all other nodes. Thus, we can rewrite the energy term $E(I, pg)$ in Eq. (6.10) as,

$$E(I, pg) = E(I, pg^1) + E(I, pg^2) \quad (6.16)$$

where

$$E(I, pg^1) = \sum_{A \in V^1} E^t(I, X(A); R_4), \quad (6.17)$$

$$E(I, pg^2) = \sum_{A \in V^2} \sum_{k \neq 4} \mathbf{1}(r(A) = k) E^t(I, X(A); R_k). \quad (6.18)$$

The term $E(I, pg^1)$ is a typical MRF type energy function with the regularization of Potts/Ising prior, which can be efficiently solved by the loopy belief propagation (LBP) method [41]. With the trick of Distance Transform, LBP algorithm can converge in $O(nkT)$ time where n is the number of superpixels, k is the number of labels and T is the number of iterations. We use the results of region labeling to initialize the

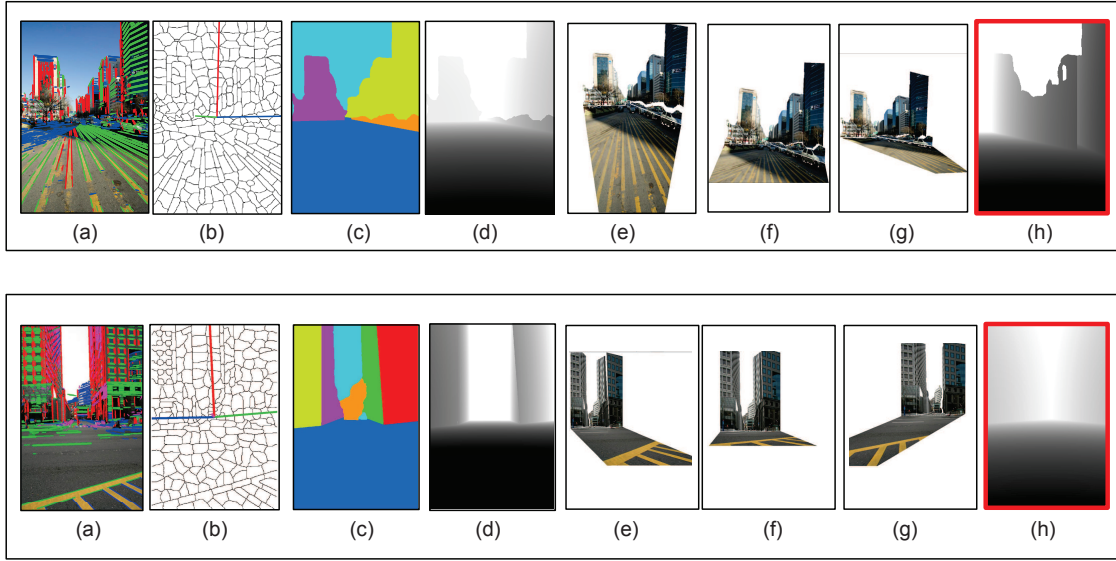


Figure 6.11: Results on urban world. Reprinted from Ref. [91] with permission.

desired parse graph. Finally, a data-driven Monte Carlo Markov Chain (DDMCMC) method is used to sample the posterior probability $P(I|W)$. This is computationally feasible because the solution space has been largely narrowed by the first two stages.

Algorithm 13 summarizes the proposed inference algorithm. It includes two bottom-up computations and an iterative sampling step that simulates the Markov Chain by a set of dynamics.

Algorithm 13: Building Parse Graph via Attributed Grammar.

Input: Single Image

Output: Optimal Parse Graph

- 1 Partition input image into superpixels;
 - 2 Estimate focal length and parallel families;
 - 3 Initialize labeling of superpixels by minimizing $E(I, pg^1)$;
 - 4 Initialize the parse graph pg ;
 - 5 Iterate until convergence,
 - Randomly select one of the five dynamics
 - Make proposals according the select dynamic to reconfigure the current parse graph;
 - Accept the change with a probability
-

Fig. 6.11 show some results on the outside world. For each cell, we can see (a) input image overlaid with families of parallel lines; (b) superpixel partition overlaid with vanishing points; (c) obtained surface normal map; (d) estimated depth map; (e-g) newly synthesized views; (h) depth map by Hoiem *et al.* [65].

6.5 Example III: Human-Centric Indoor Scene Synthesis Using Stochastic Grammar

Recent advances in visual recognition and classification through machine-learning-based computer vision algorithms have produced results comparable to or in some cases exceeding human performance by leveraging large-scale, ground-truth-labeled RGB datasets. However, indoor scene understanding remains a largely unsolved challenge due in part to the current limitations of RGB-D datasets available for training purposes. To address this deficiency, the use of synthetic image datasets as training data has increased.

In this example [123], we illustrate that Attributed AoG can be used to sample and synthesize 3D room layout and 2D images thereof, to obtain large-scale 2D/3D image data with perfect per-pixel ground-truth. An attributed spatial AoG is proposed to represent indoor scenes.

6.5.1 Representation

Formally, the A-AOG is defined as a 5-tuple: $\mathcal{G} = \langle S, V, R, P, E \rangle$, where we use notations S the root node of the scene grammar, V the vertex set, R the production rules, P the probability model defined on the attributed S-AOG, and E the contextual relations represented as horizontal links between nodes in the same layer.

Vertex Set V can be decomposed into a finite set of non-terminal and terminal nodes: $V = V_{NT} \cup V_T$.

- $V_{NT} = V^{And} \cup V^{Or} \cup V^{Set}$. The non-terminal nodes consists of three subsets. i) A set of **And-nodes** V^{And} , in which each node represents a decomposition of a larger entity (e.g., a bedroom) into smaller components (e.g., walls, furniture and supported objects). ii) A set of **Or-nodes** V^{Or} , in which each node branches to alternative decompositions (e.g., an indoor scene can be a bedroom or a living room), enabling the algorithm to reconfigure a scene. iii) A set of **Set nodes** V^{Set} , in which each node represents a nested And-Or relation: a set of Or-nodes serving as child branches are grouped by an And-node, and each child branch may include different numbers of objects.

- $V_T = V_T^r \cup V_T^a$. The terminal nodes consists of two subsets of nodes: regular nodes and address nodes. i) A **regular terminal node** $v \in V_T^r$ represents a spatial entity in a scene (e.g., an office chair in a bedroom) with attributes. In this section, the attributes include internal attributes A_{int} of object sizes (w, l, h) , external attributes A_{ext} of object position (x, y, z) and orientation $(x - y \text{ plane}) \theta$, and sampled human positions A_h . ii) To avoid excessively dense graphs, an **address terminal node** $v \in V_T^a$ is introduced to encode interactions that only occur in a certain context but are absent in all others [48]. It is a pointer to regular terminal nodes, taking values in the set $V_T^r \cup \{\text{nil}\}$, representing supporting or grouping relations as shown in Fig. 6.12.

Contextual Relations E among nodes are represented by the horizontal links in S-AOG forming MRFs on the terminal nodes. To encode the contextual relations, we define different types of potential functions for different cliques. The contextual relations $E = E_f \cup E_o \cup E_g \cup E_r$ are divided into four subsets: i) relations among furniture E_f ; ii) relations between supported objects and their supporting objects E_o (e.g., a monitor on a desk); iii) relations between objects of a functional pair E_g (e.g., a chair and a desk); and iv) relations between furniture and the room E_r . Accordingly, the cliques formed in the terminal layer could also be divided into four subsets: $C = C_f \cup C_o \cup C_g \cup C_r$. Instead of directly capturing the object-object relations, we compute the potentials using affordances as a bridge to characterize the object-human-object relations.

A hierarchical parse tree pt is an instantiation of the S-AOG by selecting a child node for the Or-nodes as well as determining the state of each child node for the Set-nodes. A parse graph pg consists of a parse tree pt and a number of contextual relations E on the parse tree: $pg = (pt, E_{pt})$. Fig. 6.13 illustrates a

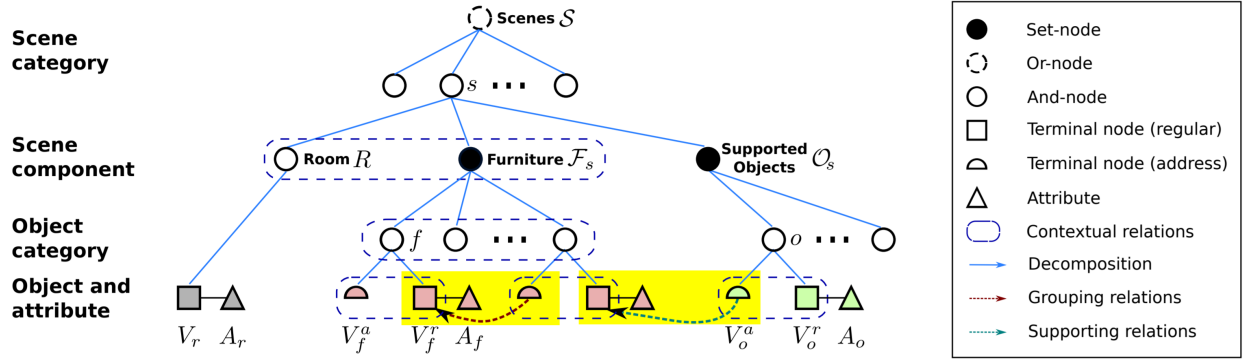


Figure 6.12: Scene grammar as an attributed S-AOG. A scene of different types is decomposed into a room, furniture, and supported objects. Attributes of terminal nodes are internal attributes (sizes), external attributes (positions and orientations), and a human position that interacts with this entity. Furniture and object nodes are combined by an address terminal node and a regular terminal node. A furniture node (*e.g.*, a chair) is grouped with another furniture node (*e.g.*, a desk) pointed by its address terminal node. An object (*e.g.*, a monitor) is supported by the furniture (*e.g.*, a desk) it is pointing to. If the value of the address node is null, the furniture is not grouped with any furniture, or the object is put on the floor. Contextual relations are defined between the room and furniture, between a supported object and supporting furniture, among different pieces of furniture, and among functional groups. Reprinted from Ref. [123] with permission.

simple example of a parse graph and four types of cliques formed in the terminal layer.

6.5.2 Probabilistic Formulation

A scene configuration is represented by a parse graph pg , including objects in the scene and associated attributes. The prior probability of pg generated by an S-AOG parameterized by Θ is formulated as a Gibbs distribution:

$$p(pg|\Theta) = \frac{1}{Z} \exp\{-\mathcal{E}(pg|\Theta)\} \quad (6.19)$$

$$= \frac{1}{Z} \exp\{-\mathcal{E}(pt|\Theta) - \mathcal{E}(E_{pt}|\Theta)\}, \quad (6.20)$$

where $\mathcal{E}(pg|\Theta)$ is the energy function of a parse graph, $\mathcal{E}(pt|\Theta)$ is the energy function of a parse tree, and $\mathcal{E}(E_{pt}|\Theta)$ is the energy term of the contextual relations.

$\mathcal{E}(pt|\Theta)$ can be further decomposed into the energy functions of different types of non-terminal nodes, and the energy functions of internal attributes of both regular and address terminal nodes:

$$\mathcal{E}(pt|\Theta) = \underbrace{\sum_{v \in V^{Or}} \mathcal{E}_{\Theta}^{Or}(v)}_{\text{non-terminal nodes}} + \underbrace{\sum_{v \in V^{Set}} \mathcal{E}_{\Theta}^{Set}(v)}_{\text{terminal nodes}} + \sum_{v \in V_T^r} \mathcal{E}_{\Theta}^{A_{in}}(v), \quad (6.21)$$

where the choice of the child node of an Or-node $v \in V^{Or}$ and the child branch of a Set-node $v \in V^{Set}$ follow different multinomial distributions. Since the And-nodes are deterministically expanded, we do not have an energy term for the And-nodes here. The internal attributes A_{in} (size) of terminal nodes follow a non-parametric probability distribution learned by kernel density estimation.

$\mathcal{E}(E_{pt}|\Theta)$ combines the potentials of the four types of cliques formed in the terminal layer, integrating

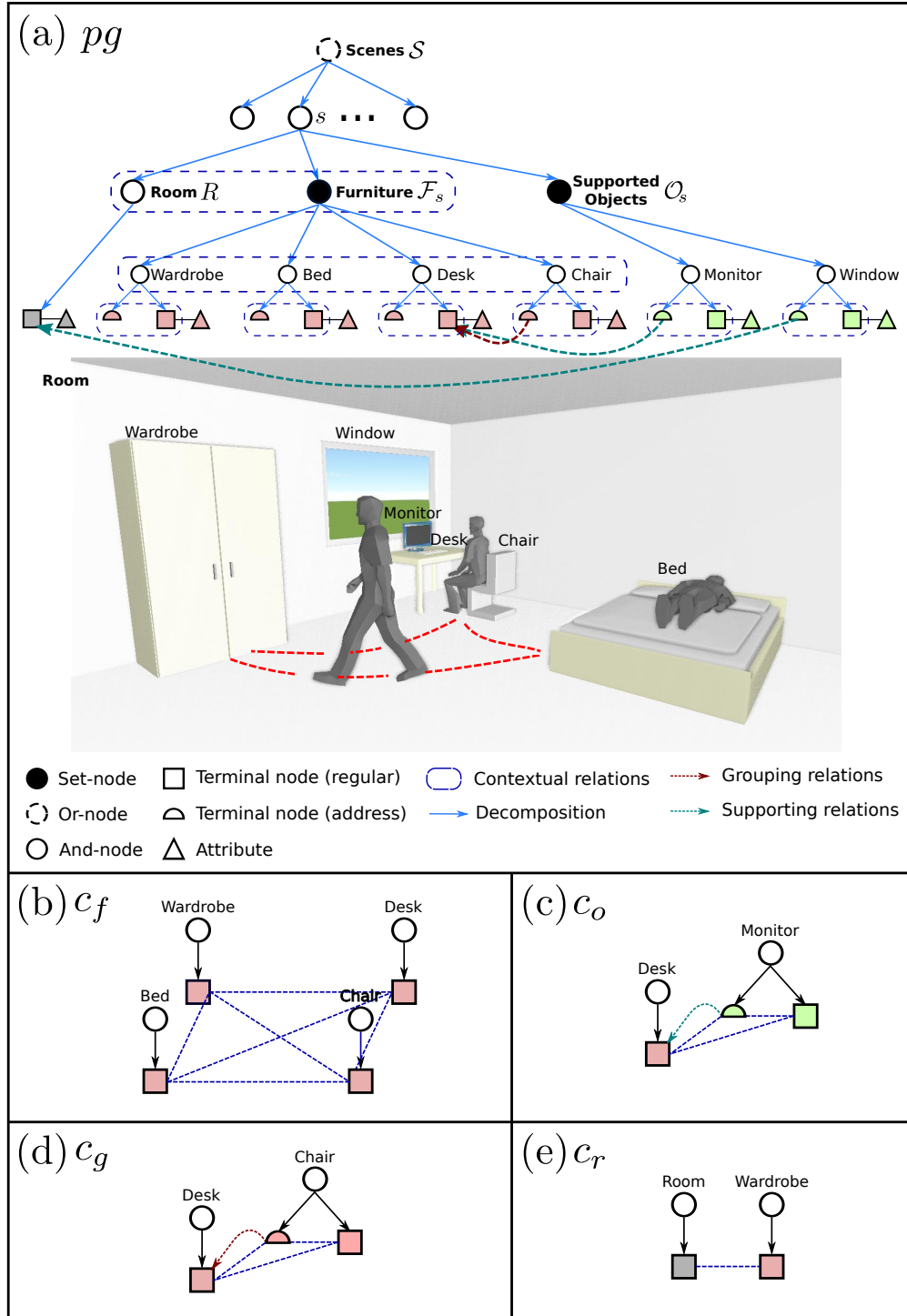


Figure 6.13: (a) A simplified example of a parse graph of a bedroom. The terminal nodes of the parse graph form an MRF in the terminal layer. Cliques are formed by the contextual relations projected to the terminal layer. Examples of the four types of cliques are shown in (b)-(e), representing four different types of contextual relations. Reprinted from Ref. [123] with permission.

human attributes and external attributes of regular terminal nodes:

$$p(E_{pt}|\Theta) = \frac{1}{Z} \exp\{-\mathcal{E}(E_{pt}|\Theta)\} \quad (6.22)$$

$$= \prod_{c \in C_f} \phi_f(c) \prod_{c \in C_o} \phi_o(c) \prod_{c \in C_g} \phi_g(c) \prod_{c \in C_r} \phi_r(c). \quad (6.23)$$

Human Centric Potential Functions:

- Potential function $\phi_f(c)$ is defined on relations between furniture (Fig. 6.13(b)). The clique $c = \{f_i\} \in C_f$ includes all the terminal nodes representing furniture:

$$\phi_f(c) = \frac{1}{Z} \exp\{-\lambda_f \cdot \langle \sum_{f_i \neq f_j} l_{col}(f_i, f_j), l_{ent}(c) \rangle\}, \quad (6.24)$$

where λ_f is a weight vector, $\langle \cdot, \cdot \rangle$ denotes a vector, and the cost function $l_{col}(f_i, f_j)$ is the overlapping volume of the two pieces of furniture, serving as the penalty of collision. The cost function $l_{ent}(c) = -H(\Gamma) = \sum_i p(\gamma_i) \log p(\gamma_i)$ yields better utility of the room space by sampling human trajectories, where Γ is the set of planned trajectories in the room, and $H(\Gamma)$ is the entropy. The trajectory probability map is first obtained by planning a trajectory γ_i from the center of every piece of furniture to another one using bi-directional rapidly-exploring random tree (RRT) [82], which forms a heatmap. The entropy is computed from the heatmap as shown in Fig. 6.14.

- Potential function $\phi_o(c)$ is defined on relations between a supported object and the supporting furniture (Fig. 6.13(c)). A clique $c = \{f, a, o\} \in C_o$ includes a supported object terminal node o , the address node a connected to the object, and the furniture terminal node f pointed by a :

$$\phi_o(c) = \frac{1}{Z} \exp\{-\lambda_o \cdot \langle l_{hum}(f, o), l_{add}(a) \rangle\}, \quad (6.25)$$

where the cost function $l_{hum}(f, o)$ defines the human usability cost—a favorable human position should enable an agent to access or use both the furniture and the object. To compute the usability cost, human positions h_i^o are first sampled based on position, orientation, and the affordance map of the supported object. Given a piece of furniture, the probability of the human positions is then computed by:

$$l_{hum}(f, o) = \max_i p(h_i^o | f). \quad (6.26)$$

The cost function $l_{add}(a)$ is the negative log probability of an address node $v \in V_T^a$, treated as a certain regular terminal node, following a multinomial distribution.

- Potential function $\phi_g(c)$ is defined on functional grouping relations between furniture (Fig. 6.13(d)). A clique $c = \{f_i, a, f_j\} \in C_g$ consists of terminal nodes of a core functional furniture f_i , pointed by the address node a of an associated furniture f_j . The grouping relation potential is defined similarly to the supporting relation potential

$$\phi_g(c) = \frac{1}{Z} \exp\{-\lambda_c \cdot \langle l_{hum}(f_i, f_j), l_{add}(a) \rangle\}. \quad (6.27)$$

Other Potential Functions:

- Potential function $\phi_r(c)$ is defined on relations between the room and furniture (Fig. 6.13(e)). A clique $c = \{f, r\} \in C_r$ includes a terminal node f and r representing a piece of furniture and a room, respectively. The potential is defined as

$$\phi_r(c) = \frac{1}{Z} \exp\{-\lambda_r \cdot \langle l_{dis}(f, r), l_{ori}(f, r) \rangle\}, \quad (6.28)$$

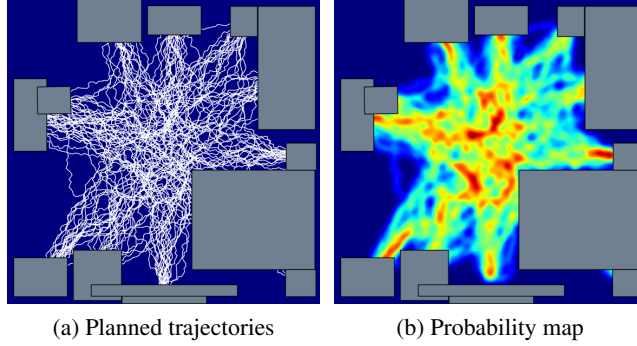


Figure 6.14: Given a scene configuration, we use bi-directional RRT to plan from every piece of furniture to another, generating a human activity probability map. Reprinted from Ref. [123] with permission.

where the distance cost function is defined as $l_{\text{dis}}(f, r) = -\log p(d|\Theta)$, in which $d \sim \ln \mathcal{N}(\mu, \sigma^2)$ is the distance between the furniture and the nearest wall modeled by a log normal distribution. The orientation cost function is defined as $l_{\text{ori}}(f, r) = -\log p(\theta|\Theta)$, where $\theta \sim p(\mu, \kappa) = \frac{e^{\kappa \cos(x-\mu)}}{2\pi I_0(\kappa)}$ is the relative orientation between the model and the nearest wall modeled by a von Mises distribution.

6.5.3 Synthesizing Scene Configurations

Synthesizing scene configurations is accomplished by sampling a parse graph pg from the prior probability $p(pg|\Theta)$ defined by the S-AOG. The structure of a parse tree pt (*i.e.*, the selection of Or-nodes and child branches of Set-nodes) and the internal attributes (sizes) of objects can be easily sampled from the closed-form distributions or non-parametric distributions. However, the external attributes (positions and orientations) of objects are constrained by multiple potential functions, hence they are too complicated to be directly sampled from. Here, we utilize a Markov chain Monte Carlo (MCMC) sampler to draw a typical state in the distribution. The process of each sampling can be divided into two major steps:

1. Directly sample the structure of pt and internal attributes A_{in} : (i) sample the child node for the Or-nodes; (ii) determine the state of each child branch of the Set-nodes; and (iii) for each regular terminal node, sample the sizes and human positions from learned distributions.
2. Use an MCMC scheme to sample the values of address nodes V^a and external attributes A_{ex} by making proposal moves. A sample will be chosen after the Markov chain converges.

We design two simple types of Markov chain dynamics which are used at random with probabilities $q_i, i = 1, 2$ to make proposal moves:

- Dynamics q_1 : translation of objects. This dynamic chooses a regular terminal node, and samples a new position based on the current position x : $x \rightarrow x + \delta x$, where δx follows a bivariate normal distribution.
- Dynamics q_2 : rotation of objects. This dynamic chooses a regular terminal node, and samples a new orientation based on the current orientation of the object: $\theta \rightarrow \theta + \delta \theta$, where $\delta \theta$ follows a normal distribution.

Adopting the Metropolis-Hastings algorithm, the proposed new parse graph pg' is accepted according to the following acceptance probability:

$$\alpha(pg'|pg, \Theta) = \min(1, \frac{p(pg'|\Theta)p(pg|pg')}{p(pg|\Theta)p(pg'|pg)}) \quad (6.29)$$

$$= \min(1, \exp(\mathcal{E}(pg|\Theta) - \mathcal{E}(pg'|\Theta))), \quad (6.30)$$

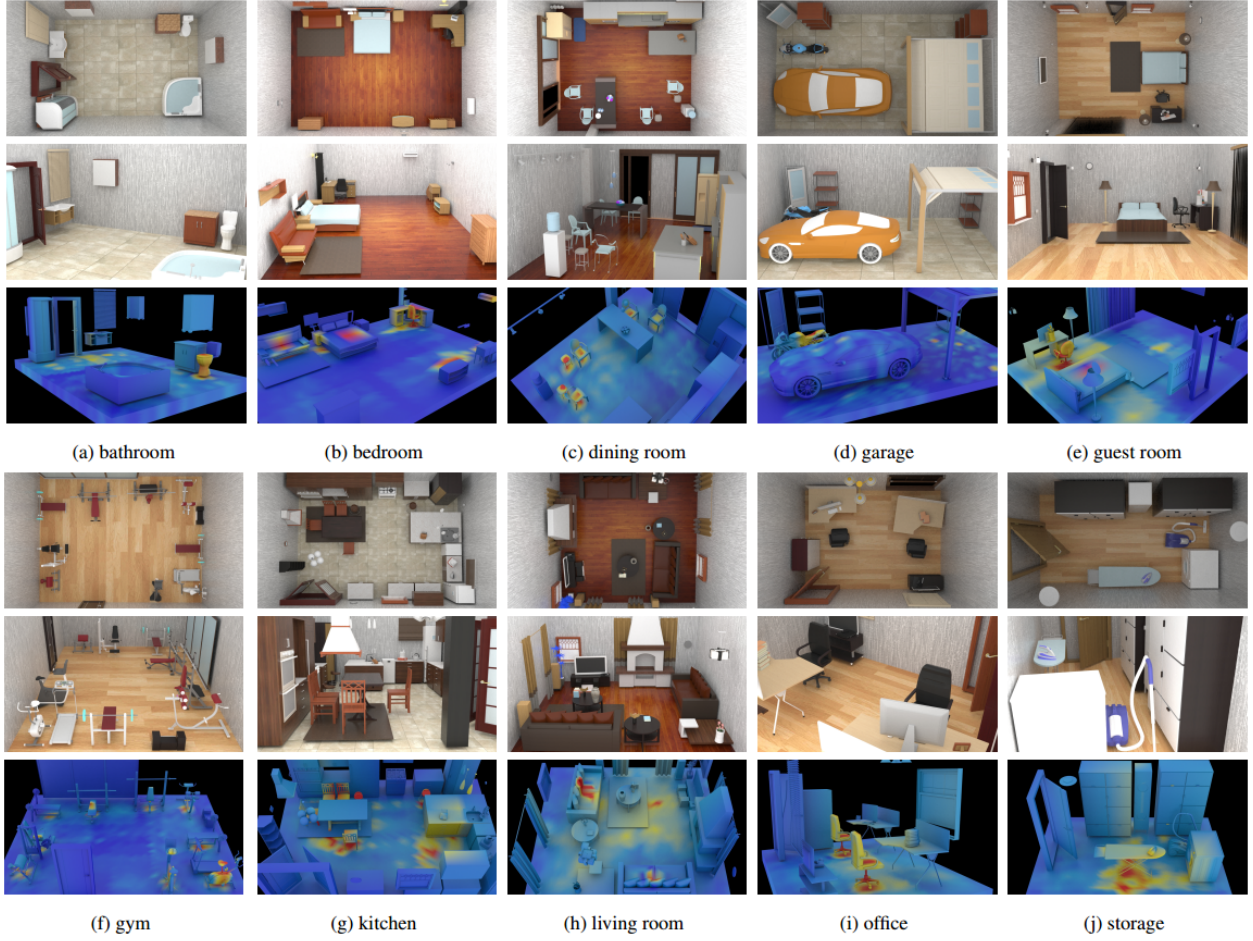


Figure 6.15: Examples of scenes in ten different categories. Top: top-view. Middle: a side-view. Bottom: affordance heatmap. Reprinted from Ref. [123] with permission.

where the proposal probability rate is canceled since the proposal moves are symmetric in probability. A simulated annealing scheme is adopted to obtain samples with high probability.

Some qualitative results are shown in Fig. 6.15.

6.6 Example IV: Joint Parsing of Human Attributes, Parts and Pose

In this example [107], we show that Attribute AoG can be used to explicitly represent pose, parts and attributes and tackle the tasks of attribute recognition, pose estimation and part localization jointly. Fig. 6.16 shows an example.

Again, we denote the A-AOG by a 5-tuple:

$$A - AOG = \langle S, V, E, X, \mathcal{P} \rangle .$$

1) The vertex set $V = V_{\text{and}} \cup V_{\text{or}} \cup V_T$ consists of three subsets: (i) a set of and-nodes V_{and} for decomposition in the phrase structure grammar; (ii) a set of or-nodes V_{or} for branching to alternative decompositions and thus enabling reconfiguration; and (iii) a set of terminal nodes V_T for grounding on image features. Each node $v \in V$ represents a body part of different granularities and has state variables designating the location

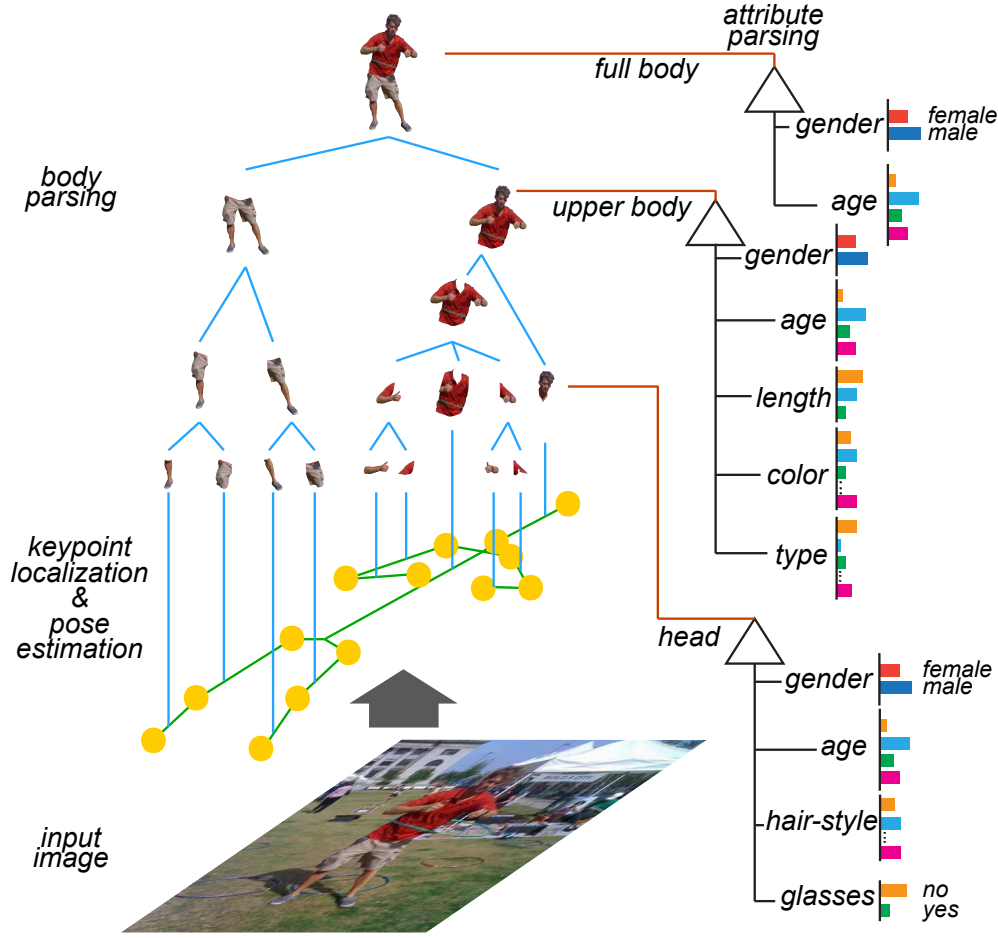


Figure 6.16: An attributed parse graph for a human image includes three components in colored edges: (i) The hierarchical whole-part decomposition (blue); (ii) the articulation of body parts (green); and (iii) the attributes associated with each node in the hierarchy (red). The parse graph also includes the probabilities and thus uncertainty at each node for attributes. Reprinted from Ref. [107] with permission of IEEE, © 2017.

(x, y) . In fact, the state variables are the geometric attributes which are propagated between parent-children nodes in the hierarchy. However, here we treat them separately from the human attributes. We define 14 atomic terminal parts, *i.e.* head, torso, l.shoulder, r.shoulder, l.upper arm, l.lower arm, r.upper arm, r.lower arm, l.hip, r.hip, l.upper leg, l.lower leg, r.upper leg and r.lower leg. These parts are defined as terminal nodes (V_T) in the grammar. We then define non-terminal nodes ($V_{\bar{T}}$), upper body and lower body by combining terminal parts. Upper body part includes head, torso, shoulders, and arms. Lower body part includes hips and legs. The root part, *i.e.* full body, is defined by upper body and lower body. We illustrate the defined grammar in Fig. 6.17.

2) The edge set $E = E_{\text{psg}} \cup E_{\text{dg}}$ consists of two subsets: (i) a set of edges with phrase structure grammar E_{psg} ; and (ii) a set of edges with dependency grammar E_{dg} .

3) The attribute set $X = \{x_1, \dots, x_9\}$ are associated with nodes in V .

4) \mathcal{P} is the probability model on the graphical representation.

The goal is to find the most probable parse graph pg from the designed grammar model given an image I . The probability model over the parse graph is formulated in a Bayesian framework, which computes the joint posterior as the product of a likelihood and prior probability, and equivalently represented as the

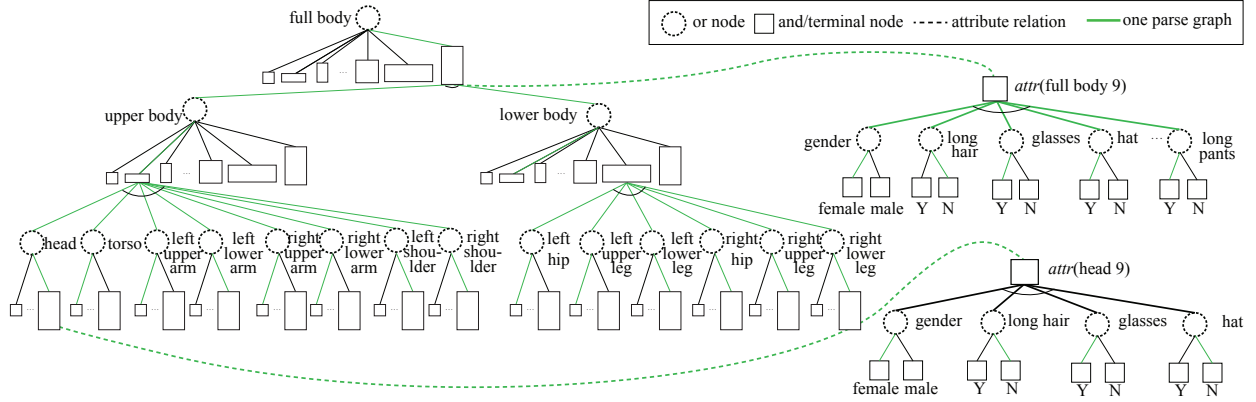


Figure 6.17: We define 14 atomic parts, 2 mid-level parts, and a root part. We define 9 part types for each part. Each attribute has corresponding parts, and connected with part types through attribute relations. In this figure, we only show attributes of head and full body, and draw phrase structure grammar relation for better illustration. Reprinted from Ref. [107] with permission of IEEE, © 2017.

following the Gibbs distribution

$$\begin{aligned} P(pg|I; \lambda) &\propto P(I|pg; \lambda)P(pg; \lambda) \\ &= \frac{1}{Z} \exp\{-\mathcal{E}(I|pg; \lambda) - \mathcal{E}(pg; \lambda)\} \end{aligned} \quad (6.31)$$

The model parameters are denoted as λ . The energy functions \mathcal{E} are further decomposed into a set of potential functions. These potentials constrain all aspects of the grammar. The likelihood term describes appearance response and is decomposed into part and attribute appearance.

$$-\mathcal{E}(I|pg; \lambda) = -\mathcal{E}_{\text{app}}^V(I|pg; \lambda) - \mathcal{E}_{\text{app}}^X(I|pg; \lambda) \quad (6.32)$$

$\mathcal{E}_{\text{app}}^V(I|pg; \lambda)$ and $\mathcal{E}_{\text{app}}^X(I|pg; \lambda)$ are appearance terms for part and attribute respectively. The prior term is used to describe relations in grammar. It is also decomposed into part and attribute relations.

$$-\mathcal{E}(pg; \lambda) = -\mathcal{E}_{\text{rel}}^V(pg; \lambda) - \mathcal{E}_{\text{rel}}^X(pg; \lambda) \quad (6.33)$$

$\mathcal{E}_{\text{rel}}^V(pg; \lambda)$ and $\mathcal{E}_{\text{rel}}^X(pg; \lambda)$ are relation terms for part and attribute respectively. We rewrite Equation 6.31 as

$$\begin{aligned} P(pg|I; \lambda) &= \frac{1}{Z} \exp\{-\mathcal{E}_{\text{app}}^V(I|pg; \lambda) - \mathcal{E}_{\text{app}}^X(I|pg; \lambda) \\ &\quad - \mathcal{E}_{\text{rel}}^V(pg; \lambda) - \mathcal{E}_{\text{rel}}^X(pg; \lambda)\} \end{aligned} \quad (6.34)$$

We, then, can express energy terms as scoring functions.

$$\begin{aligned} S(pg, I) &= -\mathcal{E}_{\text{app}}^V(I|pg) - \mathcal{E}_{\text{app}}^X(I|pg) - \mathcal{E}_{\text{rel}}^V(pg) - \mathcal{E}_{\text{rel}}^X(pg) \\ &= S_{\text{app}}^V(I, pg) + S_{\text{app}}^X(I, pg) + S_{\text{rel}}^V(pg) + S_{\text{rel}}^X(pg) \end{aligned} \quad (6.35)$$

We now have four scoring functions.

$$S_{\text{app}}^V(I, pg) \quad \text{part appearance score function}$$

$$\begin{aligned}
S_{\text{app}}^X(I, pg) & \text{ attribute appearance score function} \\
S_{\text{rel}}^V(pg) & \text{ part relation score function} \\
S_{\text{rel}}^X(pg) & \text{ attribute relation score function}
\end{aligned}$$

For the part model, one can refer to [107] for more details. Here we focus on the attribute model. Previous attribute approaches [10, 17, 179] use all defined parts (or poselets) for attribute classifications, and it means they assumed attributes are related to all body parts. However, we can simply know some parts may not be related with such attributes, and it might hurt attribute prediction if we classify attributes with unrelated parts. For examples, ‘glasses’ is not related to ‘lower body’ parts and ‘t-shirt’ is not related to ‘head’ or ‘lower body’ parts. In contrast, ‘long-hair’ attribute will be highly related to ‘head’ part. Therefore, we need to learn how attributes and parts are related. We define the set of attributes for each part v and denote them by $X(v)$. Then, $X(v)$ includes related attributes for part v .

Attribute Node. First, we can treat the set of attributes $X(v)$ for part v as a two-layer simple graph which follows and-or grammar rules as illustrated on the right side in Fig. 6.17. The root node of $X(v)$ is described by and-rule in and-or grammar. It includes corresponding attributes for part v as its child nodes. Then, each attribute includes attribute types as child nodes, it follows or-rule. It has two attribute types for binary attribute class, such as ‘gender’ and ‘wearing t-shirt’, or has more than two for multi-class attributes, such as ‘cloth types’ or ‘age’. It can be described in the production rule. For example, when part v has two corresponding attributes, ‘gender’ and ‘wearing t-shirt’, ‘gender’ can have types ‘male’ and ‘female’. ‘wearing t-shirt’ has two child nodes, ‘yes’ and ‘no’. In production rule, it could be written as follow,

attribute production rule	example
$X(v) \rightarrow \{X_1, X_2\}$	$X(v) \rightarrow \{\text{Gender, T-shirt}\}$
$X_1 \rightarrow X_{11} X_{12}$	$\text{Gender} \rightarrow \text{Female} \text{Male}$
$X_2 \rightarrow X_{21} X_{22}$	$\text{T-shirt} \rightarrow \text{Yes} \text{No}$

Attribute Relation Model. Each attribute node $X(v_i)$ is linked to part v_i through a relation as shown by a dashed line in Fig. 6.17. This relation describes properties of part node v_i and reflects the co-occurrence frequency of the attribute given the part type. For example, let the specific part type of node v (= upper body) have an appearance that is blouse-like. This will occur more frequently with female than male, and therefore the model should favor selecting this part when there is strong evidence for the female attribute. It is described by score function $S_{\text{rel}}^X(v_i, X(v_i))$, and S_{rel}^X over pg is computed as

$$S_{\text{rel}}^X(pg) = \sum_{v_i \in V(pg)} S_{\text{rel}}^X(v_i, X(v_i)) \quad (6.36)$$

Attribute Appearance Model. We define appearance model for attribute corresponding to the scoring function $S_{\text{app}}^X(pg, I)$, and it can be computed from attribute score function $S_{\text{app}}^X(X(v_i), I)$. It indicates the local appearance response of attributes of part v_i at image path centered at $v_i = (x_i, y_i)$. The score S_{app}^X over pg is computed as

$$S_{\text{app}}^X(pg, I) = \sum_{v_i \in V(pg)} S_{\text{app}}^X(X(v_i), I) \quad (6.37)$$

For the learning of attribute model, we divide it into two parts.

Inspired by R-CNN, we design a part-attribute network to generate likelihood for all part-attribute combinations using a classification layer and predict positions of parts using a regression layer.

The part-attribute relation describes the compatibility relationship between part and attribute. In most cases, the attributes are annotated unknown because of the invisible specific parts. For example, the attribute

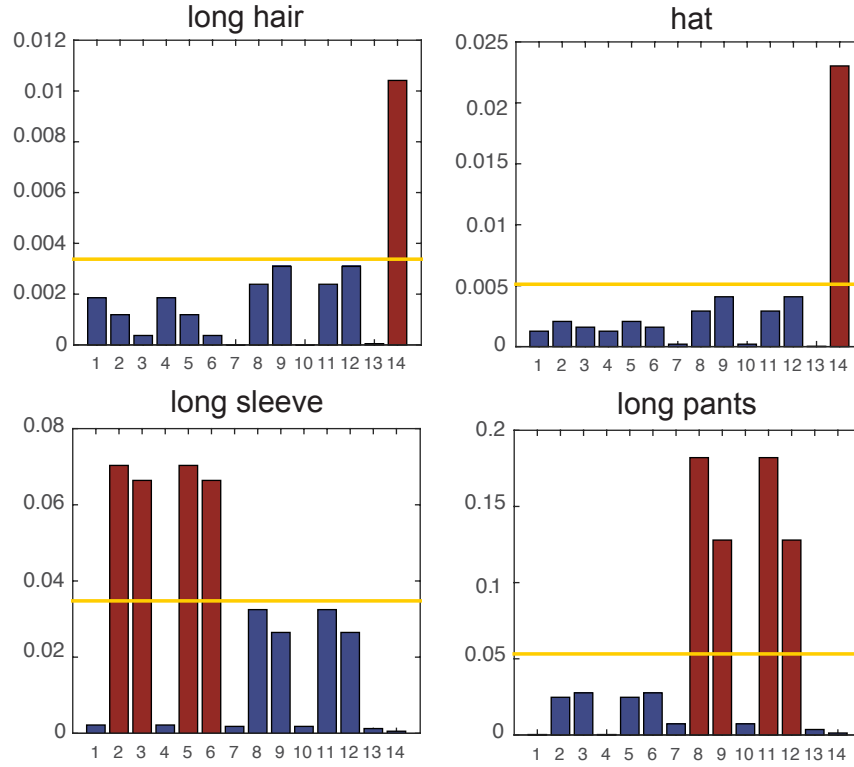


Figure 6.18: **Part-attribute relation.** We compute mutual information between atomic parts and attribute to find the parts that contribute the most to each attribute. If the mutual information value is higher than the mean value (the yellow lines), we consider the part to be associated with the attribute (the red bars). We only compute for 14 terminal parts. The mid-level and root parts will synthesize the attribute relations from child nodes. Please see the text for details and part indexes. Reprinted from Ref. [107] with permission of IEEE, © 2017.

'long pants' of most images is unknown because only the upper body is visible. Thus, these annotations actually provide strong evidence about which parts play a role in distinguishing the attribute. To find the associated part for each attribute, we compute mutual information between attributes with label 'known' or 'unknown' and 14 atomic parts with label 'visible' or 'invisible'. We show the computed mutual information in Fig. 6.18 for four attributes. For each attribute a_j , we pick the parts of which the values are above the mean value as the associated part set. If a part is picked then its parent part will also be picked. For example, 'leg' is the related part for 'jeans', and 'lower body' and 'full body' are considered as related parts for 'jeans' as well. We denote corresponding attributes for each part by $X(v)$.

6.7 Summarization

The exemplar tasks show the strong representation power of the Attributed And-Or Graph. A-AOG is one of the efficient ways to explicitly model the entities and hierarchical structures of images and scenes with the visual attributes. It applies statistical models to do the inference and sampling, making it generalizable to applications like visual recognition, image parsing, 3D reconstruction, and scene synthesis. One of the limitations of the A-AOG is that it lacks a principled method to learn the attribute categories and distributions from the data automatically, which can be compromised by the data-driven methods.

Pursuing a unified and generalizable representation to model the structure of natural and man-made worlds is still the pursuit of our team. Although there are still arguments about whether we should model the visual attributes explicitly or implicitly, by categorical labels or continuous values, we believe our model can shed a light into the correct path of understanding and interpreting the world.

Temporal And-Or Graph

7.1 Introduction

Understanding activities is one of the most fundamental problems in artificial intelligence and computer vision. As the most readily available learning source, videos of daily human activities capture a large variety of actions and events, from causal transition in the physical world to interactions of a group of people following certain social protocols. These action and event concepts constitute a large portion of human visual knowledge. In a short recording, we human can recognize and infer a lot of information regarding the past, present and even the future.

For example, as shown in Fig.7.1, with the person standing up and reaching the trashcan with a cup in his hand, we can infer that he is probably aiming to empty his cup for refilling. Further, after he opened the tea can, we can easily predict that he is refilling tea even though we haven't seen him pouring water into the cup yet. Making these kinds of inference seems easy for human, however, is extremely difficult for vision systems. As the additional temporal dimension brings more uncertainties, such an inference chain is inherently more difficult to model when handling temporal sequences than static images.

To really capture the essence of the actions and events, we need to (i) have a thorough understanding of the environment in a static view (2D and 3D), (ii) understand the unit-actions human perform, (iii) capture temporal patterns of human actions and (iv) provide reasoning on why this event is happening. We focus on the later three key requirements in this chapter as thorough scene understanding has been partially discussed in Chapter 3 and will be one of the main focuses in Chapter 9.

With the first requirement having more of a computer vision flavour, the others are not necessarily limited to the perception domain. Actions are hard to recognize in nature. With subtle differences, similar poses may actually be observed in completely different poses. As these contextual indicators are oftentimes small and occluded in videos, a good model is strongly desired for correctly understanding atomic actions. We focus on the 4D-HOI modeling in Sec.7.2 and introduce the general concepts and principles we need to follow when tackling the atomic-action recognition problem.

Building up to our ultimate goals, capturing temporal patterns of events is also not an easy task. This is caused by the flexibility of task execution and multi-tasking nature of us human. The same event can have various time-span depending on the performer's speed, which might be affected by latent factors like mood. Multiple events can also happen interchangeably with random insertions between each other. And the trickiest problem is that these events are not always repetition of some routines. This is correlated to two systems, (i) routines and social norms (U) and (ii) values and goals (V), that guide our actions in daily life. These two systems cooperate seamlessly and account for human activities jointly. We leave the interaction and connection between the two systems to future discussions for now. However, we address that this

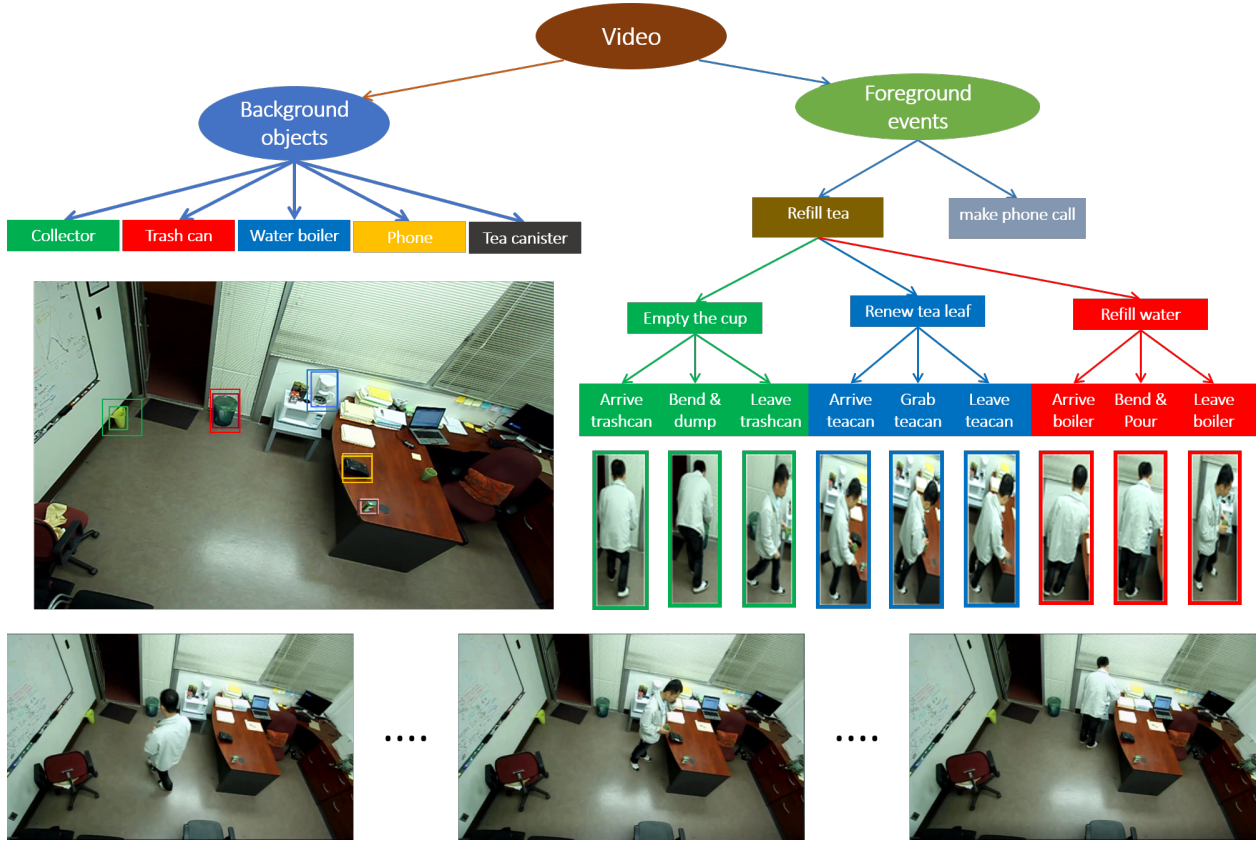


Figure 7.1: An example recording of daily activities performed in office scenario. The original video sequence (bottom), the detailed understanding of the vent (top).

ability to reason on both realms is one crucial part of human intelligence having significant impact on social development [23], and therefore necessary for us to consider in intelligent systems. In this chapter, we share insights by illustrating the Temporal And-Or Graph for both modeling daily life routines and generating goal-oriented understanding of events. We describe the representation of events under the T-AOG setting in Sec.7.3, parsing and inference in Sec.7.4 and learning these grammar-based representations in Sec.7.5.

7.2 Atomic Action Models

Atomic actions are the building blocks of daily activities in temporal grammar. For example, drinking water can be loosely decomposed into get-cup, drink-from-cup, and put-cup. In spatial grammar, atomic actions provide constraints between the human pose and the 3D environment. With a proper model, we can greatly improve holistic scene understanding with the understanding of atomic actions.

Each atomic action is defined as a set of geometrical relationships in human-object interaction (HOI) or hand-object interaction (hoi). For example, *holding-cup* contains the necessary geometrical relations between fingers and a cup that constitute the holding action, and *sitting* contains the relations between a person and a sitable object.

HOI and hoi constitute the two levels of relations in atomic actions.

- An HOI describes **coarse** relations, where the interaction is mostly easily detectable from third-view













Status of person	Symbols	Examples	Status of objects	Examples
Stand(P1)			On (phone)	
Stretch(P1)			Off (phone)	
Bend (P1)			On(screen)	
Sit (P2)			Off(screen)	

Figure 7.2: The unary relations or fluents. The left are four fluents of agent: ‘Stand’, ‘Stretch’, ‘Bend’ and ‘Sit’. The right shows two fluents (‘On’ and ‘Off’) of a phone and a laptop screen. Reprinted from Ref. [110] with permission.









Binary Fluent (A,B)	Touch (A,B)	Near (A,B)	Occlude (A,B)	In(A,B)
Symbols				
Examples				

Figure 7.3: The binary spatial relations between agents (or parts) and background objects. Reprinted from Ref. [110] with permission.

cameras. The set of HOI includes atomic actions such as sitting, standing, pouring-water, making-phone-call, *etc.* In holistic scene understanding [19], an HOI action is usually used to help optimize human pose estimation and object detection/localization by perturbing the corresponding detections to fit HOI relations.

- An hoi, on the other hand, describes **fine-grained** relations that are hard to detect from distance. The set of hoi includes holding, pinching, and different types of grasping. In practice, both the hand and the object in an hoi are hard to detect due to their size and self-occlusion. Therefore, the hoi is usually inferred from context, and the details are hallucinated for reasonable scene reconstruction.

7.2.1 2D HOI in Time - A Simplified Atomic Action Model

To model an event in a video, a Temporal And-Or Graph (T-AOG) can be used to represent the hierarchical composition of events and the temporal relations between sub-events.

Atomic actions	Grounded relations	Symbols		Video examples
		Foreground	Background	
ShakeHands(P1,P2)	Near(P1,P2) And Touch(P1.hand, P2.hand)			
UseDispenser(P3)	Reach(P3) And Near(P3,A) And Touch(P3.hand,A)			
PickUpPhone(P4)	Near(P4,B) And On(P4.hand,B)			

Figure 7.4: Some atomic actions. Each atomic action is defined on a set of grounded relations shown by 2 half circles. Unary relations ‘Bend’ and ‘On’ are defined in Figure 7.2. Binary relations ‘Near’ and ‘Touch’ are defined in Figure 7.3. For the atomic action ‘ShakeHands’, when P1 is considered as the agent, P2 is regarded as object and vice versa. Reprinted/modified from Ref. [110] with permission.

The T-AOG is defined on a set of geometrical relations that can be easily detected from video. We call these relations the *grounded relations*. In addition to the HOI relations $r(A, B)$ described above, we also include a unary relation $r(A)$ as a time-varying property of the agent of object A in the scene. It could be the agent pose, *e.g.* Stand(person1) or the object state *e.g.* Open(door).

The unary relations include existence (whether an agent or an object is out of picture), pose (the pose of an agent, can include zero or many of stand, bend, sit, raise-arm, *etc.*), and on (whether an object is turned on). The binary relations include near, in, touch and occlude.

An atomic action is a vector of grounded relations $a = (r_1, \dots, r_J)$ that happen sequentially in the joint domain of space and time. Figure 7.4 shows three atomic actions defined on the grounded relations.

An atomic action is detected when all its relations are detected with probability higher than a given threshold, and the probability of the atomic action is computed as the product of the probabilities of all its constituent relations. An atomic action $a = (r_1, \dots, r_J)$, has the following probability given a short video snippet $I_{1:t}$,

$$p(a \mid I_{1:t}) = \frac{1}{Z} \prod_{j=1}^J p(r_j) \propto \exp\{-E(a)\} \quad (7.1)$$

where

$$E(a) = - \sum_{j=1}^J \log p(r_j)$$

is the energy of a and Z is the normalizing constant for all atomic actions.

In the T-AOG, the And-nodes define the temporal order of its children-nodes, and the Set-nodes representing atomic actions model the lasting time of the atomic action by the frequency of its production rules. The temporal relations between multiple nodes are modeled by non-parametric filters to augment the T-AOG.

Figure 7.16 is an illustration of the T-AOG representing the temporal relations. E_1 is an And-node and A , B and C are three sub-nodes; τ_A , τ_B and τ_C are the lasting time of A , B and C , respectively. There is a constraint between the lasting time of A , B and C . For example, when an agent does event E_1 in a hurry, the lasting time of A , B and C will be shorter than usual, while the ratio of the lasting time between A , B and C will remain stable. This relation r is modeled by a distribution of a function response over the nodes included in the relation. We use $\tau_{E_1} = (\tau_A, \tau_B, \tau_C)$ to represent the lasting time of E_1 , and $F = (F_1, F_2, F_3)$ to represent the function on which the response of τ_{E_1} is modeled, F could be regarded as a filter and $\langle \tau_{E_1}, F \rangle$ could be regarded as a filter response. We use a histogram to model the distribution of the response, and the F^* , on which the distribution of the training data's response has the minimum entropy, are selected to model the relation as in [185]. Given τ and F^* , the probability of the relation r is

$$p(r) \sim h(\langle \tau, F^* \rangle), \quad (7.2)$$

where h is the histogram of the training data's response over F^* . One may use multiple F to model the relations if needed.

7.2.2 Modeling Human-Object Interaction in 3D and Time

When an image or a video is taken, the 3D world is projected on a 2D surface known as the canvas. Important depth information is lost during the project. In order to reconstruct the 3D world from a 2D image or a 2D video, it is crucial to recover such information. In 3D computer vision, assumptions or prior knowledge of 3D layouts provide important information for 3D reconstruction. Two examples are using the prior distribution of 3D human pose for 3D pose estimation, and using the Manhattan world assumption for 3D scene Reconstruction. In reconstructing dynamic scenes involving human-object interactions, the 3D geometric relations in HOI over time play a similar role in providing the extra information for 3D recovery.

To model the relation over time, we use a simplified 4DHOI as the atomic actions as described in the above subsection.

As Fig. 7.5 shows, the model takes an RGB-D video with 3D human skeletons from a Kinect camera as its inputs. The output includes three parts: i) a hierarchical graph segmenting the video into events and atomic events, ii) an event category label, and iii) the contextual objects localized in the RGB-D frames.

As Fig. 7.6 illustrates, the 4DHOI model is a hierarchical graph for an event. On the time axis, an event is decomposed into several ordered atomic events. For example, the event *fetch water from dispenser* is decomposed into three sequential atomic events - *approach the dispenser*, *fetch water*, and *leave the dispenser*. An atomic event corresponds to a continuous segment in a video sequence, and contains similar human poses and object interactions. These atomic events are treated as hidden variables, which are learned by mining and clustering sequence samples.

In 3D space, an atomic event is decomposed into a human pose, one or multiple objects, and the relations between the pose and the objects. The relations include semantic relations and 3D geometric relations. The semantic relations between the object classes and the atomic event are treated as hard constraints. For example, the atomic event *fetch water* consists of the pose *fetch* and the objects *dispenser*, *mug*, as is shown in Fig. 7.6.

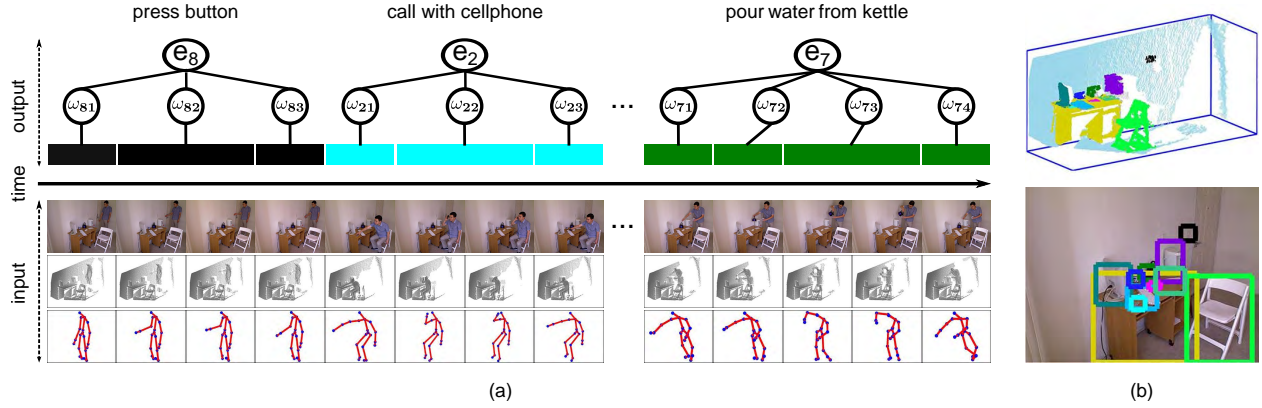


Figure 7.5: The 4DHOI model. (a) The framework of the model. The input is an RGB-D video with human skeletons. The outputs are the hierarchical interpretations of the video sequence, including event recognition, segmentation, and object localization. (b) Object recognition and localization in the 3D point cloud (upper) and the RGB image (lower) through the 4DHOI model after analyzing the video events. Reprinted from Ref. [164] with permission of IEEE, © 2016.

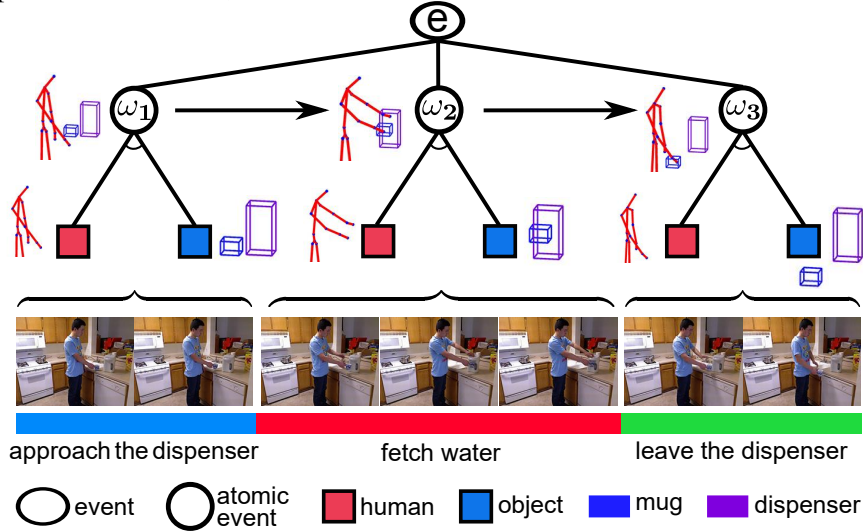


Figure 7.6: A hierarchical graph of the 4D human-object interactions for an example event *fetch water from dispenser*. Reprinted from Ref. [163] with permission.

We formulate the 4DHOI with a T-AOG. Let $V = (f_1, \dots, f_\tau)$ be a video sequence in the time interval $[1, \tau]$, where $f_t = (I_t, h_t)$ is the frame at time t . I_t is the RGB-D data. h_t is the human pose feature extracted from the 3D skeletons estimated by motion capture technology [136].

The sequence V is interpreted by the hierarchical graph $G = \langle E, L \rangle$ as follows.

i) $E \in \Delta = \{e_i | i = 1, \dots, |\Delta|\}$ is the event category such as *fetch water from dispenser*. Δ is the set of event categories.

ii) $L = (l_1, \dots, l_\tau)$ is a sequence of frame labels. $l_t = (a_t, o_t)$ is the interpretation of the frame f_t . $a_t \in \Omega_E = \{\omega_i | i = 1, \dots, K_E\}$ is the atomic event label such as *fetch water*.

Ω_E is the atomic event set of E . Each event category e_i has its distinct atomic event set Ω_{e_i} , i.e. the relations between an event and its atomic events are hard constraints.

$o_t = (o_t^1, \dots, o_t^{n_t})$ are the objects interacting with the human at time t , where n_t is the number of objects. Each object has a class label and a 3D location.

The energy that the video V is interpreted by the graph G is defined as

$$\text{En}(G|V) = \sum_{t=1}^{\tau} \Phi(f_t, l_t) + \sum_{t=2}^{\tau} \Psi(l_{1:t-1}, l_t), \quad (7.3)$$

where $\Phi(\cdot)$ is the spatial energy term of a single frame, encoding the human-object interactions in 3D space, and $\Psi(\cdot)$ is the temporal energy term encoding the temporal relations between the current frame l_t and all previous frames $l_{1:t-1}$. The variable E is omitted in the right side of Eq. (7.3) since each event has its own distinct atomic event set.

7.2.2.1 Human-Object Interactions in 3D Space

$\Phi(f_t, l_t)$ in Eq.(7.3) describes the human-object interactions in 3D space. The interactions include:

- 1) semantic co-occurrence between a specific type of human pose and the object classes.
- 2) geometric compatibility describing the 3D spatial constraints between the human pose and the objects.

Thus, $\Phi(f_t, l_t)$ is further decomposed into three terms which will be defined in the remainder of the subsection,

$$\Phi(f_t, l_t) = \phi_1(a_t, h_t) + \phi_2(a_t, o_t, I_t) + \phi_3(a_t, h_t, o_t). \quad (7.4)$$

Pose Model

$\phi_1(a_t, h_t)$ is the human pose model. A human skeleton consists of multiple 3D joints estimated by motion capture technology, like the Kinect camera [136]. The dimensionality of human pose is reduced by PCA and h_t is the PC parameter vector. We assume that h_t follows a Gaussian distribution:

$$\phi_1(a_t, h_t) = -\ln N(h_t; \mu_{a_t}, \Sigma_{a_t}) \quad (7.5)$$

where μ_{a_t} and Σ_{a_t} are respectively the mean and the covariance in the atomic event a_t .

Contextual Object Model

$\phi_2(a_t, o_t, I_t)$ is the term for fitting the contextual objects o_t to the RGB-D data I_t . Each object o_t^i includes a class label, *e.g.* *mug*, and a 3D bounding box located at z_t^i in 3D space. The 3D box is projected onto the RGB and depth images to form 2D bounding boxes, from which the RGB-D HOG features [25, 79] are extracted. Let $s(z_t^i)$ be the score of the linear SVM object detector using the RGB-D HOG features at location z_t^i . We convert this score into a probability using Platt scaling [79, 113], *i.e.* $p(z_t^i|I_t) = 1/\{1 + \exp\{\nu_1 s(z_t^i) + \nu_2\}\}$, where ν_1, ν_2 are parameters. The object model is

$$\phi_2(a_t, o_t, I_t) = -\frac{1}{n_t} \sum_{i=1}^{n_t} \ln p(z_t^i|I_t), \quad (7.6)$$

where n_t is the number of objects. $\frac{1}{n_t}$ offsets the influence of different object numbers. Because the relation between a_t and the object label is a hard constraint, we omit a_t in the right side of Eq. (7.6) for clarity.

This model defines the object location in 3D space, and the object appearance in 2D images, which is more robust to viewpoint and scale changes. The 3D boxes also provide a natural way for defining the 3D geometric relations between objects and human poses.

3D Geometric Compatibility and Object Prediction

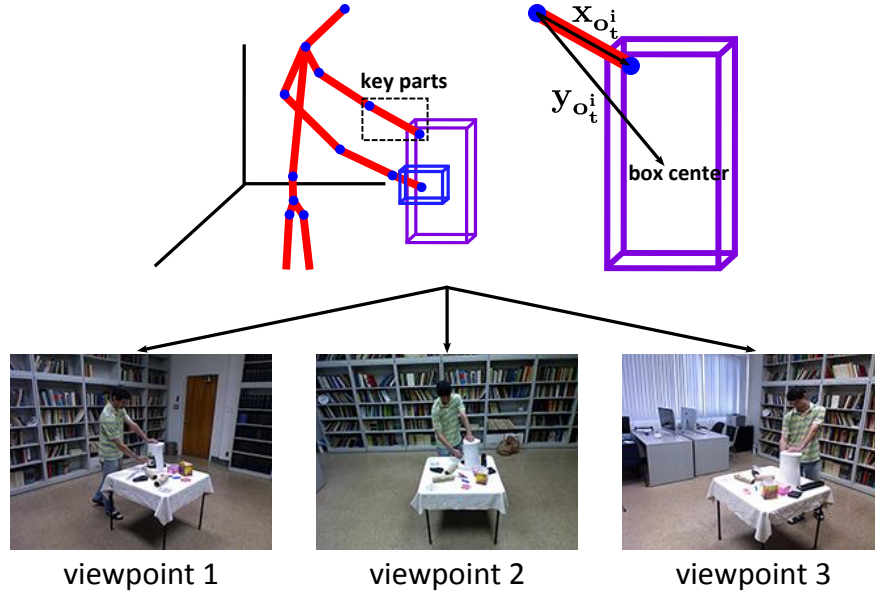


Figure 7.7: Human-object geometric relations in 3D space. Reprinted from Ref. [163] with permission.

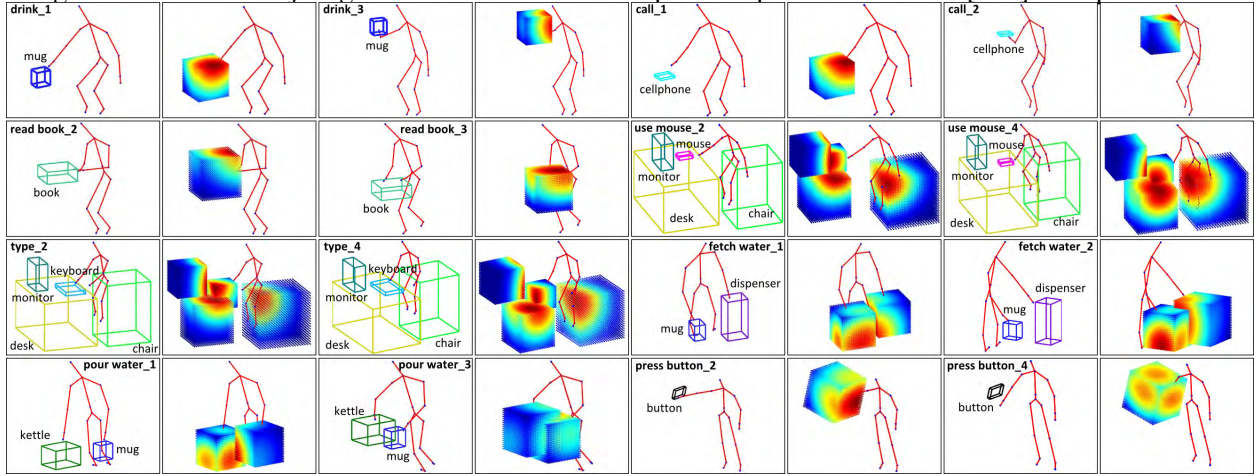


Figure 7.8: Examples of the learned geometric relations in atomic events. The odd-number columns are about the instances of the learned atomic events. The indices denote the atomic event number in each event. The even-number columns are about the probability maps of object prediction, where warmer colors indicate higher probabilities of locations where objects appear. Reprinted from Ref. [164] with permission of IEEE, © 2016.

The third energy term $\phi_3(a_t, h_t, o_t)$ measures the geometric relations between a human pose and objects. Geometric relations are mostly defined in 2D images [117, 175], and are not suitable for different viewpoints, as Fig. 7.7 illustrates. We model the geometric relations in 3D space.

Geometric relations between a human pose and objects are time-varying in different atomic events. For example, in Fig. 7.6, the human hand is far from the dispenser in the atomic event *approach the dispenser* while touches it in *fetch water*. Each atomic event has different geometric relations.

In an atomic event, an object interacts with some body parts, which we call the key parts. For example, in Fig. 7.7, the left arm interacts with the dispenser and the right arm interacts with the mug. The locations of objects in 3D space are closely related to and largely revealed by the locations and orientations of the key

parts.

As is shown in Fig. 7.7, let $y_{o_t^i}$ be the difference vector from one joint of the key parts to the object bounding box center. $x_{o_t^i}$ is the difference vector between the endpoints of the key parts. $y_{o_t^i}$ is closely related to $x_{o_t^i}$. We define $\eta_{o_t^i} = y_{o_t^i} - W_{o_t^i}^{a_t} x_{o_t^i}$, where $W_{o_t^i}^{a_t}$ is a similarity transformation matrix. $\eta_{o_t^i}$ describes the location of the object relative to the key parts. We assume $\eta_{o_t^i}$ follows a Gaussian distribution. This formulation is motivated by the observation that, for an atomic event, the instances of an object category have similar locations relative to the key parts. The 3D geometric relation is modeled as

$$\phi_3(a_t, h_t, o_t) = -\frac{1}{n_t} \sum_{i=1}^{n_t} \ln N(\eta_{o_t^i}; \mu_{o_t^i, a_t}^R, \Sigma_{o_t^i, a_t}^R), \quad (7.7)$$

where $\mu_{o_t^i, a_t}^R$ is the mean and $\Sigma_{o_t^i, a_t}^R$ is the covariance. The sign R is used to differentiate the 3D relation parameters from others. The subscript (o_t^i, a_t) indicates that the geometric relation varies for different atomic events and objects.

The vector $x_{o_t^i}$ is like a local reference, by which we can estimate $y_{o_t^i}$ and therefore predict the locations of related objects. Fig. 7.8 illustrates some examples of atomic events and the probability maps of the learned geometric relations. As Fig. 7.8 shows, according to the key parts, the probability that an object appears at a 3D location can be evaluated.

7.2.2.2 Temporal Relation

The temporal relation $\Psi(l_{1:t-1}, l_t)$ is decomposed as

$$\Psi(l_{1:t-1}, l_t) = \psi_1(a_{1:t-1}, a_t) + \psi_2(o_{t-1}, o_t), \quad (7.8)$$

where $a_{1:t-1}$ are the atomic event labels of the frames from time 1 to $t-1$. The first term encodes the atomic event transition, and the second term encodes the temporal coherence of objects.

7.2.2.3 Atomic Event Transition

In an event, the transition probability from the current atomic event ω_{k-1} to the next atomic event ω_k depends on the duration of the current atomic event, denoted by d_{k-1} . We model the time-varying transition probability with a logistic sigmoid function.

Fig. 7.9 compares two kinds of transition probabilities. Fig. 7.9 (a) is a duration-dependent transition. In interval 1 when the hand is still far from the dispenser, the probability of staying at *approach the dispenser* is much larger than the possibility of changing to the next atomic event *fetch water*. As the duration of *approach the dispenser* becomes long, in interval 3, the probability of staying at *approach the dispenser* will be close to zero, and the probability of transitioning to *fetch water* is almost 1. During interval 2, the transition will most likely happen. In contrast, if we use a duration-independent transition probability, they will be constant, regardless of the duration, as Fig. 7.9 (b) shows.

Let ω_{k-1} and ω_k be two consecutive atomic events of an event E . d_{k-1} is the duration of ω_{k-1} up to time $t-1$. We define the time-varying transition probability as

$$p(a_t = \omega_k | a_{t-1} = \omega_{k-1}, d_{k-1}) = \sigma(\beta d_{k-1} + \gamma), \quad (7.9)$$

where $\sigma(v) = 1/(1 + e^{-v})$ is a logistic sigmoid function. We simplify $p(a_t = \omega_k | a_{t-1} = \omega_{k-1}, d_{k-1})$ as $p(\omega_k | \omega_{k-1}, d_{k-1})$. The transition probability to ω_{k-1} is $p(\omega_{k-1} | \omega_{k-1}, d_{k-1}) = 1 - p(\omega_k | \omega_{k-1}, d_{k-1})$. Then the energy $\psi_1(a_{1:t-1}, a_t)$ is $-\ln p(\omega_k | \omega_{k-1}, d_{k-1})$ or $-\ln p(\omega_{k-1} | \omega_{k-1}, d_{k-1})$, up to the value of a_t .

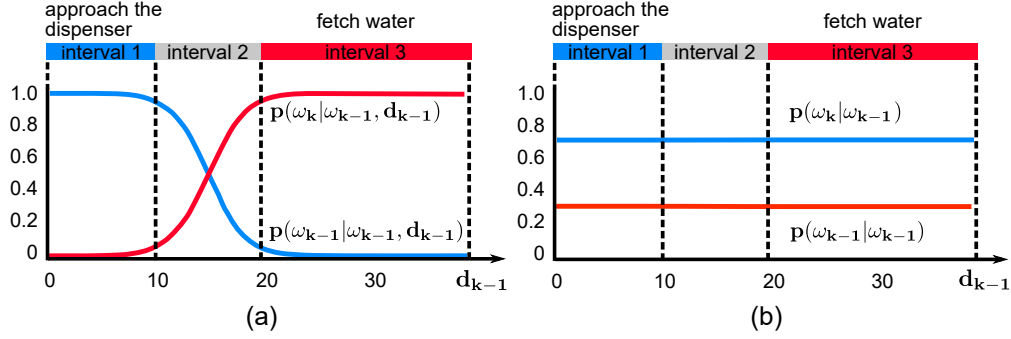


Figure 7.9: The atomic event transition probability. (a) Duration-dependent transition. (b) Duration-independent transition. Reprinted from Ref. [163] with permission.

7.2.2.4 Temporal Coherence of Objects

$\psi_2(o_{t-1}, o_t)$ describes the temporal coherence of objects. In an event, the locations of some objects, such as dispensers, are almost static, while other objects, such as mugs, move with hands. For *moveable* objects, we assume the location follows a Gaussian distribution $p(z_t^i | z_{t-1}^i) = N(z_t^i - z_{t-1}^i; \mu_{o_t^i, a_t}^Z, \Sigma_{o_t^i, a_t}^Z)$. For *static* objects, we set a threshold. If the difference of the proposed location z_t^i in the current frame and the location at the last frame z_{t-1}^i is smaller than the threshold, $p(z_t^i | z_{t-1}^i)$ is 1, otherwise 0. The energy is

$$\psi_2(o_{t-1}, o_t) = -\frac{1}{n_t} \sum_{i=1}^{n_t} \ln p(z_t^i | z_{t-1}^i). \quad (7.10)$$

7.2.3 Part-Level 3DHOI

In the previous subsection, we represented an object by its 3D bounding box. While this method works well for smaller objects, it falls apart when dealing with larger furniture such as chairs or desks when a single bounding box is not capable of representing the complexities in the object shapes. The left figure in Fig.7.10 shows how a single bounding box is not suitable for representing complex functional shapes in some objects. To overcome this issue, it is better to use a set of bounding boxes, each representing a functional part of the object, to represent it. As a result, the geometric compatibility discussed in the previous section needs an update.

Namely, we use η to denote a geometric relation between an object part and a human key part, instead of the relation between an entire object and a key part. Similar to the previous section, we assume η follows Gaussian distribution. The final geometric relation is modeled as the summation of the log pdfs of the Gaussian distributions of each η , divided by the number of object parts. Fig.7.11 demonstrates a set of sitting poses sampled according to the part-level HOI relations.

7.2.4 Hand-Object Interaction

In addition to human-object interaction, it is also important to model hand-object interaction to provide a more fine-grained scene understanding. Different from human-object interaction, the subjects of hand-object interaction are usually small and subject to heavy occlusion. Therefore, it is usually not possible to obtain detailed information from detection. Instead, we can improvise based on what we have. Namely, according to the spatial and temporal constraints that we have, hallucinate the 3D details as realistic as

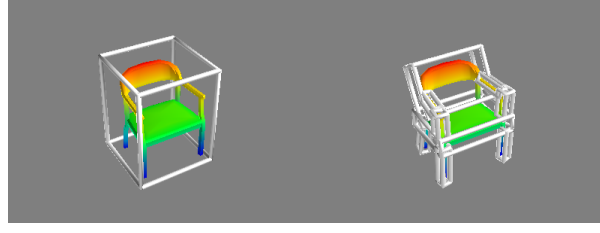


Figure 7.10: Left: Instance-level bounding box representation of a chair. Right: Part-level bounding box representation of a chair.

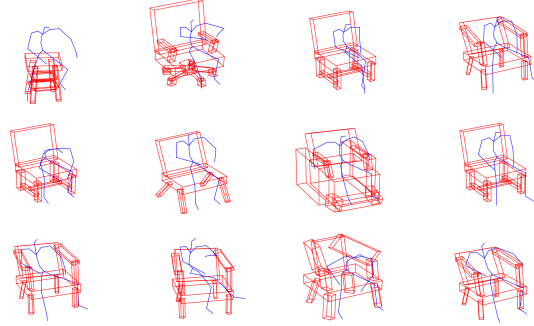


Figure 7.11: Sitting instances sampled according to part-level HOI relations.

possible. Because hand-object interaction has the same acronym as human-object interaction, we use HOI to denote human-object interaction and hoi to denote hand-object interaction.

Different Types of hoi

The main device that we human use to interact with tools is our hand. With different tasks and different requirements of precision and power, we tend to interact with tools differently. [37] organized human grasping into a hierarchical taxonomy as shown in Fig.7.12. In order to generate realistic hoi snapshots, we need a model that can generate all types of grasping, instead of only one.

Learning hoi

By examining the grasp taxonomy proposed in Fig.7.12, we make the observation that different types of grasping use different parts of the hand. In other words, the contact map on the hand follows different distribution for different grasping types. To produce different types of grasping, it is natural to produce different contact maps of the hand first, and then attach the contact regions on the hand surface to the object surface. Fig.7.13 demonstrates a system using a PointNet [118] structure to produce contact maps conditioned on the geometric relationship between the hand and the object. In this work, the authors use v^+ for the pose parameters of the hand, v^o for the shape of the object, h^+ for the latent factor that controls the type of hoi, and θ for the collection of weights of the PartNet segmentation model.

Given an input object shape v^o , an hoi type parameter h^+ , and an initial hand pose v^+ , the model first performs differentiable forward kinematics to produce the shape of the hand. Then, we compute the distance from the hand surface points to object surface using a DeepSDF model [106]. We attach the signed distance measures to the hand surface points to create a 4-dimensional point cloud, and feed that into the PartNet segmentation model σ . σ will determine whether a point is a contact point or a non-contact point. According to the result of segmentation, each hand surface point will compute its own soi (surface-object interaction) score:

		Power					Intermediate			Precision					
		Palm		Pad			Side			Pad				Side	
Opp:		3-5	2-5	2	2-3	2-4	2-5	2	3	3-4	2	2-3	2-4	2-5	3
VF:															
Thumb Abducted			1: Large Diameter 2: Small Diameter 3: Medium Wrap 10: Power Disk 11: Power Sphere	31: Ring	28: Sphere Finger	18: Extension Type 26: Sphere 4-Finger	19: Distal Type	23: Adduction Grip		21: Tripod Variation	9: Palmar Pinch 24: Tip Pinch 33: Inferior Pincer	8: Prismatic 2 Finger 14: Tripod	7: Prismatic 3 Finger 27: Quadpod	6: Prismatic 4 Finger 12: Precision Disk 13: Precision Sphere	20: Writing Tripod
Thumb Adducted	17: Index Finger Extension	4: Adducted Thumb 5: Light Tool 15: Fixed Hook 30: Palmar						16: Lateral 29: Stick 32: Ventral	25: Lateral Tripod					22: Parallel Extension	

Figure 7.12: Grasp taxonomy as described in [37]

$$e_i(c_i, d_i) = \begin{cases} \max(0, -d_i) & \text{if } c_i = \text{non-contact point} \\ d_i^2 & \text{if } c_i = \text{contact point} \end{cases} \quad (7.11)$$

where e_i denotes the point-wise soi score, c_i is the segmentation result for that hand surface point, and d_i is the signed distance between that hand surface point and the object surface. d_i is negative if the hand surface point is inside the object. The final hoi energy $\mathcal{E}(v^+|v^o, h^+; \theta)$ is the sum of soi of all hand surface points.

To learn the parameter θ , we adopt a maximum likelihood estimation method as described in [171]. Fig.7.14 shows the generated hoi's using that system.

7.2.5 Concurrent HOI's and hoi's in STC-AOG

When we combine HOI's and hoi's to form an atomic action, that atomic action becomes a basic unit in our Spatial-Temporal-Causal And-Or Graph (STC-AOG). We call it an STC-unit. Fig.7.15 illustrates what an STC-Unit looks like. The root of an STC-Unit is the atomic action label, shown in a diamond shape node. The atomic action label is a leaf node in the Temporal AOG. An atomic action label has four components: precondition, causal function, post-condition, and an ST-unit describing the spatial-temporal

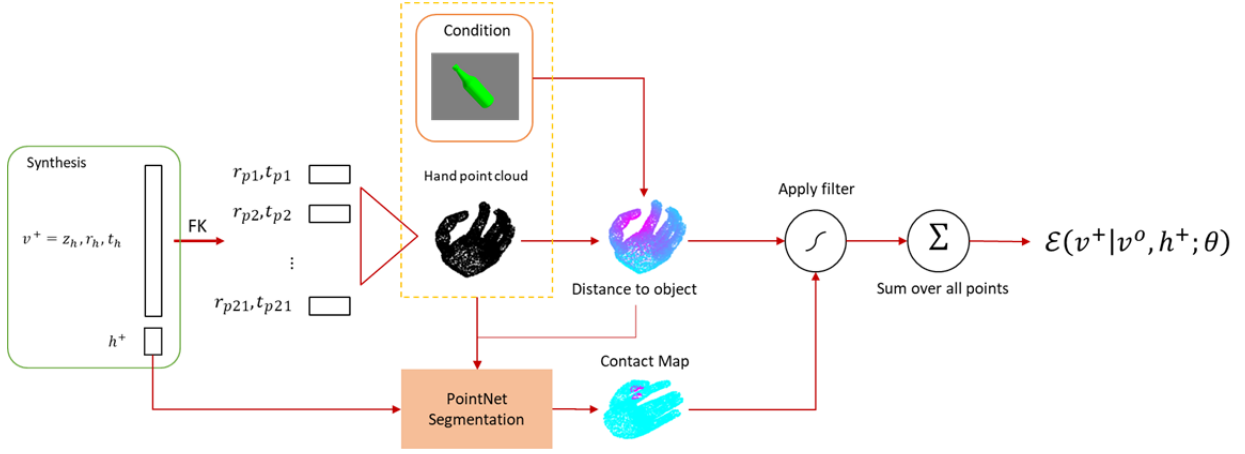


Figure 7.13: Computing HOI energy from predicting contact maps. From Ref. [90] with permission.

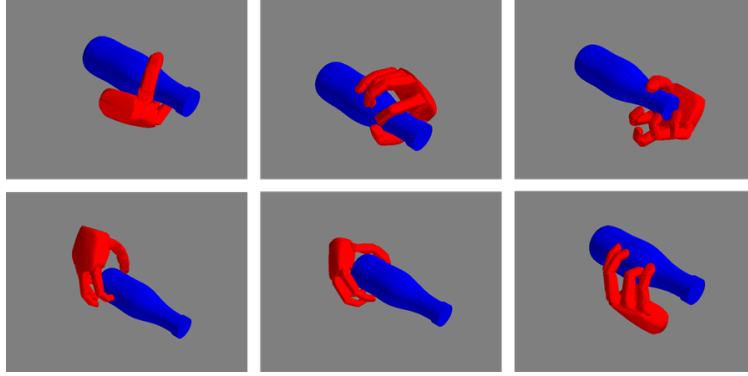


Figure 7.14: Generated hoi examples. From Ref. [90] with permission.

relations involved. The precondition, causal function and post-condition are a part of the Causal AOG. The ST-unit contains a list of binary relations between key parts of human and object, which are in turn members of the Spatial AOG. The STC-unit is the key component that wires the spatial, temporal, and causal AOG's into one organic STC-AOG.

7.3 Event representation by T-AOG

Here we introduce the T-AOG for event representation.

7.3.1 The T-AOG for Events

An T-AOG (see Figure 7.16 for an example) is specified by a 6-tuple

$$T - AOG = \langle S, V_N, V_T, R, \Sigma, P \rangle .$$

S is the root node for an event category, $V_N = V^{and} \cup V^{or}$ is the set of non-terminal nodes (events and sub-events) composed of an And-node set and an Or-node set.

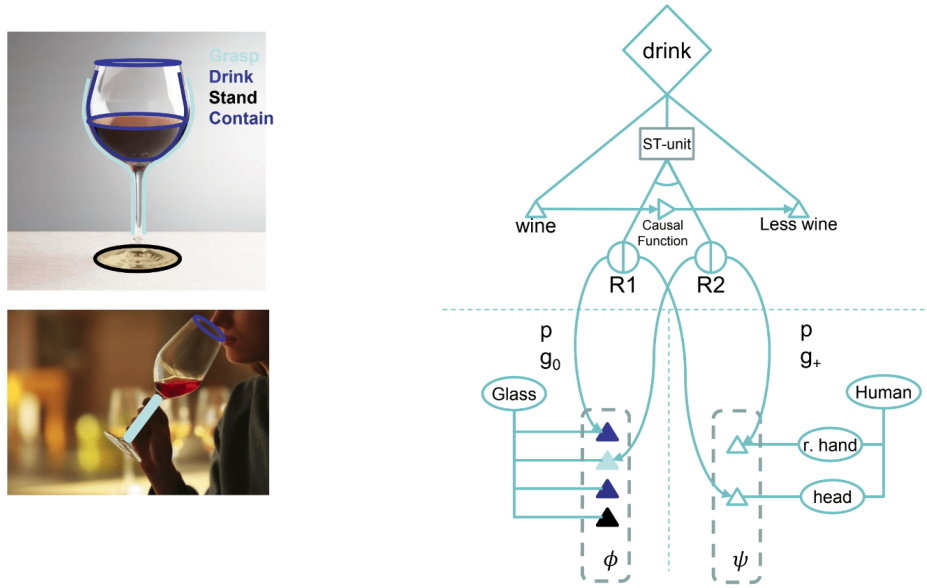


Figure 7.15: Example STC-Unit of drinking with a wine glass. Left: affordances of different parts of the object. Right: STC-Unit of the atomic action "drink", and its relation with different parts of the object.

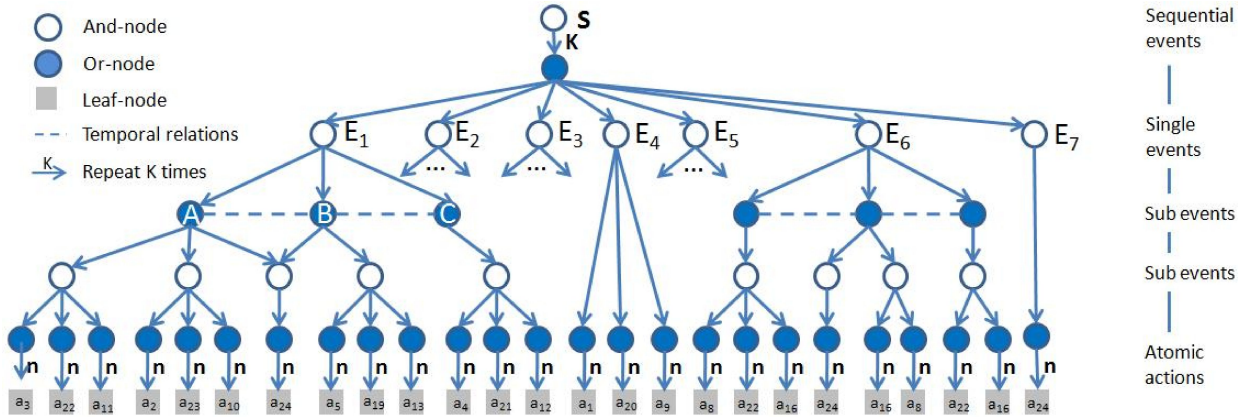


Figure 7.16: T-AOG for events in the office scene. S is the root node which represents the sequential events happened in the office. It is a Set-node and could be any combinations of K single events. For example, S could be $E_1|E_2|E_1E_2|E_3E_2E_3|...$, where $E_1, ..., E_7$ are And-nodes representing single events. The atomic actions are also represented by Set-nodes, and could last for 1 to n frames. The temporal relations are given by the ratio of the lasting time between related nodes. For clarity, only the temporal relations between sub-events are shown. Reprinted from Ref. [110] with permission.

Each And-node represents an event or sub-event, and is decomposed into sub-events or atomic actions as their children nodes. The children nodes must occur in a certain temporal order.

An Or-node has a number of alternative ways to realize an event or sub-event, and each alternative has a probability associated with it to indicate the frequency of occurrence. A Set-node is a special Or-node which can repeat m times with probability $p(m)$ and accounts for the time warping effects.

V_T is a set of terminal nodes for atomic actions. R is a number of relations between the nodes (temporal

relations), Σ is the set of all valid configurations (possible realizations of the events) derivable from the T-AOG, *i.e.* its language, and P is the probability model defined on the graph. The T-AOG for events in the office scene is shown in Figure 7.16.

7.3.2 Parse Graph

A parse graph is an instance of the T-AOG obtained by selecting variables at the Or-nodes and specifying the attributes of And-nodes and terminal nodes. We use pg to denote the parse graph of the T-AOG of a single event E_i . We denote the following components in pg :

- $V^t(pg) = \{a_1, \dots, a_{n_t(pg)}\}$ is the set of leaf nodes in pg .
- $V^{or}(pg) = \{v_1, \dots, v_{n_{or}(pg)}\}$ is the set of non-empty Or-nodes in pg , $p(v_i)$ is the probability that v_i chooses its sub-nodes in pg .
- $R(pg) = \{r_1, \dots, r_{n(R)}\}$ is the set of temporal relations between the nodes in pg . Without temporal relations, the pg reduces to a parse tree.

The energy of pg is defined as in [185]:

$$\begin{aligned} \varepsilon(pg) = & \sum_{a_i \in V^t(pg)} E(a_i) + \sum_{v_i \in V^{or}(pg)} -\log p(v_i) \\ & + \sum_{r_i \in R(pg)} -\log p(r_i) \end{aligned} \quad (7.12)$$

The first term is the data term. It expresses the energy of the detected terminal nodes (atomic actions) which is computed by Eq.7.1. The second term is the frequency term. It accounts for how frequently each Or-node decomposes in a certain way, and can be learned from the training data. The third term is the relation term which models the temporal relations between the nodes in pg and can be computed by Eq. 7.2.

Given input video I_\wedge in a time interval $\wedge = [0, T]$. We use PG to denote parse graph for a sequence of events in S and to explain the I_\wedge . PG is of the following form,

$$PG = (K, pg_1, \dots, pg_K)$$

where K is the number of parse graphs for events.

7.3.3 Example I: Synthesizing New Events by T-AoG

With events defined as atomic-action sequences, we can use the T-AoG for representing them. However, compactness guaranteed by this hierarchical representation is not the only advantage that T-AoG offers. Let's consider the event of "Refill Tea" as an example Fig.7.17. In this event, we have the root node being an And-node and three Or-nodes as the children of the root. With the root node specifying that the three nodes, "Prepare Cup", "Renew Tea" and "Refill Water", must follow the order provided by the T-AoG, the Or-nodes for each sub-event allows us to have different actions under different contexts. Depending on the status of the cup, "Prepare Cup" requires to (i) dump old tea if the cup contains tea leaves, (ii) dump water if the cup is full, and (iii) do nothing if the cup is already empty. Such a hierarchical representation gives us a clear picture of the context of each subevents (offered by Or-node) and also the execution pattern of activities and tasks (And-node).

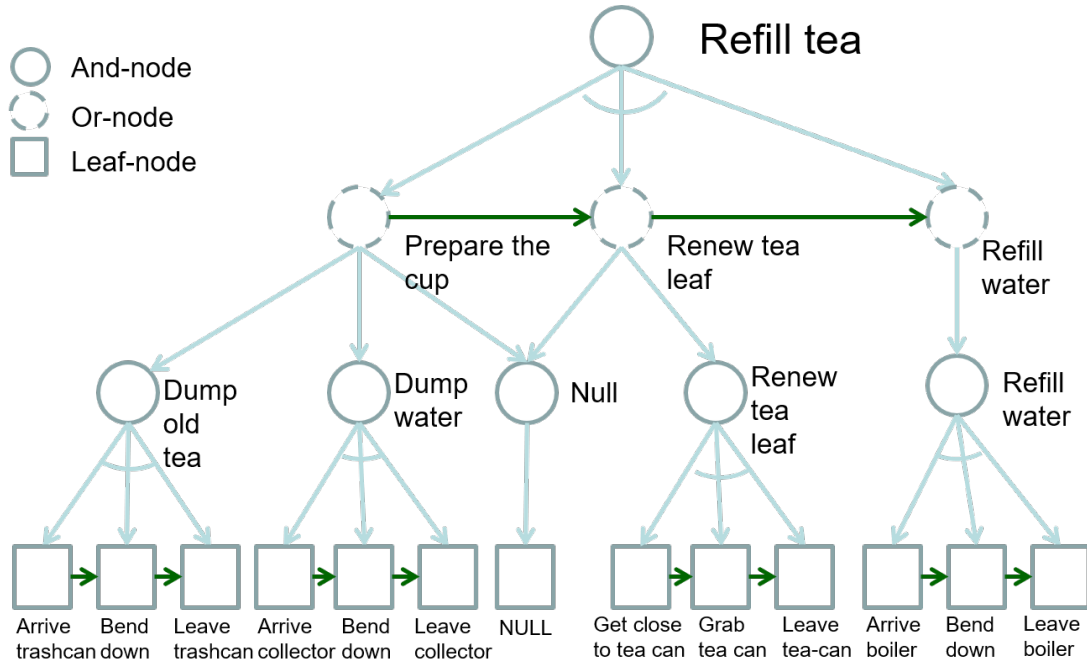


Figure 7.17: An example T-AoG for representing the event “Refill Tea” with sub-events have different alternatives captured by Or-nodes.

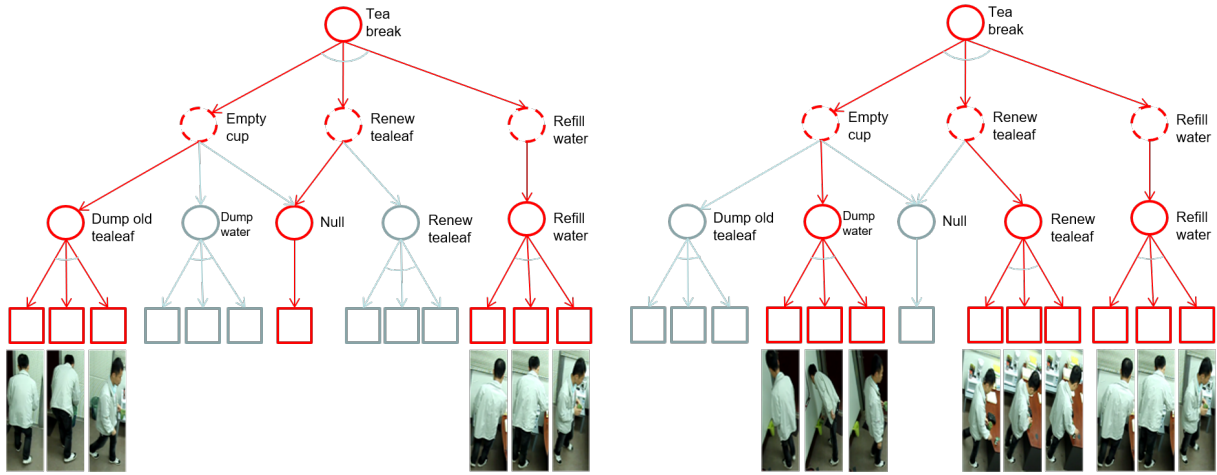


Figure 7.18: Synthesizing new activities by sampling from the T-AoG

Moreover, using this grammar model for events, we can also synthesize new events for unseen scenarios by sampling the T-AoG. Suppose that we have seen sub-events like “dump water” and “dump tea leaf” in different activities, but never as a full event. We can sample a new combination of sub-events constructing new activities that we have not observed. This can be easily achieved by sampling atomic action units following the context-free grammar the T-AoG is capturing. For example, as shown in Fig.7.18, we start from the root node of the T-AoG, randomly select one branch at each Or-node and take all sub-events under And-nodes. In this way, we can get different event sequences sampled from the T-AoG representing the event.

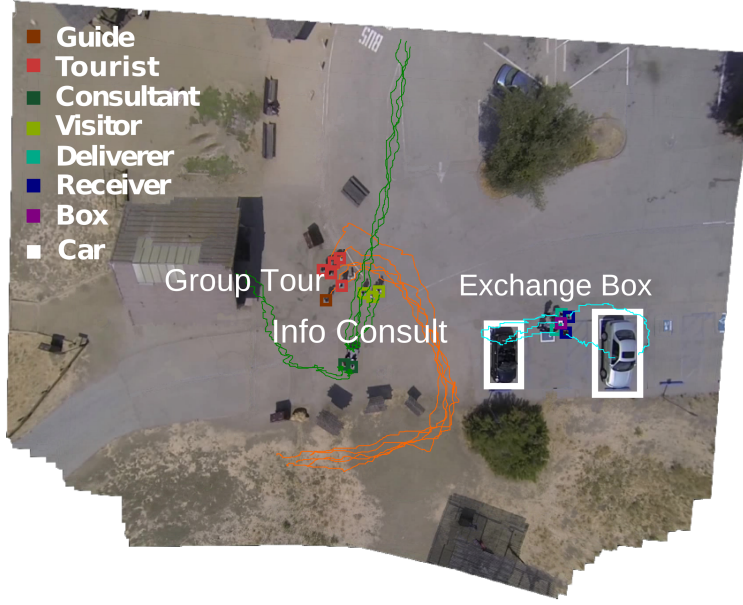


Figure 7.19: An example of group activity parsing. Reprinted from Ref. [137] with permission.

7.3.4 Example II: Group Activity Parsing by ST-AoG

In this section, we provide an example of how ST-AoG can be applied to modeling social events. Specifically, we will look at group activity parsing, which requires understanding human behaviors and discovering their social relations. This calls for a holistic understanding of social scenes as shown in Figure 7.19, which includes a joint inference of three key elements: i) social groups, ii) events that people in each social group engage in, and iii) roles of the group members.

7.3.4.1 Representing of Group Events by ST-AoG

Similar with hierarchical representation in [59, 86, 111, 112], domain knowledge can be formalized as ST-AoG, depicted in Figure 7.20. Its nodes represent the following five sets of concepts: events $\Delta_E = \{E_i\}$; sub-events $\Delta_L = \{L_a\}$; human roles $\Delta_R = \{R_j\}$; small objects that people interact with $\Delta_O = \{O_j\}$; and large objects and scene surfaces $\Delta_S = \{S_j\}$. A particular pattern of foreground trajectories observed in a given time interval gives rise to a sub-event, and a particular sequence of sub-events defines an event.

Edges of the ST-AoG represent decomposition and temporal relations in the domain. In particular, the nodes are hierarchically connected by decomposition edges into three levels, where the root level corresponds to events, middle level encodes sub-events, and leaf level is grounded onto foreground tracklets and object detections in the video. The nodes of sub-events are also laterally connected for capturing “followed-b” temporal relations of sub-events within the corresponding events.

ST-AoG has special types of nodes. An AND node, \wedge , encodes a temporal sequence of latent sub-events required to occur in the video so as to enable the event occurrence (*e.g.*, in order to *Exchange Box*, the *Deliverers* first need to approach the *Receivers*, give the *Box* to the *Receivers*, and then leave). For a given event, an OR node, \vee , serves to encode alternative space-time patterns of distinct sub-events.

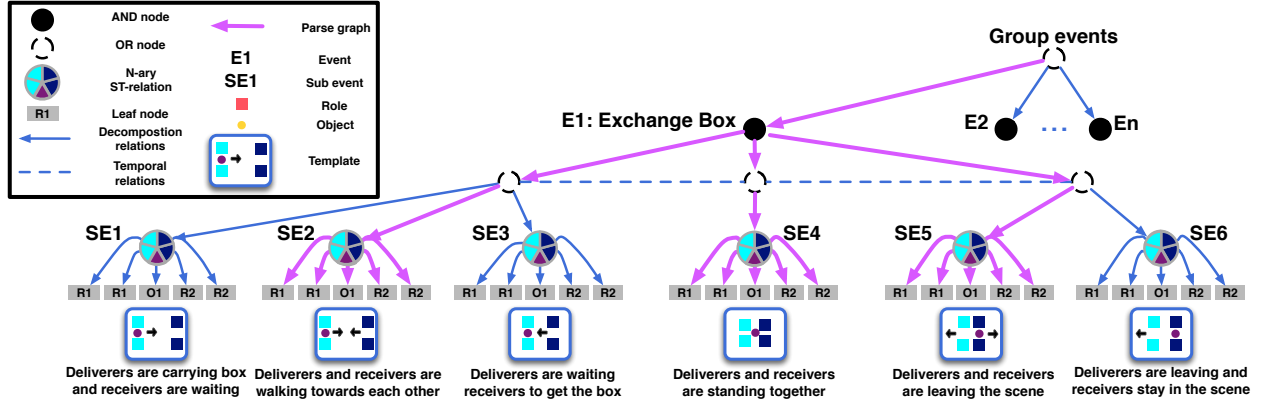


Figure 7.20: A part of ST-AoG for *Exchange Box*. The nodes are hierarchically connected (solid blue) into three levels, where the root level corresponds to events, middle level encodes sub-events, and leaf level is grounded onto foreground tracklets and small static objects in the video. The lateral connections (dashed blue) indicate temporal relations of sub-events. The colored pie-chart nodes represent templates of n -ary spatial-temporal relations among human roles and objects (see Figure 7.21). The magenta edges indicate an inferred parse graph which recognizes and localizes temporal extents of events, sub-events, human roles and objects in the video. Reprinted from Ref. [137] with permission.

7.3.4.2 Sub-events as Latent Spatiotemporal Templates

A temporal segment of foreground trajectories corresponds to a sub-event. ST-AoG represents a sub-event as the *latent* spatial-temporal template of n -ary spatial-temporal relations among foreground trajectories within a time interval, as illustrated in Figure 7.21. In particular, as an event is unfolding in the video, foreground trajectories form characteristic space-time patterns, which may not be semantically meaningful. As they frequently occur in the data, they can be robustly extracted from training videos through unsupervised clustering. We formalize these patterns with the spatial-temporal templates within the Bayesian framework using unary, pairwise, and n -ary relations among the foreground trajectories.

Unary attributes. A foreground trajectory, $\Gamma = [\Gamma^1, \dots, \Gamma^k, \dots]$, can be viewed as spanning a number of time intervals, $\tau_k = [t_{k-1}, t_k]$, where $\Gamma^k = \Gamma(\tau_k)$. Each trajectory segment, Γ^k , is associated with unary attributes, $\phi = [\mathbf{r}^k, s^k, \mathbf{c}^k]$. Elements of the role indicator vector $\mathbf{r}^k(l) = 1$ if Γ^k belongs to a person with role $l \in \Delta_R$ or object class $l \in \Delta_O$; otherwise $\mathbf{r}^k(l) = 0$. The speed indicator $s^k = 1$ when the normalized speed of Γ^k is greater than a threshold (we use 2 pixels/sec); otherwise, $s^k = 0$. Elements of the closeness indicator vector $\mathbf{c}^k(l) = 1$ when Γ^k is close to any of the large objects or types of surfaces detected in the scene indexed by $l \in \Delta_S$, such as *Building*, *Car*, for a threshold (70 pixels); otherwise, $\mathbf{c}^k(l) = 0$.

Pairwise relations. A pair of trajectory segments, Γ_j^k and $\Gamma_{j'}^k$, are aimed at capturing spatial-temporal relations of human roles or objects represented by the two trajectories, as illustrated in Figure 7.21. The pairwise relations are specified as: $\phi_{jj'} = [d_{jj'}^k, \theta_{jj'}^k, \mathbf{r}_{jj'}^k, s_{jj'}^k]$, where $d_{jj'}^k$ is the mean distance between Γ_j^k and $\Gamma_{j'}^k$; $\theta_{jj'}^k$ is the angle subtended between Γ_j^k and $\Gamma_{j'}^k$; and the remaining three pairwise relations check for compatibility between the aforementioned binary relations as: $\mathbf{r}_{jj'}^k = \mathbf{r}_j^k \oplus \mathbf{r}_{j'}^k$, $s_{jj'}^k = s_j^k \oplus s_{j'}^k$, $\mathbf{c}_{jj'}^k = \mathbf{c}_j^k \oplus \mathbf{c}_{j'}^k$, where \oplus denotes the Kronecker product.

n -ary relations. Towards encoding unique spatial-temporal patterns of a set of trajectories, we specify the following n -ary attribute. A set of trajectory segments, $G_i(\tau_k) = G_i^k = \{\Gamma_j^k\}$, can be described by a 18-bin histogram \mathbf{h}^k of their velocity vectors. \mathbf{h}^k counts orientations of velocities at every point along the trajectories in a polar coordinate system: 6 bins span the orientations in $[0, 2\pi]$, and 3 bins encode the

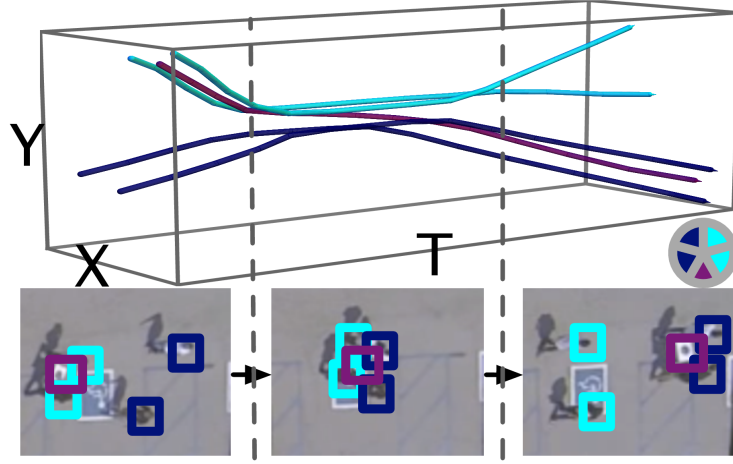


Figure 7.21: Three example templates of n -ary spatial-temporal relations among foreground trajectories extracted from the video (XYT-space) for the event *Exchange Box*. The recognized roles *Deliverers*, *Receivers* and the object *Box* in each template are marked cyan, blue and purple, respectively. Spatial-temporal templates are depicted as colored pie-chart nodes in Figure 7.20. Reprinted from Ref. [137] with permission.

locations of trajectory points relative to a given center. As the polar-coordinate origin, we use the center location of a given event in the scene.

Unsupervised Extraction of Templates. Given training videos with ground-truth partition of all their ground-truth foreground trajectories G into disjoint subsets $G = \{G_i\}$. Every G_i can be further partitioned into equal-length time intervals $G_i = \{G_i^k\}$ ($|\tau^k| = 2\text{sec}$). We can use K-means clustering to group all $\{\Gamma_{i,j}^k\}$, and then estimate spatial-temporal templates $\{L_a\}$ as representatives of the resulting clusters a .

7.3.4.3 Formulation and Learning of Templates

Given the spatial-temporal templates, $\Delta_L = \{L_a\}$, extracted by K-means clustering from training videos (see Section 7.3.4.2), we conduct inference by seeking these latent templates in foreground trajectories of the new video. To this end, we define the log-likelihood of a set of foreground trajectories $G = \{\Gamma_j\}$ given $L_a \in \Delta_L$ as

$$\begin{aligned} \log p(G|L_a) &\propto \sum_j \mathbf{w}_a^1 \cdot \phi_j + \sum_{jj'} \mathbf{w}_a^2 \cdot \phi_{jj'} + \mathbf{w}_a^3 \cdot \mathbf{h}, \\ &= \mathbf{w}_a \cdot [\sum_j \phi_j, \sum_{jj'} \phi_{jj'}, \mathbf{h}] = \mathbf{w}_a \cdot \boldsymbol{\psi}. \end{aligned} \quad (7.13)$$

where the bottom equation of (7.13) formalizes every template as a set of parameters $\mathbf{w}_a = [\mathbf{w}_a^1, \mathbf{w}_a^2, \mathbf{w}_a^3]$ appropriately weighting the unary, pairwise and n -ary relations of G , $\boldsymbol{\psi}$. Recall that our spatial-temporal templates are extracted from unit-time segments of foreground trajectories in training. Thus, the log-likelihood in Eq. (7.13) is defined only for sets G consisting of unit-time trajectory segments.

From Eq. (7.13), the parameters \mathbf{w}_a can be learned by maximizing the log-likelihood of $\{\boldsymbol{\psi}_a^k\}$ extracted from the corresponding clusters a of training trajectories.

The log-posterior of assigning template L_a to longer temporal segments of trajectories, falling in $\tau =$

(t', t) , $t' < t$, is specified as

$$\log p(L_a(\tau)|G(\tau)) \propto \sum_{k=t'}^t \log p(G^k|L_a) + \log p(L_a(\tau)) \quad (7.14)$$

where $p(L_a(\tau))$ is a log-normal prior that L_a can be assigned to a time interval of length $|\tau|$. The hyperparameters of $p(L_a(\tau))$ are estimated using the MLE on training data.

7.3.4.4 Probabilistic Model

A parse graph is an instance of ST-AoG, explaining the event, sequence of sub-events, and human role and object label assignment. The solution of our video parsing is a set of parse graphs, $W = \{pg_i\}$, where every pg_i explains a subset of foreground trajectories, $G_i \subset G$, as

$$pg_i = \{e_i, \tau_i = [t_{i,0}, t_{i,T}], \{L(\tau_{i,u})\}, \{c_{i,j}\}\}, \quad (7.15)$$

where $e_i \in \Delta_E$ is the recognized event conducted by G_i ; $\tau_i = [t_{i,0}, t_{i,T}]$ is the temporal extent of e_i in the video starting from frame $t_{i,0}$ and ending at frame $t_{i,T}$; $\{L(\tau_{i,u})\}$ are the templates (*i.e.*, latent sub-events) assigned to non-overlapping, consecutive time intervals $\tau_{i,u} \subset \tau_i$, such that $|\tau_i| = \sum_u |\tau_{i,u}|$; and $c_{i,j}$ is the human role or object class assignment to j th trajectory $\Gamma_{i,j}$ of G_i .

Our objective is to infer W that maximizes the log-posterior $\log p(W|G) \propto -\mathcal{E}(W|G)$, given all foreground trajectories G extracted from the video. The corresponding energy $\mathcal{E}(W|G)$ is specified for a given partitioning of G into N disjoint subsets G_i as

$$\mathcal{E}(W|G) \propto \sum_{i=1}^N \left[\underbrace{-\log p(\wedge_{e_i} | \vee_{\text{root}})}_{\text{select event } e_i} + \sum_u \left[\underbrace{-\log p(\wedge_{L_a} | \vee_{e_i})}_{\text{select template } L_a} - \underbrace{\log p(L_a(\tau_{i,u}) | G_i(\tau_{i,u}))}_{\text{assign template}} \right] \right] \quad (7.16)$$

where $G_i(\tau_{i,u})$ denotes temporal segments of foreground trajectories falling in time intervals $\tau_{i,u}$, $|\tau_i| = \sum_u |\tau_{i,u}|$, and $\log p(L(\tau_{i,u}) | G_i(\tau_{i,u}))$ is given by Eq. (7.14). Also, $\log p(\wedge_{e_i} | \vee_{\text{root}})$ and $\log p(\wedge_{L_a} | \vee_{e_i})$ are the log-probabilities of the corresponding switching OR nodes in ST-AoG for selecting particular events $e_i \in \Delta_E$ and spatial-temporal templates $L_a \in \Delta_L$. These two switching probabilities are simply estimated as the frequency of corresponding selections observed in training data.

7.3.4.5 Inference

Given an aerial video, we first build a video panorama and extract foreground trajectories G . Then, the goal of inference is to: (1) partition G into disjoint groups of trajectories $\{G_i\}$ and assign label event $e_i \in \Delta_E$ to every G_i ; (2) assign human roles and object labels $r_{i,j}$ to trajectories $\Gamma_{i,j}$ within each group G_i ; and (3) assign latent spatial-temporal templates $L(\tau_{i,u}) \in \Delta_L$ to temporal segments $\tau_{i,u}$ of foreground trajectories within every G_i . For steps (1) and (2) we use two distinct MCMC processes. Given groups G_i , event labels e_i and role assignment $r_{i,j}$ proposed in (1) and (2), step (3) uses dynamic programming for efficient estimation of sub-events $L(\tau)$ and their temporal extents τ . Steps (1)–(3) are iterated until convergence, *i.e.*, when $\mathcal{E}(W|G)$, given by Eq. (7.16), stops decreasing after a sufficiently large number of iterations.

Grouping Given G , we first use [52] to perform initial clustering of foreground trajectories into atomic groups. Then, we apply the first MCMC to iteratively propose either to merge two smaller groups into a merger, with probability $p(1) = 0.7$, or to split a merger into two smaller groups, with probability $p(2) =$

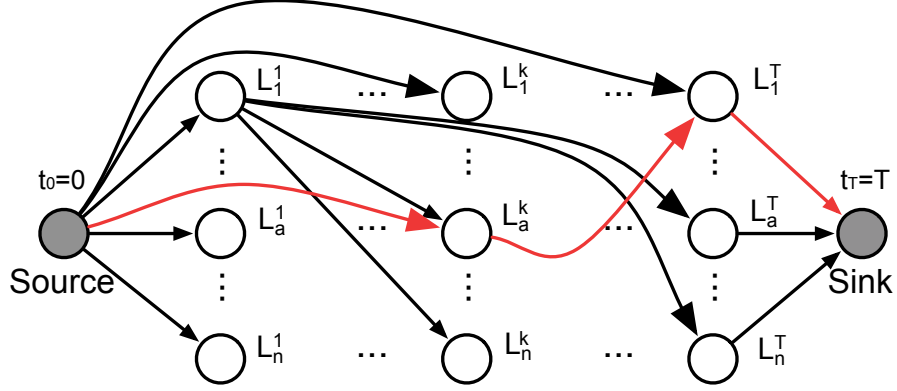


Figure 7.22: Our DP process can be illustrated by this DAG (directed acyclic graph). An edge between $L_{a'}^{k'}$ and L_a^k means the transition $L_{a'} \rightarrow L_a$ follows the rule defined in ST-AoG and the time interval $[t_{a'}, t_a]$ is assigned with template L_a . In this sense, with the transition rules and the prior defined in Eq. (7.14) (we do not consider the assignment with low prior probability), we can define the edges of such DAG. So the goal of DP is equivalent to finding a shortest path between source and sink. The red edges highlight a possible path. Suppose we find a path $source \rightarrow L_3^8 \rightarrow L_1^{20} \rightarrow sink$. This means that we decompose $[0, T]$ into 2 time intervals: $[0, 8\delta t]$, $[8\delta t, T]$, and they are assigned with template L_3 and L_1 respectively. Reprinted from Ref. [137] with permission.

0.3. Given the proposal, each resulting group G_i is labeled with an event $e_i \in \Delta_E$ (we enumerate all possible labels). In each proposal, the MCMC jumps from current solution W to a new solution W' generated by one of the dynamics. The acceptance rate is $\alpha = \min \left\{ 1, \frac{Q(W \rightarrow W')p(W'|G)}{Q(W' \rightarrow W)p(W|G)} \right\}$, where the proposal distribution $Q(W \rightarrow W')$ is one of $p(1)$ or $p(2)$ depending on the proposal, and $p(W|G)$ is given by Eq. (7.16).

Human Role Assignment Given a partitioning of G into groups $\{G_i\}$ and their event labels $\{e_i\}$, we use the second MCMC process within every G_i to assign human roles and object labels to trajectories. Each trajectory $\Gamma_{i,j}$ in G_i is randomly assigned with an initial human-role/object label $r_{i,j}$ for solution pg_i . In each iteration, we randomly select $\Gamma_{i,j}$ and change its role label to generate a new proposal pg'_i . The acceptance rate is $\alpha = \min \left\{ 1, \frac{Q(pg_i \rightarrow pg'_i)p(pg'_i|G_i)}{Q(pg'_i \rightarrow pg_i)p(pg_i|G_i)} \right\}$, where $\frac{Q(pg_i \rightarrow pg'_i)}{Q(pg'_i \rightarrow pg_i)} = 1$ and $p(pg'_i|G_i)$ is maximized by dynamic programming specified in the next section 7.3.4.5.

Detection of Latent Sub-events with DP From steps (1) and (2), we have obtained the trajectory groups $\{G_i\}$, and their event $\{e_i\}$ and role labels $\{r_{i,j}\}$. Every G_i can be viewed as occupying time interval of $\tau_i = [t_{i,0}, t_{i,T}]$. The results of steps (1) and (2) are jointly used with detections of large objects $\{S_i\}$ to estimate all unary, pairwise, and n -ary relations ψ_i of every G_i . Then, we apply dynamic programming for every G_i in order to find latent templates $L(\tau_{i,u}) \in \Delta_L$ and their optimal durations $\tau_{i,u} \subset [t_{i,0}, t_{i,T}]$. In the sequel, we drop notion i for the group, for simplicity.

The optimal assignment of sub-events can be formulated using a graph, shown in Figure 7.22. To this end, we partition $[t_0, t_T]$ into equal-length time intervals $\{[t_{k-1}, t_k]\}$, where $t_k - t_{k-1} = \delta t$, $\delta t = 2\text{sec}$. Nodes L_a^k in the graph represent the assignment of templates $L_a \in \Delta_L$ to the intervals $[t_{k-1}, t_k]$. The graph also has the source and sink nodes.

Directed edges in the graph are established only between nodes $L_a^{k'}$ and L_a^k , $1 \leq k' < k$, to denote a

possible assignment of the very same template L_a to the temporal sequence $[t_{k'}, t_k]$. The directed edges are assigned weights (a.k.a. belief messages), $m(L_a^{k'}, L_a^k)$, defined as

$$m(L_a^{k'}, L_a^k) = \log p(L_a(t_{k'}, t_k) | G_i(t_{k'}, t_k)), \quad (7.17)$$

where $\log p(L_a(t_{k'}, t_k) | G_i(t_{k'}, t_k))$ is given by Eq. (7.14). Consequently, the belief of node L_a^k is defined as

$$b(L_a^k) = \max_{k', a'} b(L_a^{k'}) + m(L_a^{k'}, L_a^k). \quad [\text{Forward pass}] \quad (7.18)$$

Here $b(L_a^0) = 0$. We compute the optimal assignment of latent sub-events using the above graph in two passes. In the *forward pass*, we compute the beliefs of all nodes in the graph using Eq. (7.18). Then, in the *backward pass*, we backtrace the optimal path between the sink and source nodes, in the following steps:

- 0: Let $t_k \leftarrow t_T$;
- 1: Find the optimal sub-event assignment at time t_k as $L_{a^*}^k = \arg \max_a b(L_a^k)$; let $a \leftarrow a^*$;
- 2: Find the best time moment in the past t_{k^*} , $k^* < k$, and its best sub-event assignment as $L_{a^*}^{k^*} = \max_{a', k'} b(L_a^{k'}) + m(L_a^{k'}, L_a^k)$; Let $a \leftarrow a^*$ and $k \leftarrow k^*$.
- 3: If $t_k > t_0$, go to Step 2.

7.4 Parsing with Event Grammars

In this section, we first show the event parsing process by assuming that there is only one agent in the scene in Section 7.4.1 - 7.4.4. In Section 7.4.5 we show how to parse events when there are multiple agents in the scene.

7.4.1 Formulation of Event Parsing

The input of our algorithm is a video I_\wedge in a time interval $\wedge = [0, T]$, and atomic actions are detected at every frame I_t . We denote by \wedge_{pg_i} the time explained by parse graph pg_i . $PG = (K, pg_1, \dots, pg_K)$ is regarded as an interpretation of I_\wedge where

$$\begin{cases} \bigcup_{i=1}^K \wedge_{pg_i} = \wedge \\ \wedge_{pg_i} \cap \wedge_{pg_j} = \emptyset \quad \forall i, j \quad i \neq j \end{cases} \quad (7.19)$$

We use a small T-AOG in Figure 7.23(a) to illustrate the algorithm. Figure 7.23(b) shows a sample input of atomic actions. Note that there are multiple atomic actions at each time point. Figure 7.23(c), (d) and (e) show three possible parse graphs (interpretations) of the input up to time t_4 . $PG_1 = (1, pg_1)$ in figure 7.23(c) is an interpretation of the video $I_{[t_1, t_4]}$ and it segments $I_{[t_1, t_4]}$ into one single event E_1 at the event level, and segments $I_{[t_1, t_4]}$ into three atomic actions a_1, a_3 and a_4 at the atomic action level. $PG_2 = (2, pg_2, pg_3)$ in Figure 7.23(d) segments $I_{[t_1, t_4]}$ into two single events E_1 and E_2 , where E_2 is inserted in the process of E_1 . Similarly $PG_3 = (2, pg_4, pg_5)$ in Figure 7.23(e) is another parse graph and segments $I_{[t_1, t_4]}$ into two single events E_1 and E_2 .

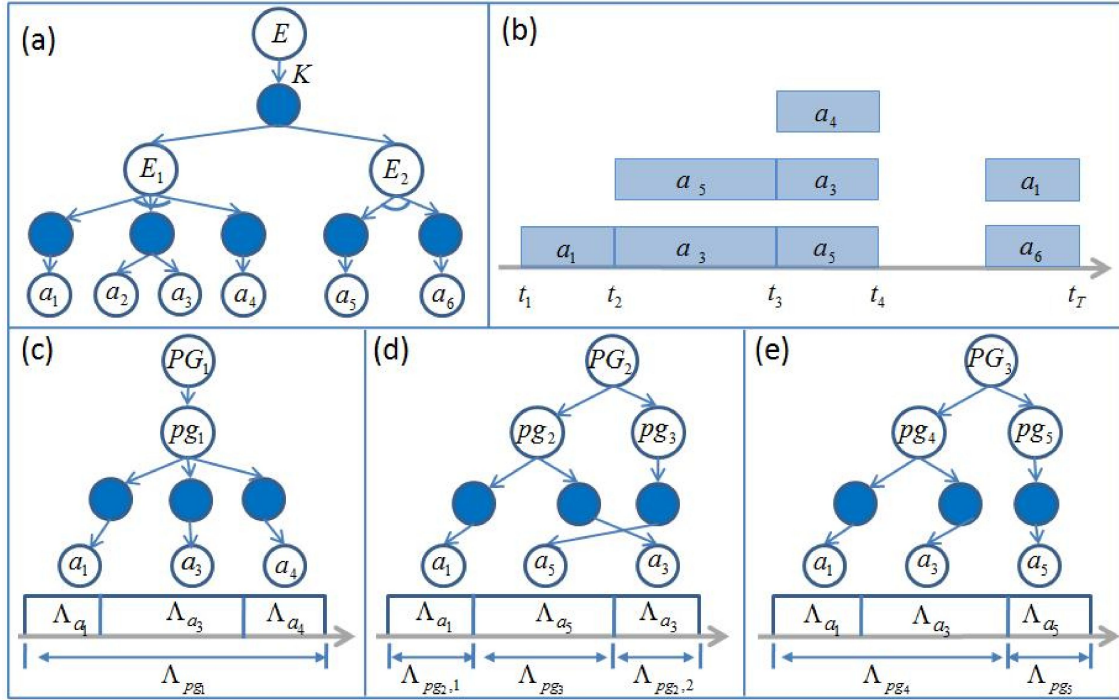


Figure 7.23: (a) A small T-AOG. (b) A typical input of the algorithm. (c), (d) and (e) are three possible parse graphs (interpretations) of the video $I_{\wedge[t_1, t_4]}$. Each interpretation segments the video $I_{\wedge[t_1, t_4]}$ into single events at the event level and into atomic actions at the atomic action level. Reprinted from Ref. [110] with permission.

We can see that the segmentation of events is automatically integrated in the parse process and each interpretation could segment the video I_{\wedge} into single events, and remove the ambiguities in the detection of atomic actions by the event context. The energy of PG is

$$E(PG | I_{\wedge}) = p(K) \sum_{k=1}^K (\varepsilon(pg_k | I_{\wedge_{pg_k}}) - \log p(k)) \quad (7.20)$$

where $p(k)$ is the prior probability of the single event whose parse graph in PG is pg_k , and $p(K)$ is a penalty item that follows the poisson distribution as $p(K) = \frac{\lambda_T^K e^{-\lambda_T}}{K!}$ where λ_T is the expected number of parse graphs in I_{\wedge} . The probability for PG is of the following form

$$p(PG | I_{\wedge}) = \frac{1}{Z} \exp\{-E(PG | I_{\wedge})\} \quad (7.21)$$

where Z is the normalization factor and is summed over all PG . The most likely interpretation of I_{\wedge} can be found by maximizing the following posterior probability

$$PG^* = \arg \max_{PG} p(PG | I_{\wedge}) \quad (7.22)$$

When the most possible interpretation is obtained, the goal at frame I_T can be inferred as the single event whose parse graph pg_i explains I_T , and the intent can be predicted by the parse graph pg_i .

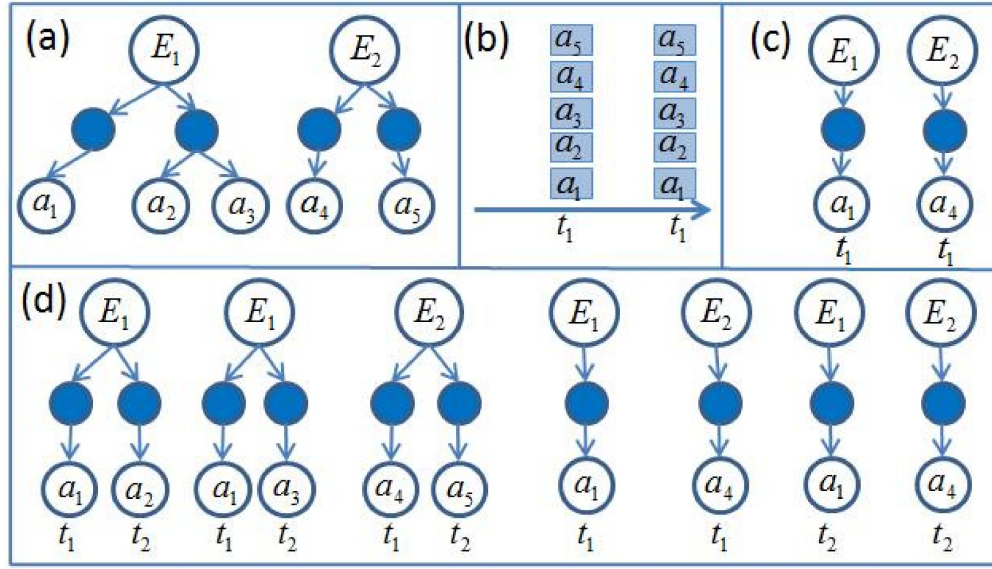


Figure 7.24: (a) The two T-AOGs of single event E_1 and E_2 . (b) The input in the worst case. (c) The parse graphs at time t_1 . (d) The parse graphs at time t_2 . Reprinted from Ref. [110] with permission.

7.4.2 Generating Parse Graphs of Single Events

Insights can be drawn from natural language parsing for the parsing problem discussed in Section 7.4.1. As we discussed in Chapter 5, we adopt grammar parsing techniques for generating parse graphs according to T-AoGs. To better take prior context information into consideration, we choose the top-down Earley parser for generating parse graphs. Treating each atomic-action segment as a similar unit to word tokens, the Earley parser operations: predict, scan and complete can be done in a similar way after we translate videos into a sequence of atomic action segments. With the top-down predict operations in Earley parser, we can even make future prediction given the prior context information embedded in the T-AoG. Here we use two simple T-AOGs of E_1 and E_2 without set nodes for illustration to review the basic operations of Earley parser for event parsing.

Here we consider the worst case, that is, at each time, the input will contain all the atomic actions in E_1 and E_2 as shown in Figure 7.24(b). At time t_0 , in the prediction step, E_1 's first atomic action a_1 and E_2 's first atomic action a_4 are put in the open list. At time t_1 , in the scanning step, since a_1 and a_4 are in the input, they are scanned in and there are two partial parse graphs at t_1 as shown in Figure 7.24(c). Notice that we do not remove a_1 and a_4 from the open list. This is because the input is ambiguous, if the input at t_1 is really a_1 , then it cannot be a_4 and should not be scanned in and should stay in the open list waiting for the next input. It is the same that if the input at t_1 is really a_4 . Then based on the parse graphs, a_2, a_3 and a_5 are predicted and put in the open list. Then at time t_1 , we have a_1, a_2, a_3, a_4, a_5 in the open list. At time t_2 , all of the five nodes in the open list are scanned in and we will have 7 parse graphs (five new parse graphs plus the two parse graphs at t_1) as shown in Figure 7.24(d). The two parse graphs at t_1 are kept unchanged at t_2 to preserve the ambiguities in the input. This process will continue iteratively and all the possible parse graphs of E_1 and E_2 will be generated.

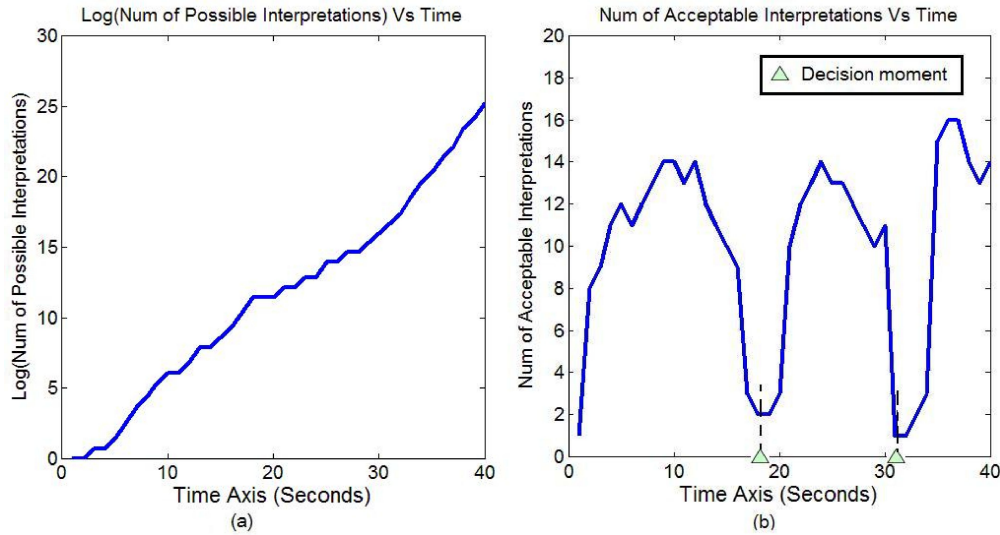


Figure 7.25: (a) The number of possible interpretations (in logarithm) vs. time (in seconds). (b) The number of acceptable interpretations vs. time. The decision moments are the time points on which the critical actions happen and the number of acceptable interpretations drops sharply. Reprinted from Ref. [110] with permission.

7.4.3 Runtime Incremental Parsing

As time passes, the number of parse graphs will increase rapidly and the number of the possible interpretations of the input will become huge, as Figure 7.25(a) shows. However, the number of acceptable interpretations (PG with probability higher than a given threshold) does not keep increasing, instead it will fluctuate and drop sharply at certain time, as shown in Figure 7.25(b). We call these time points the “decision moments”. This resembles human cognition. When people watch others taking some actions, the number of possible events could be huge, but at certain times, when some critical actions occurred, most of the alternative interpretations can be ruled out. After adapting Earley parser to the temporal And-Or graph parsing problem, we can see that the algorithm behaves in a similar way like Figure 7.25. At each frame, we compute the probabilities of all the possible interpretations and only the acceptable interpretations are kept. The parse graphs which are not contained in any of these acceptable interpretations are pruned. This will reduce the complexity of the proposed algorithm greatly.

7.4.4 Generalized Earley Parser

As T-AOGs model events in a symbolic fashion, the nature of observations sets an unbridgable gap between symbolic grammars and sequence data like videos or audios. Parsing sequence data is therefore crucial for T-AOG models to be useful and applicable. Since traditional grammar parsers (*e.g.*, Earley parser) only take symbolic sentences as inputs, the data has to be segmented and labeled so that a symbolic segmentation can be used for grammar parsing. Such methods are inaccurate since the segmentation process was done without any grammatical constraints, the output symbolic sequence will oftentimes be invalid because of the noisy input data. See the example shown in Fig.7.26, in most of the cases, we use a classifier for recognizing actions. To apply parsing algorithms, we need to first tokenize this frame-wise action probability usually through dynamic programming. However, as we show in the same figure, such a dynamic programming segmentation of event is usually noisy and inaccurate. This leads to a significant problem for grammar

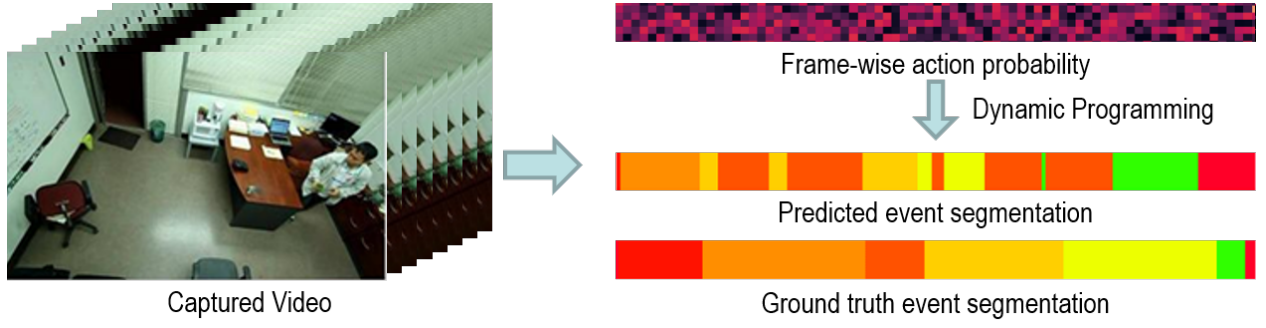


Figure 7.26: Visualization of the full process parsing T-AoG with Earley parser

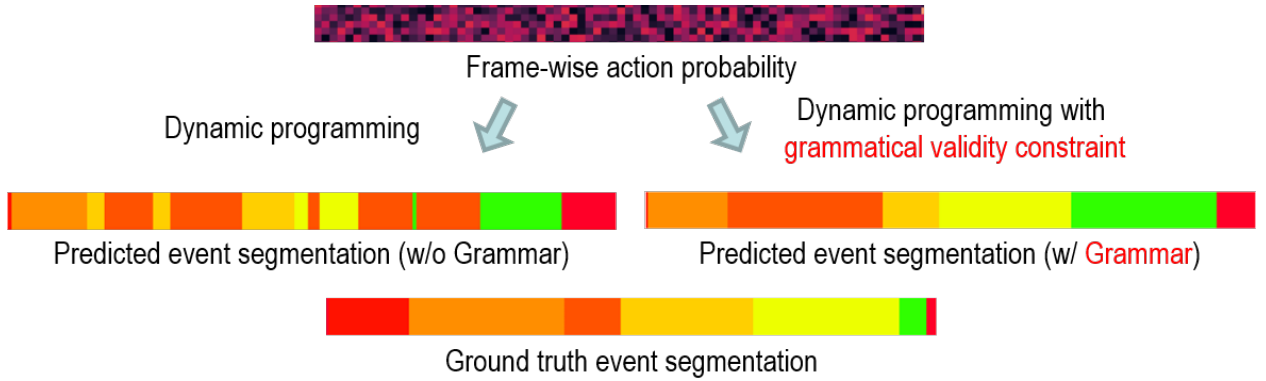


Figure 7.27: The desired effect of adding grammar constraints in action segmentation for better T-AoG parsing and prediction.

parsers as noisy symbol sentences are not valid in grammar and therefore can not be parsed even if they are capturing events that follow the T-AoG in the broader sense.

To solve this issue, one intuitive way is to consider grammar constraints in the segmentation process for generating the action sequence. The desired effect of adding this feature, as shown in Fig.7.27, is to reduce the noise and invalidity of generated action sequences and therefore boosts the parsing process. Instead of following the two-step segment-parse process, we introduce Generalized Earley Parser (GEP), which integrates a grammar parser with a classifier to operate directly on sequence data for finding the optimal parse, and more importantly, parse sequence data with grammatical constraints.

Instead of taking symbolic sentences as input, we aim to design an algorithm that can parse raw sequence data \mathbf{x} of length T (e.g., videos or audios) into a sentence l of labels (e.g., actions or words) of length $|l| \leq T$, where each label $k \in \{0, 1, \dots, K\}$ corresponds to a segment of a sequence. To achieve that, a classifier (e.g., a neural network) is first applied to each sequence \mathbf{x} to get a $T \times K$ probability matrix \mathbf{y} (e.g., softmax activations of the neural network), with y_t^k representing the probability of frame t being labeled as k . The proposed generalized Earley parser takes \mathbf{y} as input and outputs the sentence l^* that best explains the data according to a grammar G of Chomsky normal form. Formally, our objective for parsing the event sequence now becomes

$$l^* = \arg \max_l p(l | x_{0:t}, G)$$

We can calculate this parsing probability by dynamic programming

$$p(l|x_{0:t}, G) = q(a|x_t)(p(l|x_{0:t-1}, G) + p(l^-|x_{0:t-1}, G)) = y_t^a(p(l|x_{0:t-1}, G) + p(l^-|x_{0:t-1}, G))$$

where $q(a|x_t)$ denotes the frame detection probability of action a at time t . Recall that in the conventional segmentation, we do not require l to be grammatically correct. To solve the issue, we can constraint l to be a sentence or a partial sentence of sentence generated by G . The core idea is to use the original Earley parser to help construct a prefix tree according to the grammar as illustrated in Figure 7.29. A prefix tree is composed of terminal symbols and terminations that represent ends of sentences. The root node of the tree is the “empty” symbol. The path from the root to any node in the tree represents a partial sentence (prefix). For each prefix, we can compute the probability that the best label sentence starts with this prefix. This probability is used as a heuristic to search for the best label sentence in the prefix tree: the prefix probabilities prioritize the nodes to be expanded in the prefix tree. The parser finds the best solution when it expands a termination node in the tree. It then returns the current prefix string as the best solution. With this prefix tree, we can calculate probabilities for possible l which can either be a prefix of a full sentence indicating unfinished event, or the full event sequence following a heuristic that looks for the most probable parse. Some examples of prefix tree generated by the Earley parser is shown in Fig.7.28.

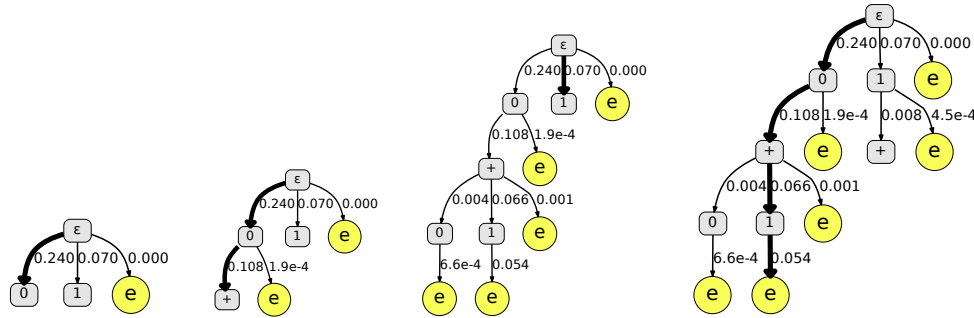
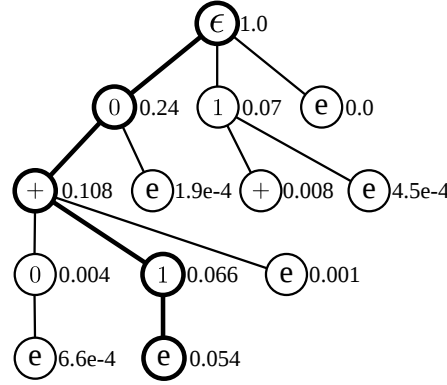


Figure 7.28: The parsing process of the example in Figure 7.29. It performs a heuristic search in the prefix tree according to the prefix/parsing probability. It iteratively expands the tree and computes the probabilities as it expands the tree. The search ends when it hits a parsing terminal e . The paths in bold indicate the best candidates at each search step. Reprinted from Ref. [121] with permission of IEEE, © 2020.

This heuristic search is realized by generalizing the Earley parser. Specifically, the scan operation in the Earley parser essentially expands a new node in the grammar prefix tree. For each prefix l , we can compute $p(l|x_{0:T})$ and $p(l...|x_{0:T})$ based on \mathbf{y} , where $p(l|x_{0:T})$ is the probability of l being the best label, and $p(l...|x_{0:T})$ is the probability of l being the prefix of the best label of $x_{0:T}$. The formulations for $p(l|x_{0:T})$ and $p(l...|x_{0:T})$ are derived in Equation 7.24.

Each scan operation will create a new set $S(m, n) \in S(m)$, where m is the length of the scanned string, n is the total number of the terminals that have been scanned at position m . This can be thought of as creating a new leaf node in the prefix tree, and $S(m)$ is the set of all created nodes at level m . A priority queue q is kept for state sets for prefix search. Scan operations will push the newly created set into the queue with priority $p(l...)$, where l is the parsed string of the state being scanned.

Each state is a tuple $(A \rightarrow \alpha \cdot \beta, i, j, l, p(l...))$ augmented from the original Earley parser by adding $j, l, p(l...)$. Here l is the parsed string of the state, and i, j are the indices of the set that this rule originated. The parser then repeatedly executes three operations: prediction, scanning, and completion modified from Earley parser:



$\Gamma \rightarrow R$	1.0
$R \rightarrow N$	0.4
$R \rightarrow N " + " N$	0.6
$N \rightarrow "0"$	0.3
$N \rightarrow "1"$	0.7

Input (classifier output):

frame	"0"	"1"	"+"
0	0.8	0.1	0.1
1	0.8	0.1	0.1
2	0.1	0.1	0.8
3	0.1	0.8	0.1
4	0.1	0.8	0.1

Figure 7.29: Prefix search according to grammar. A classifier is applied to a 5-frame signal and outputs a probability matrix (bottom right) as the input to our algorithm. The proposed algorithm expands a grammar prefix tree (left), where “e” represents termination. It finally outputs the best label “0 + 1” with probability 0.054. The probabilities of children nodes do not sum to 1 since the grammatically incorrect nodes are eliminated from the search. Courtesy of [122].

Generalized Earley Parser Operations

- **Prediction:** for every state in $S(m, n)$ of the form $(A \rightarrow \alpha \cdot B\beta, i, j, l, p(l...))$, add $(B \rightarrow \cdot\gamma, m, n, l, p(l...))$ to $S(m, n)$ for every production in the grammar with B on the left-hand side.
- **Scanning:** for every state in $S(m, n)$ of the form $(A \rightarrow \alpha \cdot a\beta, i, j, l, p(l...))$, append the new terminal a to l and compute the probability $p((l + a)...) \dots$. Create a new set $S(m + 1, n')$ where n' is the current size of $S(m + 1)$. Add $(A \rightarrow \alpha a \cdot \beta, i, j, l + a, p((l + a)...) \dots)$ to $S(m + 1, n')$. Push $S(m + 1, n')$ into q with priority $p((l + a)...) \dots$.
- **Completion:** for every state in $S(m, n)$ of the form $(A \rightarrow \gamma \cdot, i, j, l, p(l...))$, find states in $S(i, j)$ of the form $(B \rightarrow \alpha \cdot A\beta, i', j', l', p(l'...))$ and add $(B \rightarrow \alpha A \cdot \beta, i', j', l, p(l'...))$ to $S(m, n)$.

This parsing process is efficient since we do not need to search through the entire tree. As shown in Figure 7.29 and algorithm 14, the best label sentence l is returned when the probability of termination is larger than any other prefix probabilities. As long as the prefix probability is computed correctly, it is guaranteed to return the best solution.

The original Earley parser is a special case of the generalized Earley parser. Intuitively, for any input sentence to Earley parser, we can always convert it to one-hot vectors and apply the proposed algorithm. On the other hand, the original Earley parser cannot be applied to segmented one-hot vectors since the labels are often grammatically incorrect. Hence we have the following proposition.

Proposition 1. *Earley parser is a special case of the generalized Earley parser.*

Proof. Let $L(G)$ denote the language of grammar G , $h(\cdot)$ denote a one-to-one mapping from a label to a one-hot vector. $L(G)$ is the input space for Earley parser. $\forall l \in L(G)$, the generalized Earley parser accepts

$h(l)$ as input. Therefore the proposition follows. \square

The parsing probability $p(l|x_{0:T})$ is computed in a dynamic programming fashion as we previously discussed. Here we provide a more formal definition with boundary cases considered. Let k be the last label in l . For $t = 0$, the probability is initialized by:

$$p(l|x_0) = \begin{cases} y_0^k & l \text{ contains only } k \\ 0 & \text{otherwise} \end{cases} \quad (7.23)$$

Let l^- be the label sentence obtained by removing the last label k from the label sentence l . For $t > 0$, the last frame t must be classified as k . The previous frames can be labeled as either l or l^- . Then we have:

$$p(l|x_{0:t}) = y_t^k (p(l|x_{0:t-1}) + p(l^-|x_{0:t-1})) \quad (7.24)$$

It is worth mentioning that when y_t^k is wrongly given as 0, the dynamic programming process will have trouble correcting the mistake. Even if $p(l^-|x_{0:t-1})$ is high, the probability $p(l|x_{0:t})$ will be 0. Fortunately, since the softmax function is usually adopted to compute y , y_t^k will not be 0 and the solution will be kept for consideration.

Then we compute the prefix probability $p(l...|x_{0:T})$ based on $p(l^-|x_{0:t})$. For l to be the prefix, the transition from l^- to l can happen at any frame $t \in \{0, \dots, T\}$. Once the label k is observed (the transition happens), l becomes the prefix and the rest frames can be labeled arbitrarily. Hence the probability of l being the prefix is:

$$p(l...|x_{0:T}) = p(l|x_0) + \sum_{t=1}^T y_t^k p(l^-|x_{0:t-1}) \quad (7.25)$$

In practice, the probability $p(l|x_{0:t})$ decreases exponentially as t increases and will soon lead to numeric underflow. To avoid this, the probabilities need to be computed in log space. The time complexity of computing the probabilities is $O(T)$ for each sentence l because $p(l^-|x_{0:t})$ are cached. The worst-case complexity of the entire parsing is $O(T|G|)$.

As we only used the bottom-up likelihood from the classifier output in Equation 7.24 and Equation 7.25, we can also formulate the grammar prior probability into the iterative calculation of parsing probability. We first give the definition of grammar prefix probability $p(l...|G)$ and later incorporate it into the parsing and prefix probability formulation.

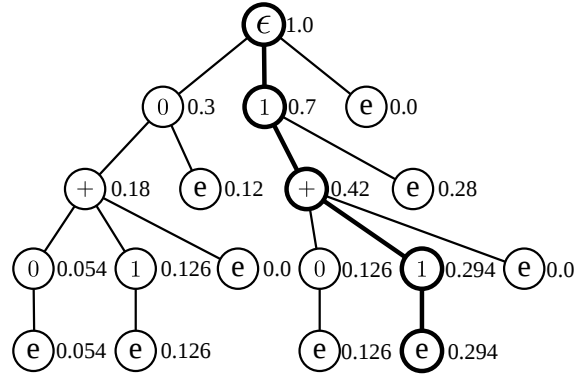


Figure 7.30: Grammar prefix probabilities computed according to the grammar in Figure 5.28. The numbers next to the tree nodes are prefix probabilities according to the grammar. The transition probabilities can be easily computed from this tree, *e.g.*, $p("1"|"1+", G) = p("1+1" \dots | G) / p("1+" \dots | G) = 0.294 / 0.42 = 0.7$. Reprinted from Ref. [121] with permission of IEEE, © 2020.

Grammar Prefix Probability

- **Prediction.** For $(A \rightarrow \alpha \cdot B\beta, i, [\mu, \nu]) \Rightarrow (B \rightarrow \cdot \gamma, m, [\mu', \nu'])$, the new probabilities are given by

$$\mu' + = \alpha \cdot P(B \rightarrow \gamma), \nu' = P(B \rightarrow \gamma).$$

- **Scanning.** For $(A \rightarrow \alpha \cdot a\beta, i, [\mu, \nu]) \Rightarrow (A \rightarrow \alpha a \cdot \beta, i, [\mu', \nu'])$, we have

$$\mu' = \mu, \nu' = \nu.$$

- **Completion.** For $(A \rightarrow \gamma \cdot, j, [\mu'', \nu''])$ and $(B \rightarrow \alpha \cdot A\beta, i, [\mu, \nu]) \Rightarrow (B \rightarrow \alpha A \cdot \beta, i, [\mu', \nu'])$, we have

$$\mu' + = \mu \cdot \nu'', \nu' = \nu \cdot \nu''.$$

Notice that we can calculate the probability of taking symbol k into sentence l^- to sentence l

$$p(k|l^-, G) = \frac{p(l \dots | G)}{p(l^- \dots | G)}$$

An example is shown in Figure 7.30 for a better intuition. We address two important remarks here. 1) This prior prefix probability is different from the prefix probability based on the likelihood. The prior is the probability that a string is the prefix of a sentence in the language defined by the grammar, without seeing any data; the likelihood is the probability that a string is the prefix of a video's label. 2) This grammar-based transition probability is non-Markovian, since the new symbol is conditioned on the entire history string that has a variable length.

Now, incorporating the grammar transition probability, for $t = 0$, the probability is initialized by:

$$p(l|x_0, G) \propto \begin{cases} p(k|\epsilon, G) y_0^k & l \text{ contains only } k, \\ 0 & \text{otherwise,} \end{cases} \quad (7.26)$$

Frame	ϵ	0	1	0 +	1 +	0 + 0	0 + 1
0	0.000	0.240	0.070	0.000	0.000	0.000	0.000
1	0.000	0.192	0.007	0.014	0.004	0.000	0.000
2	0.000	0.019	7.0e-04	0.104	0.007	4.3e-04	0.001
3	0.000	0.002	5.6e-04	0.012	7.1e-04	0.003	0.059
4	0.000	1.9e-04	4.5e-04	0.001	1.1e-04	6.6e-04	0.054
prefix	1.000	0.240	0.070	0.108	0.008	0.004	0.066

Figure 7.31: Cached probabilities. Reprinted from Ref. [121] with permission of IEEE, © 2020.

where $p(k|\epsilon, G)$ is the probability of appending k to the empty string ϵ , which is equivalent to $p(k...|G)$ or $p(l...|G)$. Notice that the equal sign is replaced by \propto since the right hand side should be normalized by the prior $p(x_0)$ to get the correct posterior.

Whenever we append a new symbol to our sentence, we multiply the probability by the transition probability. Hence for $t > 0$ we have:

$$p(l|x_{0:t}, G) \propto y_t^k (p(l|x_{0:t-1}, G) + p(k|l^-, G)p(l^-|x_{0:t-1}, G)). \quad (7.27)$$

Comparing to Equation 7.24, we multiply the second term by $p(k|l^-, G)$ to account for the transition to symbol k .

Finally the posterior probability of l being the prefix of the label sentence for data x is:

$$p(l...|x_{0:T}, G) = p(l|x_0, G) + \sum_{t=1}^T p(k|l^-, G)y_t^k p(l^-|x_{0:t-1}, G). \quad (7.28)$$

Here we also visualize the parsing process of the example shown in Figure 7.29. First, we show the calculated parsing probabilities in Figure 7.31.

We further visualize the state sets generated from generalized Earley parser and also the searching process similar to what we did in Section 7.4.2. The state sets are shown in Figure 7.32 and the searching process is shown in Figure 7.28.

state #	rule	μ	ν	prefix	comment
$S(0, 0) : l = \epsilon, p(l G) = 1.000, p(l x, G) = 0.000, p(l... x, G) = 1.000$					
(0)	$\Gamma \rightarrow \cdot R$	1.000	1.000	" ϵ "	start rule
(1)	$R \rightarrow \cdot N$	0.400	0.400	" ϵ "	predict: (0)
(2)	$R \rightarrow \cdot N + N$	0.600	0.600	" ϵ "	predict: (0)
(3)	$N \rightarrow \cdot 0$	0.300	0.300	" ϵ "	predict: (1),(2)
(4)	$N \rightarrow \cdot 1$	0.700	0.700	" ϵ "	predict: (1),(2)
$S(1, 0) : l = "0", p(l G) = 0.300, p(l x, G) = 1.9e - 04, p(l... x, G) = 0.240$					
(0)	$N \rightarrow 0 \cdot$	0.300	0.300	"0"	scan: S(0, 0)(3)
(1)	$R \rightarrow N \cdot$	0.120	0.120	"0"	complete: (0) and S(0, 0)(1)
(2)	$R \rightarrow N \cdot + N$	0.180	0.180	"0"	complete: (0) and S(0, 0)(2)
(3)	$\Gamma \rightarrow R \cdot$	0.120	0.120	"0"	complete: (1) and S(0, 0)(0)
$S(1, 1) : l = "1", p(l G) = 0.700, p(l x, G) = 4.5e - 04, p(l... x, G) = 0.070$					
(0)	$N \rightarrow 1 \cdot$	0.700	0.700	"1"	scan: S(0, 0)(4)
(1)	$R \rightarrow N \cdot$	0.280	0.280	"1"	complete: (0) and S(0, 0)(1)
(2)	$R \rightarrow N \cdot + N$	0.420	0.420	"1"	complete: (0) and S(0, 0)(2)
(3)	$\Gamma \rightarrow R \cdot$	0.280	0.280	"1"	complete: (1) and S(0, 0)(0)
$S(2, 0) : l = "0 +", p(l G) = 0.180, p(l x, G) = 0.001, p(l... x, G) = 0.108$					
(0)	$R \rightarrow N + \cdot N$	0.180	0.180	"0+"	scan: S(1, 0)(2)
(1)	$N \rightarrow \cdot 0$	0.054	0.300	"0+"	predict: (0)
(2)	$N \rightarrow \cdot 1$	0.126	0.700	"0+"	predict: (0)
$S(2, 1) : l = "1 +", p(l G) = 0.420, p(l x, G) = 1.1e - 04, p(l... x, G) = 0.008$					
(0)	$R \rightarrow N + \cdot N$	0.420	0.420	"1+"	scan: S(1, 1)(2)
$S(3, 0) : l = "0 + 0", p(l G) = 0.054, p(l x, G) = 6.6e - 04, p(l... x, G) = 0.004$					
(0)	$N \rightarrow 0 \cdot$	0.054	0.300	"0 + 0"	scan: S(2, 0)(1)
$S(3, 1) : l = "0 + 1", p(l G) = 0.126, p(l x, G) = \mathbf{0.054}, p(l... x, G) = 0.066$					
(0)	$N \rightarrow 1 \cdot$	0.126	0.700	"0 + 1"	scan: S(2, 0)(2)
(1)	$R \rightarrow N + N \cdot$	0.126	0.126	"0 + 1"	complete: (0) and S(2, 0)(0)
(2)	$\Gamma \rightarrow R \cdot$	0.126	0.126	"0 + 1"	complete: (1) and S(0, 0)(0)

Final output: $l^* = "0 + 1"$ with probability 0.054

Figure 7.32: A run-through of the algorithm. Reprinted from Ref. [121] with permission of IEEE, © 2020.

7.4.5 Multi-agent Event Parsing

When there are multiple agents in the scene, we can do event parsing for each agent separately. That is, for each agent in the scene, the atomic actions are detected (all other agents are regarded as objects in the scene) and parsed as mentioned above, then the interpretations of all the agents in the scene are obtained.

7.5 Learning the T-AoG

7.5.1 Information Projection

The unsupervised learning of stochastic T-AOG is conducted under the information projection and minimum description length principle [139]. Here we provide a review of the related theoretical instruments.

Let $\mathcal{X}_+ = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ be positive examples (*e.g.* observed video clips) governed by an unknown target distribution $f(\mathbf{x})$. Let \mathcal{X}_- be a large set of random negative examples governed by a reference distribution

Algorithm 14: Generalized Earley Parser

```
Input : Grammar  $G$ , probability matrix  $y$ 
Output: Best label string  $l^*$ 
/* For brevity, we denote  $p(\cdot; x_{0:t})$  as  $p(\cdot)$  */
/* Initialization */
1  $S(0, 0) = \{(\Gamma \rightarrow \cdot R, 0, 0, \epsilon, 1.0)\}$ 
2  $q = \text{priorityQueue}()$ 
3  $q.\text{push}(1.0, (0, 0, \epsilon, S(0, 0)))$ 
4 while  $(m, n, l^-, \text{currentSet}) = q.\text{pop}()$  do
5   for  $s = (r, i, j, l, p(l...)) \in \text{currentSet}$  do
6     if  $p(l) > p(l^*)$ :  $l^* = l$  then  $l^* = l$ 
7     if  $r$  is  $(A \rightarrow \alpha \cdot B\beta)$  then // predict
8       for each  $(B \rightarrow \Gamma)$  in  $G$  do
9          $r' = (B \rightarrow \cdot \Gamma)$ 
10         $s' = (r', m, n, l, p(l...))$ 
11         $S(m, n).\text{add}(s')$ 
12      end
13    end
14    else if  $r$  is  $(A \rightarrow \alpha \cdot a\beta)$  then // scan
15       $r' = (A \rightarrow \alpha a \cdot \beta)$ 
16       $m' = m + 1, n' = |S(m + 1)|$ 
17       $s' = (r', i, j, l + a, p((l + a)...))$ 
18       $S(m', n').\text{add}(s')$ 
19       $q.\text{push}(p((l + a)...), (m', n', S(m', n')))$ 
20    end
21    else if  $r$  is  $(B \rightarrow \Gamma \cdot)$  then // complete
22      for each  $((A \rightarrow \alpha \cdot B\beta), i', j')$  in  $S(i, j)$  do
23         $r' = (A \rightarrow \alpha B \cdot \beta)$ 
24         $s' = (r', i', j', l, p(l...))$ 
25         $S(m, n).\text{add}(s')$ 
26      end
27    end
28    if  $p(l^-) > p(l)$ ,  $\forall$  un-expanded  $l$  then return  $l^*$ 
29  end
30 end
31 return  $l^*$ 
```

$q(\mathbf{x})$ (here q is an i.i.d. uniform distribution). For each example \mathbf{x} , a list of spatial relations

$$(r_1(\mathbf{x}), r_2(\mathbf{x}), \dots, r_D(\mathbf{x}))$$

are extracted from the video clip. These relations form a predefined alphabet, just like the set of weak classifiers in adaboost. Our objective of learning is to pursue a model $p(\mathbf{x})$ to approximate $f(\mathbf{x})$ in a series of steps:

$$q(\mathbf{x}) = p_0(\mathbf{x}) \rightarrow p_1(\mathbf{x}) \rightarrow \dots \rightarrow p_T(\mathbf{x}) = p(\mathbf{x}) \approx f(\mathbf{x})$$

starting from q .

The above model updates are performed by selecting a most informative subset from all the spatial relations. The model p after T iterations contains T selected spatial relations $\{r_t : t = 1, \dots, T\}$. If the selected spatial relations capture all the related information about the scene semantics in \mathbf{x} , it can be shown by variable transformation [170] that:

$$\frac{p(\mathbf{x})}{q(\mathbf{x})} = \frac{p(r_1, \dots, r_T)}{q(r_1, \dots, r_T)}.$$

So p can be constructed by reweighting q with the marginal likelihood ratio on selected spatial relations.

Under the maximum entropy principle, $p(\mathbf{x})$ can be expressed in the following log-linear form:

$$p(\mathbf{x}) = q(\mathbf{x}) \prod_{t=1}^T \left[\frac{1}{z_t} \exp \{ \beta_t r_t(\mathbf{x}) \} \right]. \quad (7.29)$$

where β_t is the parameter for the t -th selected spatial relation r_t and z_t ($z_t > 0$) is the individual normalization constant determined by β_t :

$$z_t = \sum_{r_t} q(r_t) \exp \{ \beta_t r_t \}.$$

By the information projection principle [27, 170, 186], we adopt a step-wise procedure for selecting spatial relations. In particular, the t -th spatial relation r_t is selected and model p_t is updated by:

$$\begin{aligned} p_t &= \arg \min \mathcal{K}(p_t | p_{t-1}) \\ s.t. \quad E_{p_t}[r_t] &= \frac{1}{N} \sum_{i=1}^N r_t(\mathbf{x}_i) \end{aligned} \quad (7.30)$$

where \mathcal{K} denotes the Kullback-Leibler divergence, and by minimizing it we select a most informative spatial relation r_t to augment p_{t-1} towards p_t . The constraint equation in Eq. (7.30) ensures that the updated model is consistent with the observed training examples on marginal statistics. The optimal β_t can be found by a simple line search or gradient descent to satisfy the constraint in Eq. (7.30).

7.5.2 Block Pursuit on Data Matrix

Data matrix. Firstly we set up a data matrix \mathbf{r} using spatial relations of positive training examples as shown in Figure 7.33. Each row of \mathbf{r} is the vector of spatial relations detected from one example (or video clip) in \mathcal{X}_+ . For simplicity, we assume all positive training examples are aligned and have the same dimensionality. Therefore \mathbf{r} is a matrix with N (number of positive examples) rows and D (number of all candidate spatial relations) columns, and each entry

$$r_{ij} = r_j(\mathbf{x}_i)$$

is a binary response. $r_{ij} = 1$ means the spatial relation j holds in example \mathbf{x}_i .

Block pursuit. On the data matrix, we pursue large homogeneous blocks $\{\mathcal{B}_k : k = 1, \dots, K\}$. A block is specified by a set of common spatial relations (columns) that co-occur in a set of examples (rows). Each block corresponds to a frequent verb concept, *i.e.* a terminal node or And-node composed of several spatial relations. For example, the verb concept a_{02} (arrive at the door) in Table 7.2 is composed by two spatial relations: $\text{near}(\text{agent}, \text{door})$ and $\text{stand}(\text{agent})$. The verb concept emerges from data because it appears frequently and with high confidence, thus it is readily represented by an AND node that strongly binds its

	t = 1					t = 2					t = T					
	r ₁	r ₂	r ₃	...	r _K	r ₁	r ₂	r ₃	...	r _K	...	r ₁	r ₂	r ₃	...	r _K
Clip 1	0	1	0		1	0	0	1		1		1	1	0		0
Clip 2	0	1	0		1	0	0	1		1		1	1	0		0
Clip 3	0	1	0		1	0	0	1		1		1	1	0		0
...	...															
Clip 9	0	0	1		1	0	1	1		1		0	1	1		0
Clip 10	0	0	1		1	0	1	1		1		0	1	1		0
...	...															
Clip 55	0	1	0		0	0	1	1		1		1	0	0		1
...	...															

Figure 7.33: Pursuing homogeneous blocks from the data matrix. Each block corresponds to a terminal node or an And-node in T-AOG. Reprinted from Ref. [139] with permission.

constituent relations. Quantitatively, we can measure this by the information gain of block \mathcal{B}_k , computed by the summation over the block:

$$\text{Gain}(\mathcal{B}_k) = \sum_{\substack{i \in \text{rows}(\mathcal{B}_k) \\ j \in \text{cols}(\mathcal{B}_k)}} (\beta_{k,j} r_{i,j} - \log z_{k,j}) \quad (7.31)$$

where $\text{rows}(\cdot)$ and $\text{cols}(\cdot)$ denote the rows and columns of block \mathcal{B}_k . $\text{cols}(\mathcal{B}_k)$ correspond to the selected spatial relations, capturing their co-occurrence in space and time. And $\text{rows}(\mathcal{B}_k)$ are the examples that belong to the k -th block. $\beta_{k,j}$ is the multiplicative parameter of selected spatial relation j , and $z_{k,j}$ is the individual normalizing constant determined by $\beta_{k,j}$. Eq. (7.31) measures the information gain by explaining the submatrix covered by \mathcal{B}_k using the foreground model p instead of the background model q . [138] provides more detailed explanation about the block pursuit algorithm applied on learning visual parts from images. Similar approaches have also been adopted in the grammar learning of textual data [150].

Recall that we pursue a series of models starting from $q(\mathbf{x})$ to approximate the target distribution $f(\mathbf{x})$ governing training positives \mathcal{X}_+ . This corresponds to maximizing the log-likelihood $\log p(\mathbf{x})$ on \mathcal{X}_+ . Initially $p = q$, and the data matrix has a log-likelihood $L_0(\mathbf{r})$. After pursuing K blocks, the resulting image log-likelihood is:

$$L = L_0 + \sum_{k=1}^K \text{Gain}(\mathcal{B}_k). \quad (7.32)$$

The block pursuit algorithm is a greedy procedure that maximizes the log-likelihood in Eq. (7.32). Each time we select rows and columns of the data matrix to pursue the block with the largest gain as computed in Eq. (7.31). The entries covered by the block are then explained away and excluded from subsequent block pursuit. This procedure is repeated until the information gain of the newly pursued block is negligible.

To penalize the model complexity, we apply a constant penalty for each additional block learned. This is equivalent to imposing a Laplacian prior on the size of the learned grammar.

The above block pursuit procedure can be implemented either by clustering, which produces multiple blocks or non-terminal nodes at the same time, or by stepwise pursuit, which produces one block or non-terminal node at a time.

Name	Definition	Description
r_{01}	absent(agent)	not found in the frame
r_{02}	near(agent, other_agent)	near other agent
r_{03}	near(agent, board)	near the white board
r_{04}	near(agent, door)	near the door
r_{05}	near(agent, dispenser)	near the water dispenser
r_{06}	near(agent, trash_can)	near the trash can
r_{07}	near(agent, mug)	near the mug
r_{08}	near(agent, laptop)	near the laptop
r_{09}	near(agent, phone)	near the phone
r_{10}	near(agent, basin)	near the basin
r_{11}	near(agent, microwave)	near the microwave
r_{12}	near(agent, tea_box)	near the tea box
r_{13}	in(agent, door)	agent is in the door
r_{14}	touch(agent, keyboard)	typing on keyboard
r_{15}	touch(agent, mug)	grabbing the mug
r_{16}	touch(agent, phone)	grabbing the phone
r_{17}	touch(agent, tea_box)	grabbing the tea box
r_{18}	bend(agent)	bending down
r_{19}	sit(agent)	sitting on something
r_{20}	raise_arm(agent)	raising arm
r_{21}	stand(agent)	standing straight
r_{22}	occlude(soccer match, screen)	soccer match on the screen
r_{23}	on(phone)	phone is in use
r_{24}	on(screen)	screen is on

Table 7.1: The grounded unary and binary relations of event grammar: relations directly detectable from video. From Ref. [111].

The block pursuit procedure for T-AOG is carried out into two stages. (1) Learn a set of terminal nodes as blocks on the data matrix of grounded spatial relations. These terminal nodes account for *atomic events* which directly specify spatial-temporal configurations of grounded relations. This is done by clustering. (2) Learn non-terminal nodes as blocks on the data matrix of atomic actions, to account for longer events composed of atomic actions.

7.5.3 Detecting Grounded Spatial Relations

As a preprocessing step, we perform one round of bottom-up detection for grounded spatial relations.

Firstly we use a standard background subtraction algorithm to segment moving agent and fluent changes of objects, and use a commercial surveillance system to track the detected agent.

The relations of agents' location are detected by the distance between agent and objects which belongs to normal distribution. The location of the agent is detected by combining foreground segmentation and skin color detection that locates head and hands of the agent. Then the distances between agent and objects are computed directly as the locations of objects are known (automatically detected or manually labeled).

The agent pose is inferred by the nearest neighbor classifier using both pixels and foreground segmentation map within the estimated bounding box for the agent. The agent-environment interaction `touch(agent,`

Node Name	Semantic Name	Contained Relations
a_{01}	absent	r_{01}
a_{02}	arrive at door	r_{04}, r_{21}
a_{03}	enter door	r_{04}, r_{21}, r_{13}
a_{04}	stand near phone	r_{09}, r_{21}
a_{05}	sit near phone	r_{09}, r_{19}
a_{06}	stand and use phone	$r_{09}, r_{21}, r_{16}, r_{23}$
a_{07}	sit and use phone	$r_{09}, r_{19}, r_{16}, r_{23}$
a_{08}	arrive at trashcan	r_{06}, r_{21}
a_{09}	throw trash	r_{06}, r_{18}
a_{10}	arrive at basin	r_{10}, r_{21}
a_{11}	dump water	r_{10}, r_{18}, r_{15}
a_{12}	arrive at dispenser	r_{05}, r_{21}, r_{15}
a_{13}	use dispenser	r_{05}, r_{18}, r_{15}
a_{14}	arrive at tea box	r_{12}, r_{21}, r_{15}
a_{15}	use tea box	$r_{12}, r_{21}, r_{15}, r_{17}$
a_{16}	arrive at board	r_{03}, r_{21}
a_{17}	discussion	r_{03}, r_{21}, r_{02}
a_{18}	arrive at laptop	r_{08}, r_{21}
a_{19}	sit near laptop	r_{08}, r_{19}
a_{20}	watch soccer	$r_{08}, r_{19}, r_{22}, r_{24}$
a_{21}	celebrate	$r_{08}, r_{20}, r_{22}, r_{24}$
a_{22}	use laptop	$r_{08}, r_{19}, r_{14}, r_{24}$
a_{23}	arrive at microwave	r_{11}, r_{21}
a_{24}	use microwave	r_{11}, r_{18}
a_{25}	arrive at mug	r_{07}, r_{19}
a_{26}	take mug	r_{07}, r_{19}, r_{15}

Table 7.2: Learned atomic actions. From Ref. [111].

keyboard) and `touch(agent, phone)` are detected by checking whether there is enough skin color within the designated area for the laptop and phone, which are static objects in the office environment. The relation `touch(agent, mug)` and `touch(agent, tea box)` are also detected using skin color, and also the unique color and shape of the mug and tea box. When a relation involves an object, the object is tracked until the relation finishes and the new position of the object will be updated.

The environment relations `occlude(soccer match, screen)` is determined by checking whether there is large amount of green color within the designated area of laptop. The `on` relations are detected by the properties of the object area such as intensity histogram of the bounding box.

Using the techniques described above, we detect grounded relations for every video frame. The detection result is organized as a spatial-temporal table where each row corresponds to a time frame. Each column corresponds to a grounded relation.

7.5.4 Learning Atomic Actions

We define *atomic actions* to be simple and transient events composed spatially and temporally by grounded relations. To learn an alphabet of atomic actions, we use a temporal scanning window spanning 5 frames

to collect a large number of small clips. Each 5-frame clip is described by the detected relation vector: $\{(\mathbf{r}_{1,1}, \dots, \mathbf{r}_{1,D}, \dots, \mathbf{r}_{5,1}, \dots, \mathbf{r}_{5,D})\}$ where $D = 24$ is the number of grounded relations detected per frame. A k-means clustering is then performed on the grounded relation vectors of these 5-frame clips, using the simple Hamming distance as the metric. And a centroid of a cluster is simply determined as the grounded relation vector that has the minimal distance to all the cluster members. As the time span is very small, we can assume that the grounded relations (*e.g.* agent location, pose) stay constant during the short period. So we constrain the centroids to be stationary, *i.e.* $\mathbf{r}_{1,d} = \mathbf{r}_{2,d} = \dots = \mathbf{r}_{5,d}, \forall d = 1, \dots, D$. For each cluster, we estimate the symbol probabilities $p(\mathbf{r}_1), \dots, p(\mathbf{r}_{24})$ by counting the member sub-sequences of the cluster. And we represent this stochastic model by its mode (the most likely sub-sequence) as the cluster prototype $\mathbf{r}_{1:24}^{(k)}$ for brevity. Each cluster corresponds to a block pursued in the data matrix in Figure 7.33.

The result of clustering is a list of 26 atomic actions shown in Table 7.2. Each atomic action is represented by a list of grounded relations that are activated. The semantic description for these atomic actions is in Table 7.2. The atomic actions that happen most frequently include a_{19} (sit near laptop), a_{22} (use laptop), a_{20} (watch soccer) and a_{03} (enter door). a_{19} , a_{22} can be considered as constituent components of a longer event “working by laptop”. a_{03} indicates the student is entering or leaving. The learned atomic actions and their relative frequencies are representative and truthful to the video data.

Now the sequence of multi-dimensional relations is encoded by the alphabet of 26 atomic actions. For the computational efficiency in discovering longer events, we use hard assignments by computing the most likely atomic action per every 5 frames. The resulting sequence of atomic actions is

$$\mathbf{w}_{1:T} = (w_1, \dots, w_T), \quad \text{where } w_t \in \{a_{01}, \dots, a_{26}\}$$

and T is the total number of video frames divided by 5.

7.5.5 Learning Longer Events and T-AOG

There is a large variation in the duration of atomic actions. For example, a student may repeatedly enter the office, work for a varying time and leave the office. If we naively group atomic actions into longer ones, we get a large number of repetitive patterns of various lengths, providing little information. To deal with temporal variation, we perform a simple compression operation: every repetitive sub-sequence is summarized into one symbol (*e.g.* $bbbb$ substituted by b). We may interpret this operation as learning a large number of grammar rules in the form $\tilde{N} \rightarrow NN\dots N$ with various lengths of repetition. We estimate a non-parametric model (Figure 7.34) for the length of repetition, or *duration* under maximum likelihood principle. These duration models are used to adjust the data term $E(a_i)$ in Eq. (7.12).

After compression, the original sequence of atomic actions $\mathbf{w}_{1:T}$ is transformed into a much shorter one $\mathbf{c}_{1:M}$ ($M \ll T$) where each symbol c_i takes value from the same domain as w_i .

There will be some frames that none of the relations are activated except r_{21} , that is, in these frames, the agent just stands somewhere not near any interested objects. These frames are regarded as background frames, that is during these frames, no interest event or action happened. The background frames and the frames in which absence is detected are used to separate the video into different sequences, each sequence is a single event.

We then scan the sequence $\mathbf{c}_{1:M}$ to collect subsequences of length l ($l = 2$ in our system) and form a data matrix. Now the columns of this data matrix are atomic actions instead of grounded relations. A large number of homogeneous blocks (*i.e.* frequent sub-sequences) are identified from the data matrix. They are candidates for the right hand side of production rules in the event grammar. From the candidates, we select a subset of production rules in a step-wise fashion.

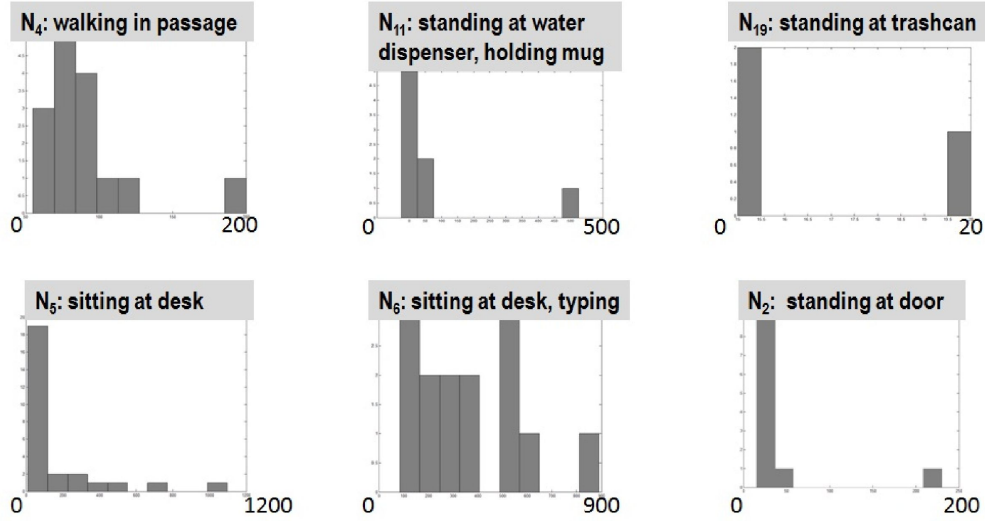


Figure 7.34: The duration model for the length of repetition. Reprinted from Ref. [111] with permission of Elsevier, © 2013.

The proposed candidate production rule takes the form $\alpha \rightarrow \beta\gamma$. It re-encodes the current sequence into a new sequence by replacing all occurrences of $\beta\gamma$ by α . By doing this, the reduction in description length is computed as:

$$\text{reduction} = \Delta_1 + \Delta_2 + \Delta_3 - \text{constant} \quad (7.33)$$

and,

$$\begin{aligned} \Delta_1 &= n'_\alpha \cdot \left(\log \frac{n_\alpha}{n'} - \log \frac{n_\beta}{n} - \log \frac{n_\gamma}{n} \right) \\ \Delta_2 &= n'_\beta \cdot \left(\log \frac{n'_\beta}{n'} - \log \frac{n_\beta}{n} \right) + n'_\gamma \cdot \left(\log \frac{n'_\gamma}{n'} - \log \frac{n_\gamma}{n} \right) \\ \Delta_3 &= (n' - n'_\beta - n'_\gamma - n'_\alpha) \cdot \log \frac{n}{n'} \end{aligned}$$

where $n'_\alpha, n'_\beta, n'_\gamma$ are the frequencies of α, β, γ in the new sequence respectively, n_β, n_γ are the corresponding frequencies in the current sequence. n is the length of the current sequence. $n' = n - n'_\alpha$ is the length of the new sequence. We rank the candidate production rules using Eq.7.33 and select the largest one. This learning procedure is recursively carried out, until the reduction of description length is too small for any new candidate production rule.

The terminal nodes $\{a_1, a_2, \dots\}$ and non-terminal And-nodes form a compositional hierarchy. By learning them altogether, we greatly reduce the ambiguity of segmenting video into events and atomic actions.

7.5.6 Learning the Parameters of T-AOG

After the structure (*i.e.* And-Or nodes) of T-AOG is learned, we can compute the probability of each branch of OR-Node by counting the time each branch appears. This is essentially a maximum likelihood estimation. The details can be found in [185]. Let V_i^{or} be an Or-node and v be an index of one of V_i^{or} 's branches, then

$$p(V_i^{or} = v) = \frac{\sum_{pg \in PG} \mathbf{1}_{V_i(pg)=v}}{|PG|}$$

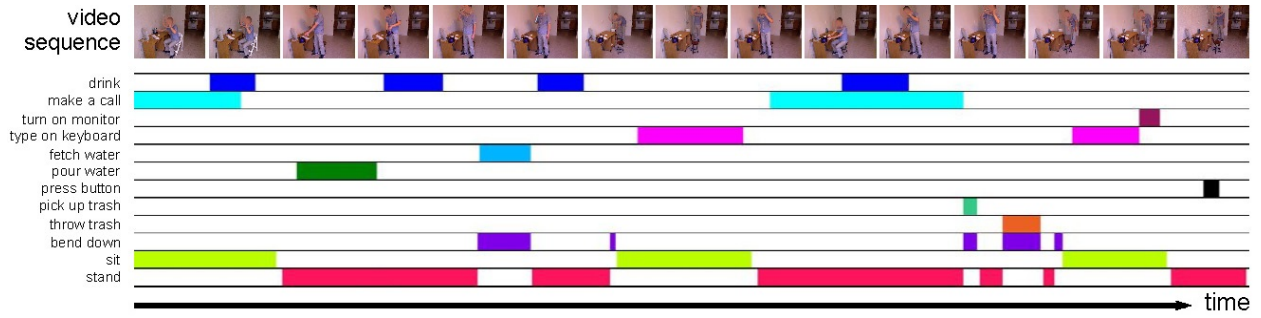


Figure 7.35: An example video and the action annotations from the human activity dataset [165]. Each colored bar denotes the start/end time of an occurrence of an action.

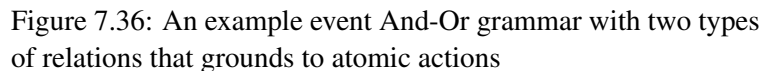
where PG is the set of all parse graphs on the training data.

7.5.7 Example: Learning Event Grammars

We applied our approach to learn event grammars from human activity data. The first dataset contains 61 videos of indoor activities, *e.g.*, using a computer and making a phone call [165]. The atomic actions and their start/end time are annotated in each video, as shown in Figure 7.35. Based on this dataset, we also synthesized a more complicated second dataset by dividing each of the two most frequent actions, sitting and standing, into three subtypes and assigning each occurrence of the two actions randomly to one of the subtypes. This simulates the scenarios in which the actions are detected in an unsupervised way and therefore actions of the same type may be regarded as different because of the difference in the posture or viewpoint.

We employed three different methods to apply our grammar learning approach on these two datasets. The first method is similar to that proposed in [139]. For each frame of a video in the dataset, we construct a binary vector that indicates which of the atomic actions are observed in this frame. In this way, each video is represented by a sequence of vectors. Consecutive vectors that are identical are merged. We then map each distinct vector to a unique ID and thus convert each video into a sequence of IDs. Our learning approach is applied on the ID sequences, where each terminal node represents an ID and each And-node specifies the temporal “followed-by” relation between its child nodes. In the second and third methods, instead of the ID sequences, our learning approach is directly applied to the vector sequences. Each terminal node now represents an occurrence of atomic actions. In addition to the “followed-by” relation, an And-node may also specify the “co-occurring” relation between its child nodes. In this way, the resulting And-Or grammar is directly grounded to the observed atomic actions and is therefore more flexible and expressive than the grammar learned from IDs as in the first method. Figure 7.36 shows such a grammar. The difference between the second and the third method is: in the second method we require the And-nodes with the “co-occurring” relation to be learned before any And-node with the “followed-by” relation is learned, which is equivalent to applying the first method based on a set of IDs that are also learned; on the other hand, the third method does not restrict the order of learning of the two types of And-nodes.

Note that in our learning algorithm we assume that each training sample consists of a single pattern generated from the target grammar, but here each video may contain multiple unrelated events. We slightly modified our algorithm to accommodate this issue: right before the algorithm terminates, we change the top-level And-nodes in the grammar to Or-nodes, which removes any temporal relation between the learned events in each training sample and renders them independent of each other. When parsing a new sample using the learned grammar, we employ the CYK algorithm to efficiently identify all the subsequences that



	Data 1	Data 2
ADIOS [142]	0.810	0.204
SPYZ [139]	0.756	0.582
Ours (f)	0.831	0.702
Ours (c+f)	0.768	0.624
Ours (cf)	0.767	0.813

We used 55 samples of each dataset as the training set and evaluated the learned grammars on the remaining 6 samples. On each testing sample, the events identified by the learned grammars were compared against manual annotations. We measured the purity (the percentage of the identified event durations overlapping with the annotated event durations) and inverse purity (the percentage of the annotated event durations overlapping with the identified event durations), and report the F-measure (the harmonic mean of purity and inverse purity). We compared our approach with two previous approaches [139, 142], both of which can only learn from ID sequences.

Table 7.3 shows the experimental results. It can be seen that our approach is competitive with the previous approaches on the first dataset and outperforms the previous approaches on the more complicated second dataset. Among the three methods of applying our approach, the second method has the worst performance, mostly because the restriction of learning the “co-occurring” relation first often leads to premature equating of different vectors. The third method leads to the best overall performance, which implies the advantage of grounding the grammar to atomic actions and simultaneously learning different relations. Note that the third method has better performance on the more complicated second dataset, and our analysis suggests that the division of sitting/standing into subtypes in the second dataset actually helps the third method to avoid learning erroneous compositions of continuous sitting or standing.

8

Fluent and Causal And-Or Graph

In this chapter, a framework—Causal And-Or Graph is given for the unsupervised learning of causal relations from video, learning which actions cause objects to change status. Then we extend the C-AOG to a sequential model representing actions and their effects on objects over time, and we build a probability model for it.

8.1 Perceptual Causality and Fluent

In this section, we start from daily observation to provide some background knowledge to bring causality and vision together. Imagine: A man approaches a closed door. He reaches out to grasp the handle and then stands there. Is it locked? Does he not have the key? He knocks and waits, but the door remains closed. Is there no one on the other side to open it?

Watching these events unfold, humans can readily answer these questions by reasoning with causal knowledge. Our innate ability to learn causal relationships from daily observation leads to a sophisticated understanding of the semantics of the events, and thus has been a topic of interest for cognitive science researchers [55]. An analogous form of observation by a computer comes through video. Learning causal relationships through video will enable a computer to acquire the same commonsense knowledge.

8.1.1 What are Cause and Fluent?

In order to advance causal models of visual data, it helps to take a step back and ask what do we, as humans, mean when we say, “The man caused the door to open”?

In [92], Mackie addresses the philosophical question of what humans consider a cause. He suggests that humans give “cause” status to INUS conditions, “*insufficient* but *necessary* parts of a condition which is itself *unnecessary* but *sufficient* for the effect”. The acts of unlocking and pushing the door are insufficient in themselves for opening the door, but they compose necessary parts of one sufficient way to open a door. This particular way of opening the door is unnecessary as there are multiple alternative ways the door could have been opened—e.g., someone could have exited the room, opening the door from the other side.

The INUS conditions focuses on are actions, studying short term action causes where an agent’s observable action causes a change in an observable fluent. The concept of fluents was first coined in 1736 by Newton in [101] and has since been adopted in the commonsense reasoning literature to represent time-varying properties of objects [98]. The status of a door, for example, can be open at time t_1 , but closed at time t_2 .

In a scene, there are many types of fluents:

1. *Unary fluents* give the statuses of an object, e.g., the door is open or the light is on.
2. *Compositional fluents* allow objects to be composed of subparts, e.g., the doorknob is part of the door.
3. *Relational fluents* provide spatio-temporal relations between objects and times, e.g., the agent is near the door or the book is on the desk.

Here unary fluents are our main focus. For unary fluents, $F(t)$ is a function returning the status of an object from some predefined set at time t . Each object has its own fluent function, and it is possible to describe an object using more than one set. For a door, examples of unary fluents are $F(t) \in \{\text{open, closed}\}$ and $F(t) \in \{\text{wood, metal, glass}\}$.

Fluents can be further classified according to their pattern of change:

1. *Long-term/Permanent fluents* maintain a constant value over time, such as the color of a door.
2. Some fluents change over time: *periodic fluents* change at regular intervals (such as the alignment of hands of a clock), while *intermittent fluents* are less predictable.

The fluent’s underlying relationship to an agent’s action is classified by:

1. *Active fluents* directly reflect a current status of an agent, related to the agent’s motion. E.g., the agent’s position is moved left means that the agent is moving left.
2. *Passive fluents* change only by an agent applying action to the scene.

8.1.2 Perceptual Causality

Traditional causal discovery methods are insufficient for computer vision tasks. Most importantly, true causal discovery does not necessarily align with human perceptions. Secondly, the traditional methods leave many questions: they do not inform which detection variables humans would indicate as causes or effects (from pixels over time, to features aggregating pixels, to object classifiers using features, to action detectors, to hierarchies thereof); they do not indicate how to divide the video stream to create examples (too small a clip might omit causes; too large of one introduces noise); and they do not encode a prior understanding of likely causes that could be used in detections.

Perceptual causality as studied by cognitive science researchers fills in these gaps.

Humans link, for example, a change in an object status with the action of an agent [130]. Humans award the “cause” distinction to the agent’s action of opening the door (decomposed at a high level into unlocking and pulling open the door), ahead of individual pixels, the door, and the lock (part of the door). We limit ourselves to agentive actions as potential causes of fluent changes. In order to make detections from video, these sets of actions and fluents must be pre-specified so appropriate detectors can be trained.

Considering actions alone is not enough. Actions come hierarchically defined, where, for example, the person opening the door performs the actions unlock and pull. The method we present can correctly select from a hierarchy, as shown in Section 8.4.3.

Humans consider cause and effect relationships when the temporal lag between the two is short and cause precedes effect [60]. We construct examples from the video that only consider actions occurring within a small window preceding a given effect.

Finally, humans link states of the world to perceived causing conditions by measuring co-occurrence [55]. We propose a method to learn this perceptual causality, acquiring the knowledge of causal relationships in the world, illustrated by the dashed arrows in Figure 8.1.

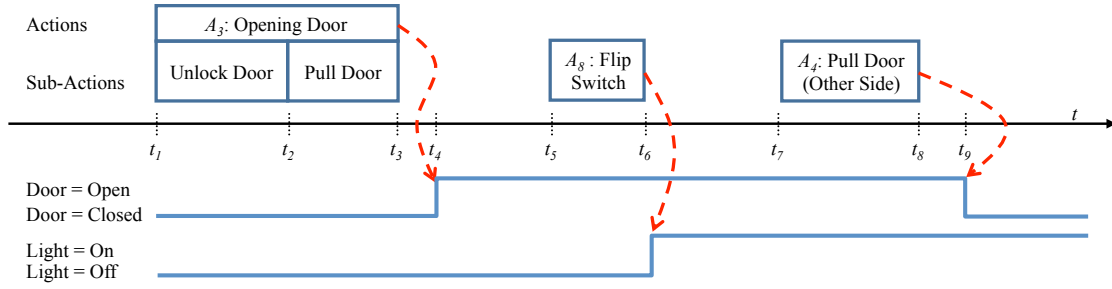


Figure 8.1: Key actions over time are shown at the top, and status changes below. Dashed arrows indicate perceived causal links. A link can be found between any action in an action hierarchy and its relevant fluent change. The joint actions of unlocking and pulling cause the door to open at t_4 . From t_5 to t_6 , a person turns a light on. From t_7 to t_8 , a person pulls the door from the other side, resulting in a closed door at t_9 .

8.2 Vision and Causality

8.2.1 Converting Perceptual Causality to Heuristics

Perceptual causality as presented above grounds causal discovery on video, and distinguishes perceptual causality from the causal modeling typically done in the social and biologic sciences [109], [129]. We now present the heuristics of perceptual causality.

The Heuristics:

1. Agentive actions are causes ,

$$\text{Action} \rightarrow \text{Effect}.$$

This heuristic informs the set of potential causes: It's not the pixels we see or the human that we detect, but it's the human *doing something*.

2. Temporal lag between cause and effect is short, with cause preceding effect,

$$0 < \text{Time}(\text{Effect}) - \text{Time}(\text{Causing Action}) < \epsilon.$$

This provides a method for breaking the video stream into clips to create examples. Determining ϵ is challenging: taking it too small might exclude the cause, and taking it too large creates too much noise. We examine various temporal lags, as well as different ways of measuring the temporal lag, in Experiment ??.

3. Perceptual causal relationships are obtained by measuring co-occurrence between actions and effects.

In this paper, we examine co-occurrence while simultaneously building our model following an information projection pursuit. In Experiment ??, we find our method outperforms Hellinger's χ^2 measure for co-occurrence. In Experiment ??, we show that our method outperforms the treatment effect.

When the computer examines the co-occurrence of Heuristic 3, restricted by Heuristics 1 and 2, then we assume the model determined represents perceptual causality.

8.2.2 Assumptions and Structural Equation Models

In addition to assuming Heuristics 1-3, we also make some assumptions standard to traditional causal discovery.

We assume that our detections (and the hierarchies used for such) are sufficient. In particular, the set of pre-specified actions is sufficient, and the computer is able to generally detect these elements in the scene when they occur.

We assume causal faithfulness: multiple causes do not exactly cancel. When we detect no correlation, we match this to the perception of no causal connection.

We assume each effect is a function of its immediate causes and an independent error. Each action, A_i depends on its own exogenous variable, u_{A_i} . Using ΔF_j to denote fluent change j , we notate in terms of structural equations:

$$A_i = g_{A_i}(u_{A_i}) \text{ for } i = 1, \dots, n_A \quad (8.1)$$

$$\Delta F_j = g_{\Delta F_j}(\mathbf{A}_j, u_{\Delta F_j}) \text{ for } j = 1, \dots, n_{\Delta F} \quad (8.2)$$

where \mathbf{A}_j denotes specifically those actions that are in a causal relationship with ΔF_j . $u_{\Delta F_j}$ are exogenous.

8.2.3 Potential Effects: The Space of Fluent Changes

Given a fluent that can take n_F values, there are $n_{\Delta F} = n_F^2$ possible transitions from time t to $t+1$. With the door, for example, where the fluent could be “open” or “closed”, there are four possible sequences: the door changes from “open” to “closed”, changes from “closed” to “open”, remains “open”, or remains “closed”. We notate the fluent change for a clip with ΔF .

Per the commonsense reasoning literature [98], a lack of change-inducing action (referred to here as “non”-action) causes the fluent to maintain its status, denoted $\Delta F = 0$; for example, a door that is closed will remain closed until some action changes that status. Figure 8.1 shows the door and the light maintaining their statuses for varied durations, punctuated by periods of change due to action.

The space of fluent changes possible over the objects in the video is pre-specified and denoted by

$$\Omega_{\Delta F} = \{\Delta F\}.$$

8.2.4 Potential Causes: The Space of Action Detections

Action parsing provides Ω_A , the space of actions. Ω_A contains actions at high levels of an action hierarchy. An action detection hierarchy (e.g., [110]) aggregates pixels into objects, relates these objects spatially and temporally to define atomic actions, groups those into sub-actions (such as pushing or pulling the door), and hierarchically combines even further (for example, unlocking and pulling the door). Figure 8.1 shows actions from different levels of the hierarchy.

In our case, Ω_A is limited to top-level action or sub-actions from a pre-designed action hierarchy, following Heuristic 1.

8.3 Perceptual Causal Relations

In this section, we formalize our key building block for causal structure, the notion of a perceptual causal relation between an action and a fluent change.

8.3.1 Defining Perceptual Causal Relations

Combining the fluent changes with the actions, we define the space of potential causal relations.

Definition 5 (Space of Causal Relations). *The space of causal relations is given by*

$$\Omega_{CR} = \Omega_A \times \Omega_{\Delta F}. \quad (8.3)$$

The space, Ω_{CR} , provides the basic units for learning. Elements $\mathbf{cr} \in \Omega_{CR}$ specify an action and fluent change, and provide the framework for the 2×2 tables as shown in Table 8.3.1.

Table 8.1: Causal relation.

$\mathbf{cr} :$		Action	\neg Action
	Effect	c_0	c_1
	\neg Effect	c_2	c_3

Labeling the individual cells of the table, $\mathbf{cr} = (c_0, c_1, c_2, c_3)$ where c_i functions as a binary indicator. When applied to a sufficiently short video clip (defined in the next section), the elements of Ω_{CR} identify whether or not the clip has the action and/or fluent change.

When these video clips show strong evidence for elements of Ω_{CR} , we award perceptual causal status and add the elements to our model.

8.3.2 Preparing the Data: Creating Clips from the Video

In order to determine the elements of Ω_{CR} which have the most evidence for being true causal relations, we evaluate the elements using video.

A long video sequence \mathbf{V} is decomposed into shorter video clips, $\mathbf{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_n\}$. Following Heuristic 2 for limiting temporal lag, only actions occurring within a pre-specified τ_{\max} of the fluent change are included in \mathbf{v}_i , to be considered as potential causes. The function $\tau(t_A, t_F)$ measures time between the action completion, t_A , and the fluent change, t_F . Some example functions for $\tau(t_A, t_F)$ include:

1. Counting the number of frames between t_A and t_F . We consider τ_{\max} between 15 and 90 seconds.
2. Counting the number of actions detected between t_A and t_F . We consider τ_{\max} ranging from 1 to 6 recent actions.
3. Combinations of the first two. We consider τ_{\max} to be max or min over combinations of 15, 45 seconds and 1, 2, 3 actions. For example, taking the maximum of 15 seconds and 2 actions creates clips *at most* 15 seconds long or with *at most* 2 action detections. Taking the minimum of 15 seconds and 2 actions creates clips of *at least* 15 seconds or 2 action detections.

These are explored in experiments in Section ???. It is intuitive to expect a dependence between clip length definition and performance. If the clip is not long enough to include the causing action, then the ability to detect causes diminishes. However, if clip length is too long, then there will only be a few examples, not enough information to rise above the noise.

8.3.3 Evaluating Causal Relations

Aggregating the values from $\mathbf{cr} \in \Omega_{CR}$ across the clips, \mathbf{v}_i , we obtain relative frequencies for the particular action and fluent change:

Definition 6 (Relative Frequencies of a Causal Relation). *Given a causal relation \mathbf{cr} and video \mathbf{V} that has been broken into clips $\{\mathbf{v}_1, \dots, \mathbf{v}_n\}$, the relative frequencies of \mathbf{cr} are given by*

$$RF(\mathbf{cr}) = \frac{1}{n} \sum_{i=1}^n \mathbf{cr}(\mathbf{v}_i). \quad (8.4)$$

The relative frequencies from the video’s action and fluent detections are denoted by $\mathbf{f} = (f_0, f_1, f_2, f_3)$.

Our causal model is built by greedily augmenting action and fluent distribution with causal relations, linking actions to fluent changes. At any iteration, there is the model that has been built so far (the “current model”), and the observed data from the video. The limiting relative frequencies under the current model are denoted by $\mathbf{h} = (h_0, h_1, h_2, h_3)$. Table 8.2 summarizes these statistics.

ΔF	A	Current Model	Observed Data
0	0	h_0	f_0
0	1	h_1	f_1
1	0	h_2	f_2
1	1	h_3	f_3

Table 8.2: Relative Frequencies.

We construct our model by electing the most informative causal relations sequentially in terms of maximizing the information gain. Intuitively, this information gain is linked to the difference between \mathbf{f} and \mathbf{h} .

For a causing action, \mathbf{f} is shown in Figure 8.2a, together with the relative frequencies of \mathbf{cr} under a probability model assuming independence, \mathbf{h} . The greatest difference between these histograms occurs in the f_1/h_1 and f_3/h_3 components. The relative frequencies \mathbf{f} and \mathbf{h} for a non-causing action in (b) look equivalent, indicating independence between the fluent and action.

We select the relations that show the greatest difference between \mathbf{f} and \mathbf{h} , as measured by the KL-divergence, thereby adding perceptual causal semantics to the model.

8.4 Pursuit of the Causal Relations

In this section we develop the theoretical framework for the learning theory, formulas of which were provided in [45]. From the space of all possible relations, Ω_{CR} , we now show how to sequentially select \mathbf{cr} and build a joint probability model incorporating them.

The video clips, \mathbf{v}_i , are assumed to be drawn from an unknown distribution of perceptual causality, $f(\mathbf{v})$. We incrementally build a series of models approximating f

$$p_0(\mathbf{v}) \rightarrow p_1(\mathbf{v}) \rightarrow \dots \rightarrow p(\mathbf{v}) \rightarrow p_+(\mathbf{v}) \rightarrow \dots \rightarrow p_k(\mathbf{v}) \approx f(\mathbf{v}), \quad (8.5)$$

where each new model incorporates a new causal relation as illustrated in Figure 8.3. We use an information projection approach (see, e.g., [24]).

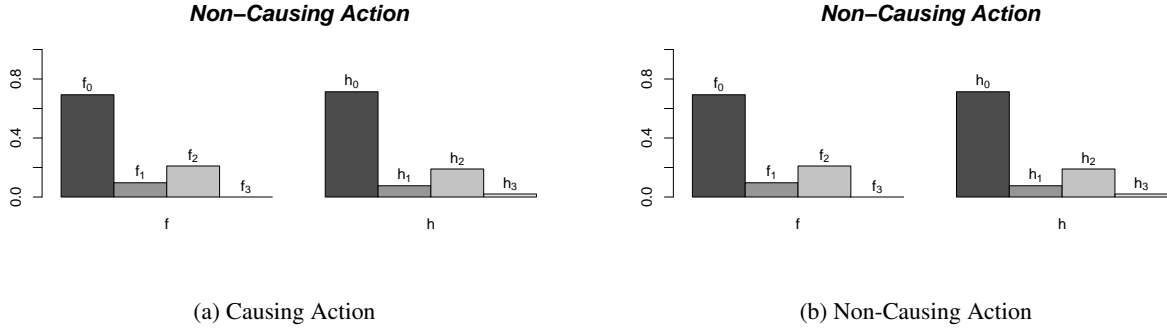


Figure 8.2: Relative frequencies of \mathbf{cr} for the observations is shown on the left of each pair, and for the model of independence on the right.

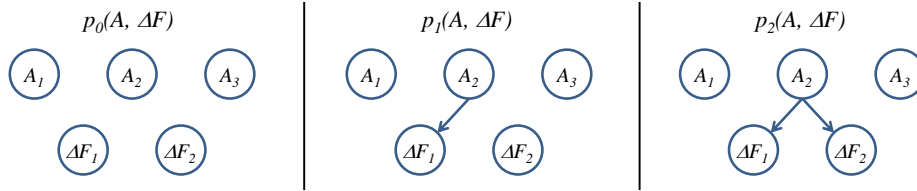


Figure 8.3: The perceptual causal structure is incrementally constructed. Here, the action is flipping the light switch, which can turn the light on or off.

Shown in the first panel of Figure 8.3, learning initializes by independently considering action and fluent distributions, p_A and $p_{\Delta F}$, respectively:

$$p_0(\mathbf{v}) = p_A(\mathbf{v})p_{\Delta F}(\mathbf{v}). \quad (8.6)$$

In this paper, we initialize $p_A(\mathbf{v})$ with the proportion of clips, \mathbf{v} , that contain action A ; similarly for $p_{\Delta F}$.

In a single iteration, we fix the previous model, p , and augment to a new model, p_+ . Under the information projection framework, learning proceeds in two steps. In the first step, we select the causal relation to add to the model by maximizing the KL-divergence between p_+ and p , also known as the information gain. In step two, we fit the selected causal relation to the data by minimizing the KL-divergence between p_+ and p .

Any model over the video clips that considers fluent changes independently from causing actions, such as p_0 , will fail to match f on true causal relations. However, given a selected relation, the latter step requires that the new model match the observed data on the newly selected causal relation

$$E_{p_+}[\mathbf{cr}_+] = E_f[\mathbf{cr}_+] \approx \mathbf{f}. \quad (8.7)$$

The probability distribution with minimum KL-divergence, $\text{KL}(p_+||p)$, subject to that constraint is

$$p_+(\mathbf{v}) = \frac{1}{z_+} p(\mathbf{v}) \exp(-\langle \lambda_+, \mathbf{cr}_+(\mathbf{v}) \rangle) \quad (8.8)$$

where $\lambda_+ = (\lambda_0, \lambda_1, \lambda_2, \lambda_3)$ is a scalar vector corresponding to the components of $\mathbf{cr}_+(\mathbf{v}) = (c_0(\mathbf{v}), c_1(\mathbf{v}), c_2(\mathbf{v}), c_3(\mathbf{v}))$ shown in Table 8.3.1 and described in Section 8.3.1 and z_+ is a normalizing constant. When p_0 is uniform, this procedure yields the maximum entropy distribution.

8.4.1 Fitting the Causal Relation

Unlike in other information projection applications to vision (e.g., [27] [186]), λ_+ can be computed analytically thanks to the binary nature of the causal relation:

Proposition 2. *To add the causal relation \mathbf{cr}_+ to the model in Equation 8.8, the parameters are given by:*

$$\lambda_i = \log \left[\frac{h_i}{h_0} \cdot \frac{f_0}{f_i} \right] \quad (8.9)$$

for $i = 0, \dots, 3$, where h_i and f_i are as found in Table 8.2.

of Prop. 2. Consider adding a single causal relation to the probability distribution, $p(\mathbf{v}) = \frac{1}{Z} \exp(-\mathcal{E}(\mathbf{v}))$. This gives a new probability distribution

$$p_+(\mathbf{v}) = \frac{1}{z_+} p(\mathbf{v}) \exp(-\langle \lambda_+, \mathbf{cr}_+(\mathbf{v}) \rangle). \quad (8.10)$$

Since $\sum_{i=0}^3 c_i = 1$, there is 1 degree of freedom in λ_+ ; without loss of generality, set $\lambda_0 = 0$.

From the observed data, the expected value under the true distribution, f , is best estimated by the quantity from the data,

$$E_f(c_i(\mathbf{v})) = f_i. \quad (8.11)$$

Further, $E_p(c_i(\mathbf{v})) = h_i$.

$$E_{p_+(\mathbf{v})}(c_i(\mathbf{v})) = \int p_+(\mathbf{v}) c_i(\mathbf{v}) d\mathbf{v} \quad (8.12)$$

$$= \int \frac{1}{z_+} p(\mathbf{v}) \exp(-\langle \lambda_+, \mathbf{cr}_+(\mathbf{v}) \rangle) c_i(\mathbf{v}) d\mathbf{v} \quad (8.13)$$

$$= E_p \left(\frac{1}{z_+} \exp(-\langle \lambda_+, \mathbf{cr}_+(\mathbf{v}) \rangle) c_i(\mathbf{v}) \right) \quad (8.14)$$

$$= \frac{1}{z_+} h_i \exp(-\lambda_i) \quad (8.15)$$

The last equation holds because only one of the $c_i(\mathbf{v})$ will be nonzero at a time.

Equating the matched statistics,

$$f_i = \frac{1}{z_+} h_i \exp(-\lambda_i). \quad (8.16)$$

Since $\lambda_0 = 0$, $f_0 = \frac{h_0}{z_+}$, or

$$z_+ = \frac{h_0}{f_0}. \quad (8.17)$$

Hence,

$$\lambda_i = \log \left[\frac{h_i}{h_0} \cdot \frac{f_0}{f_i} \right]. \quad (8.18)$$

□

Intuitively, the h_i/h_0 component “undoes” the independent consideration under the current model, and the f_0/f_i component inserts the new information joining the action and fluent change.

In experiments, $p_0(\mathbf{v})$ is defined over a finite set, and \mathbf{h} is computable.

8.4.2 Pursuing Causal Relations by Information Projection

While Proposition 2 provides a formula to add a causal relation to a model, the best causal relation, \mathbf{cr}_+ , is selected at each step through a greedy pursuit which leads to the maximum reduction of the KL divergence [27], [186]:

$$\mathbf{cr}_+ = \arg \max \mathbf{cr} (\text{KL}(f||p) - \text{KL}(f||p_+)) . \quad (8.19)$$

Equivalently, \mathbf{cr}_+ is added to maximize the information gain:

$$\mathbf{cr}_+ = \arg \max \mathbf{cr} IG_+ := \arg \max \mathbf{cr} \text{KL}(p_+||p) \geq 0, \quad (8.20)$$

moving the model closer to the true distribution f with each new causal relation.

An analytic formula provides the best causal relation:

Proposition 3. *The next best relation, \mathbf{cr}_+ , to add to the model is given by*

$$\mathbf{cr}_+ = \arg \max \mathbf{cr} \text{KL}(p_+||p) = \arg \max \mathbf{cr} \text{KL}(\mathbf{f}||\mathbf{h}) \quad (8.21)$$

where \mathbf{f} and \mathbf{h} are as found in Section 8.3.3.

A proof of this proposition can be found in the Section ??.

of Prop. 3.

$$\text{KL}(p_+||p) = \int p_+(\mathbf{v}) \log \frac{p_+(\mathbf{v})}{p(\mathbf{v})} d\mathbf{v} \quad (8.22)$$

$$= \int p_+(\mathbf{v}) \log \left(\frac{1}{z_+} \exp(-\langle \lambda_+, \mathbf{cr}_+(\mathbf{v}) \rangle) \right) d\mathbf{v} \quad (8.23)$$

$$= \int p_+(\mathbf{v}) \log \frac{1}{z_+} d\mathbf{v} - \int p_+(\mathbf{v}) (\langle \lambda_+, \mathbf{cr}_+(\mathbf{v}) \rangle) d\mathbf{v} \quad (8.24)$$

$$= \log \frac{1}{z_+} - E_{p_+}(\langle \lambda_+, \mathbf{cr}_+(\mathbf{v}) \rangle) \quad (8.25)$$

$$= \log \frac{1}{z_+} - E_f(\langle \lambda_+, \mathbf{cr}_+(\mathbf{v}) \rangle) \quad (8.26)$$

$$= \log \frac{1}{z_+} - \langle \lambda_+, \mathbf{f} \rangle. \quad (8.27)$$

Applying the formula for λ_i ,

$$\lambda_i f_i = f_i \log \left[\frac{h_i}{h_0} \cdot \frac{f_0}{f_i} \right] \quad (8.28)$$

$$= f_i \log \frac{f_0}{h_0} + f_i \log \frac{h_i}{f_i}. \quad (8.29)$$

Continuing from Equation 8.27 and substituting Equations 8.17 and 8.29,

$$\text{KL}(p_+||p) = \log \frac{f_0}{h_0} - \sum_{i=0}^3 \left(f_i \log \frac{f_0}{h_0} + f_i \log \frac{h_i}{f_i} \right) \quad (8.30)$$

$$= (1 - f_1 - f_2 - f_3) \log \frac{f_0}{h_0} + \sum_{i=1}^3 f_i \log \frac{f_i}{h_i} \quad (8.31)$$

$$= f_0 \log \frac{f_0}{h_0} + \sum_{i=1}^3 f_i \log \frac{f_i}{h_i} \quad (8.32)$$

$$= \text{KL}(\mathbf{f}||\mathbf{h}). \quad (8.33)$$

□

Therefore, in order to determine which causal relation is best to add to the model, we calculate the KL-divergence between the current model and the data for each potential causal relation, selecting the one that maximizes the information gain.

Once the relation is selected, perceptual causal arrows can be assigned between A and ΔF according to Heuristic 1 as shown in Figure 8.3.

Algorithm 15 summarizes Propositions 2 and 3.

Algorithm 15: Learning the causal relations.

Input : Action and fluent change detections from the video, $\tau(t_A, t_F)$ and τ_{\max}

Output : Probability distribution over a learned structure of perceptual causality

- 1 Create video clips according to τ and τ_{\max} ;
 - 2 Tally observations;
 - 3 Initialize model estimates (e.g., with proportions of action/fluent change occurrence);
 - 4 **repeat**
 - 5 **foreach** candidate causal relation **do**
 - 6 └ Compute its information gain by Proposition 3;
 - 7 Select **cr** that maximizes information gain;
 - 8 Calculate λ_+ by Proposition 2;
 - 9 Update model estimates using λ_+ ;
 - 10 **until** information gain is smaller than a threshold;
-

8.4.3 Precise Selection of **cr** When Actions are Hierarchical

In recent computer vision literature, human actions are organized into hierarchical representations, such as stochastic event grammar [70] or the Temporal And-Or Graph [110]. In such representations, actions can be decomposed into sub-actions (where all parts compose the action) and alternative actions. The Temporal And-Or Graph represents these as And-nodes and Or-nodes, respectively.

As instances of a parent and its children often compete, our learning method must have the precision to select the correct node as the cause of the fluent change. Fortunately, as the information gain for each action node in the action hierarchy is tested, these parent/child interactions are automatically taken into account.

We first consider a parent action, A , that is a choice between two children, A_1 or A_2 , as shown with the Or-node on the left of Figure 8.4. Intuitively, if A_1 is a cause, but not A_2 , then A_1 will exhibit the strongest relationship with the fluent change. A will have the second highest, as some of the time it is activated when A_1 occurs and some of the time it is activated when A_2 occurs.

For a cause, the information gain is dominated by the $f_3 \log f_3/h_3$ contribution. Let f_A , f_{A_1} , and f_{A_2} be f_3 from Table 8.2 for A , A_1 , and A_2 , respectively. Further, let β be the Or-probability of selecting A_1 . In this case,

$$f_A = \beta \cdot f_{A_1} + (1 - \beta) \cdot f_{A_2}, \quad (8.34)$$

and therefore,

$$\min(f_{A_1}, f_{A_2}) \leq f_A \leq \max(f_{A_1}, f_{A_2}). \quad (8.35)$$

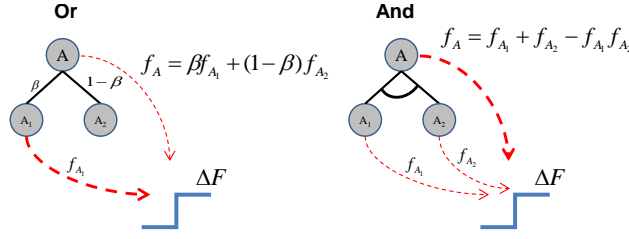


Figure 8.4: Graphical demonstration for when our algorithm encounters an Or-node or an And-node in the action hierarchy. When encountering an Or-node, where action A is identified through one of the child actions A_1 or A_2 with prior probability of A_1 of β , the pursuit process prefers the child node showing the strongest causal relation. For an And-node, where the action A is identified as a composition of A_1 and A_2 , the parent is preferred.

Further, let h_A , h_{A_1} , and h_{A_2} be defined similarly. Since A happens if A_1 or A_2 happen, $h_A > h_{A_1}$. Finally, if $h_3 < f_3$ as is the case on a distribution considering A and ΔF independently, then

$$h_{A_1} < h_A < f_A \leq f_{A_1}, \quad (8.36)$$

and the contribution on the information gain for A_1 will be larger than for A . In the case of an Or-node, the causing child node will be selected over the parent under pursuit by information gain.

Next, let A be a parent that groups its children A_1 and A_2 as in the right side of Figure 8.4. In this case, A happens if both children A_1 and A_2 happen and so $h_A < h_{A_1}$ and

$$f_A = f_{A_1} + f_{A_2} - f_{A_1} f_{A_2}. \quad (8.37)$$

It follows that

$$f_A \geq f_{A_1}, f_{A_2} \quad (8.38)$$

and

$$h_A < h_{A_1} < f_{A_1} \leq f_A. \quad (8.39)$$

Therefore, for an And-node where both children must happen in order for the parent node to happen, our method selects the parent node.

8.5 The Causal And-Or Graph

The And-Or structure that is used to represent actions was first introduced to computer vision for representing compositional concepts defined by grammar [185], and it can be used to collect the perceived causal relations. An example of the Causal And-Or Graph for a fluent value [45] is shown in Figure 8.5. Nodes are hierarchically defined: And-nodes activate if all their children activate, while Or-nodes activate if any of their children activate.

The Causal And-Or Graph provides a detangled view of the causal structure, separating the causes. Or-nodes represent fluent values, whose children are the alternate causes for that fluent value. For example, ΔF as an Or-node could be caused by any of the alternative causes A_1 , A_2 , and A_3 :

$$\Delta F \leftarrow A_0 \vee A_1 \vee A_2. \quad (8.40)$$

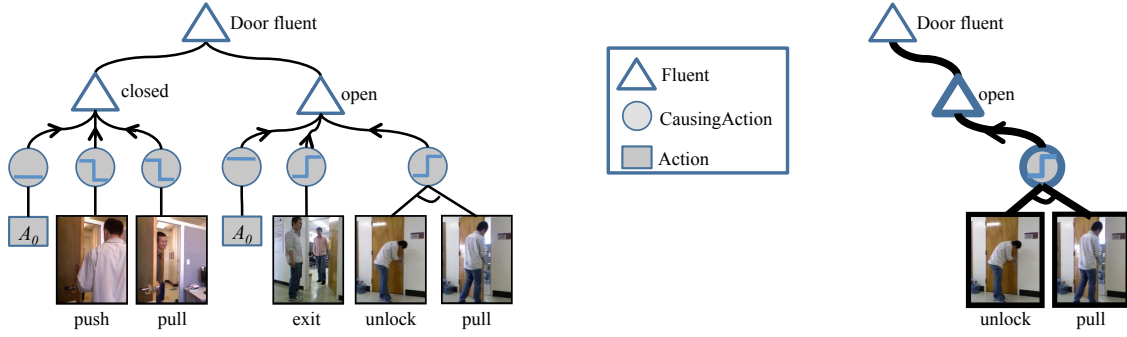


Figure 8.5: The Causal And-Or Graph (left) and a parse graph (right). Each causing action node shows an action from a high level of the hierarchy. Arrows point from these actions (causes) to the fluent (effect). Children of And-nodes are grouped by arcs. A_0 represents non-action, causing a fluent to maintain status. Reprinted/modified from Ref. [45] with permission.

Arrows point from causes to effects. These Or-nodes represent a choice in the causing condition. Here, actions cause fluent values to change. Similarly, non-actions (shown in Figure 8.5 with A_0) maintain a fluent's value.

Action recognition, while beyond the scope of this paper, works by detecting spatio-temporal relationships in the video (e.g., detecting computer use through relative positions of skeleton joints and proximity to the computer [163]). These spatio-temporal relationships are really compositions of fluents (as ambient conditions or as the visual decomposition of actions). In the Causal And-Or Graph, these compositions are represented with And-nodes, e.g.,

$$A_2 := f_1 \wedge f_2 \quad (8.41)$$

where $:=$ represents definition.

The Causal And-Or Graph provides a hierarchical, computationally efficient decomposition that is useful in computer vision for detections. A selection on the Or-nodes provides parse graphs (*pg*) from the grammar and represents simpler causal explanations; an example of which is shown in the left side of Figure 8.5. This parse graph provides the causal explanation for the video clip: the door is open because an agent unlocked and pulled. These untangled networks allow faster inference.

Further, the Or-nodes encode *prior* information on the different causes. (This is different from Bayesian structural equation modeling, which places a prior over the parameters [131].) Humans have an intuitive understanding of causation that they use to answer questions amid missing or hidden information. Without seeing what happened or knowing what the circumstances are in the room, they can answer: Why is the door closed? (Because no one opened it.) Why did the light turn on? (Because someone toggled the switch.) A prior on causality is important for computer vision as it enables guesses on the particular causal relationship (both the cause and effect together) in play when only partial information is available and thus can fill in detections.

Note that the learned Causal And-Or Graph depends on both the pre-specified fluents of interest and the action recognition hierarchy used. For example, here we learned the joint actions of unlock and push open the door. This could more accurately be represented by changing the lock's fluent, coupled with the pushing action. Regardless, the learning method still produces a graph structure that is useful.

8.5.1 The Probability Model on the Causal And-Or Graph

The Causal And-Or Graph is a graphical representation of the joint probability distribution learned in Section 8.4, conditioned on the fluent value. This natural probability distribution over the Causal And-Or Graph provides the prior on causality.

More concretely, probability is defined over the parse graphs, pg , in the Causal And-Or Graph, and is formed by conditioning on the fluent value in the jointly pursued model:

$$p_C(pg) = p(pg|F) \propto \exp(-\mathcal{E}_C(pg)) \quad (8.42)$$

where

$$\mathcal{E}_C(pg) = \mathcal{E}_0(pg) + \sum_{a \in CR(pg)} \lambda_a(w(a)). \quad (8.43)$$

$\mathcal{E}_0(pg)$ is the energy from the model p_0 in Equation 8.6, limited to the actions and fluents relevant to the included causal relations. $CR(pg)$ is the set of all non-empty, causal relations included in the parse graph (Or-nodes). $w(a)$ is the choice of causing action a (the selection of the child from the Or-node). λ_a comes from Equation 2 and represents the switch probability on the Or-nodes for cr_a , providing a measure for how frequently an action causes the fluent status.

This prior on causality allows common knowledge to overcome ambiguous or missing spatio-temporal detections. When this prior distribution over the parse graphs is combined with a likelihood model, MAP inference provides instances of perceptual causality in video.

This probability on the Causal And-Or Graph can be thought of as a scoring mechanism for detection purposes. In particular, detections of fluents and actions contribute to the score, and the prior on causality contributes a favorable amount to the score if the actions and fluents detected are linked.

8.6 C-AoG Inference and Reasoning

In this section, we focus on the inference and reasoning process on the Causal And-Or Graph (C-AOG). This was first done by formulating the C-AOG as a probabilistic model representing actions and their effects on objects over time. With this probabilistic model over state-action sequences, inference algorithms like Viterbi algorithm can be applied over the probabilistic detections from video to finish causal reasoning tasks like filling in hidden and misdetected actions and statuses.

Following daily observations, object fluents are connected to actions as preconditions or triggers (e.g., an *empty* cup gets filled by a *thirsty* person) or as effects (e.g., using the mouse or keyboard turns the monitor *on*). Because of limitations on visibility and detectability, the values of these fluents are often hidden (e.g., the fill-level of a cup). Changes in fluent value may be caused by human action (e.g., a light turns on when a person flips the switch) or by an internal mechanism (e.g., a screensaver activates on a monitor). Non-changes are explained by inaction (e.g., a light that is on stays on until it's turned off) or by maintaining action (e.g., continued computer use keeps the monitor awake). Actions can be detectable (e.g., using a computer) or hard to detect (e.g., making a phone call). Some actions are even defined by their causal effects: a “blowing” action is not detectable, but can be reasoned from the expanding balloon.

Connecting triggering conditions to actions and effects, Figure 8.6 shows an inference possible by long-term reasoning. Seeing a man raise a phone to his head, we can infer he's talking to someone on the phone, perhaps because it rang. The man moved the mouse to wake the monitor, his thirst motivated him to fill the cup and drink, and he threw something away. Without seeing the person flipping a light switch (the switch

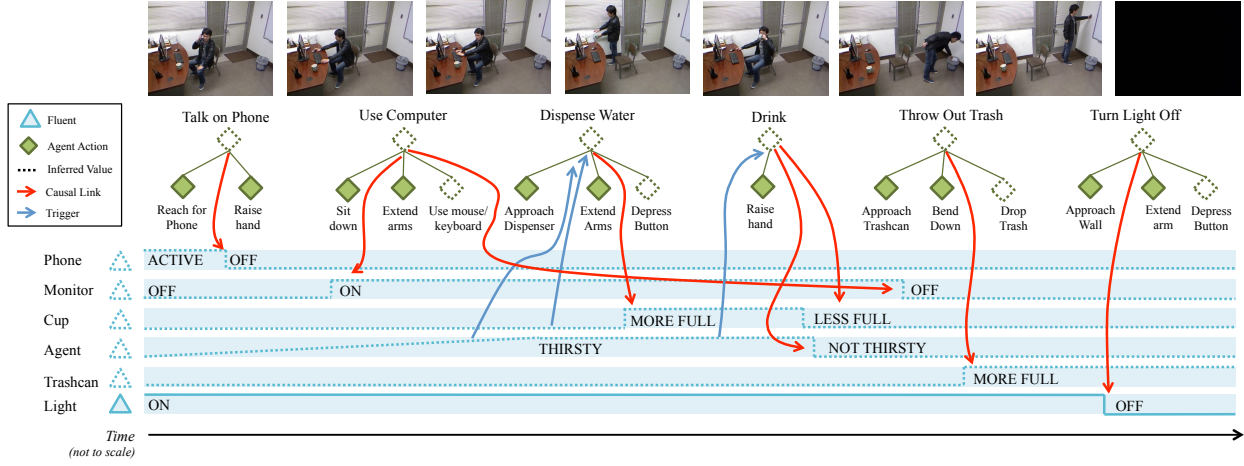


Figure 8.6: Example causal inference. Over time, observed actions are used to infer values of hidden fluents, and values of observed fluents are similarly used to infer hidden actions. Reprinted from Ref. [46] with permission.

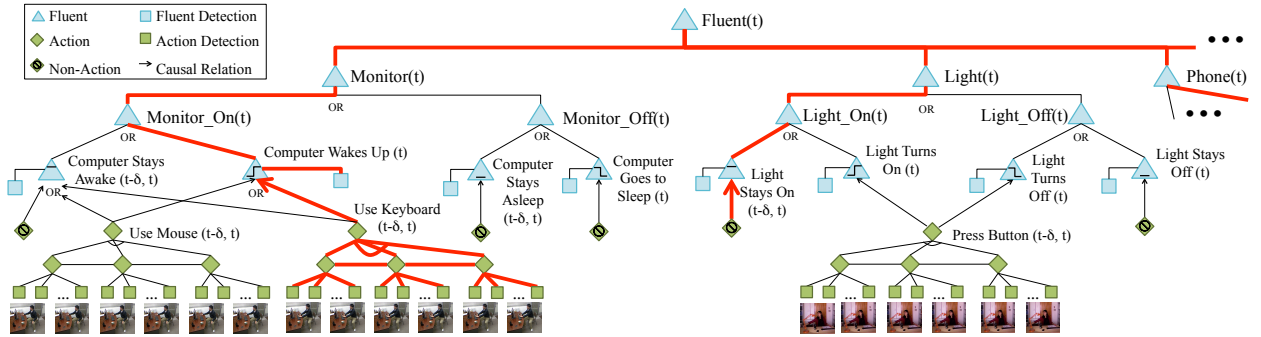


Figure 8.7: A C-AOG for an office at time t . Fluent values are consequences of their children. Arcs connect children of And-nodes. A single selection at the Or-nodes (red, bold lines here) provides a parse graph, explaining the current instance of time. Terminal leaf nodes ground the C-AOG on video, linking input from detected features. Step functions indicate types of fluent changes: step up for turning “on”, step down for “off”. Reprinted from Ref. [46] with permission.

is not detectable), we still reason that he performed that action based on the observed effect. By the end of the clip, we might infer that the monitor is inactive.

Compared with the C-AOG concepts we previously discussed, we need special focus on the continuity of time when applying it into a real-world problem. We can formalize the causal transition of a given video as follows. Given a short video sequence, $V[t - \delta, t]$, the C-AOG represents causal explanations for fluents at time t where causing actions occur within the δ time window (e.g., modeling that using the keyboard causes the monitor to display and the light remains on at t , as shown with thick red in Figure 8.7).

Also as we previously discussed, we use Or-nodes to represent alternate means of causation (e.g., a monitor can be woken by someone using a mouse *or* a keyboard), And-nodes to group sub-actions and conditions (e.g., the sub-actions used to detect “use keyboard”) and Terminal nodes to represent low-level features for detecting actions and fluent changes in video. Horizontal links connect nodes with temporal relationships (e.g., a person nears the computer before using it). Arrows point from causes to effects.

A parse graph (pg) is from the C-AOG is therefore formed by making a selection at each Or-node (*e.g.*, the thicker, red lines in Figure 8.7) and captures the causal reason that the fluent changed value at time t . The best parse graph at t is given by selecting the best children according to the posterior probability

$$P(pg_t|V[t - \delta, t]) \propto P(pg_t; \Theta) \prod_{l \in L(pg_t)} P(l|pg_t) \quad (8.44)$$

where $L(pg)$ is the set of terminal leaf nodes included in pg . Here $P(pg; \Theta)$ defines a prior on causality, indicating a level of prior belief for the current fluent value and why the fluent took that value. Both the prior and the likelihood term could be represented by a energy-based model we discussed in the previous chapters using different energy terms for nodes in AOG. Here we quickly go over the energy based model formulation for AOG.

Let's assume that $P(pg; \Theta) \propto \exp(-\mathcal{E}(pg))$, where $\mathcal{E}(pg)$ is recursively propagated to the top-level nodes in the C-AOG by the following rules:

Or-nodes The energy of an Or-node, O , can be derived as

$$\mathcal{E}(O) = \max_{v \in ch(O)} (\mathcal{E}(v) + \langle \Theta_v, \lambda_v \rangle)$$

where $ch(O)$ represents the children of Or-node O . Θ_v indicates how likely each child is of causing the parent, and λ_v indicates which child is selected. $\langle \Theta_v, \lambda_v \rangle$ returns the prior probability of selecting that particular child. Θ_v can be learned by MLE, giving the proportion of training examples that included child λ_v . The learned Θ_v favors the status quo, *i.e.*, that the fluent maintained status a priori.

And-nodes The energy of an And-node, A , with children $ch(A)$ passes probabilities from all children up to the top node, and is given by

$$\mathcal{E}(A) = \sum_{v \in ch(A)} \mathcal{E}(v|A).$$

Temporal relations Top-level actions are detected as triads of sub-actions, with each allowing a variable number of pose detections. Relations preserve the temporal order of sub-actions. For relation R across nodes $\tilde{v} = v_{i_1}, \dots, v_{i_k}$, $\mathcal{E}(R) = \psi_{\tilde{v}}(\tilde{v})$.

Leaf nodes Terminal leaf nodes anchor the C-AOG to features extracted from video. The fluent energies, $\mathcal{E}(l_F|F)$, and the action energies, $\mathcal{E}(l_A|A)$ are calculated from the detected features, trained separately with machine learning approaches. Treated independently, $\mathcal{E}(l_A|A)$ and $\mathcal{E}(l_F|F)$ sum to provide $\mathcal{E}(l|pg)$. $\mathcal{E}(A)$ and $\mathcal{E}(O)$ recursively compute energies for all included nodes. Decomposing the recursion,

$$\mathcal{E}(pg_t|V[t - \delta, t]) = \sum_{l_F \in L_F(pg)} \mathcal{E}(l_F|F) + \sum_{l_A \in L_A(pg)} \mathcal{E}(l_A|A) + \sum_{\tilde{v} \in R} \psi_{\tilde{v}}(\tilde{v}) + \sum_{v \in O(pg)} \langle \Theta_v, \lambda_v \rangle, \quad (8.45)$$

where $L_F(pg)$, $L_A(pg)$, $R(pg)$, and $O(pg)$ are the sets of included fluent leaves, action leaves, relations, and Or-nodes, respectively.

Detections of actions and fluents are jointly considered for pg where temporal spacing between the two is within a latent time, δ , which can be pre-learned by optimizing the hit rate as latency increases. Latent time between flipping a switch and the light turning on is kept near instantaneous, whereas latent time between pushing an elevator call button and the elevator's arrival affords more leniency.

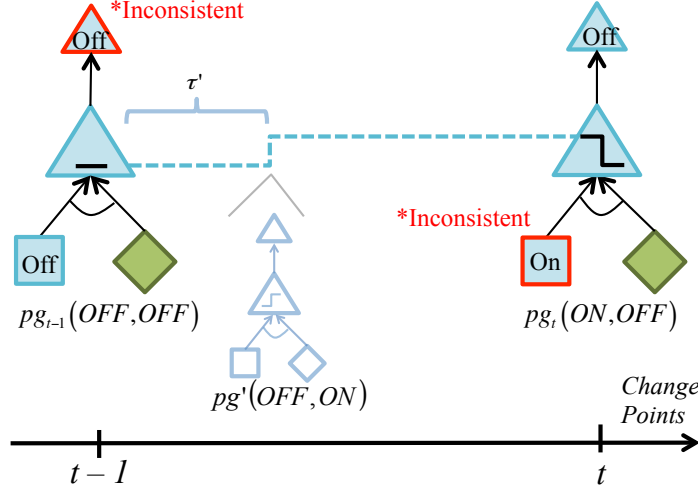


Figure 8.8: Inconsistent state transition. Reprinted from Ref. [46] with permission.

8.6.1 Reasoning over Time

Over time, a fluent takes a sequence of values (F_1, \dots, F_n) and a series of actions (A_1, \dots, A_k) are performed. The C-AOG models causal relationships as the fluent value transitions from F_{t-1} to F_t . In this section, we bind the C-AOGs sequentially to model a sequence of parse graphs, $\mathbf{PG} = (pg_1, \dots, pg_n)$, explaining a longer video. Greedily connecting the pg yields two concerns: (1) Subsequent parse graphs must be consistent, and (2) The process is non-Markovian.

When reasoning over time, we have the constraint that subsequent states should be consistent. Using our notation, this is to say that subsequent pg_{t-1} and pg_t from \mathbf{PG} both contain the fluent value at $t-1$. Combining the parse graphs pg_t and pg_{t-1} shown in Figure 8.8 requires pg' to maintain consistency—the final value of the former must match the incoming value of the latter. For example, multiple detections of flipping a light switch cannot all cause the light to turn on unless the light is turned off between them. The following state transition probability enforces consistency between subsequent parse graphs:

$$P(pg_t | pg_{t-1}) = \begin{cases} 0, & \text{if } pg_{t-1}, pg_t \text{ inconsistent} \\ 1, & \text{otherwise.} \end{cases} \quad (8.46)$$

Next, we also have the necessity to model non-markovian transitions where actions happened long before will affect the current state. For example, fluents of the computer monitor are non-Markovian: rather than following an exponential fall-off, the screensaver activates after a set amount of time (usually 5 minute increments), following a predictable distribution such as shown in Figure 8.9. Further, while a Markov process can insert the hidden trigger “thirst” between two subsequent observations of “drink”, it has difficulty consistently matching human estimates as to where the insertion should go.

To have this elements in our model, we need to model the duration for which a given fluent maintains a particular value so that we can have time modeled to capture these properties and satisfy the constraints. An intuitive way of modeling is through a probability, $P(\tau|F)$, which captures the duration information of fluent values. And if we assume subsequent durations are independent, given the fluent value. $P(\tau|F)$ can be approximated with step functions, discretizing the probability model. The models for $P(\tau|F)$ can be directly coded (*e.g.*, screensaver) where commonsense knowledge is available. If not, we can also use real-world data to learn this probability by MLE and feed in the inference engine.

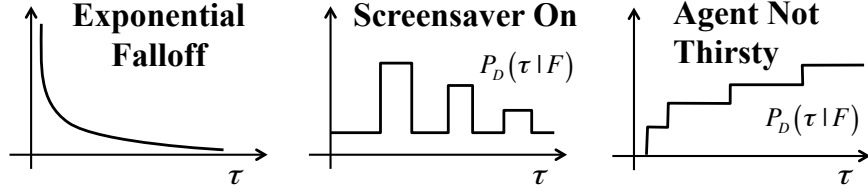


Figure 8.9: Fluent durations. Courtesy of Ref. [46].

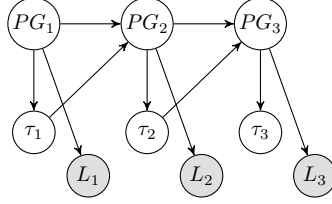


Figure 8.10: Hidden semi-Markov model. Courtesy of Ref. [46].

To incorporate duration as well as causal transition when reasoning over time, we introduce the hidden semi-Markov model for inferencing sequential parse graphs.

A hidden semi-Markov model [100] can accommodate the non-Markovian duration terms while enforcing consistency. The graphical model shown in Figure 8.10 captures our assumed dependencies. In this model, PG_t from the C-AOG is repeated for a duration of τ_t . L_t represents the sequence of observed fluents and actions under PG_t . The following conditional probability distributions govern the state transitions as well as handle a counter for the duration:

$$\begin{aligned} &P(PG_t = pg | PG_{t-1} = pg', \tau_{t-1} = d) \\ &= \begin{cases} 1(pg, pg'), & \text{if } d > 0 \text{ (remain in same state)} \\ P(pg|pg'), & \text{if } d = 0 \text{ (transition per Eq. 8.46).} \end{cases} \end{aligned} \quad (8.47)$$

$$\begin{aligned} &P(\tau_t = d' | PG_t = pg) \\ &= \begin{cases} 1(d', d-1), & \text{if } d > 0 \text{ (decrement)} \\ P(\tau|F), & \text{if } d = 0 \text{ (per Sec. ??).} \end{cases} \end{aligned} \quad (8.48)$$

d and d' count down the duration, and 1 is the Dirac delta function. The optimal sequence explaining the video is given by

$$\mathbf{PG}^*, \tilde{\tau}^* = \underset{\mathbf{PG}, \tilde{\tau}}{\operatorname{argmax}} P(\mathbf{PG}, \tilde{\tau} | V), \quad (8.49)$$

where $\tilde{\tau} = (\tau_1, \dots, \tau_n)$ represents the durations corresponding to elements of \mathbf{PG} . To calculate \mathbf{PG}^* and $\tilde{\tau}^*$, we run a Viterbi algorithm with equations

$$V_t(pg, \tau) \quad (8.50)$$

$$\begin{aligned} &\triangleq \max_{pg', \tau'} \left(P \left(\begin{array}{l} PG_t = pg, \tau_t = \tau, \\ PG_{t-1} = pg', \tau_{t-1} = \tau', \\ L_{1:t} = l_{1:t} \end{array} \right) \right) \end{aligned} \quad (8.51)$$

$$\begin{aligned} &= P(l_{t-\tau+1:t} | pg) \\ &\quad \max_{pg', \tau'} P(pg|pg') P(\tau|F) V_{t-\tau}(pg', \tau'). \end{aligned} \quad (8.52)$$

By defining $V_t(pg) \triangleq \max_{\tau} V_t(pg, \tau)$, we can separate the maximization over τ from the state space:

$$V_t(pg) = \max_{\tau} \left[\frac{P(l_{t-\tau+1:t}|pg) P(\tau|F)}{\max_{pg'} P(pg|pg') V_{t-\tau}(pg')} \right]. \quad (8.53)$$

Derivations are provided in the supplemental materials. By precomputing $P(l_{t-\tau+1:t}|pg)$, the complexity is $O(T \cdot |PG|^2 \cdot |\tau|)$ where $|\tau|$ is the maximum duration considered. This model can be approximated by an HMM with the addition of more nodes, increasing complexity.

To reduce complexity, we index t over detected change points (time points with either a fluent change or action detection). In order to accommodate this simplification, we assume at most one missed fluent change occurred between them. In particular, we consider it possible that a light gets turned off between two detections of turning on, but we ignore the chance that there would be multiple missed detections of on/off. If pg_{t-1} and pg_t are inconsistent, we try to optimally insert a new change point, $t' \in (t-1, t)$ as shown in Figure 8.8, interpreting the inconsistency as missed information. $P(\tau|F)$ informs where to insert this change.

In general, all instances between these change points are best explained by the non-action causal parse graph: the fluent maintains status because no change-inducing action occurred. By jointly optimizing the parse graphs over time, we avoid early decisions, allowing new information to revise previous conflicts.

8.6.2 Evaluating the Inference Model

From a computer vision perspective, evaluating models' performance on causal recognition is hard, especially for most of the current datasets which are defined for short-term action recognition purposes. Therefore, to test the models, we need to have a special focus on the dataset contents aside from the 3D representations for environments as shown in 8.11. For example, we need to design specifically for the object categories, fluent values and also the preconditions (causing actions) like what was done in the dataset proposed by [46], shown in 8.3:

Unlike the previous activity recognition datasets, a causality dataset showcases cause and effect relationships between actions and object responses/fluent changes. With special focus on the causal inference, we might even need to add more complexity to the actions performed. In this case, the activities are placed among human-centric causal contexts; the included fluents reflect a cross-section of those that are detectable (*e.g.*, the light is on or off), confusable (*e.g.*, the refrigerator door fluent is confused with the office door fluent), and inferable (*e.g.*, that the waterstream is on is inferred when the filling cup action is detected); the actions could be ambiguous or occluded that cause confusion if not conducting long-term reasoning.

With the subjectiveness when annotating the occluded world-state transitions, we can not have a one-pass annotation for all the fluent values or actions. Instead, when annotating such a dataset, annotators should provide an estimation of the state over a scale instead of a fixed label (*e.g.*, Did the human dispense water to the cup? Is the cup more full, less full, or the same as in the previous clip? Is the human thirsty?) and iteratively update their belief. In intuitively, the responses by annotators are various, producing a higher number of distinct annotations for ambiguous (occluded) scenes than for "detectable" ones, shown in Figure 8.13. When asked for the monitor's status, humans produced the probabilities shown in the heat maps at the bottom of Figure 8.13. The computer screen is not visible, and humans (generally and specifically) exhibited large variability in examining hidden values. While they all agreed that the actor in this case was using the computer, they lacked a consensus as to whether the screen was on or off or transitioning between the two. Each distinct response provides a different interpretation for the events in the scene (such as the two ways to interpret the Necker cube). Therefore, instead of using a single label for each frame, we need

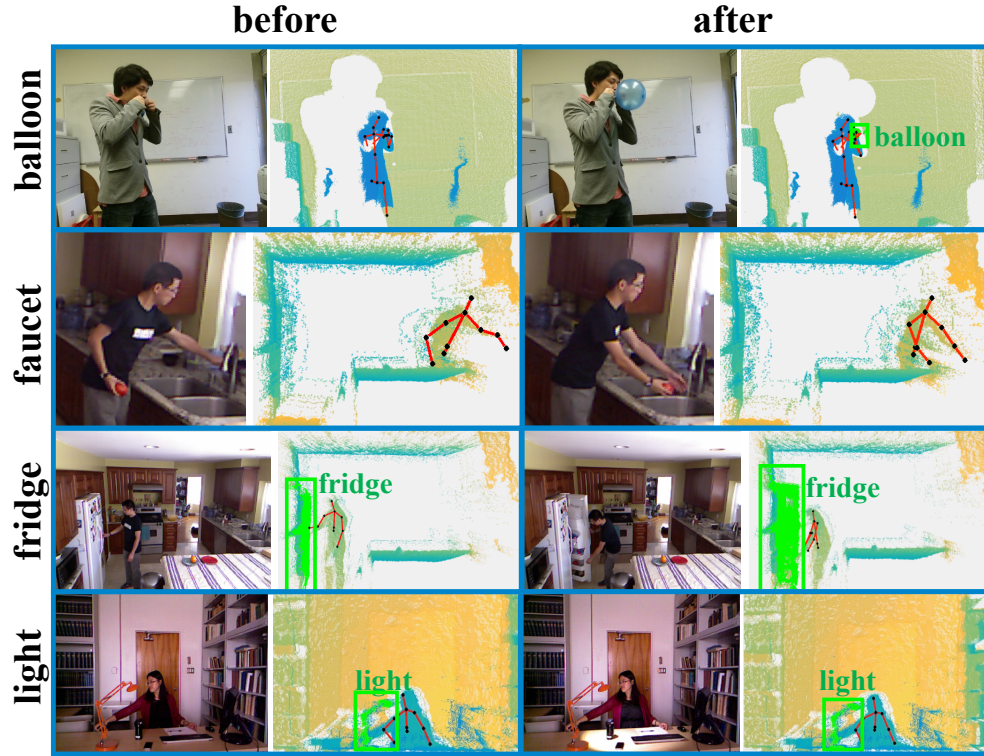


Figure 8.11: Human poses and depth images (before and after a fluent change) for actions as captured by the Kinect, together with sample frames. Reprinted from Ref. [46] with permission.

to accept possible ground truth (*i.e.*, a valid interpretation of the scene) and preserve all valid interpretations from the annotators.

In this way, we can evaluate a causal inference model with the data and annotations discussed previously. Here, we show an example on the bottom-up fluent and action detections (results of the inference) in Figure 8.12. As we can see, the results are improved (and clarified) by applying the sequential C-AOG developed in this paper, having more clear boundaries on the transition of the world states and actions, even with action detectors (second and third plots) that use only pose to detect open/close actions, without distinguishing objects.

Figure 8.13 shows another result from detectors and the sequential C-AOG for light and screen fluents. The fluent detectors erroneously detect multiple light and monitor changes as the light turns on (once) and the camera adjusts; the sequential C-AOG mostly corrects these.

8.7 Discussion

As mentioned in Section 8.2.2, the Causal And-Or graph provides a qualitative view of the causal structure between events and actions. The Or-nodes speak to disjunctive logic operator in a causal structural model, whereas the And-nodes specify the conjunctive logic operator. The door opening example in Section ?? illustrates these properties. That is, the Or-nodes in in Figure 8.5 represent possible fluent values and describe the alternative causes for the fluent value, which could be built out of atomic propositions and disjunctive logical operators in causal structural equations. Similarly, the And-nodes in Figure 8.5, specify the compo-

Table 8.3: Dataset Included Action/Fluent Relationships

Object	Fluent Values	Causing Actions	nScenes	nClips	nFrames
door	open/closed	open door, close door	4	50	10611
light	on/off	turn light on/off	4	34	16631
screen	on/off	use computer	4	179	56632
phone	active/off	use phone	5	68	30847
cup	more/less/same	fill cup, drink	3	48	16564
thirst	thirsty/not	drink	3	48	16564
waterstream	on/off	fill cup	3	40	14061
trash	more/less/same	throw trash out	4	11	2586
microwave	open/closed, running/not	open door, close door, turn on	1	3	4245
balloon	full/empty	blow up balloon	1	3	664
fridge	open/closed	open door, close door	1	2	2751
blackboard	written on/clear	write on board, erase	1	2	5205
faucet	on/off	turn faucet on/off	1	2	3013

sition of actions that leads to fluent changes, could be represented by propositions and conjunctive operators in the structural equations.

As mentioned in Section 8.2.2, the Causal And-Or graph provides a qualitative view of the causal structure between events and actions. The Or-nodes speak to disjunctive logic operator in a causal structural model, whereas the And-nodes specify the conjunctive logic operator. The door opening example in Section ?? illustrates these properties. That is, the Or-nodes in in Figure 8.5 represent possible fluent values and describe the alternative causes for the fluent value, which could be built out of atomic propositions and disjunctive logical operators in causal structural equations. Similarly, the And-nodes in Figure 8.5, specify the composition of actions that leads to fluent changes, could be represented by propositions and conjunctive operators in the structural equations.

The Causal And-Or graph in Section ?? tells us qualitative information about which variables (action nodes and fluent nodes) depend upon which others on a symbolic level, but it does not gives us what is the quantitative functional form of the dependence. In the door opening example, it tells us how actions (*push*, *pull*, *unlock*, etc.) may change the fluent of the door (*Open* or *Closed*). Every time an action is detected in the video frames, the fluent of the door will be updated according to the Causal And-Or graph. However, it fails to tell to what extent an action is able to change the fluent, for example, how much force is adequate to *open* the door. Considering a man cracking a walnut, as shown in Figure 8.14, with different force exerted the walnut may be cracked, smashed, or even remains uncracked if the exerted force is too little to crack the walnut. The question here is that how much force, pressure, or other related physical quantities is adequate to enable a fluent change from one to another (otherwise remains the same fluent), which drive us move to a quantitative causal structural equation for future.

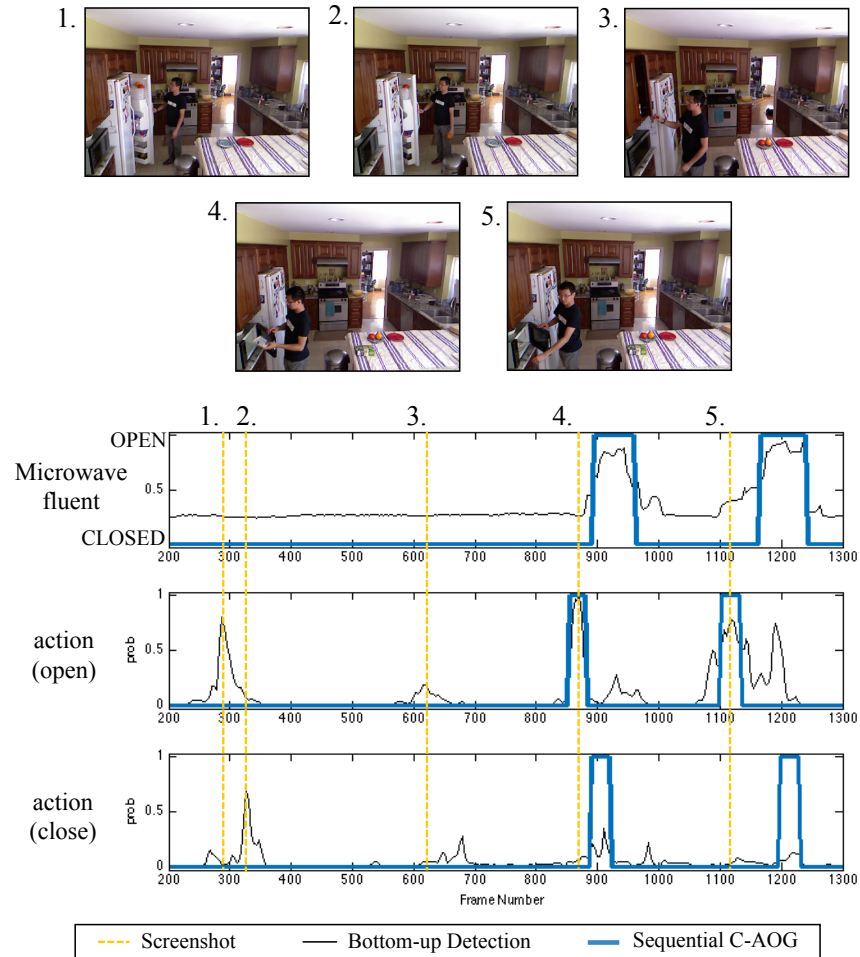


Figure 8.12: Microwave. Results from fluent and action detectors, superimposed with causal reasoning results. Step functions mark fluent changes—up for turning on, down for turning off. Reprinted from Ref. [46] with permission.

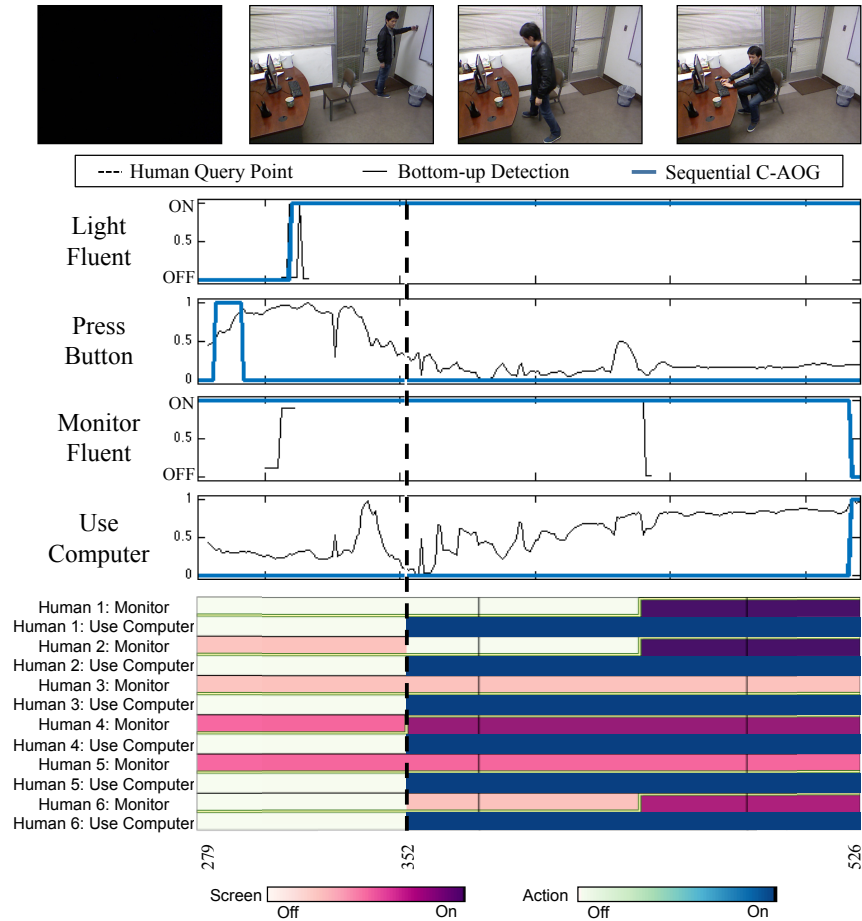


Figure 8.13: Screen and light fluents with human answers. The dashed line separates the two query points for humans. Human responses varied widely (*e.g.*, Human 1 was certain that the monitor changed from off to on at the second query point, human 3 thought that the screen remained off, and human 4 thought the screen was on). Reprinted from Ref. [46] with permission.

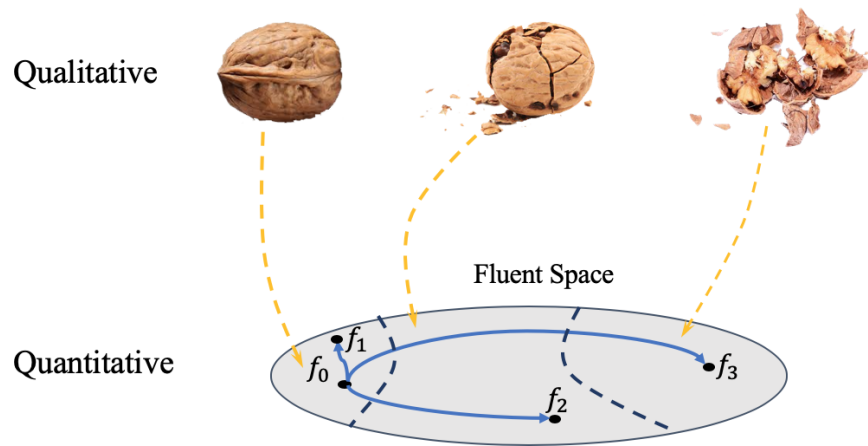


Figure 8.14: The fluent space is partitioned into three subsets which represent uncracked, cracked, and smashed respectively (from left to right). Each element in the same subset could be quantitatively different, but they are qualitatively the same. For example, there are infinite types of cracked walnut (quantitatively different), but they are all cracked (qualitatively the same). The blue line represents a fluent change. For example, the line from f_0 to f_2 indicates a walnut from uncracked to cracked.

9

Joint Parsing

In this chapter, we present how to seamlessly integrate the aforementioned AoG graph, e.g., Spatial, Temporal, and Causal And-Or Graph (STC-AoG), to jointly parse the given scene. Joint parsing is essential for robust vision, going beyond visual recognition to scene understanding under numerous real-life scenarios since the reasoning captures information from multiple sources in long-range spatial, temporal, and causal contexts in the joint inference process. We start by discussing the motivations and benefits of joint parsing and presenting the joint parsing of complex 3D scene understanding. Then we go from view centered parsing to scene centered parsing to show comprehensive examples on space and time inference for various computer vision tasks such as the cross-view person recognition and cross-view person tracking. We also show some applications in the intersection of natural language and computer vision like Visual Question Answering(VQA).

9.1 Introduction

The classic definition of computer vision proposed by the pioneer David Marr is to look at “what” is “where.” Here, “what” refers to object recognition (object vision), and “where” denotes three-dimensional (3D) reconstruction and object localization (spatial vision). Over the past several years, progress has been made in object detection and localization with the rapid advancement of deep neural networks (DNNs), fueled by hardware accelerations and the availability of massive sets of labeled data to learn feedforward features. However, we are still far from solving computer vision or real machine intelligence; the inference and reasoning abilities of current computer vision systems are narrow and highly specialized, require large sets of labeled training data designed for special tasks.

On the other hand, psychological studies have shown that human visual experience is much richer than “what” and “where.” As early as infancy, humans quickly and efficiently perceive causal relationships (e.g., perceiving that object A launches object B), agents and intentions (e.g., understanding that one entity is chasing another), and the consequences of physical forces (e.g., predicting that a precarious stack of rocks is about to fall in a particular direction). Such physical and social concepts can be perceived from both media as rich as videos and much sparser visual inputs. To enable an artificial agent with similar capabilities, we call for joint reasoning algorithms on a joint representation that integrates (i) the “visible” traditional recognition and categorization of objects, scenes, actions, events, and so forth; and (ii) the “dark” higher level concepts of fluent, causality, physics, functionality, affordance, intentions/goals, utility, and so forth.

Fig. 9.1 shows an example of the human-like understanding of a scene that requires a joint understanding of various tasks and sub-problems. However, current DNN methods require a huge amount of data for a single task while neglecting the intertwined relations between the sub-problems. They also fail in numerous

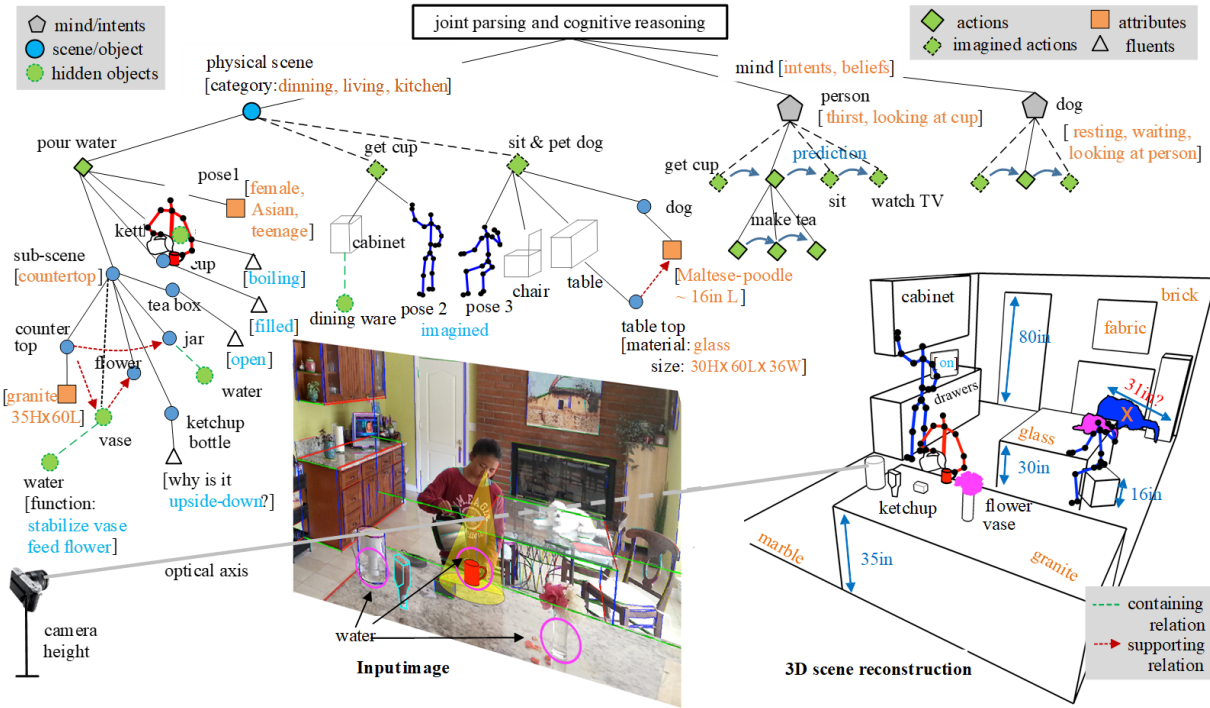


Figure 9.1: An example of the human-like understanding of a scene. From a single image, a computer vision system should be able to jointly (i) reconstruct the 3D scene, (ii) estimate camera parameters, materials, and illumination, (iii) parse the scene hierarchically with attributes, fluents, and relationships, (iv) reason about the intentions and beliefs of agents (*e.g.*, the human and dog in this example), (v) predict their actions in time, and (vi) recover invisible elements such as water, latent object states, *etc.* We, as humans, can effortlessly (i) predict that water is about to come out of the kettle, (ii) reason that the intent behind putting the ketchup bottle upside down is to utilize gravity for easy use, and (iii) see that there is a glass table, hard-to-detect with existing computer vision methods, under the dog; without seeing the glass table, parsing results would violate the laws of physics, as the dog would appear to be floating in midair. These perceptions can only be achieved by reasoning about unobservable factors in the scene not represented by pixels, requiring us to build an AI system with human-like core knowledge and common sense, which are largely missing from current computer vision research. Reprinted from Ref. [187] with permission of Elsevier, © 2020.

conditions such as uncanonical views, unusual poses and scales, occlusion and illumination, etc.

Joint parsing/inference serves as a tool to solve the aforementioned problems. Specifically, we represent the entities, relations, scenes, and events with Spatial-Temporal-Causal parse graph (STC-PG), which is an instantiation of the previously introduced STC-AOG. In a STC-PG, each node can be inferred from many sources or visual pathways, which brings in inference from long-range spatial, temporal, and causal context. As shown in Fig. 9.2, according to the obtained spatial PG, temporal& causal PG, and grammar PG, we (i) perform spatial parse to represent the person and objects inside the given video, (ii) conduct the temporal and causal parse to understand the behavior of the person with uncertainty, and (iii) use the text parser to generate text descriptions about the ongoing joint parsing process simultaneously. Moreover, we also employ the natural language query techniques to further explain the joint parsing process by answering the user query, as illustrated in Fig. 9.3. The core of joint inference is to leverage the coupled tasks and data from multiple sources to simulate the human reasoning process, which shadows the light to high-level cognitive

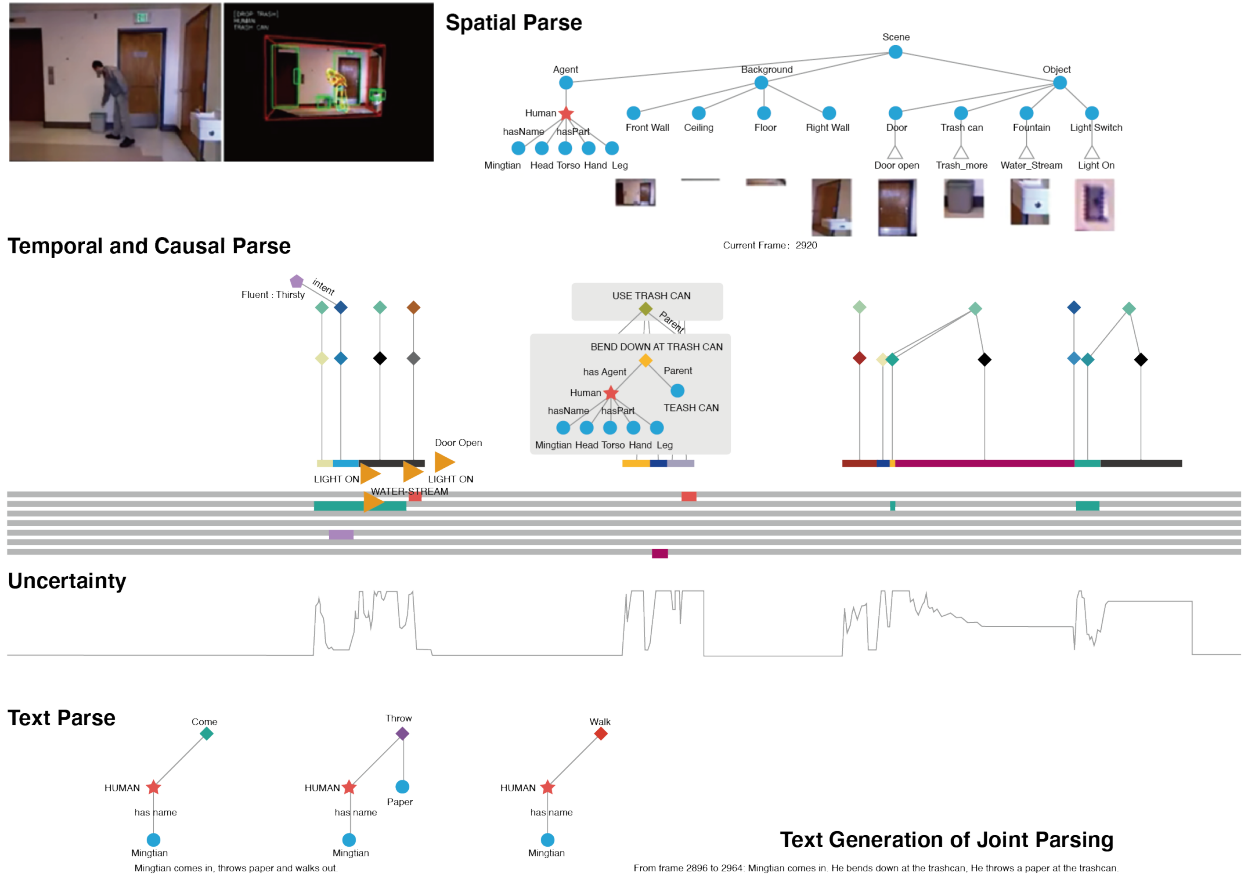


Figure 9.2: An example of spatial, temporal and causal inference based on joint parsing.

reasoning with multiple tasks and limited data.

One drawback of modelling more and more complex system is the difficulties in optimization for the "best" configuration or parse graph. Since the solution space is often complex and non-differentiable, sampling method is the common approach to solve this problem. In the following sections, we present representative examples of using joint parsing to solve three challenging vision and language tasks.

9.2 Holistic⁺⁺ 3D Scene understanding

In this section we introduce how to jointly tackle two tasks: (i) holistic scene parsing and reconstruction—3D estimations of object bounding boxes, camera pose, and room layout, and (ii) 3D human pose estimation, which is a challenging 3D scene understanding problem from a single RGB image. The intuition behind is to leverage the coupled nature of the two tasks by exploiting two critical and essential connections between these two tasks: (i) human-object interaction (HOI) to model the fine-grained relations between human agents and objects in the scene, and (ii) physical commonsense to model the physical plausibility. The optimal configuration of the 3D scene, represented by a parse graph, is inferred using Markov Chain Monte Carlo (MCMC), which efficiently traverses through the non-differentiable joint solution space.

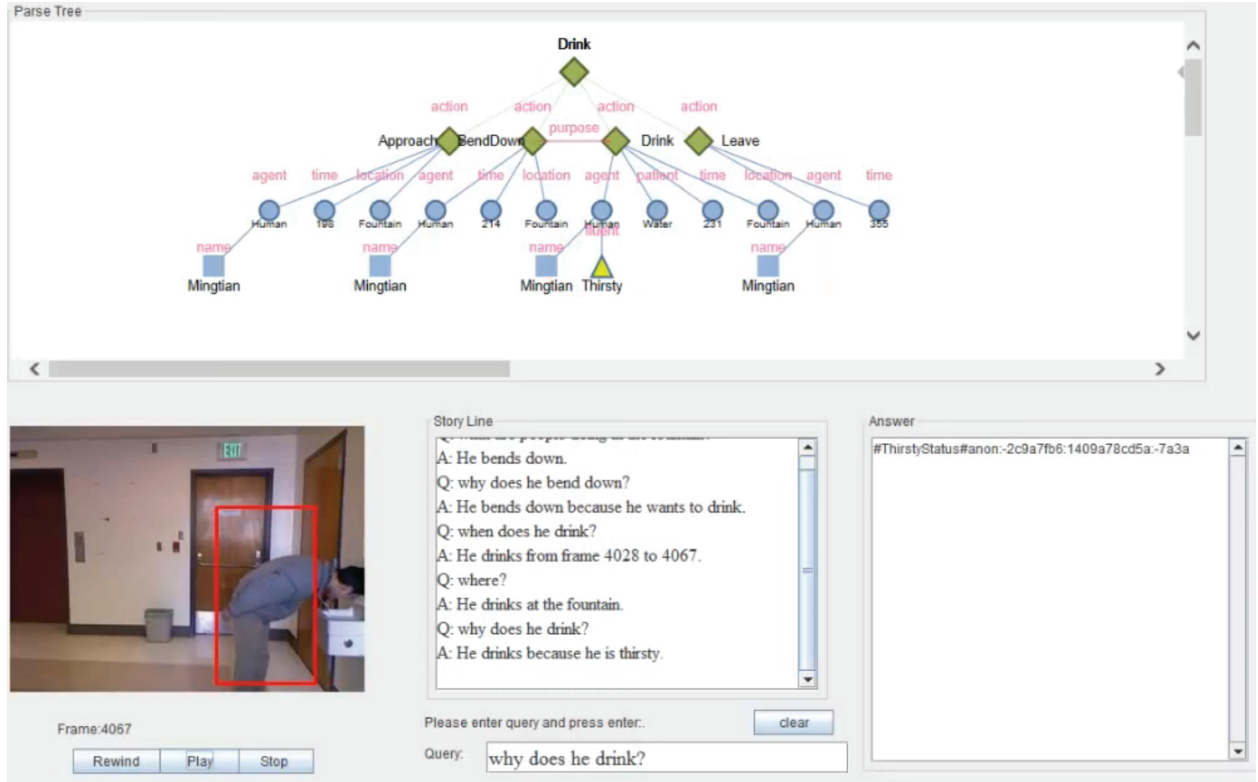


Figure 9.3: An example of natural lanague query (storeline) based on joint parsing.

9.2.1 Representation

We represent the configuration of an indoor scene by a parse graph $pg = (pt, E)$ as shown in Figure 9.4. It combines a parse tree pt and contextual relations E among the leaf nodes. Here $pt = (V, R)$ and we denote $V = V_r \cup V_m \cup V_t$ the vertex set and R the decomposing rules. The tree has three levels. The first level is the root node V_r that represents the scene, and the second level V_m has three nodes (objects, human, and room layout). The third level (terminal nodes V_t) contains child nodes of the second level nodes, representing the detected instances of the parent node in this scene. $E \subset V_t \times V_t$ is the set of contextual relations among the terminal nodes, represented by horizontal links.

Terminal Nodes V_t in pg can be further decomposed as $V_t = V_{\text{layout}} \cup V_{\text{object}} \cup V_{\text{human}}$:

- The room layout $v \in V_{\text{layout}}$ is represented by a 3D bounding box $X^L \in \mathbb{R}^{3 \times 8}$ in the world coordinate. The 3D bounding box is parametrized by the node's attributes, including its 3D size $S^L \in \mathbb{R}^3$, center $C^L \in \mathbb{R}^3$, and orientation $Rot(\theta^L) \in \mathbb{R}^{3 \times 3}$. See the supplementary for the parametrization of the 3D bounding box.
- Each 3D object $v \in V_{\text{object}}$ is represented by a 3D bounding box with its semantic label. We keep the same parameterization of the 3D bounding box as the one for room layout.
- Each human $v \in V_{\text{human}}$ is represented by 17 3D joints $X^H \in \mathbb{R}^{3 \times 17}$ with their action labels. These 3D joints are parametrized by the pose scale $S^H \in \mathbb{R}$, pose center (*i.e.*, hip) $C^H \in \mathbb{R}^3$, local joint position $Rel^H \in \mathbb{R}^{3 \times 17}$, and pose orientation $Rot(\theta^H) \in \mathbb{R}^{3 \times 3}$. Each person is also attributed by a concurrent action label a , which is a multi-hot vector representing the current actions of this person: one can “sit” and “drink”, or “walk” and “make phone call” at the same time.

Contextual Relations E contains three types of relations in the scene $E = \{E_s, E_c, E_{hoi}\}$. Specifically:

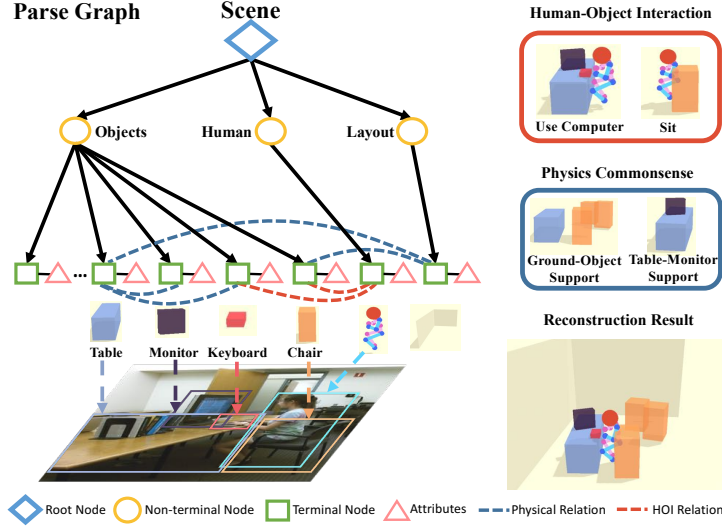


Figure 9.4: Jointly recovering a parse graph that represents the scene, including human poses, objects, camera pose, and room layout, all in 3D. Reasoning human-object interaction (HOI) helps reconstruct the detailed spatial relations between humans and objects. Physical commonsense (*e.g.*, physical property, plausibility, and stability) further refines relations and improves predictions. Reprinted (or modified), with permission, from (Chen, *et al.*, 2019) [19].

- E_s and E_c denote support relation and physical collision, respectively. These two relations penalize the physical violations among objects, between objects and layout, and between human and layout, resulting in a physically plausible and stable prediction.
- E_{hoi} models HOI and gives us more constraints to reconstruct 3D from 2D. For instance, if a person is detected as sitting on the chair, we can constrain the relative 3D positions between this person and chair using a pre-learned spatial relation of “sitting”.

9.2.2 Probabilistic Formulation

The parse graph pg is a comprehensive interpretation of the observed image I . The goal of the holistic⁺⁺ scene understanding is to infer the optimal parse graph pg^* given I by a maximum a posteriori (MAP) estimation:

$$\begin{aligned}
 pg^* &= \arg \max_{pg} p(pg|I) = \arg \max_{pg} p(pg) \cdot p(I|pg) \\
 &= \arg \max_{pg} \frac{1}{Z} \exp\{-\mathcal{E}_{phy}(pg) - \mathcal{E}_{hoi}(pg) - \mathcal{E}(I|pg)\},
 \end{aligned} \tag{9.1}$$

We model the joint distribution by a Gibbs distribution, where the prior probability of parse graph can be decomposed into physical prior and HOI prior.

Physical Prior $\mathcal{E}_{phy}(pg)$ represents physical commonsense in a 3D scene. We consider two types of physical relations among the terminal nodes: support relation E_s and collision relation E_c . Therefore, the energy of physical prior is defined as $\mathcal{E}_{phy}(pg) = \lambda_s \mathcal{E}_s(pg) + \lambda_c \mathcal{E}_c(pg)$, where λ_s and λ_c are balancing factors. Specifically:

- *Support Relation* $\mathcal{E}_s(pg)$ defines the energy between the supported object/human and the supporting ob-

ject/layout:

$$\mathcal{E}_s(pg) = \sum_{(v_i, v_j) \in E_s} \mathcal{E}_o(v_i, v_j) + \mathcal{E}_{height}(v_i, v_j), \quad (9.2)$$

where $\mathcal{E}_o(v_i, v_j) = 1 - \text{area}(v_i \cap v_j) / \text{area}(v_i)$ is the overlapping ratio in the xy-plane, and $\mathcal{E}_{height}(v_i, v_j)$ is the absolute height difference between the lower surface of the supported object v_i and the upper surface of the supporting object v_j . We define $\mathcal{E}_o(v_i, v_j) = \mathcal{E}_{height}(v_i, v_j) = 0$ if the supporting object is floor or wall.

- *Physical Collision* $\mathcal{E}_c(pg)$ denotes the physical violations. We penalize the intersection among human, objects, and room layout except the objects in HOI and objects that could be a container. The potential function is defined as:

$$\mathcal{E}_c(pg) = \sum_{v \in (V_{object} \cup V_{human})} \mathcal{C}(v, V_{layout}) + \sum_{\substack{v_i \in V_{object} \\ v_j \in V_{human} \\ (v_i, v_j) \notin E_{hoi}}} \mathcal{C}(v_i, v_j) + \sum_{\substack{v_i, v_j \in V_{object} \\ v_i, v_j \notin V_{container}}} \mathcal{C}(v_i, v_j), \quad (9.3)$$

where $\mathcal{C}()$ denotes the volume of intersection between entities. $V_{container}$ denotes the objects that can be a container, such as a cabinet, desk, and drawer.

Human-object Interaction Prior $\mathcal{E}_{hoi}(pg)$ is defined on the interactions between human and objects:

$$\mathcal{E}_{hoi}(pg) = \sum_{(v_i, v_j) \in E_{hoi}} \mathcal{K}(v_i, v_j, a_{v_j}), \quad (9.4)$$

where $v_i \in V_{object}$, $v_j \in V_{human}$, and \mathcal{K} is an HOI function that evaluates the interaction between an object and a human given the action label a :

$$\mathcal{K}(v_i, v_j, a_{v_j}) = -\log l(v_i, v_j | a_{v_j}), \quad (9.5)$$

where $l(v_i, v_j | a_{v_j})$ is the likelihood of the relative position between node v_i and v_j given an action label a , and λ_a the balancing factor. We formulate the action detection as a *multi-label classification*. The likelihood $l(\cdot)$ models the distance between key joints and the center of the object; *e.g.*, for “sitting”, it models the relative spatial relation between the hip and the center of a chair. The likelihood can be learned from 3D HOI datasets with a multivariate Gaussian distribution $(\Delta x, \Delta y, \Delta z) \sim \mathcal{N}_3(\mu, \Sigma)$, where Δx , Δy , and Δz are the relative distances in the directions of three axes.

Likelihood $\mathcal{E}(I|pg)$ characterizes the consistency between the observed 2D image and the inferred 3D result. The projected 2D object bounding boxes and human poses can be computed by projecting the inferred 3D objects and human poses onto a 2D image plane. The likelihood is obtained by comparing the directly detected 2D bounding boxes and human poses with projected ones from inferred 3D results:

$$\mathcal{E}(I|pg) = \sum_{v \in V_{object}} \lambda_o \cdot \mathcal{D}_o(B(v), B'(v)) + \sum_{v \in V_{human}} \lambda_h \cdot \mathcal{D}_h(Po(v), Po'(v)), \quad (9.6)$$

where $B()$ and $B'()$ are the bounding boxes of detected and projected 2D objects, $Po()$ and $Po'()$ the poses of detected and projected 2D humans, $\mathcal{D}_o(\cdot)$ the intersection-over-union (IoU) between the detected 2D bounding box and the convex hull of the projected 3D bounding box, and $\mathcal{D}_h(\cdot)$ the average pixel-wise Euclidean distance between two 2D poses.

9.2.3 Joint Inference

Given a single RGB image as the input, the goal of joint inference is to find the optimal parse graph that maximizes the posterior probability $p(pg|I)$. The joint parsing is a four-step process: (i) 3D scene initialization of the camera pose, room layout, and 3D object bounding boxes, (ii) 3D human pose initialization that

Algorithm 16: Joint Inference Algorithm

```
1 Given: Image  $I$ , initialized parse graph  $pg_{init}$ 
2 Phase 1: for Different temperatures do
3   | Inference with physical commonsense  $\mathcal{E}_{phy}$  but without HOI  $\mathcal{E}_{hoi}$ : randomly select from room
   | layout, objects, and human poses to optimize  $pg$ 
4 end
5 Phase 2: Match each agent with their interacting objects
6 Phase 3: for Different temperatures do
7   | Inference with total energy  $\mathcal{E}$ , including physical commonsense and HOI: randomly select from
   | layout, objects, and human poses to optimize  $pg$ 
8 end
9 Phase 4: Top-down sampling by HOIs
```

estimates rough 3D human poses in a 3D scene, (iii) concurrent action detection, and (iv) joint inference to optimize the objects, layout, and human poses in 3D scenes by maximizing the posterior probability.

9.2.3.1 Initialization

Following [69], we initialize the 3D objects, room layout, and camera pose cooperatively, where the room layout and objects are parametrized by 3D bounding boxes.

For pose initialization, we take 2D poses as the input and predict 3D poses in a local 3D coordinate following [147], where the 2D poses are detected and estimated by [13]. The local 3D coordinate is centered at the human hip joint, and the z-axis is aligned with the up direction of the world coordinate. To transform this local 3D pose into the world coordinate, we find the 3D world coordinate $\mathbf{v}_{3D} \in \mathbb{R}^3$ of one visible 2D joint $\mathbf{v}_{2D} \in \mathbb{R}^2$ (e.g., head) by solving a linear equation with the camera intrinsic parameter K and estimated camera pose R per the pinhole camera projection model.

9.2.3.2 Concurrent Action Detection

We formulate the concurrent action detection as a multi-label classification problem to ease the ambiguity in describing the action. We define a portion of the action labels (e.g., “eating”, “making phone call”) as the HOI labels, and the remaining action labels (e.g., “standing”, “bending”) as general human poses without HOI. The mixture of HOI actions and non-HOI actions covers most of the daily human actions in indoor scenes. We manually map each of the HOI action labels to a 3D HOI relation learned from the SHADE dataset, and use the HOI actions as cues to improve the accuracy of 3D reconstruction by integrating it as prior knowledge in our model. The concurrent action detector takes 2D skeletons as the input and predicts multiple action labels with a three-layer multi-layer perceptron (MLP).

9.2.3.3 Inference

Given an initialized parse graph, we use MCMC with simulated annealing to jointly optimize the room layout, 3D objects, and 3D human poses through the non-differentiable energy space; see algorithm 16 as a summary. To improve the efficiency of the optimization process, we adopt a scheduling strategy that divides the optimization process into following four phases with different focuses: (i) Optimize objects, room layout, and human poses without HOIs. (ii) Assign HOI labels to each human in the scene, and search the interacting objects of each human. (iii) Optimize objects, room layout, and human poses jointly with HOIs. (iv) Generate possible miss-detected objects by top-down sampling.

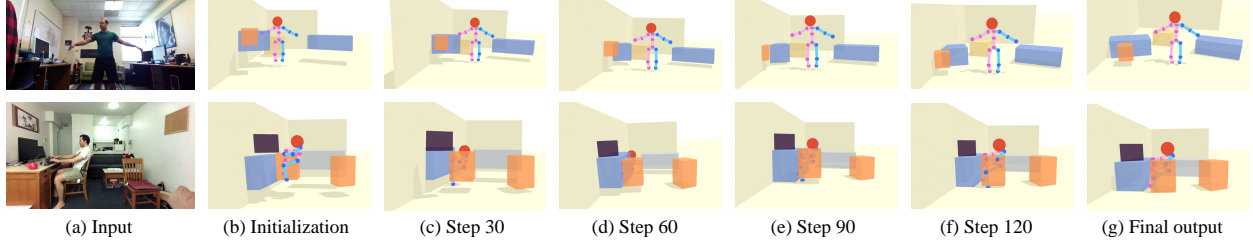


Figure 9.5: The optimization process of the scene configuration by simulated annealing MCMC. Each step is the number of accepted proposal. Reprinted (or modified), with permission, from (Chen, *et al.*, 2019) [19].

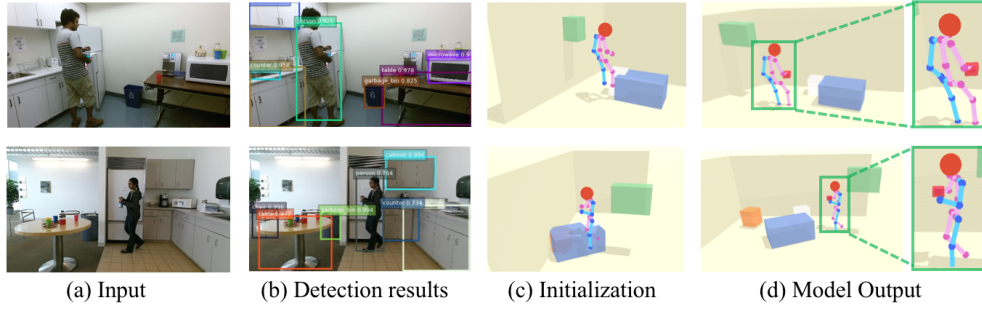


Figure 9.6: Illustration of the top-down sampling process. The object detection module misses the detection of the bottle held by the person, but our model can still recover the bottle by reasoning HOI. Reprinted (or modified), with permission, from (Chen, *et al.*, 2019) [19].

Dynamics. In Phase (i) and (iii), we use distinct MCMC processes. To traverse non-differentiable energy spaces, we design Markov chain dynamics q_1^o, q_2^o, q_3^o for objects, q_1^l, q_2^l for room layout, and q_1^h, q_2^h, q_3^h for human poses.

- **Object Dynamics:** Dynamics q_1^o adjusts the position of an object, which translates the object center in one of the three Cartesian coordinate axes or along the depth direction. The depth direction starts from the camera position and points to the object center. Translation along depth is effective with proper camera pose initialization. Dynamics q_2^o proposes rotation of the object with a specified angle. Dynamics q_3^o changes the scale of the object by expanding or shrinking corner positions of the cuboid with respect to object center. Each dynamic can diffuse in two directions: each object can translate in the direction of ‘+x’ and ‘-x,’ or rotate in the direction of clockwise and counterclockwise. To better traverse in energy space, the dynamics may propose to move along the gradient descent direction with a probability of 0.95 or the gradient ascent direction with a probability of 0.05.

- **Human Dynamics:** Dynamics q_1^h proposes to translate 3D human joints along x, y, z, or depth direction. Dynamics q_2^h is designed to rotate the human pose with a certain angle. Dynamics q_3^h adjusts the scale of human poses by a scaling factor on the 3D joints with respect to the pose center.

- **Layout Dynamics:** Dynamics q_1^l translates the wall towards or away from the layout center. Dynamics q_2^l adjusts the floor height, equivalent to change the camera height.

In each sampling iteration, the algorithm proposes a new pg' from current pg under the proposal probability of $q(pg \rightarrow pg'|I)$ by applying one of the above dynamics. The generated proposal is accepted with respect to an acceptance rate $\alpha(\cdot)$ as in the Metropolis-Hastings algorithm [63]:

$$\alpha(pg \rightarrow pg') = \min(1, \frac{q(pg' \rightarrow pg) \cdot p(pg'|I)}{q(pg \rightarrow pg') \cdot p(pg|I)}), \quad (9.7)$$

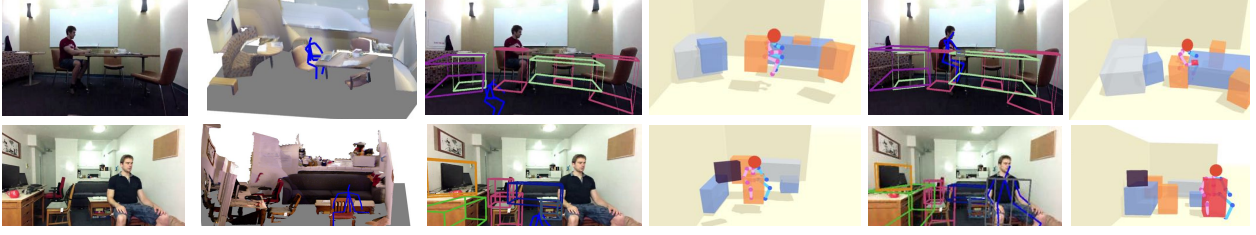


Figure 9.7: Qualitative results of the proposed method. The proposed model improves the initialization with accurate spatial relations and physical plausibility and demonstrates an outstanding generalization. Reprinted (or modified), with permission, from (Chen, *et al.*, 2019) [19].

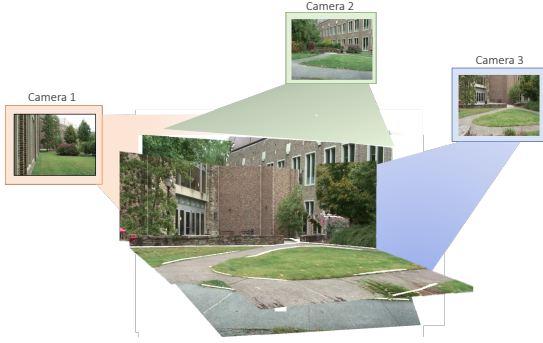


Figure 9.8: An example of the multi-camera scenario in a visual scene captured by three cameras with overlapping fields-of-views. Reprinted (or modified), with permission, from (Xu, *et al.*, 2017) [172].

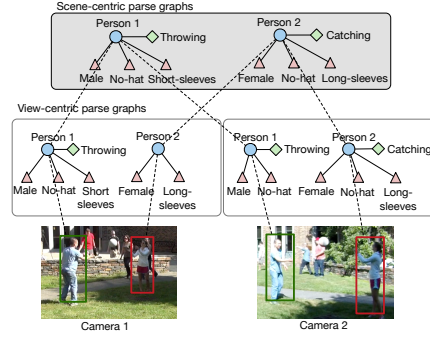


Figure 9.9: An example of the spatio-temporal semantic parse graph hierarchy in a visual scene captured by two cameras. Reprinted (or modified), with permission, from (Qi, *et al.*, 2018) [120].

A simulated annealing scheme is adopted to obtain pg with high probability.

Top-down sampling. By top-down sampling objects from HOIs, the proposed method can recover the interacting 3D objects that are too small or novel to be detected by the state-of-the-art 2D object detector. In Phase (iv), we propose to sample an interacting object from the person if the confidence of HOI is higher than a threshold. Specifically, we minimize the HOI energy in Equation 9.4 to determine the category and location of the object; see examples in Figure 9.6.

Some qualitative results are shown in Figure 9.7.

9.3 From View Centered Parsing to Scene Centered Parsing

In this example, we present a Spatio-temporal Attributed Grammar model for cross-view people tracking in crowded scenes under wild conditions. Given videos from multiple cameras with overlapping field of view (FOV), our goal is to parse all human trajectories in the scene into a scene-centered representation which explicitly encodes various fine-grained attributes of humans in both space and time domains. Our representation encodes two principles: (i) compositionality, i.e. decomposing an trajectory into sub-trajectories into boxes, using multiple cues in both 2D and 3D, e.g., ground occupancy consistency, appearance similarity, motion coherence, which are mutually complementary while tracking people over time; (ii) attribution, i.e. augmenting each trajectory elements with a set of fine-grained semantic attributes (e.g., activities), or geometric attributes (e.g., facing directions, postures and actions), to enhance cross-view tracklet associations.

The inference of the optimal representation is solved by iteratively grouping tracklets with cluster sampling and estimating people semantic attributes by dynamic programming.

9.3.1 Representation

We first introduce the proposed ST-APG model and augmented attributes.

9.3.1.1 Spatio-temporal Attributed Parse Graph

Under the multi-view setting, activities in a scene S are captured by multiple different cameras $\{C_1, C_2, \dots, C_n\}$ with overlapping field of view (FOV). Videos from these cameras are synchronized in time. Given such data, our goal is to discover the trajectories Γ of every person within the scene, that is,

$$\Gamma = \{\Gamma_i : i = 1, \dots, K\}, \quad (9.8)$$

where K indicates the total number of people appearing in the scene over a time period.

We follow a default tracking-by-detection pipeline and apply [125] to obtain detected bounding boxes. After that, we associate them into short trajectory fragments, i.e., tracklets, similar to [68, 160]. Tracklets are used as the basic units in this paper and can be regarded as a mid-level representation to reduce the computation complexity, similar to super-pixels/voxels in segmentation. Normally, the duration of a tracklet is short (less than 300 frames, usually 50-200 frames). The person identity and motion within a tracklet usually remain consistent. A tracklet τ consists of a short sequence of object bounding boxes, which can be denoted as

$$\tau = \{(b_k, t_k) : k = 1, 2, \dots, |\tau|\}, \quad (9.9)$$

where b_k indicates the bounding box and t_k the corresponding frame number. Note that the 3D ground position is calculated by projecting the foot point of the 2D bounding box onto the world reference frame. For convenience, we denote the start time and end time of a tracklet by t^s and t^e , respectively.

Given a tracklet set $\Gamma = \{\tau_j, j = 1, 2, \dots, N\}$, we can re-write the scene-center trajectory of a person Γ_i as

$$\Gamma_i = \{\tau_j : l(\tau_j) = l_i, j = 1, 2, \dots, N\}, \quad (9.10)$$

where K indicates the total number of existing people in the scene. Each tracklet τ_j will be assigned with a label $l_i \in \{0, 1, \dots, K\}$, which can be regarded as the person ID which it belongs to. We also add $l_i = 0$ to denote this tracklet belongs to background.

Therefore, the problem of multi-view tracking can be formulated as a tracklet grouping problem, i.e. clustering tracklets of the same person into scene-centered trajectories. We further associate these tracklets with attributes and represent the scene as a Spatio-temporal Attributed Parse Graph (ST-APG) M , as illustrated in Fig. 7.5. A ST-APG consists of four components:

$$M = (S, X(S), \Gamma, X(\Gamma)), \quad (9.11)$$

where $X(S)$ denotes the global attributes (i.e., homographs $\{H_1, H_2, \dots, H_n\}$ for each camera $\{C_1, C_2, \dots, C_n\}$), $X(\Gamma)$ denotes the semantic attributes for tracklets. Therefore, solving multi-view people tracking is equivalent to finding the optimal ST-APG.

9.3.1.2 Attributes

Besides the identity label $l(\cdot)$, a tracklet τ_i is enriched with four kinds of attributes:

$$x(\tau_i) = (l(\tau_i), f(\tau_i), h(\tau_i), \{\vec{v}_{i,k}\}_{k=1}^{|\tau_i|}, \{a_{i,k}\}_{k=1}^{|\tau_i|}), \quad (9.12)$$

where $f(\tau_i)$ denotes the appearance attribute, $h(\tau_i)$ denotes the geometry attribute, $\vec{v}_{i,k}$ denotes the motion attribute of tracklet τ_i and $a_{i,k}$ the pose/action attribute at time $t_{i,k}$, i.e., the k -th frame of tracklet τ_i . The motion attributes are computed within a time window of 5 frames.

Appearance attribute. Instead of using traditional descriptors (e.g., HOG, SIFT, color histograms, MSCR) to measure the appearance discrepancy, we employ the powerful DCNN to model people's appearance variations. Notice that most DCNNs are trained over generic object categories and insufficient to provide fine-grained level of information about people's identities [173]. We therefore fine-tune the Alex-Net using people image samples with identity labels. Note the training samples are augmented from unlabeled data and identity labels are obtained in an unsupervised way. Similar to bag-of-words (BoW), our DCNN plays the role of a codebook, which codes a person image with common people appearance templates. We use this 1000-dimensional output as our appearance descriptor. The new DCNN consists of 5 convolutional layers, 2 max-pooling layers, 3 fully-connected layers and a final 1000-dimensional output. The last two layers are discarded and replaced by random initializations. The output is new 1000 labels on people's identities.

Geometric attribute. Similar to the literature, we define the geometry attribute $h(\tau_i)$ as the 2D object bounding boxes and projected footprints on the 3D ground plane. Besides appearance and geometry attributes, we further leverage two kinds of human semantic attributes to specifically handle the task of people tracking.

Motion attribute. We assume the facing direction of a person is same as his/her motion direction. The average speed \vec{v}_i is computed for each tracklet τ_i . However, 2D view-based motion not only suffers from the scale problem, but also is useless for cross-view comparisons. We thus transform the 2D view-based motion into the 3D real motion. Given the camera calibration, the foot point of each 2D bounding box is calculated and projected back onto the 3D ground. The speed and facing direction are thus computed and regarded as the motion attributes.

Semantic attribute. We further leverage a set of human actions and poses a_i to describe certain behaviors and activities within people's trajectories. We include 6 kinds of actions: standing, sitting, bending, walking, running, riding bike, which covers people's common type of actions/poses in public. The collected training set consists 5000 images. We thus also fine-tune a 7 layer Alex-Net, with 5 convolutional layers, 2 max-pooling layers, 3 fully-connected layers. The final output gives us a 6d human pose/action confidence score and can be regarded as the local attribute probability $p(a_i)$. Besides the unary pose/action confidence, we further learn a binary temporal consistency table $p(a_i, a_j)$ to describe the possible transitions between two successive pose/action attributes.

9.3.2 Bayesian Formulation

According to Bayes' rule, M can be solved by maximizing a posterior (MAP), that is,

$$\begin{aligned} M^* &= \arg \max_M p(M|I; \theta) \\ &\propto \arg \max_M p(I|M; \theta) \cdot p(M; \theta) \\ &= \arg \max_M p(I|M; \theta) \cdot \frac{1}{Z} \exp \{-\mathcal{E}(M; \theta)\}, \end{aligned} \quad (9.13)$$

where θ indicates the model parameters.

Likelihood term $\mathcal{E}(I|M; \theta)$ measures how well the observed data (video bundle) satisfies a certain object trajectory. The video data I is only dependent on the terminal nodes V_T and can be further decomposed as

$$\begin{aligned} p(I|M; \theta) &= \left(\prod_{\tau_i \in \Gamma} \prod_{b_k \in \tau_i} p^{fg}(b_k) \right) \cdot \prod_{b_k \in I \setminus \Gamma} p^{bg}(b_k) \\ &= \prod_{\tau_i \in \Gamma} \prod_{b_k \in \tau_i} \frac{p^{fg}(b_k)}{p^{bg}(b_k)} \cdot \prod_{b_k \in I} p^{bg}(b_k), \end{aligned} \quad (9.14)$$

where $p^{fg}(\cdot)$ and $p^{bg}(\cdot)$ are foreground and background probabilities, respectively. The second term $\sum_{b_k} p^{bg}(b_k)$ measures the background probability over the entire video data and thus can be treated as a constant, and the first term measures the divergence between foreground and background, which can be analogous to a probabilistic foreground/background classifier. We use the detection scores to approximate this log-likelihood ratio.

Prior term $\mathcal{E}(M; \theta)$ to impose constraints on people trajectories and their interactions. In this paper, we utilize an adjacency graph $G = \langle \Gamma, E \rangle$ to describe relations among tracklets. Each tracklet $\tau_i \in \Gamma$ is treated as a graph vertex and each edge $e_{ij} = \langle \tau_i, \tau_j \rangle \in E$ describes the relation between two adjacent (neighboring) tracklets τ_i and τ_j . In this paper, two tracklets τ_i and τ_j are regarded as neighbors $\tau_i \in nbr(\tau_j)$ if only their temporal difference is no more than $\Delta_t = 30$ frames and no far than $\Delta_d = 5m$. Given an adjacency graph, we consider two factors: intra-tracklet relations and inter-tracklet relations:

$$\mathcal{E}(M; \theta) = \sum_{\tau_i \in \Gamma} \mathcal{E}_v(\tau_i; \theta) + \sum_{e_{ij} \in E} \mathcal{E}_e(\tau_i, \tau_j). \quad (9.15)$$

9.3.2.1 Intra-tracklet and Inter-tracklet Relations

Assuming the likelihood of each bundle is calculated independently given the partition, then $\mathcal{E}_v(M; \theta)$ can be written as

$$\mathcal{E}_v(M; \theta) = \sum_{\tau_i \in \Gamma} \mathcal{E}_v(\tau_i; \theta). \quad (9.16)$$

Each term $\mathcal{E}_v(\tau_i|M; \theta)$ measures how the tracklet τ_i closely related within the trajectory. We estimate $\mathcal{E}(\tau_i|M; \theta)$ as a Hidden Markov Chain structure, where the unary term $\mathcal{E}_v(a_i)$ is the attribute classification entropy, and the pairwise term $\mathcal{E}_v(a_i, a_j)$ is the attribute consistency in two successive frames, that is

$$\begin{aligned} \mathcal{E}_v(\tau_i; \theta) &= \sum_{k=1}^{|\tau_i|} \mathcal{E}_v(a_{i,k}) + \sum_{k=1}^{|\tau_i|-1} \mathcal{E}_v(a_{i,k}, a_{i,k+1}), \\ &= - \sum_{k=1}^{|\tau_i|} \log(p(a_{i,k})) - \sum_{k=1}^{|\tau_i|-1} \log(p(a_{i,k}, a_{i,k+1})), \end{aligned} \quad (9.17)$$

Note the motion information is trivial for successive frames and we thus ignore this part.

We regard edges generated by five types of constraints as cooperative edges E^+ . The edge set E is further extended with conflicting edges E^- , that is, $E = E^+ \cup E^-$. Both types of relationships are utilized to help us group tracklets with similar characteristics together and with conflicting characteristics being dispelled.

The graph edge energy $\mathcal{E}_e(M; \theta)$ can be decomposed into pairwise potentials between every two adjacent

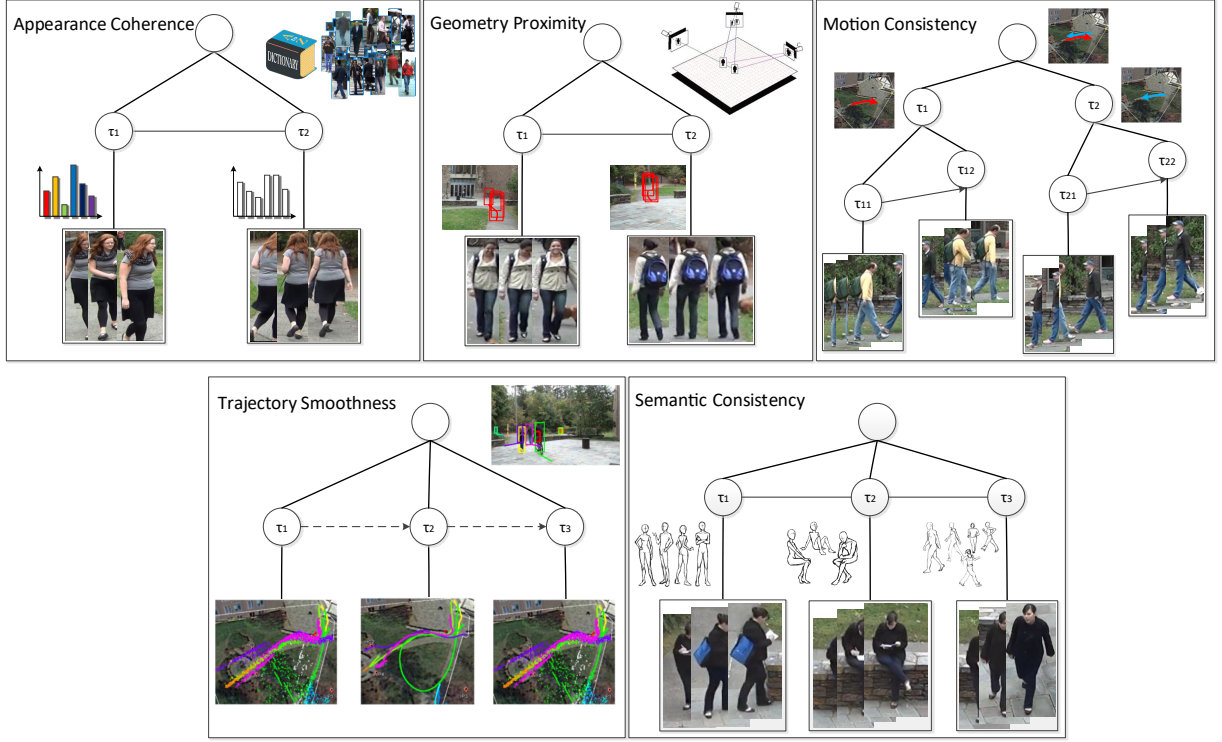


Figure 9.10: An illustration of five types of relations. Reprinted (or modified), with permission, from (Xu, *et al.*, 2017) [172].

tracklets within G , that is,

$$\mathcal{E}_e(M; \theta) = \sum_{l_i=l_j, e_{ij} \in E^+} \mathcal{E}_e^+(\tau_i, \tau_j) + \sum_{l_i=l_j, e_{ij} \in E^-} \mathcal{E}_e^-(\tau_i, \tau_j), \quad (9.18)$$

where $\mathcal{E}_e^+(\cdot)$ and $\mathcal{E}_e^-(\cdot)$ are the corresponding cooperative and conflicting edge energies.

For cooperative edges, we develop five types (i.e., $E^+ = \{E^{app}, E^{geo}, E^{mov}, E^{sem}, E^{smo}\}$) of relations between two tracklets, as illustrated in Fig. 9.10. Given two tracklets, we consider traditional visual relations (i.e., appearance and geometry), temporal relations (i.e., motion and smoothness) and leveraged semantic relations (i.e., pose/action).

Appearance Coherence E^{app} . This constraint assumes that the same person should share similar appearance across time and cameras. We use this 1000-dimensional output as our appearance descriptor. Given two tracklets τ_i and τ_j , the appearance coherence constraint $\mathcal{E}_e^{app}(\tau_i, \tau_j)$ is defined as

$$\mathcal{E}_e^{app}(\tau_i, \tau_j) = \sum_1^{|\tau_i|} \sum_1^{|\tau_j|} \frac{\|f(\tau_i) - f(\tau_j)\|_2}{|\tau_i| \cdot |\tau_j|}, \quad (9.19)$$

where $f(\tau_i)$ denotes the encoded feature vector (appearance attribute) of τ_i . This term actually measures the mean complete-link appearance dissimilarities among object bounding boxes belonging to two tracklets.

Geometry proximity E^{geo} . Given tracklets from a single view or cross views, we first project them on the world reference frame to measure their geometric distances uniformly. However, considering tracklets

with different time stamps and lengths, it is not a trivial task to determine whether the two given tracklets belong to the same object or not. The reason lies in: i) the time stamps of tracklet pairs might not be well aligned; ii) the localizations across views usually lead to remarkable amount of errors.

In order to address these issues, we introduce a kernel to measure these time series samples. The kernel $K(\tau_i, \tau_j)$ to measure the distance between two tracklets τ_i and τ_j is defined as the product of two kernel distances in space and time where $\phi_l(h(\tau_i), h(\tau_j))$ and $\phi_t(t_n, t_m)$ are two RBF kernels between two points. We use different σ_l and σ_t values for the two kernels, respectively. This new kernel acts like a sequential convolution filter and takes both spatial and temporal proximities into consideration.

Given a set of training samples Λ ,

$$\Lambda = \{(\tau_i, \tau_j, y_n) : n = 1, \dots, |\Lambda|\}, \quad (9.20)$$

where $y_n \in \{1, 0\}$ indicates whether or not the two tracklets τ_i and τ_j belong to the same identity, we can train a kernel SVM with the energy function

$$\min_w \frac{1}{2} \langle w, w \rangle + C \sum_n \max(0, 1 - y_n \langle w, K(\tau_i, \tau_j) \rangle), \quad (9.21)$$

where C is a regularization factor.

We therefore interpret the opposite of the classification margin as the energy term $\mathcal{E}_e^{geo}(\tau_i, \tau_j)$.

Motion consistency E^{mov} . Given two proximate tracklets, the motion direction actually provides a solid evidence to show whether these tracklets belong to a same person or two persons crossing each other. Therefore, we can compute the angle between two motion directions. that is,

$$\mathcal{E}_e^{mov}(\tau_i, \tau_j) = \frac{1}{|\tau_i \cap \tau_j|} \sum_{t_m \in \tau_i \cap \tau_j} \arccos \frac{\vec{v}_{i,m} \cdot \vec{v}_{j,m}}{|\vec{v}_{i,m}| |\vec{v}_{j,m}|}. \quad (9.22)$$

If the angle is large, this probably indicates that two persons are moving in different directions.

Trajectory smoothness E^{smo} . We model the motion information of a tracklet τ_i as a continuous function of its 3D ground positions b w.r.t. time t , i.e., $b = \tau(t)$. We define a constraint on two tracklets that they can be interpreted with the same motion function. However, finding this motion pattern is a challenging problem. The reason lies in two-fold: i) inaccurate 3D positions due to perspective effects, detection errors and false alarms; ii) missing detections and object inter-occlusions in certain views, especially for crowded scenarios. In this paper, we address these issues in the following two aspects.

Firstly, we employ the b-spline function to represent the motion pattern of the trajectory. B-spline functions can enforce high-order smoothness constraints, which enables learning from sparse and noisy data. Considering a tracklet τ with 3D positions $\{b_i : i = 1, \dots, |\tau|\}$, starting time t^s and ending time and t^e , the spline function $\tau(t)$ uses some quadratic basis functions $B_k(t)$, and represents the motion path as a linear combination of $\Phi_k(t)$:

$$\begin{aligned} \tau(t) &= \sum_k \alpha_k \Phi_k(t), \\ s.t. \quad \tau''(t^s) &= \tau''(t^e) = 0, \end{aligned} \quad (9.23)$$

where $\tau''(t)$ denotes the second derivative of $\tau(t)$. The constraints enforce zero curvature at the starting and the ending point.

Secondly, we take advantages of the multi-view setting and derive feasible regions for object 3D positions to further confine the fitted motion curve. As illustrated in Fig. 9.11, given bounding boxes of a single object in the views (a), (b) and (c), we first perform exhaustive search to find the two anchor points (yellow dots in the image) along two sides of the foot position of each object. An anchor point is defined

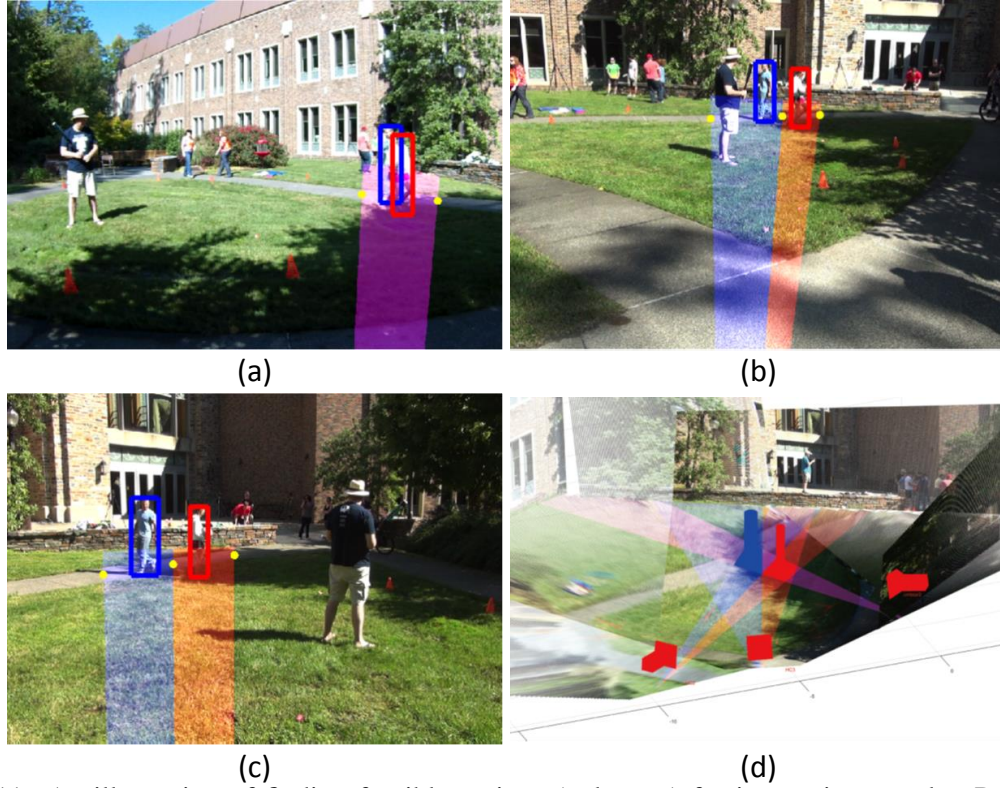


Figure 9.11: An illustration of finding feasible regions (polygons) for interacting people. Reprinted (or modified), with permission, from (Xu, *et al.*, 2017) [172].

as a position where the surrounding 8×8 area contains most of background regions. Note that we generate background masks by GMM background modeling.

Once obtaining all the anchor points for an object, we can find the union area Ω , i.e., a polygon on the world ground plane, as shown the shaded area in (d). These polygons serve as additional localization feasibility constraints on the motion pattern. That is, the spline fitting is formulated as minimizing the following objective function:

$$\mathcal{E}_e^{smo*}(\tau_i, \tau_j) = \min_{\alpha_k, \Phi_k} \sum_{(b_n, t_n) \in \tau_i \cup \tau_j} \left(b_n - \sum_k \alpha_k \Phi_k(t_n) \right)^2, \quad (9.24)$$

$$s.t. \quad \alpha_k \Phi_k(t_n) \in \Omega_n.$$

This is a constrained convex programming problem considering that all polygons are convex. We refer the readers to find more details about b-spline and robust fitting algorithms in [31].

The energy term $\mathcal{E}_e^{smo}(\tau_i, \tau_j)$ is defined upon the averaged residuals for spline fitting, i.e.,

$$\mathcal{E}_e^{smo}(\tau_i, \tau_j) = \frac{1}{|\tau_i \cup \tau_j|} \mathcal{E}_e^{smo*}(\tau_i, \tau_j). \quad (9.25)$$

Semantic consistency E^{sem} . Noticing the pose/action of a same person across different views should also be consistent, we thus use the learned temporal consistency table $p(a_i, a_j)$ to describe the consistency

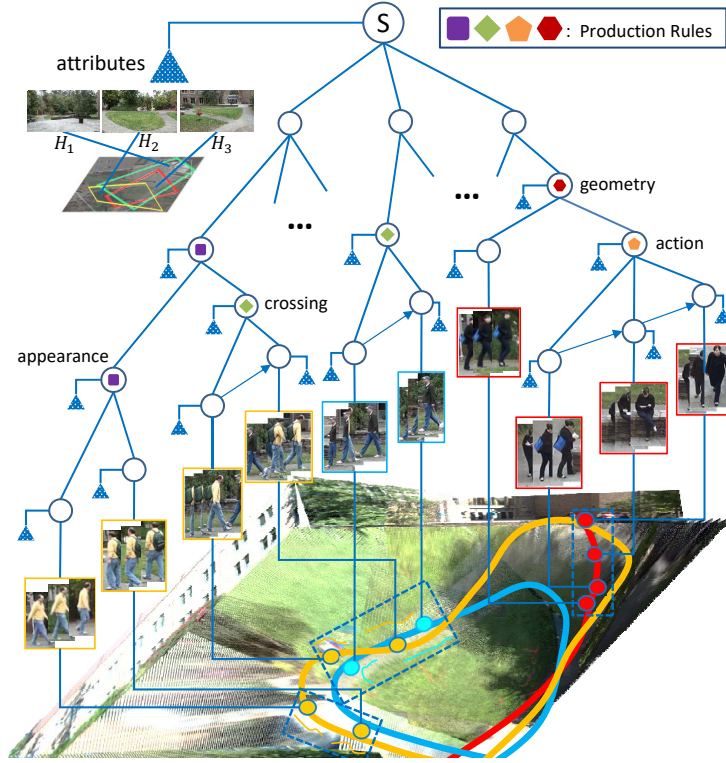


Figure 9.12: An illustration of our inference process. A Spatio-temporal Attributed Parse Graph (ST-APG). The scene S is generated by 3D reconstruction and associated with certain global attributes (e.g., homograph H_1, \dots, H_n), and can be decomposed into tracklets belonging to different persons. Each tracklet is also leveraged with five types of semantic attributes (i.e., the blue triangles connected to nodes) and hierarchically organized in a parse graph. Reprinted (or modified), with permission, from (Xu, *et al.*, 2017) [172].

between two actions/poses. The rule is computed as

$$\mathcal{E}_e^{sem}(\tau_i, \tau_j) = \frac{1}{|\tau_i \cap \tau_j|} \sum_{t_m \in \tau_i \cap \tau_j} \mathbf{1}(a_{i,m} \neq a_{j,m}), \quad (9.26)$$

where $\mathbf{1}(\cdot)$ is the indicator function. Note that we only consider such relation among overlapped frames of two tracklets τ_i and τ_j .

Conflicting relations E^- . We enforce hard constraints to guarantee that i) two tracklets from the same view with temporal overlap will never be grouped together; ii) two adjacent tracklets with same identities will never have impossible pose/action transitions defined in temporal consistency table $p(a_i, a_j)$. The energy term $\mathcal{E}_e^-(\tau_i, \tau_j)$ is thus defined as

$$\mathcal{E}_e^-(\tau_i, \tau_j) = \begin{cases} th_{e-}, & l_i = l_j, e_{ij} \in E^- \\ 0, & \text{otherwise} \end{cases}, \quad (9.27)$$

where th_{e-} is a cutoff energy threshold and we set $th_{e-} = 5$ in this paper.

9.3.3 Inference

Given a scenario, finding the optimal ST-APG includes two sub-tasks: (1) partitioning tracklet set Γ into trajectories belonging to different people Γ_i , (2) inferring the semantic human attributes for each person. Noticing that sub-task (1) is a combinatorial optimization problem and jointly solving these two sub-tasks is infeasible, we therefore propose an inference algorithm to optimize these two sub-tasks iteratively. The inference process is illustrated in Fig. 9.12: For sub-task (1), we apply a stochastic clustering algorithm, i.e., Swendsen-Wang Cuts [7], which could efficiently and effectively traverses through the grouping solution space; For sub-task (2), given grouped tracklets, we can use Dynamic Programming to update the semantic attributes of tracklets within every group (i.e., person trajectory). These two algorithms are iterated one after another until convergence.

9.3.3.1 Associating Tracklets by Stochastic Clustering

Traditional sampling algorithms usually suffer from the efficiency issues. On the contrary, cluster sampling algorithm overcomes this issue by randomly grouping clusters and re-sampling cluster as a whole. The algorithm consists of two steps:

(I) **Generating cluster set.** Given an adjacency graph $G = \langle \Gamma, E \rangle$ and the current state M , we regard every edge e_{ij} in this graph as a switch. We turn on every edge e_{ij} probabilistically with its edge probability p_e . Afterwards, we regard candidates connected by "on" positive edges as a cluster V_{cc} and collect separate clusters to produce the cluster set.

(II) **Relabeling cluster set.** We randomly choose a cluster V_{cc} from the produced cluster set and randomly change the label of the selected cluster, which generates a new state M' . This is essentially changing the ID of a group of tracklets. This group of tracklets can either be merged into another trajectory, or set to background noises. Following the Markov chain Monte Carlo principal, we accept the transition from state M to new state M' with a rate $\alpha(\cdot)$ defined by the Metropolis-Hastings method [96]:

$$\alpha(M \rightarrow M') = \min(1, \frac{p(M' \rightarrow M) \cdot p(M'|\Gamma)}{p(M \rightarrow M') \cdot p(M|\Gamma)}), \quad (9.28)$$

where $p(M' \rightarrow M)$ and $p(M \rightarrow M')$ are the state transition probability, $p(M'|\Gamma)$ and $p(M|\Gamma)$ the posterior defined in Equation.(9.13). This guarantees the stochastic algorithm can find better states and obtains reversible jumps between any two states.

Following instructions in [7], the transition probability ratio can be calculated as

$$\frac{p(M' \rightarrow M)}{p(M \rightarrow M')} \propto \frac{p(V_{cc}|M')}{p(V_{cc}|M)} \propto \frac{\prod_{e \in E_{M'}^*} (1 - p_e)}{\prod_{e \in E_M^*} (1 - p_e)}, \quad (9.29)$$

where E^* denotes the sets of edges being turned off around V_{cc} , that is,

$$E^* = \{e \in E : \tau_i \in V_{cc}, \tau_j \notin V_{cc}, l(\tau_i) = l(\tau_j)\}. \quad (9.30)$$

9.3.3.2 Assigning Semantic Attributes by DP

Given a trajectory, we first find trajectory gaps (i.e., no bounding box presented) below 60 frames, we then apply a linear interpolation to fill-in the missing bounding boxes.

After that, assigning the semantic attribute is similar to estimating the likelihood term $p(\tau_i|G)$. The whole trajectory is also treated as a Markov chain structure. We therefore apply the standard factor graph belief propagation (sum-product) algorithm to infer the semantic human attributes of a trajectory.

A short summary of our proposed inference algorithm is shown in Algorithm 17.

Algorithm 17: Sketch of our inference algorithm

Input: Tracklet set Γ , global attributes $X(S)$
Output: Spatio-temporal Attributed Parse Graph M

- 1 Assign semantic attributes for each tracklet τ_i by DP ;
- 2 Construct adjacency graph G by computing cooperative and conflicting relations among Γ ;
- 3 Initialize $K = |\Gamma|$, $l_i = i$;
- 4 **repeat**
- 5 Generate a cluster V_{cc} ;
- 6 Randomly relabel cluster V_{cc} and obtain a new state M' ;
- 7 Accept the new state with acceptance rate $\alpha(M \rightarrow M')$;
- 8 Re-run DP on each new trajectory to update semantic attributes ;
- 9 **until** convergence;

9.4 Joint Video and Text Parsing

In this section, we present a multi-modal analysis framework to process video and text jointly for event understanding. Specifically, we propose a compositional knowledge representation, Spatial-temporal-causal And-Or Graph (STC-AOG), which jointly models possible hierarchical compositions of objects, scenes and events as well as their interactions and mutual contexts. Based on the representation, we propose a joint parsing framework consisting of three modules: video parsing, text parsing and joint inference. The objectives of this framework are three-ways: (i) we aim at deep semantic parsing of video and text that goes beyond the traditional bag-of-words approaches; (ii) we perform parsing and reasoning across the spatial, temporal and causal dimensions based on the joint STC-AOG representation; (iii) we show that deep joint parsing facilitates subsequent applications such as generative narrative text descriptions and answering queries in the forms of who, what, when, where and why.

9.4.1 Representation

In this section, we present the STC-AOG representing the compositional structures of objects, scenes and events.

Spatial And-Or Graph A spatial And-Or Graph (S-AOG) models the spatial decompositions of objects and scenes. An And-node in the S-AOG represents a scene or object that is the spatial composition of a set of parts. For example, a table can be the spatial composition of a table top above four legs; and the background of an indoor scene can be the spatial composition of multiple 2D planes (*i.e.* the walls, floor and ceiling) that are hinged. An Or-node represents alternative spatial configurations of a scene or object, such as different styles of a table and different viewpoints of an indoor scene. A terminal node in the S-AOG represents an atomic object that cannot be further decomposed. Figure 9.13(a) shows an example of S-AOG.

Temporal And-Or Graph A temporal And-Or Graph (T-AOG) models the temporal decompositions from events to sub-events and then to atomic actions. An And node in T-AOG represents an event that can be decomposed into a set of sub-events, while an Or node represents alternative configurations of an event. A terminal node in T-AOG represents an atomic action which is defined by properties of the action initiator(s) and the action target(s). Figure 9.14(a) shows an example of T-AOG. In this example, the event “buy from

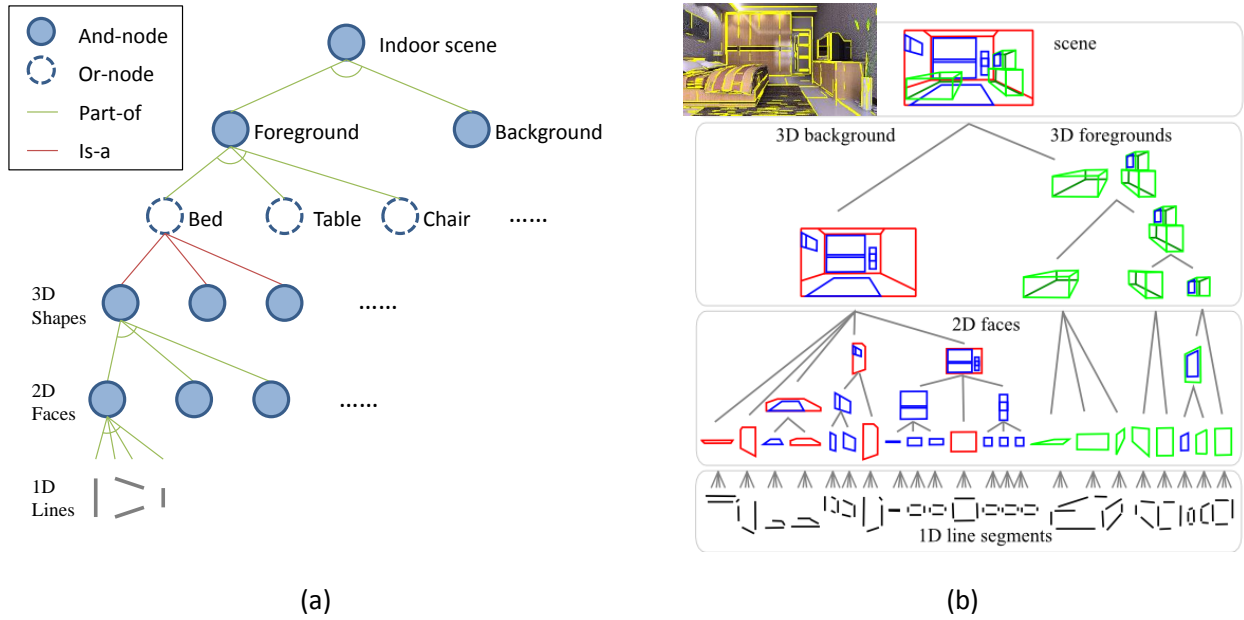


Figure 9.13: (a) An example of S-AOG. (b) An example of temporal parse graph sampled from (a), which is a realization of the S-AOG by making decisions at Or-nodes. Reprinted from Ref. [152] with permission of IEEE, © 2014.

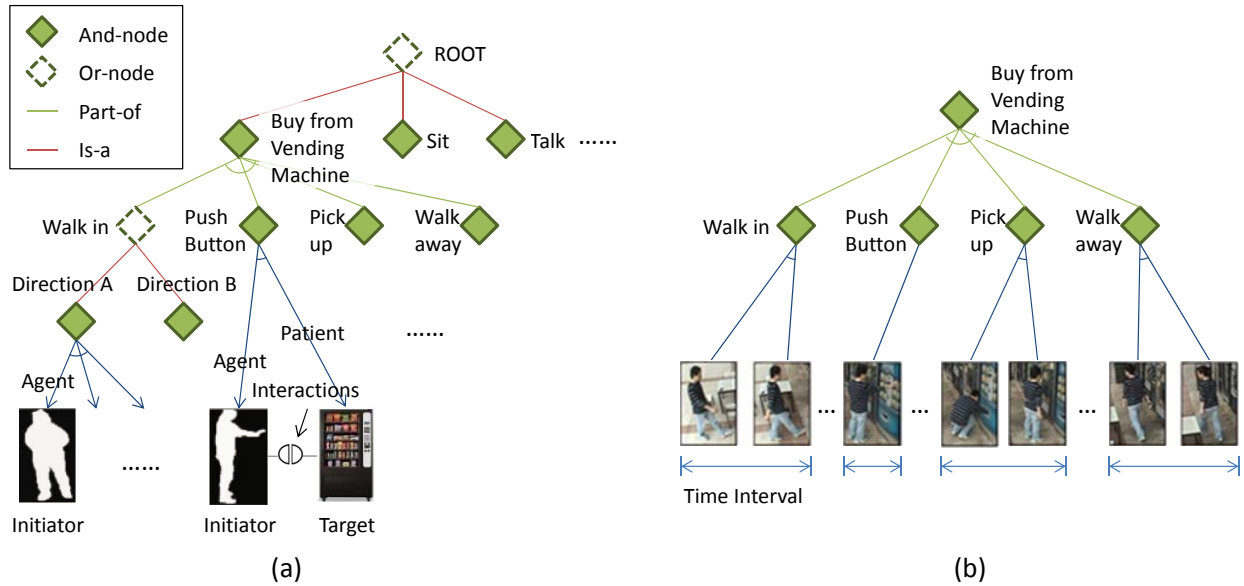


Figure 9.14: (a) An example of T-AOG. (b) An example of temporal parse graph sampled from (a), which is a realization of the T-AOG by making decisions at Or-nodes. Reprinted from Ref. [152] with permission of IEEE, © 2014.

vending machine" is an And node and can be decomposed into sub-events (walk in, push button, pick up, walk away). The "walk in" is an Or node and can have alternatives as walk to direction A or direction B. The atomic actions of the initiator are terminal nodes.

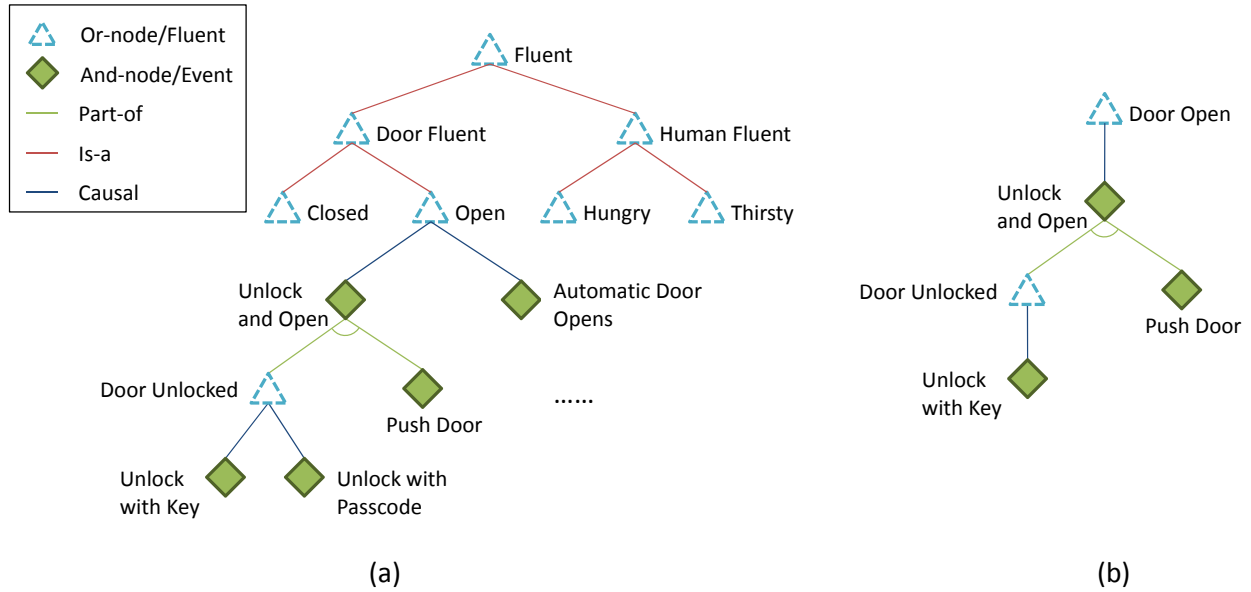


Figure 9.15: (a) An example of C-AOG representing the fluent changes between events. (b) An example of temporal parse graph sampled from (a), which is a realization of the C-AOG by making decisions at Or-nodes. Reprinted from Ref. [152] with permission of IEEE, © 2014.

Causal And-Or Graph A causal And-Or Graph (C-AOG) models the causal relation between events and fluent changes. An And node in C-AOG represents a composition of conditions and events that can cause a fluent change. An Or node in a C-AOG represents alternative causes that can result in a fluent change. A terminal node in C-AOG is either an event or a fluent. Figure 9.15(a) shows an example of T-AOG. In this example, the fluent change in “unlock and open a door” is an And node and can be decomposed into the fluent change of “door unlocked” and “push door”. The “door unlocked” is an Or node and has alternative fluents as “unlock with key” and “unlock with passcode”.

Joint STC-AOG The joint STC-AOG combines the S-AOG, T-AOG and C-AOG together by modeling the interrelation among them. The connections across the three types of AOGs enable the propagation of information between them during parsing. This also leads to a more coherent and comprehensive joint interpretation.

Parse Graph Similar as subsection 9.3.1, we represent an interpretation of input video and text as a parse graph. A parse graph is a realization of the S/T/C-AOG by selecting at each Or node one of alternative configurations. Figure 9.13(b), 9.14(b), 9.15(b) show a example of spatial, temporal and causal parse graph respectively.

9.4.2 Joint Parsing and Inference

We present the joint parsing framework with three modules: video parsing, text parsing and joint inference. Figure 9.16 shows an example of joint parsing with video and text parse graphs.

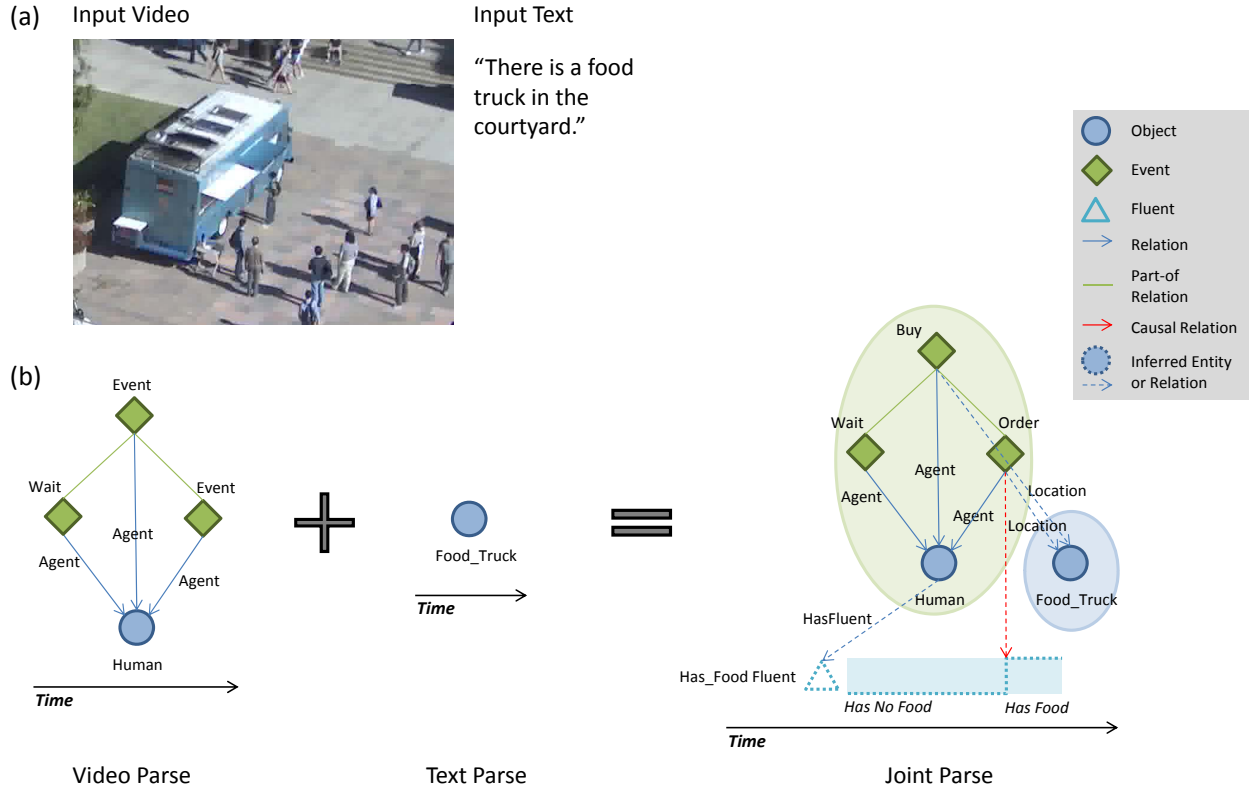


Figure 9.16: An example of joint inference with video and text parsing. Reprinted from Ref. [152] with permission of IEEE, © 2014.

9.4.2.1 Probabilistic Modeling

We define a probabilistic distribution over parse graph by assigning energy terms to the AOGs. Typically, the energy of a parse graph pg is defined as:

$$E(pg) = \sum_{v \in V^{or}(pg)} E_{or}(v) + \sum_{r \in R(pg)} E_R(v) \quad (9.31)$$

where $V^{or}(pg)$ is the set of Or nodes in pg , $E_{or}(v)$ is the energy associated to the configuration selected at the Or node v , $R(pg)$ is the set of relations specified at the And nodes in pg , and $E_R(r)$ is the energy associated with the relation r . Then the energy term for the joint STC-AOG can be defined as a summation of four terms:

$$E_{STC}(pg) = E_S(pg) + E_T(pg) + E_C(pg) + \sum_{r \in R^*(pg)} E_R(v) \quad (9.32)$$

where $E_S(pg)$, $E_T(pg)$, $E_C(pg)$ are energy terms defined by S-AOG, T-AOG and C-AOG respectively and $R^*(pg)$ is the set of relations connecting the three types of AOGs.

Given the above definition, a prior probability of a parse graph pg can be defined as:

$$p(pg) = \frac{1}{Z} \exp^{-E_{STC}(pg)} \quad (9.33)$$

where Z is the normalization factor.

9.4.2.2 Video Parsing

The objective energy function for video parsing is defined as:

$$E_v(pg_{vid}) = E_{STC}(pg_{vid}) - \log p(vid|pg_{vid}) \quad (9.34)$$

where E_{STC} is the prior knowledge of the video parse parph pg_{vid} as defined in Equation 9.32 and $p(vid|pg_{vid})$ is the probability of the input video vid given the objects, scenes and events specified in the parse graph pg_{vid} . In the following part, we briefly introduce how to perform spatial, temporal and causal parsing on the video respectively.

Spatial parsing We follow [181] to use hierarchical cluster sampling on each video frame for spatial parsing. The approach consists of two stages. In the first stage, the approach performs bottom-up clustering in which lower-level visual entities (*e.g.* line segments) of a video frame are composed into possible higher level objects according to the S-AOG. In the second stage, the parser applies the Metropolis-Hastings algorithm in the space of parse graphs. The sampler performs two types of operations to change the current spatial parse graph: the first is to add a candidate object composed in the first stage into the parse graph, where the proposal probability is defined based on the energy of the candidate object as well as the compatibility of the object with the rest of the parse graph; the second type of operation is to randomly prune the current parse graph. At convergence the approach outputs the optimal spatial parse graph.

Temporal parsing We follow [110] to use a Earley parser for temporal parsing. Given a sequence of video frames, the temporal parser reads in frames sequentially, and at each frame maintains a state set that contains pending derivations of And nodes that are consistent with the input video up to the current frame. Specifically, the And nodes at the top level of the T-AOG are added into the state set of frame 0, with all their child nodes pending. With each new frame read in, three basic operations are iteratively executed on the state set of the new frame: the *prediction* operation adds into the state set new And nodes that are expected according to the existing pending derivations in the state set; the *scanning* operation checks the detected agents, objects and fluents in the new frame and advances the pending derivations in the state set accordingly; the *completion* operation identifies And-nodes whose derivations are complete and advances the pending derivations of their parent And nodes. During this process, we prune derivations that have high energies to make the sizes of state sets tractable. After all the frames are processed, a set of candidate parses of the whole video can be constructed from the state sets.

Causal parsing We run similar process as in T-AOG to use the Earley parser for Causal parsing. For each fluent change detected in the video using detectors, we collect the set of events and run the Earley parser on the sub-graph of the C-AOG rooted at the detected fluent change.

9.4.2.3 Text Parsing

We perform text semantic parsing to convert the input text description into the text parse graph pg_{txt} . The processing pipeline consists of four steps: text filtering, Part-of-Speech (POS) tagging and dependency inference, dependency filtering, and parse graph generation.

Text Filtering This step preprocesses the text by locating and labeling the certain categories of words or phrases in the input text. It first performs the named entity recognition (NER) to identify text elements related to names of persons, time, organizations and locations. Second, it recognizes certain compound noun phrases that refer to a single entity (*e.g.* “food truck” and “vending machine”).

POS Tagging and Dependency Inference This step performs POS tagging and syntactic dependency parsing using the Stanford Lexicalized Parser. During POS tagging, each word is assigned a grammatical category such as noun, verb, adjectives, adverb, article, conjunct and pronoun. The parser then performs dependency analysis to identify relations between the words in the sentence. It generates a dependency

graph in which each node is a word and each directed edge represents a basic grammatical relation between the words.

Dependency Filtering In this step, we map each word in the sentence to an entity type defined in the ontology or to a literal type.

Parse Graph Generation In this step, we design an attribute grammar to parse the set of dependencies and infer the semantic relations. The final text parse graph is then generated from the semantic relations inferred in this step as well as the entity types produced from the previous step.

9.4.2.4 Joint Inference

In joint inference, we construct the joint parse graph from the video and text parse graphs produced by the first two modules.

Matching. We match a node a from the video parse graph with a node b from the text parse graph, and merge them into a single node c which inherits all the edges attached to either node a or node b .

Deduction. We insert a new subgraph into the joint parse graph, such that the prior probability of the joint parse graph as defined by the STC-AOG is increased. Specifically, for each entity in the joint parse graph, we find the type of this entity in the STC-AOG and insert an instantiation of its immediate surrounding subgraph in the AOG into the joint parse graph.

Revision. We either remove a subgraph from the parse graph, or change the annotation of a node or edge in the parse graph, in order to solve a conflict as defined in the STC-AOG.

Stop criterion for deduction. We may deduce multiple candidate subgraphs that have similar energy and are mutually exclusive. Specifically, we cancel the deduction operator if the entropy of the deduced subgraph is higher than a threshold:

$$H(pg_{de}|pg_{jnt}) = - \sum_{i=1}^N p(pg_{de}^i|pg_{jnt}) \log p(pg_{de}^i|pg_{jnt})$$

$$> \frac{\log N}{c}$$

where pg_{jnt} is the joint parse graph before applying a deduction operator, pg_{de} denotes the subgraph inserted by the deduction operator, N is the number of possible candidate subgraphs that can be deduced by the operator, and $c > 1$ is a pre-specified constant.

9.5 Query Answering from Joint Parse Graphs

Visual Question Answering (VQA) has been widely explored in the deep learning literature. In contrast to the previous feature encoding-decoding framework, in this section we propose to perform query answering from joint parse graphs as proposed in section 9.4, aiming at presenting a reasoned, explainable procedure for question answering.

9.5.1 Answering Natural Language Queries

The joint parse graph is a semantic representation of the objects, scenes and events contained in the input video and text, which is very useful in many applications. In this section, we show how the joint parse graph can be used in semantic queries to answer questions, such as (a) questions in the forms of who, what, when, where and why; and (b) summary of scenes or events, e.g. how many persons.

9.5.1.1 Text Query Parsing

Given a simple plain text question about the objects, scenes and events in the input video and text, we parse the question into a formal semantic query representation. We use the same text parsing procedure as in section 9.4 for query parsing, where the attribute grammar for analyzing the dependencies is extended to include interrogative wh-words such as *who*, *what*, *when*, *where* and *why*. These wh-words indicate what the objective of the query is, and the rest of the text query defines the query conditions. For example, the question “who has a cap?” is parsed into the following dependencies and attributes: The first dependency

```
nsubj(Event, who)
root(ROOT-0, Event); Event.type = Possess
det(Object, Det)
dobj(Event, Object); Object.type = Cap
```

indicates that the query is asking for the agent of an event. The rest of the dependencies specify what the event type (“Possess”) and the event patient (“Cap”) are.

9.5.1.2 Computing answer probabilities from multiple interpretations

As illustrated in section 9.4, the input video and text can have different interpretations represented by different parse graphs. To answer a query accurately, we execute query on all interpretations and then compare the collected answers. We associate the answers from different interpretations by matching their semantic types and spatial-temporal annotations. For answers that are matched, their probabilities are combined. Formally, the probability of an answer a is computed as:

$$p(a) = \sum_{pg} p(pg) \mathbb{1}(pg \models a) \quad (9.35)$$

where $p(pg)$ denotes the posterior probability of the joint parse graph pg and $\mathbb{1}(pg \models a)$ is the indicator function of whether parse graph pg entails the answer a . In this way, different possible answers can be ranked based on their marginal probabilities.

9.5.2 Visual Turing Test

The restricted Visual Turing Test system (VTT) presents a testing platform for joint video and query parsing. The VTT task is a story-line based visual query answering in long-term and multi-camera captured videos. By “restricted”, we mean the queries are designed based a selected ontology. Figure 9.17 shows an example of a full workflow of the VTT system.

9.5.2.1 Queries

A query is a first-order logic sentence (with modification) composed using variables, predicates, logical operators (\wedge , \vee , \neg), arithmetic operators, and quantifiers (\exists , \forall). The answer to a query is either true or false meaning whether the fact stated by the sentence holds given the data and the system’s state of belief.

Object definition queries. To define an object, specifications of object type, time and location are three components. Object type is specified by object predicates in the ontology. A time t is either a view-centric frame number in a particular video or a scene-centric wall clock time. A location is either a point (x, y) or a bounding box (x_1, y_1, x_2, y_2) represented by its two diagonal points, where a point can be specified either in view-centric coordinates (i.e. pixels) or in scene-centric coordinates (i.e. latitude-longitude, or coordinates

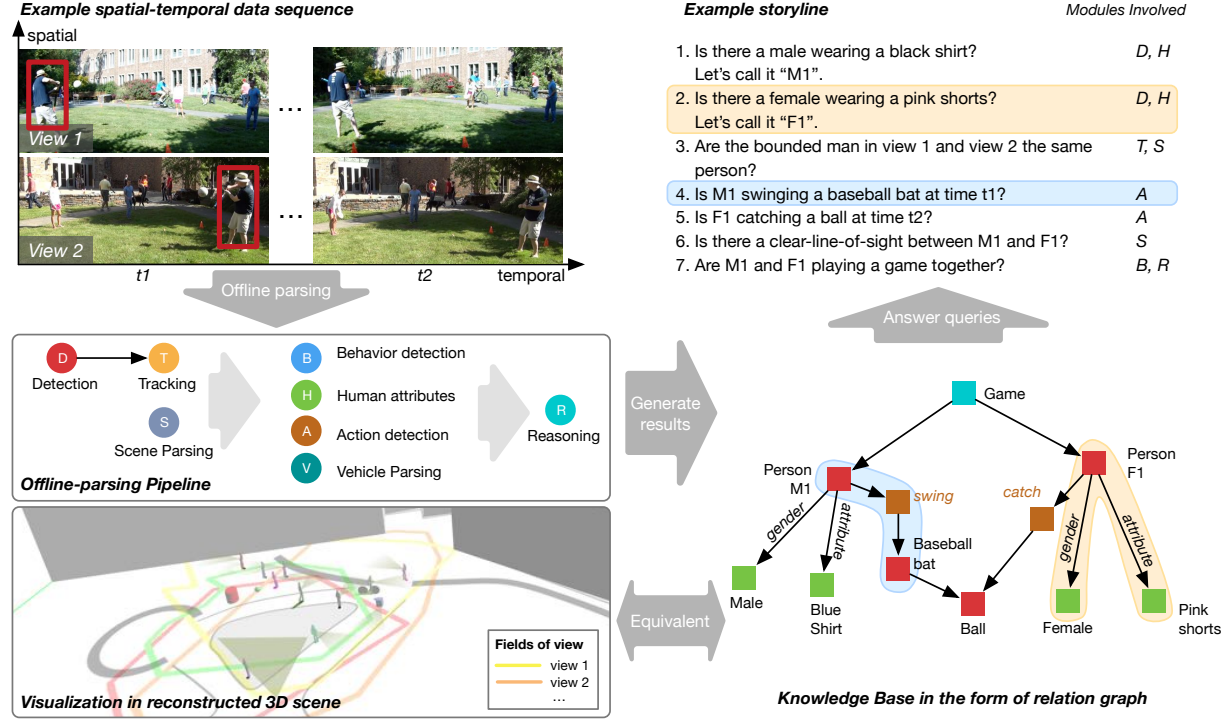


Figure 9.17: Illustration of our prototype vision system for VTT. Top-left: input videos with people playing baseball games. Middle-Left: Illustration of the offline parsing pipeline which performs spatial-temporal parsing in the input videos. Bottom-Left: Visualization of the parsed results. Bottom-Right: The knowledge base constructed based on the parsing results in the form of a relation graph. Top-Right: Example storyline and queries. Graph segments used for answering two of the queries are highlighted. Reprinted from Ref. [119] with permission.

in a customized reference coordinate system, if defined). For example, an object definition query regarding a person in the form of first-order logic sentence would look like:

$$\exists p \text{ person}(p; \text{time} = t; \text{location} = (x_1, y_1, x_2, y_2))$$

where the designated location is a bounding box.

Non-definition queries. Non-definition queries in a story line explores a system's spatial, temporal and causal understanding of events in a scene regarding the detected objects. The query space consists of all possible combinations of predicates in the ontology with the detected objects (and/or objects interacting with the detected ones) being the arguments. When expressing complex activities or relationships, multiple predicates are typically conjuncted by \wedge to form a query. For example, suppose M_1 and F_1 are two detected people confirmed by object detection queries, the following query states " M_1 is a male, F_1 is a female, and there is a clear line of sight between them at time t_1 ":

$$\text{male}(M_1) \wedge \text{female}(F_1) \wedge \text{clear-line-of-sight}(M_1, F_1; \text{time} = t_1)$$

Note that the location is not specified, because once M_1 and F_1 is identified and detected, we assume the vision system can track them over time.

9.5.2.2 Restricted visual turing test system

As illustrated in Figure 9.17, the proposed VTT consists of four components:

Multi-camera video dataset collection. Our multiple-camera video dataset includes a rich set of activities in both indoor and outdoor scenes. Videos are collected by multiple cameras with overlapping field-of-views during the same time window. A variety types of sensors are used: stationary HD video cameras located on the ground and rooftop, moving cameras mounted on bicycles and automobiles, and infrared cameras. The camera parameters are provided as meta data. The videos capture daily activities of a group of people and different events in a scene which include routine ones and abnormal ones with large appearance and structural variations exhibited.

Ontology guided story-line based query/answer collection. The ontology is sufficiently expressive to represent different aspects of spatial, temporal, and causal understanding in videos from basic level (*e.g.* identifying objects and parts) to fine-grained level. Based on the ontology, we build a toolkit for story-line query generation. Queries organized in multiple story lines are designed to evaluate a computer vision system from basic object detection queries to more complex relationship queries, and further probe the system's ability in reasoning from the physical and social perspectives, which entails human-like commonsense reasoning.

Integrated vision system. We build a computer vision system that can be used to study the organization of modules designed for different tasks and interactions between them to improve the overall performance. It is designed with two principles in mind: first, well-established computer vision tasks shall be incorporated so that we can built upon the existing achievements; second, the modules shall be loosely coupled so that it allows user to replace one or more modules with alternatives to study the performance in an integrated environment.

Q/A evaluation server. The evaluation server iterates through a stream of queries grouped by scenes. In each scene, queries are further grouped into story lines. A query is not available to the system until the previous story lines and all previous queries in the same story line have finished. The correct answer is provided to the system after each query. This information can be used by the system to be adaptive with the ability to learn from the provided answers. The answer can be used to update the previous understanding such that any conflict has to be resolved and wrong interpretations can be discarded.

10

Grounded Grammar Induction

In this Chapter, we describe how to induce the grounded grammar from vision and language. The image grammar introduced in ?? and ?? are mostly pre-defined or learned with supervision, which is not flexible and generalizable to novel instances.

In this work, we structurally align visual and textual elements to facilitate grammar induction in both vision and language. The intuition behind this practice is that forcing the multimodal alignment can reduce the inherent ambiguity of grammar induction for individual modalities, and the induced grammar can be more effective for downstream tasks with its structured representation.

10.1 Introduction

Part-whole relationships are crucial in semantic structures. For example, the constituent “two arms” in fig. 10.1 does not simply refer to a chair, but instead refers to the chair’s arms. Thus, it is necessary to align the language grammar with the hierarchical structures in physical objects. As shown in fig. 10.1, a visual object can be parsed into parts with hierarchical structures, and constituents that describe parts of an object can be naturally grounded with the parts at different hierarchies.

While the study of the hierarchical structure of images has a long history, the structure is mainly pre-defined by human and static across images. Therefore, challenges remain as: (1) how to represent flexible part-whole hierarchies that vary with images using an identical network, and (2) how to learn structure automatically without pre-defined templates. One possible way is to learn the image grammar that parses an object into parts. Instead of allocating neurons to represent nodes in the parse graph, we can use neurons to represent grammar rules. The grammar rules are general for all the images and can be recursively re-used to handle arbitrarily complicated objects (*e.g.*, a chair can have an arbitrary number of legs).

Inspired by the above ideas, we present VLGrammar, a framework that jointly learns image and language grammar. Specifically, we use compound probabilistic context-free grammars (compound PCFGs), which parameterize a PCFG’s rule probabilities with neural networks and relax the context-free constraints with a latent compound variable. To achieve grounded learning, we calculate an alignment score between the image parse tree and the language parse tree, and use a contrastive loss to learn the compound PCFGs for both image and language jointly.

10.2 Grounded Grammar Induction

In this section, we introduce the proposed VLGrammar shown in fig. 10.2 for grounded grammar induction in both vision and language. Our model starts from the compound PCFG for inducing the language

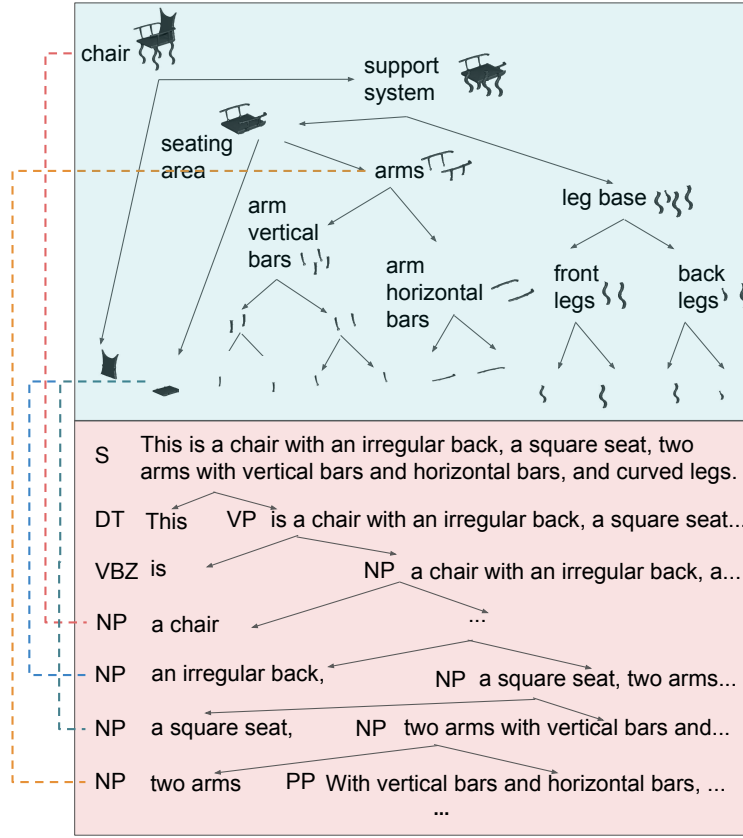


Figure 10.1: **An example of a sentence parse tree aligned with an image parse tree.** The arrow lines represent production rules of the image grammar and the language grammar. The dashed lines represent alignment between the constituents of two modalities.

grammar [76, 180] and generalizes this idea to vision, which are jointly optimized by a contrastive loss.

10.2.1 Compound PCFG for Language

A context-free grammar (CFG) can be defined as a 5-tuple $\mathcal{G} = (S, \mathcal{N}, \mathcal{P}, \Sigma, \mathcal{R})$, where S is the start symbol, \mathcal{N} is a finite set of nonterminal nodes, \mathcal{P} is a finite set of preterminal nodes, Σ is a finite set of terminal nodes, and \mathcal{R} is a set of production rules in the Chomsky normal form:

$$\begin{aligned}
 S &\rightarrow A, & A &\in \mathcal{N} \\
 A &\rightarrow BC, & A &\in \mathcal{N}, B, C \in \mathcal{N} \cup \mathcal{P} \\
 T &\rightarrow w, & T &\in \mathcal{P}, w \in \Sigma
 \end{aligned} \tag{10.1}$$

In natural language, nonterminals \mathcal{N} are constituent labels and preterminals \mathcal{P} are part-of-speech tags. A terminal node w is a word from a sentence, and Σ is the vocabulary. During implementation, we do not have the ground truth constituent labels and part-of-speech tags. Therefore, nonterminals and preterminals are sets of nodes (or clusters) which implicitly represent their functions.

A probabilistic context-free grammar (PCFG) extends a grammar \mathcal{G} with rule probabilities $\pi = \{\pi_r\}_{r \in \mathcal{R}}$, such that the rule r has probability π_r , and $\sum_{r: A \rightarrow \gamma} \pi_r = 1$. Kim et al. [76] propose a neural parameterization where rule probabilities are based on distributed representations. They extend neural PCFGs to com-

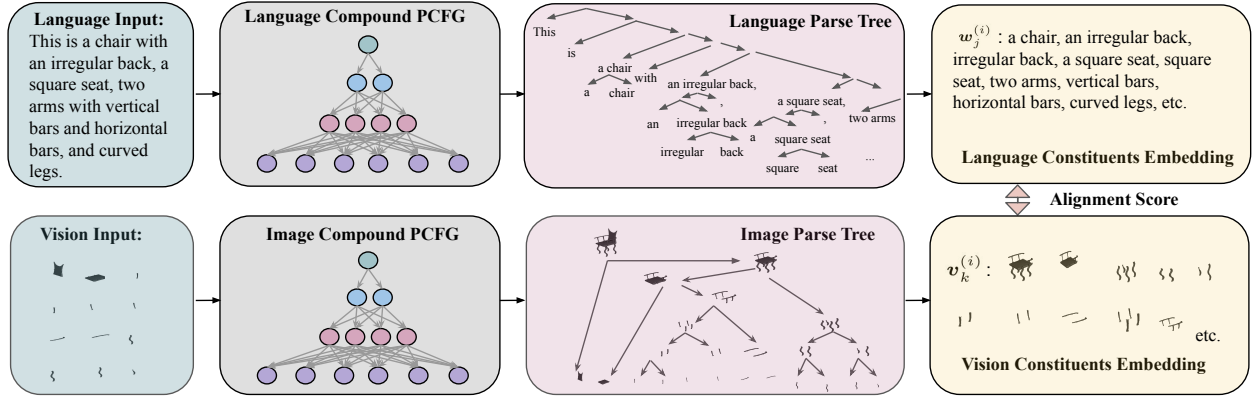


Figure 10.2: **Our proposed VLGrammar framework.** We implement image grammar induction and language grammar induction via compound PCFGs. Parse trees are derived from the grammars. We compute alignment scores between the vision and language constituents in the parse trees to guide the joint learning procedure.

pound PCFGs and mitigate the context-free assumptions by holding it conditioned on compound probability distribution [12]:

$$\mathbf{z} \sim p_{\lambda}(\mathbf{z}), \pi_{\mathbf{z}} = f_r(\mathbf{z}; E_{\mathcal{G}}) \quad (10.2)$$

where $E_{\mathcal{G}} = \mathbf{w}_N$ ($N \in \{S\} \cup \mathcal{N} \cup \mathcal{P}$) are the symbol embeddings. $p_{\lambda}(\mathbf{z})$ is a prior distribution of the latent variable \mathbf{z} with spherical Gaussian λ , and the per-sentence rule probability $\pi_{\mathbf{z}}$ is parameterized by $E_{\mathcal{G}}$ with a neural network f_r . $\pi_{\mathbf{z}}$ takes one of the following forms [76, 180]:

$$\pi_{S \rightarrow A} = \frac{\exp(\mathbf{u}_A^T f_1([\mathbf{w}_S; \mathbf{z}]))}{\sum_{A' \in \mathcal{N}} \exp(\mathbf{u}_{A'}^T f_1([\mathbf{w}_S; \mathbf{z}]))} \quad (10.3)$$

$$\pi_{A \rightarrow BC} = \frac{\exp(\mathbf{u}_{BC}^T [\mathbf{w}_A; \mathbf{z}])}{\sum_{B', C' \in \mathcal{M}} \exp(\mathbf{u}_{B'C'}^T [\mathbf{w}_A; \mathbf{z}])} \quad (10.4)$$

$$\pi_{T \rightarrow w} = \frac{\exp(\mathbf{u}_w^T f_2([\mathbf{w}_T; \mathbf{z}]))}{\sum_{w' \in \Sigma} \exp(\mathbf{u}_{w'}^T f_2([\mathbf{w}_T; \mathbf{z}]))} \quad (10.5)$$

where \mathbf{u} is a parameter vector, \mathcal{M} denotes $(\mathcal{N} \cup \mathcal{P}) \times (\mathcal{N} \cup \mathcal{P})$. $[\cdot; \cdot]$ indicates vector concatenation, and $f_1(\cdot)$ and $f_2(\cdot)$ are feedforward neural networks that encode the inputs.

In the compound PCFG, the log marginal likelihood $\log p_{\theta}(\mathbf{w})$ of the observed sentence $\mathbf{w} = w_1 w_2 \dots w_n$ can be obtained by summing out the latent tree structure using the inside algorithm [6]:

$$\log p_{\theta}(\mathbf{w}) = \log \left(\int_{\mathbf{z}} \sum_{t \in \mathcal{T}_{\mathcal{G}}(\mathbf{w})} p_{\theta}(t | \mathbf{z}) p_{\lambda}(\mathbf{z}) d\mathbf{z} \right) \quad (10.6)$$

where $\mathcal{T}_{\mathcal{G}}$ consists of all parses of the sentence \mathbf{w} under a grammar \mathcal{G} . Since the integral over \mathbf{z} makes this likelihood intractable, Compound PCFGs use amortized variational inference and compute the loss based on the evidence lower bound (ELBO):

$$\begin{aligned} \mathcal{L}_g(\mathbf{w}; \phi, \theta) &= -\text{ELBO}(\mathbf{w}; \phi, \theta) \\ &= -\mathbb{E}_{q_{\phi}(\mathbf{z} | \mathbf{w})} [\log p_{\theta}(\mathbf{w} | \mathbf{z})] + \text{KL}[q_{\phi}(\mathbf{z} | \mathbf{w}) || p_{\lambda}(\mathbf{z})] \end{aligned} \quad (10.7)$$

where $q_{\phi}(\mathbf{z} | \mathbf{w})$ is a variational posterior modeled by a neural network parameterized by ϕ .

10.2.2 Compound PCFG for Image

Compound PCFGs can be naturally extended to image grammar. In a compound PCFG for image, S denotes an object, *e.g.*, a chair. Nonterminals \mathcal{N} are types of middle-level coarse parts. Preterminals \mathcal{P} are types of fine-grained leaf-parts. The middle-level parts can be further decomposed into sub-parts which are either middle-level parts or leaf-parts; for example, the base of a chair is decomposed into the central support and the leg system, and the leg system is further decomposed into several legs.

eq. (10.3) and eq. (10.4) can be directly applied to represent the compound PCFG for image. However, eq. (10.5) does not work for image, since we do not have a fixed vocabulary for images, and terminal nodes are varied *w.r.t* pixels. To address this problem, we design a bottom-up perception module to substitute the top-down generation in eq. (10.5).

10.2.2.1 Bottom-Up Perception

Preterminals T can be viewed as a set of clusters that group the terminal nodes since we do not have ground-truth labels. Therefore, instead of inducing the top-down grammar, we use a bottom-up perception module to propose terminal nodes for T .

We consider the terminal nodes to be a sequence of leaf-parts of an object $\mathbf{v} = v_1 v_2 \dots v_n$. We want to assign a tag T to each leaf part v_i .

$$s(T, v_i) = \mathbf{u}_T^T f_t(\psi(v_i)) \quad (10.8)$$

where ψ is a perception module, *i.e.*, ResNet-18 in our model. f_t is a clustering model, which is a single-layer feed-forward neural network that gives the score of clustering leaf-part v_i to the tag T and \mathbf{u}_T is a parameter vector for the tag T . The rule probability of a preterminal to a leaf-part is thus:

$$\pi_{T \rightarrow v_i} = \frac{\exp(s(T, v_i))}{\sum_{v' \in \Sigma} \exp(s(T, v'))} \quad (10.9)$$

All leaf parts in a training batch constitute Σ .

We maximize the log-likelihood of the part sequence with ELBO:

$$\mathcal{L}_g(\mathbf{v}; \phi, \theta) = -\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{v})} [\log p_\theta(\mathbf{v} | \mathbf{z})] - \text{KL} [q_\phi(\mathbf{z} | \mathbf{v}) \| p_\lambda(\mathbf{z})] \quad (10.10)$$

where $q_\phi(\mathbf{z} | \mathbf{v})$ is a variational posterior.

Note that the image sequence \mathbf{v} is independent of \mathbf{z} given the tags $\mathbf{T} = T_1 T_2 \dots T_n$ of \mathbf{v} . Therefore,

$$\begin{aligned} p_\theta(\mathbf{v} | \mathbf{z}) &= \sum_{\mathbf{T}} p_{\theta_\psi}(\mathbf{v} | \mathbf{T}) p_{\theta_G}(\mathbf{T} | \mathbf{z}) \\ &\propto \sum_{\mathbf{T}} p_{\theta_\psi}(\mathbf{T} | \mathbf{v}) p_{\theta_G}(\mathbf{T} | \mathbf{z}) \end{aligned} \quad (10.11)$$

where we sum over all possible tags for the part. θ_ψ denotes the parameters of the clustering module, and θ_G denotes the parameters of eq. (10.3) and eq. (10.4) in the image grammar.

We notice that if \mathbf{T} has higher probability given by the grammar module, $p_{\theta_G}(\mathbf{T} | \mathbf{z})$ has a larger value, thus gives larger weight for $p_{\theta_\psi}(\mathbf{T} | \mathbf{v})$. This means \mathbf{T} is more likely to be the accurate clusters over the images if it conforms to the current grammar. Therefore, the grammar module can boost the training of the clustering module, and *vice versa*. In practice, a pre-trained clustering module can speed up the training.

10.2.3 Joint Learning by Alignment

We propose to jointly learn the grammars for image and language by aligning the paired image and sentence. Similar to [135] and [180], we use an end-to-end contrastive learning framework. While they align each language constituent with a single image, we compute an alignment score between each language constituent and each visual constituent.

Given a sentence $\mathbf{w} = w_1 \dots w_m$ where m is the total number of words, a language constituent is defined as a span over this sentence, denoted as $\mathbf{w}_j = w_a \dots w_b \in [\mathbf{w}]$ where $0 < a < b \leq m$ and $[\mathbf{w}]$ denotes the set of all possible spans over \mathbf{w} . We use a Bi-LSTM to obtain the embedding of a language constituent:

$$\mathbf{w}_j = f_w \left(\frac{1}{b-a+1} \sum_{l=a}^b \mathbf{h}_l \right) \quad (10.12)$$

where \mathbf{h}_l is the hidden state of the Bi-LSTM, and f_w is an affine transformation. We average the label-specific representations like in [180].

Given an object $\mathbf{v} = v_1 \dots v_n$ where n is the total number of parts, a visual constituent is defined as a span over this part sequence, denoted as $\mathbf{v}_k = v_c \dots v_d \in [\mathbf{v}]$ where $0 < c < d \leq n$ and $[\mathbf{v}]$ denotes the set of all possible sub-parts over \mathbf{v} . We define the embedding of a visual constituent as:

$$\mathbf{v}_k = f_v \left(\frac{1}{d-c+1} \sum_{l=c}^d \psi(v_l) \right) \quad (10.13)$$

where ψ is the perception module from eq. (10.8) and f_v is an affine transformation.

The alignment score between a language constituent and a visual constituent is defined as their cosine similarity:

$$s(\mathbf{w}_j, \mathbf{v}_k) \triangleq \cos(\mathbf{w}_j, \mathbf{v}_k) \quad (10.14)$$

The alignment score between a sentence and an image is:

$$\begin{aligned} \mathcal{S}(\mathbf{w}, \mathbf{v}) &= \sum_{\substack{t_w \in \mathcal{T}_{\mathcal{G}_w}(\mathbf{w}) \\ t_v \in \mathcal{T}_{\mathcal{G}_v}(\mathbf{v})}} p(t_w|\mathbf{w})p(t_v|\mathbf{v}) \sum_{\substack{\mathbf{w}_j \in t_w \\ \mathbf{v}_k \in t_v}} s(\mathbf{w}_j, \mathbf{v}_k) \\ &= \sum_{\substack{\mathbf{w}_j \in [\mathbf{w}] \\ \mathbf{v}_k \in [\mathbf{v}]}} \sum_{\substack{t_w \in \mathcal{T}_{\mathcal{G}_w}(\mathbf{w}) \\ t_v \in \mathcal{T}_{\mathcal{G}_v}(\mathbf{v})}} \mathbb{1}_{\{\mathbf{w}_j \in t_w\}} \mathbb{1}_{\{\mathbf{v}_k \in t_v\}} p(t_w|\mathbf{w})p(t_v|\mathbf{v}) s(\mathbf{w}_j, \mathbf{v}_k) \\ &= \sum_{\substack{\mathbf{w}_j \in [\mathbf{w}] \\ \mathbf{v}_k \in [\mathbf{v}]}} p(\mathbf{w}_j|\mathbf{w}; \mathcal{G}_w) p(\mathbf{v}_k|\mathbf{v}; \mathcal{G}_v) s(\mathbf{w}_j, \mathbf{v}_k) \end{aligned} \quad (10.15)$$

where $p(\mathbf{w}_j|\mathbf{w}; \mathcal{G}_w) = \sum_{t_w \in \mathcal{T}_{\mathcal{G}_w}(\mathbf{w})} \mathbb{1}_{\{\mathbf{w}_j \in t_w\}} p(t_w|\mathbf{w})$ and $p(\mathbf{v}_k|\mathbf{v}; \mathcal{G}_v) = \sum_{t_v \in \mathcal{T}_{\mathcal{G}_v}(\mathbf{v})} \mathbb{1}_{\{\mathbf{v}_k \in t_v\}} p(t_v|\mathbf{v})$ are the conditional probabilities of a constituent given the sentence/object, marginalized over all possible parse trees under the current grammars. They can be efficiently computed with the inside algorithm and automatic differentiation [32].

Given a training batch $\mathcal{D} = \{\mathcal{W}, \mathcal{V}\} = \{(\mathbf{w}^{(i)}, \mathbf{v}^{(i)})\}$, the contrastive loss is defined as:

$$\begin{aligned} \mathcal{L}_C(\mathcal{W}, \mathcal{V}) &= \sum_{i, m \neq i} [\mathcal{S}(\mathbf{w}^{(m)}, \mathbf{v}^{(i)}) - \mathcal{S}(\mathbf{w}^{(i)}, \mathbf{v}^{(i)}) + \delta]_+ \\ &\quad + \sum_{i, m \neq i} [\mathcal{S}(\mathbf{w}^{(i)}, \mathbf{v}^{(m)}) - \mathcal{S}(\mathbf{w}^{(i)}, \mathbf{v}^{(i)}) + \delta]_+ \end{aligned} \quad (10.16)$$

where δ is a constant margin, and $[\cdot]_+$ denotes $\max(0, \cdot)$.

The overall training loss function is then:

$$\mathcal{L} = \lambda_w \mathcal{L}_{\mathcal{G}}(\mathcal{W}; \phi_w, \theta_w) + \lambda_v \mathcal{L}_{\mathcal{G}}(\mathcal{V}; \phi_v, \theta_v) + \lambda_C \mathcal{L}_C(\mathcal{W}, \mathcal{V}) \quad (10.17)$$

where $\lambda_w, \lambda_v, \lambda_C$ are hyperparameters, and $\phi_t, \theta_t, \phi_v, \theta_v$ denote the parameters of the language and visual compound PCFGs, respectively.

10.2.4 Qualitative Study

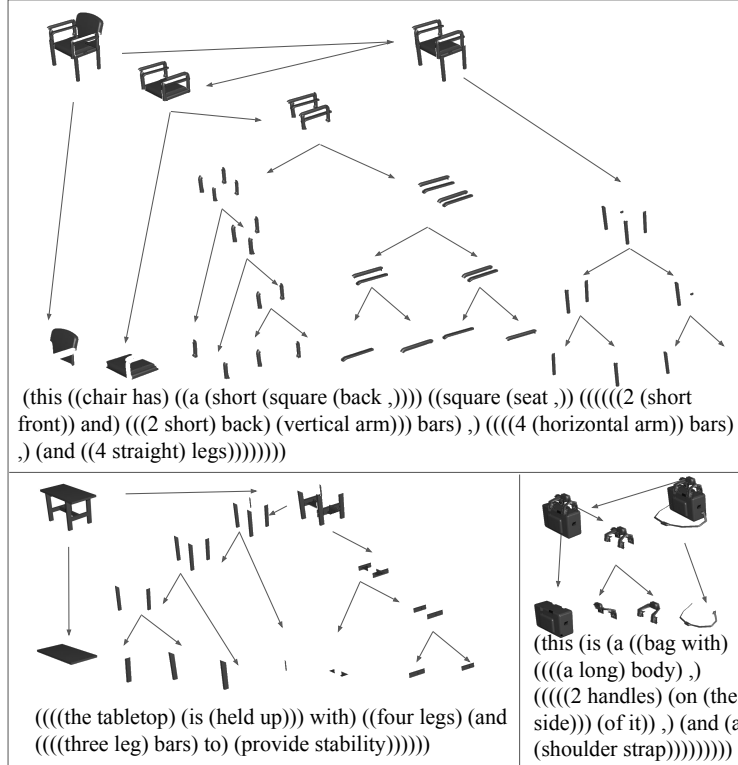


Figure 10.3: **Qualitative examples of parse trees predicted by VLGrammar.** We visualize the image parse trees and the language parse trees derived by the VLGrammar. Since the language parse trees are large, we use a bracket form to represent them.

fig. 10.3 visualizes several examples of parse trees predicted by VLGrammar. We summarize following observations from these examples:

Part-Whole Hierarchies The VLGrammar can capture precise part-whole hierarchies of the images. The objects can be parsed into parts of various hierarchies. For instance, the chair can be parsed into the upper part and supporting system. The latter can be further divided into seating area and chair base. The seating area has a chair seat and arms with vertical bars and horizontal bars, which are grouped separately. In the table, the base can be divided into legs and leg bars.

Recursive Structures One interesting question is how VLGrammar deals with recursive structures. A chair can have an arbitrary number of legs, which shall be in the the same hierarchy. However, since context-free grammar is defined on binary trees, recursive grammar is used to group the parts of same functionality. We find that VLGrammar can at least learn three types of recursive structures: (1) Pair-wise grouping: VLGrammar first groups the parts into pairs according to positional information (*e.g.*, front legs and back

legs, left horizontal arm bars and right horizontal arm bars, as shown in fig. 10.3), and then group the pairs. (2) Right-branching: the vertical bars of the chair arms are grouped using right-branching binary trees. (3) Left-branching: *e.g.*, the grouping of the legs of the table. Right-branching and left-branching are effective when dealing with an arbitrary number of parts at the same level, and when there are no salient patterns to pair them. One example is the star leg base, where the legs are in arbitrary order and form a circle.

Language Phrases VLGrammar excels at grouping phrases that refer to parts in the images. For example, VLGrammar can capture phrases such as “a short square back”, “four legs”, “three leg bars”, “2 handles”, “a shoulder strap”, and so on. This merit comes from the learned alignment between the phrases and the referred visual parts.

Bibliography

- [1] Malcolm Acock. Vision: A computational investigation into the human representation and processing of visual information. by david marr. *The Modern Schoolman*, 62(2):141–142, 1985.
- [2] Yali Amit and Alain Trouvé. Pop: Patchwork of parts models for object recognition. *International Journal of Computer Vision (IJCV)*, 75(2):267–282, 2007.
- [3] Krishna B Athreya and AN Vidyashankar. Branching processes. *Handbook of statistics*, 19:35–53, 2001.
- [4] Shai Avidan. Spatialboost: Adding spatial reasoning to adaboost. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2006.
- [5] Meg Aycinena, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Learning grammatical models for object recognition. 2008.
- [6] J. Baker. Trainable grammars for speech recognition. *Journal of the Acoustical Society of America*, 65, 1979.
- [7] Adrian Barbu and Song-Chun Zhu. Generalizing swendsen-wang to sampling arbitrary posterior probabilities. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 27(8):1239–1253, 2005.
- [8] Irving Biederman. Recognition-by-components: A theory of human image understanding. *Psychological review*, 94(2):115, 1987.
- [9] Eran Borenstein and Shimon Ullman. Combined top-down/bottom-up segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 30(12):2109–2125, 2008.
- [10] Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Describing people: A poselet-based approach to attribute classification. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2011.
- [11] R Burkard, M DellAmico, and S Martello. Society for industrial and applied mathematics, assignment problems. philadelphia (pa.): Siam. *Society for Industrial and Applied Mathematics*, 6, 2009.
- [12] L. L. Cam. Asymptotic methods in statistical decision theory. 1986.
- [13] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [14] Bruno Caprile and Vincent Torre. Using vanishing points for camera calibration. *International Journal of Computer Vision (IJCV)*, 4(2):127–139, 1990.

- [15] Sharon A Caraballo and Eugene Charniak. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298, 1998.
- [16] Hong Chen, Zi Jian Xu, Zi Qiang Liu, and Song-Chun Zhu. Composite templates for cloth modeling and sketching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.
- [17] Huizhong Chen, Andrew Gallagher, and Bernd Girod. Describing clothing by semantic attributes. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2012.
- [18] Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159, 2001.
- [19] Yixin Chen, Siyuan Huang, Tao Yuan, Siyuan Qi, Yixin Zhu, and Song-Chun Zhu. Holistic++ scene understanding: Single-view 3d holistic scene parsing and human pose estimation with human-object interaction and physical commonsense. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2019.
- [20] Zhiyi Chi and Stuart Geman. Estimation of probabilistic context-free grammars. *Computational linguistics*, 24(2):299–305, 1998.
- [21] John Cocke. *Programming languages and their compilers: Preliminary notes*. New York University, 1969.
- [22] James M Coughlan and Alan L Yuille. Manhattan world: Orientation and outlier detection by bayesian inference. *Neural computation*, 15(5):1063–1088, 2003.
- [23] Gergely Csibra and György Gergely. ‘obsessed with goals’: Functions and mechanisms of teleological interpretation of actions in humans. *Acta psychologica*, 124(1):60–78, 2007.
- [24] Imre Csiszár and Paul C Shields. Information theory and statistics: A tutorial. 2004.
- [25] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.
- [26] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.
- [27] Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 19(4):380–393, 1997.
- [28] M Fatih Demirci, Bram Platel, Ali Shokoufandeh, Luc LMJ Florack, and Sven J Dickinson. The representation and matching of images using top points. *Journal of Mathematical Imaging and Vision*, 35(2):103–116, 2009.
- [29] M Fatih Demirci, Ali Shokoufandeh, Yakov Keselman, Lars Bretzner, and Sven Dickinson. Object recognition as many-to-many feature matching. *International Journal of Computer Vision (IJCV)*, 69(2):203–222, 2006.
- [30] Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.
- [31] Paul HC Eilers and Brian D Marx. Flexible smoothing with b-splines and penalties. *Statistical science*, 11(2):89–121, 1996.

- [32] Jason Eisner. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *SPNLP@EMNLP*, 2016.
- [33] Boris Epshtein, Ita Lifshitz, and Shimon Ullman. Image interpretation by a single bottom-up top-down cycle. *Proceedings of the National Academy of Sciences (PNAS)*, 105(38):14298–14303, 2008.
- [34] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision (IJCV)*, 111(1):98–136, 2015.
- [35] Lifeng Fan, Yixin Chen, Ping Wei, Wenguan Wang, and Song-Chun Zhu. Inferring shared attention in social scene videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [36] Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 28(4):594–611, 2006.
- [37] Thomas Feix, Javier Romero, Heinz-Bodo Schmiedmayer, Aaron M Dollar, and Danica Kragic. The grasp taxonomy of human grasp types. *IEEE Transactions on human-machine systems*, 46(1):66–77, 2015.
- [38] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(9):1627–1645, 2009.
- [39] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision (IJCV)*, 59(2):167–181, 2004.
- [40] Pedro F Felzenszwalb and Daniel P Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision (IJCV)*, 61(1):55–79, 2005.
- [41] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient belief propagation for early vision. *International Journal of Computer Vision (IJCV)*, 70(1):41–54, 2006.
- [42] Pedro F Felzenszwalb and David McAllester. Object detection grammars. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2011.
- [43] Robert Fergus, Pietro Perona, and Andrew Zisserman. Weakly supervised scale-invariant learning of models for visual recognition. *International Journal of Computer Vision (IJCV)*, 71(3):273–303, 2007.
- [44] Michael Fink and Pietro Perona. Mutual boosting for contextual inference. 2004.
- [45] Amy Fire and Song-Chun Zhu. Using causal induction in humans to learn and infer causality from video. In *Proceedings of the Annual Meeting of the Cognitive Science Society (CogSci)*, 2013.
- [46] Amy Fire and Song-Chun Zhu. Inferring hidden statuses and actions in video by causal reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [47] Martin A Fischler and Robert C Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

- [48] Arthur Fridman. Mixed markov models. *Proceedings of the National Academy of Sciences (PNAS)*, 100(14):8092–8096, 2003.
- [49] King Sun Fu. Introduction to syntactic pattern recognition. In *Syntactic pattern recognition, applications*, pages 1–30. Springer, 1977.
- [50] Meirav Galun, Eitan Sharon, Ronen Basri, and Achi Brandt. Texture segmentation by multiscale aggregation of filter responses and shape elements. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2003.
- [51] Ru-Xin Gao, Tian-Fu Wu, Song-Chun Zhu, and Nong Sang. Bayesian inference for layer representation with mixed markov random field. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.
- [52] Weina Ge, Robert T Collins, and R Barry Ruback. Vision-based analysis of small groups in pedestrian crowds. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 34(5):1003–1016, 2012.
- [53] Stuart Geman and Mark Johnson. Probability and statistics in computational linguistics, a brief review. *Mathematical foundations of speech and language processing*, pages 1–26, 2004.
- [54] Dileep George, Wolfgang Lehrach, Ken Kansky, Miguel Lázaro-Gredilla, Christopher Laan, Bhaskara Marthi, Xinghua Lou, Zhaoshi Meng, Yi Liu, Huayan Wang, et al. A generative vision model that trains with high data efficiency and breaks text-based captchas. *Science*, 358(6368), 2017.
- [55] Thomas L Griffiths and Joshua B Tenenbaum. Structure and strength in causal induction. *Cognitive psychology*, 51(4):334–384, 2005.
- [56] Cheng-En Guo, Song-Chun Zhu, and Ying Nian Wu. Modeling visual patterns by integrating descriptive and generative methods. *International Journal of Computer Vision (IJCV)*, 53(1):5–29, 2003.
- [57] Cheng-en Guo, Song-Chun Zhu, and Ying Nian Wu. Primal sketch: Integrating structure and texture. *Computer Vision and Image Understanding (CVIU)*, 106(1):5–19, 2007.
- [58] Abhinav Gupta, Alexei A Efros, and Martial Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2010.
- [59] Abhinav Gupta, Praveen Srinivasan, Jianbo Shi, and Larry S Davis. Understanding videos, constructing plots learning a visually grounded storyline model from annotated videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.
- [60] York Hagmayer and Michael R Waldmann. How temporal assumptions influence causal judgments. *Memory & Cognition*, 30(7):1128–1137, 2002.
- [61] Feng Han and Song-Chun Zhu. Bottom-up/top-down image parsing by attribute graph grammar. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2005.
- [62] Feng Han and Song-Chun Zhu. Bottom-up/top-down image parsing with attribute grammar. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 31(1):59–73, 2008.
- [63] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

- [64] Bernd Heisele, Thomas Serre, and Tomaso Poggio. A component-based framework for face detection and identification. *International Journal of Computer Vision (IJCV)*, 74(2):167–181, 2007.
- [65] Derek Hoiem, Alexei A Efros, and Martial Hebert. Closing the loop in scene interpretation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.
- [66] Derek Hoiem, Alexei A Efros, and Martial Hebert. Putting objects in perspective. *International Journal of Computer Vision (IJCV)*, 80(1):3–15, 2008.
- [67] Tsai-Hong Hong and Azriel Rosenfeld. Compact region extraction using weighted pixel linking in a pyramid. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, (2):222–229, 1984.
- [68] Chang Huang, Yuan Li, and Ramakant Nevatia. Multiple target tracking by learning-based hierarchical association of detection responses. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 35(4):898–910, 2012.
- [69] Siyuan Huang, Siyuan Qi, Yinxue Xiao, Yixin Zhu, Ying Nian Wu, and Song-Chun Zhu. Cooperative holistic scene understanding: Unifying 3d object, layout, and camera pose estimation. *arXiv preprint arXiv:1810.13049*, 2018.
- [70] Yuri A. Ivanov and Aaron F. Bobick. Recognition of visual activities and interactions by stochastic parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 22(8):852–872, 2000.
- [71] Ya Jin and Stuart Geman. Context and hierarchy in a probabilistic image model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.
- [72] Bela Julesz. Textons, the elements of texture perception, and their interactions. *Nature*, 290(5802):91–97, 1981.
- [73] Gaetano Kanizsa. *Organization in vision: Essays on Gestalt perception*. Praeger Publishers, 1979.
- [74] Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*, 1966.
- [75] Daniel Keenan. General pattern theory: A mathematical study of regular structures (u. grenander). *SIAM Review*, 37(2):258–261, 1995.
- [76] Yoon Kim, Chris Dyer, and Alexander M. Rush. Compound probabilistic context-free grammars for grammar induction. *ArXiv*, abs/1906.10225, 2019.
- [77] Donald Ervin Knuth. *The art of computer programming*, volume 3. Pearson Education, 1997.
- [78] Jana Košecká and Wei Zhang. Extraction, matching, and pose recovery based on dominant rectangular structures. *Computer Vision and Image Understanding (CVIU)*, 100(3):274–293, 2005.
- [79] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Detection-based object labeling in 3d scenes. In *Proceedings of International Conference on Robotics and Automation (ICRA)*, 2012.
- [80] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

- [81] Michael Land, Neil Mennie, and Jennifer Rusted. The roles of vision and eye movements in the control of activities of daily living. *Perception*, 28(11):1311–1328, 1999.
- [82] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998.
- [83] Tai Sing Lee and David Mumford. Hierarchical bayesian inference in the visual cortex. *JOSA A*, 20(7):1434–1448, 2003.
- [84] Robert B Lees. Syntactic structures, 1957.
- [85] Anat Levin and Yair Weiss. Learning to combine bottom-up and top-down segmentation. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2006.
- [86] Liang Lin, Haifeng Gong, Li Li, and Liang Wang. Semantic event representation and recognition using syntactic attribute graph grammar. *Pattern Recognition Letters*, 30(2):180–186, 2009.
- [87] Liang Lin, Tianfu Wu, Jake Porway, and Zijian Xu. A stochastic graph grammar for compositional object representation and recognition. *Pattern Recognition*, 42(7):1297–1307, 2009.
- [88] Tony Lindeberg. *Scale-space theory in computer vision*, volume 256. Springer Science & Business Media, 2013.
- [89] Ce Liu, Jenny Yuen, and Antonio Torralba. Nonparametric scene parsing via label transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 33(12):2368–2382, 2011.
- [90] Tengyu Liu. *Hierarchical modeling of human-object interactions: From concurrent action parsing to physics-based grasping*. PhD thesis, University of California, Los Angeles, 2021.
- [91] Xiaobai Liu, Yibiao Zhao, and Song-Chun Zhu. Single-view 3d scene parsing by attributed grammar. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [92] John L Mackie. Causes and conditions. *American philosophical quarterly*, 2(4):245–264, 1965.
- [93] Sara C Madeira and Arlindo L Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM transactions on computational biology and bioinformatics*, 1(1):24–45, 2004.
- [94] Stéphane G Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on signal processing*, 41(12):3397–3415, 1993.
- [95] Kevin E Mark, Michael I Miller, and Ulf Grenander. Constrained stochastic language models. In *Image Models (and Their Speech Model cousins)*, pages 131–140. Springer, 1996.
- [96] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.
- [97] Detmar Meurers. Lecture notes in introduction to computational linguistics, January 2006.
- [98] Erik T Mueller. *Commonsense reasoning: An event calculus based approach*. Morgan Kaufmann, 2014.
- [99] David Mumford and Agnès Desolneux. *Pattern theory: the stochastic analysis of real-world signals*. CRC Press, 2010.

- [100] Kevin P Murphy. Hidden semi-markov models (hsmms). *unpublished notes*, 2, 2002.
- [101] Isaac Newton. *The Method of Fluxions and Infinite Series: With Its Application to the Geometry of Curve Lines*. Nourse, 1736.
- [102] Mark Nitzberg, David Mumford, and Takahiro Shiota. *Filtering, segmentation and depth*. 1993.
- [103] Mark Nitzberg and David Bryant Mumford. *The 2.1-D sketch*. IEEE Computer Society Press, 1990.
- [104] Bruno A Olshausen and David J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.
- [105] Jukka Paakki. Attribute grammar paradigms—a high-level methodology in language implementation. *ACM Computing Surveys (CSUR)*, 27(2):196–255, 1995.
- [106] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deep sdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [107] Seyoung Park, Bruce Xiaohan Nie, and Song-Chun Zhu. Attribute and-or grammar for joint parsing of human pose, parts and attributes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 40(7):1555–1569, 2017.
- [108] Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.
- [109] Judea Pearl et al. Models, reasoning and inference. *Cambridge, UK: Cambridge University Press*, 19, 2000.
- [110] Mingtao Pei, Yunde Jia, and Song-Chun Zhu. Parsing video events with goal inference and intent prediction. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2011.
- [111] Mingtao Pei, Zhangzhang Si, Benjamin Z Yao, and Song-Chun Zhu. Learning and parsing video events with goal and intent prediction. *Computer Vision and Image Understanding (CVIU)*, 117(10):1369–1383, 2013.
- [112] Hamed Pirsiavash and Deva Ramanan. Parsing videos of actions with segmental grammars. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [113] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.
- [114] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2011.
- [115] Jake Porway, Benjamin Yao, and Song Chun Zhu. Learning compositional models for object categories from small sample sets. *Object categorization: computer and human vision perspectives*, pages 1–17, 2008.
- [116] Jake Porway, ZY Yao, and Song-Chun Zhu. Learning an and-or graph for modeling and recognizing object categories. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (1892):67–92, 2007.

- [117] Alessandro Prest, Cordelia Schmid, and Vittorio Ferrari. Weakly supervised learning of interactions between humans and objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 34(3):601–614, 2011.
- [118] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [119] Hang Qi, Tianfu Wu, Mun-Wai Lee, and Song-Chun Zhu. A restricted visual turing test for deep scene and event understanding. *arXiv preprint arXiv:1512.01715*, 2015.
- [120] Hang Qi, Yuanlu Xu, Tao Yuan, Tianfu Wu, and Song-Chun Zhu. Scene-centric joint parsing of cross-view videos. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2018.
- [121] Siyuan Qi, Baoxiong Jia, Siyuan Huang, Ping Wei, and Song-Chun Zhu. A generalized earley parser for human activity parsing and prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020.
- [122] Siyuan Qi, Baoxiong Jia, and Song-Chun Zhu. Generalized earley parser: Bridging symbolic grammars and sequence data for future prediction. In *Proceedings of International Conference on Machine Learning (ICML)*, 2018.
- [123] Siyuan Qi, Yixin Zhu, Siyuan Huang, Chenfanfu Jiang, and Song-Chun Zhu. Human-centric indoor scene synthesis using stochastic grammar. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [124] Jan Rekers and A Schürr. *A parsing algorithm for context-sensitive graph grammars*. Rijksuniversiteit Leiden. Vakgroep Informatica, 1995.
- [125] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 28:91–99, 2015.
- [126] Xiaofeng Ren and Jitendra Malik. Learning a classification model for segmentation. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2003.
- [127] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025, 1999.
- [128] Brandon Rothrock, Seyoung Park, and Song-Chun Zhu. Integrating grammar and segmentation for human pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.
- [129] Donald B Rubin. Causal inference using potential outcomes: Design, modeling, decisions. *Journal of the American Statistical Association*, 100(469):322–331, 2005.
- [130] Rebecca Saxe, JB Tenenbaum, and Susan Carey. Secret agents: Inferences about hidden causes by 10-and 12-month-old infants. *Psychological Science*, 16(12):995–1001, 2005.
- [131] Richard Scheines, Herbert Hoijtink, and Anne Boomsma. Bayesian estimation and testing of structural equation models. *Psychometrika*, 64(1):37–52, 1999.

- [132] Henry Schneiderman and Takeo Kanade. Object detection using the statistics of parts. *International Journal of Computer Vision (IJCV)*, 56(3):151–177, 2004.
- [133] G. Schwarz. Estimating the dimension of a model. *Ann. Statist.*, 6(2):464–464, 1978.
- [134] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 29(3):411–426, 2007.
- [135] Haoyue Shi, Jiayuan Mao, Kevin Gimpel, and Karen Livescu. Visually grounded neural syntax acquisition. *ArXiv*, abs/1906.02890, 2019.
- [136] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.
- [137] Tianmin Shu, Dan Xie, Brandon Rothrock, Sinisa Todorovic, and Song Chun Zhu. Joint inference of groups, events and human roles in aerial videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [138] Zhangzhang Si, Haifeng Gong, Ying Nian Wu, and Song-Chun Zhu. Learning mixed templates for object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.
- [139] Zhangzhang Si, Mingtao Pei, Benjamin Yao, and Song-Chun Zhu. Unsupervised learning of event and-or grammar and semantics from video. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2011.
- [140] Zhangzhang Si and Song-Chun Zhu. Learning hybrid image templates (hit) by information projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 34(7):1354–1367, 2011.
- [141] Zhangzhang Si and Song-Chun Zhu. Learning and-or templates for object recognition and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 35(9):2189–2205, 2013.
- [142] Zach Solan, David Horn, Eytan Ruppin, and Shimon Edelman. Unsupervised learning of natural languages. *Proceedings of the National Academy of Sciences (PNAS)*, 102(33):11629–11634, 2005.
- [143] Valentin I Spitkovsky, Hiyun Alshawi, Dan Jurafsky, and Christopher D Manning. Viterbi training improves unsupervised dependency parsing. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, 2010.
- [144] Julian Straub, Guy Rosman, Oren Freifeld, John J Leonard, and John W Fisher. A mixture of manhattan frames: Beyond the manhattan world. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [145] Joseph Tighe and Svetlana Lazebnik. Superparsing: Scalable nonparametric image parsing with superpixels. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2010.
- [146] Sinisa Todorovic and Narendra Ahuja. Region-based hierarchical image matching. *International Journal of Computer Vision (IJCV)*, 78(1):47–66, 2008.
- [147] Denis Tome, Chris Russell, and Lourdes Agapito. Lifting from the deep: Convolutional 3d pose estimation from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

- [148] Antonio Torralba. Contextual priming for object detection. *International Journal of Computer Vision (IJCV)*, 53(2):169–191, 2003.
- [149] Antonio Torralba, Kevin P Murphy, and William T Freeman. Sharing visual features for multiclass and multiview object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 29(5):854–869, 2007.
- [150] Kewei Tu and Vasant Honavar. Unsupervised learning of probabilistic context-free grammar using iterative biclustering. In *International Colloquium on Grammatical Inference*, 2008.
- [151] Kewei Tu and Vasant Honavar. Unambiguity regularization for unsupervised learning of probabilistic grammars. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012.
- [152] Kewei Tu, Meng Meng, Mun Wai Lee, Tae Eun Choe, and Song-Chun Zhu. Joint video and text parsing for understanding events and answering queries. *IEEE MultiMedia*, 21(2):42–70, 2014.
- [153] Kewei Tu, Maria Pavlovskaya, and Song-Chun Zhu. Unsupervised structure learning of stochastic and-or grammars. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2013.
- [154] Zhuowen Tu, Xiangrong Chen, Alan L Yuille, and Song-Chun Zhu. Image parsing: Unifying segmentation, detection, and recognition. *International Journal of Computer Vision (IJCV)*, 63(2):113–140, 2005.
- [155] Zhuowen Tu and Song-Chun Zhu. Image segmentation by data-driven markov chain monte carlo. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 24(5):657–673, 2002.
- [156] Shimon Ullman, Erez Sali, and Michel Vidal-Naquet. A fragment-based approach to object representation and classification. In *International workshop on visual form*, 2001.
- [157] Shimon Ullman, Michel Vidal-Naquet, and Erez Sali. Visual features of intermediate complexity and their use in classification. *Nature neuroscience*, 5(7):682–687, 2002.
- [158] Paul Viola and Michael J Jones. Robust real-time face detection. *International Journal of Computer Vision (IJCV)*, 57(2):137–154, 2004.
- [159] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.
- [160] Bing Wang, Gang Wang, Kap Luk Chan, and Li Wang. Tracklet association with online target-specific metric learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [161] Shuo Wang, Yizhou Wang, and Song-Chun Zhu. Learning hierarchical space tiling for scene modeling, parsing and attribute tagging. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 37(12):2478–2491, 2015.
- [162] Yizhou Wang and Song-Chun Zhu. Perceptual scale-space and its applications. *International Journal of Computer Vision (IJCV)*, 80(1):143–165, 2008.
- [163] Ping Wei, Yibiao Zhao, Nanning Zheng, and Song-Chun Zhu. Modeling 4d human-object interactions for event and object recognition. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2013.

- [164] Ping Wei, Yibiao Zhao, Nanning Zheng, and Song-Chun Zhu. Modeling 4d human-object interactions for joint event segmentation, recognition, and object localization. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 39(6):1165–1179, 2016.
- [165] Ping Wei, Nanning Zheng, Yibiao Zhao, and Song-Chun Zhu. Concurrent action detection with structural prediction. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2013.
- [166] Andrew Witkin. Scale-space filtering: A new approach to multi-scale description. In *ICASSP’84. IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1984.
- [167] Tian-Fu Wu, Gui-Song Xia, and Song-Chun Zhu. Compositional boosting for computing hierarchical image structures. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.
- [168] Tianfu Wu and Song-Chun Zhu. A numerical study of the bottom-up and top-down inference processes in and-or graphs. *International Journal of Computer Vision (IJCV)*, 93(2):226–252, 2011.
- [169] Ying Nian Wu, Cheng-En Guo, and Song-Chun Zhu. From information scaling of natural images to regimes of statistical models. *Quarterly of Applied Mathematics*, pages 81–122, 2008.
- [170] Ying Nian Wu, Zhangzhang Si, Haifeng Gong, and Song-Chun Zhu. Learning active basis model for object detection and recognition. *International Journal of Computer Vision (IJCV)*, 90(2):198–235, 2010.
- [171] Ying Nian Wu, Jianwen Xie, Yang Lu, and Song-Chun Zhu. Sparse and deep generalizations of the frame model. *Annals of Mathematical Sciences and Applications*, 3(1):211–254, 2018.
- [172] Yuanlu Xu, Xiaobai Liu, Lei Qin, and Song-Chun Zhu. Cross-view people tracking by scene-centered spatio-temporal parsing. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2017.
- [173] Yuanlu Xu, Bingpeng Ma, Rui Huang, and Liang Lin. Person search in a scene by jointly modeling people commonness and person uniqueness. In *Proceedings of the 22nd ACM international conference on Multimedia*, 2014.
- [174] Zijian Xu, Hong Chen, and Song-Chun Zhu. A high resolution grammatical model for face representation and sketching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.
- [175] Bangpeng Yao and Li Fei-Fei. Recognizing human-object interactions in still images by modeling the mutual context of objects and human poses. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 34(9):1691–1703, 2012.
- [176] Benjamin Yao, Xiong Yang, and Song-Chun Zhu. Introduction to a large-scale general purpose ground truth database: Methodology, annotation tool and benchmarks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.
- [177] Benjamin Z Yao, Xiong Yang, Liang Lin, Mun Wai Lee, and Song-Chun Zhu. I2t: Image parsing to text description. *Proceedings of the IEEE*, 98(8):1485–1508, 2010.
- [178] Daniel H Younger. Recognition and parsing of context-free languages in time n^3 . *Information and control*, 10(2):189–208, 1967.

- [179] Ning Zhang, Ryan Farrell, Forrest Iandola, and Trevor Darrell. Deformable part descriptors for fine-grained recognition and attribute prediction. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2013.
- [180] Yanpeng Zhao and Ivan Titov. Visually grounded compound pcfgs. *ArXiv*, abs/2009.12404, 2020.
- [181] Yibiao Zhao and Song-Chun Zhu. Image parsing with stochastic scene grammar. *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 24:73–81, 2011.
- [182] Song-Chun Zhu. Embedding gestalt laws in markov random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 21(11):1170–1187, 1999.
- [183] Song-Chun Zhu. Statistical modeling and conceptualization of visual patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 25(6):691–712, 2003.
- [184] Song-Chun Zhu and David Mumford. Prior learning and gibbs reaction-diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 19(11):1236–1250, 1997.
- [185] Song-Chun Zhu and David Mumford. *A stochastic grammar of images*. Now Publishers Inc, 2007.
- [186] Song-Chun Zhu, Ying Nian Wu, and David Mumford. Minimax entropy principle and its application to texture modeling. *Neural computation*, 9(8):1627–1660, 1997.
- [187] Yixin Zhu, Tao Gao, Lifeng Fan, Siyuan Huang, Mark Edmonds, Hangxin Liu, Feng Gao, Chi Zhang, Siyuan Qi, Ying Nian Wu, et al. Dark, beyond deep: A paradigm shift to cognitive ai with humanlike common sense. *Engineering*, 6(3):310–345, 2020.