

Condensation of Boolean Formulas

Youichi Hanatani, Takashi Horiyama, and Kazuo Iwama

Graduate School of Informatics, Kyoto University,

Kyoto 606-8501, Japan.

E-mail: {hanatani,horiyama,iwama}@i.kyoto-u.ac.jp

Backgrounds and Motivations. Conventional complexity theory gives us only asymptotic information and does not give us any information about the complexity of each individual instance. It is also true, however, that many of us are feeling that the complexity of each instance is quite different from one to another. *Instance complexity*, denoted by $ic(x : A)$, has been thus introduced [7, 9] as a measure of the complexity of an individual instance x for a decision problem A . $ic(x : A)$ is defined as the length of the shortest program that gives the correct answer to the query “ $x \in A$?” and that does not make any mistake for other inputs (although “don’t know” answers are permitted). It is closely related with (at least, upper bounded by) Kolmogorov complexity, which is the length of the shortest program that computes x from the empty input.

Under this new measure, each element in A can have a different instance complexity; some are easy and some are hard. Now it is obviously desirable if we can convert a hard instance into an easy one by reducing its instance complexity. More concretely, this can be done by designing a mapping (algorithm) δ such that for each instance x , (i) $\delta(x) \in A$ iff $x \in A$ and (ii) $ic(\delta(x) : A) < ic(x : A)$. Note that determining the answer (yes/no) of an instance x is a special case of a complexity reduction, i.e., the *complete* reduction which converts x into a trivial instance whose answer is instantly known. Thus a (*partial*) complexity reduction should be useful if it can be done in a significantly smaller amount of time compared to its complete version. A little surprisingly, however, we have virtually no literature on this problem. The reason is probably this: Considering the (very abstract) nature of the above definition of instance complexity, it appears to be quite difficult to design nontrivial algorithms for its reduction.

Our Contribution. In this paper, we focus on the Boolean satisfiability problem and our basic goal is exactly as mentioned above, i.e., we wish to convert a hard instance (formula) into an easy one. What we do, however, is not a complexity reduction but a *density extension* (DE): Let f be a Boolean formula of n variables. Then its *density*, denoted by $d(f)$, is defined as $d(f) = (\text{the number of } f\text{'s satisfying assignments}) / 2^n$. One can see that $d(f)$ is the probability that random sampling of an assignment hits a satisfying assignment. This intuitively means that a formula of high density is easier than a formula of low density; it is certainly desirable if we can extend the density of a formula f . (We sometimes say that “we condense f .”)

Now we should be more careful: When we use the random testing as mentioned above, its time complexity depends not only on $d(f)$ but also on $|f|$ (= the length of f) since we need to evaluate the value of f for each assignment. Therefore,

$$|f|/d(f)$$

might be more reasonable as a measure for our purpose. We call it the *normalized length* of f , denoted by $l^*(f)$, and the problem of reducing it the *normalized-length reduction* (NLR) problem. It should be noted that $l^*(f)$ is approximately equal to the expected time complexity of the random testing.

Both DE and NLR are formalized as a problem which, given a formula f , produces another (better) formula g . It is probably reasonable to assume that f is given as a 3-CNF formula, but there are at least three possibilities for g : (i) Only CNFs are allowed. (ii) The shape of g is not restricted, but we can only use logical AND, OR and negation operations. (iii) Functional substitutions are allowed. As for the case (i), it is unlikely to be able to get a nice algorithm; if we would get one, then we could use it repeatedly, resulting in some implication that is unbelievable from the current complexity theory. The case (iii) is relatively straightforward, namely, there is a polynomial-time algorithm which can increase the density exponentially. The remaining case (ii) appears most interesting.

In this paper, two different approaches to this problem are presented: One is to obtain g by increasing the number of variables and the other by reducing the number of variables, both of which are based

on existing SAT algorithms. The former approach is applied to two probabilistic algorithms; random flipping of variables, and the probabilistic local search by Schönig [11]. In this approach, we introduce new variables to represent random selections. In the latter approach, we use a search-space-splitting algorithm that repeats assigning constants to some of the variables in f . We evaluate the proposed algorithms using both measures $d(g)$ and $|g|/d(g)$. Interestingly, our result shows that the performance of the condensation algorithms does not follow that of the original SAT algorithms; space splitting is the best of all for $|g|/d(g)$.

Related Work. Our general goal is to modify a given formula f so that it will satisfy some desirable property. One of the most interesting results in this category is given by Papadimitriou and Wolfe [10]. They showed that from a given CNF formula f , one can construct g in polynomial time such that if f is satisfiable then so is g and if f is not satisfiable then neither is g but it becomes satisfiable if any one clause is removed from g . Iwama [6] investigated how to relate the satisfiability of the original formula f and the length of resolution proofs for the target formula g . If we wish to keep not only the satisfiability but the functionality, there has been a large literature, for example, converting a CNF monotone formula into an equivalent DNF formula in pseudo-polynomial time [4].

Also there is a huge literature for SAT algorithms; its very partial list is as follows: The history of 3-SAT bounds begins with a 1.839^n -time deterministic algorithm [8] based on the search-space-splitting technique by Monien and Speckenmeyer. In [8], they slightly improved the bound to 1.769^n by considering closer clauses. Recently, Schönig showed a probabilistic local search algorithm [11], which works in 1.334^n time. The algorithm starts with a random initial assignment, and repeats flipping an assignment to a variable randomly selected from unsatisfied clauses. Its improved version [5] is known as the fastest probabilistic algorithm with time complexity 1.331^n . (Very recently 1.329^n has been claimed [2].) Dantsin et al. has proposed its derandomized version [3], which achieves a time complexity of 1.481^n .

Condensation Algorithms. We propose three different condensation algorithms based on SAT algorithms. The first one is based on random flipping; (i) select an initial assignment $a \in \{0, 1\}^n$ uniformly at random, (ii) flip the value of a randomly selected variable, and (iii) repeat step (ii) for t times. If a satisfying assignment is found during the repetition, output “True”, otherwise output “False”.

We construct a formula g that simulates the above algorithm. Let $x^0 = (x_1^0, \dots, x_n^0)$ denote the initial assignment, $x^k = (x_1^k, \dots, x_n^k)$ denote the assignment after the k -th flip, and $v^k = (v_1^k, \dots, v_{\lceil \log n \rceil}^k)$ denote the random choice of a variable in the k -th flip, where v^k is considered as a binary representation to denote the selected variable x_{v^k} . Then, the following formula simulates the random flipping algorithm:

$$\sum_{k=0}^t f(x_1^k, \dots, x_n^k). \quad (1)$$

The assignment (x_1^k, \dots, x_n^k) can be computed from variables $x^0, v^1, v^2, \dots, v^k$ by

$$x_i^k = x_i^0 \oplus (v^1 = i) \oplus (v^2 = i) \oplus \dots \oplus (v^k = i). \quad (2)$$

Our output formula $g(x^0, v^1, \dots, v^t)$ is now obtained by substituting (2) into (1).

The second one is based on Schönig’s probabilistic local search algorithm [11]: It is similar to the previous one but in step (ii), the variable to be flipped is selected from an unsatisfied clause. Unlike the original Schönig’s algorithm, we always select the leftmost unsatisfied clause in each iteration of step (ii). This modification does not change the probability of finding a satisfying assignment a^* , since the probability analysis in [11] depends on the fact that every selected clause has at least one literal (out of three) whose value does not match a^* . (Otherwise, the clause becomes to be satisfied.)

Let $u^k = (y^k, z^k)$ denote the random choice of a literal in the k -th flip, where $u^k = (0, 1)$ (respectively, $(1, 0)$ and $(1, 1)$) represents the case that the first (respectively, the second and the third) literal in the selected clause is flipped. Then, x_i^{k+1} is obtained as

$$x_i^{k+1} = x_i^k \oplus \sum_{\ell} \text{s.t. } x_i \in C_{\ell} \text{ or } \bar{x}_i \in C_{\ell} \left(\prod_{1 \leq j < \ell} \overline{C_j(x^k)} \right) C_{\ell}(x^k) \text{ pos}_{i,\ell}(y^k, z^k),$$

where $C_j(x^k)$ denotes the formula such that the variables (x_1, \dots, x_n) in C_j are replaced by variables (x_1^k, \dots, x_n^k) , and $\text{pos}_{i,\ell}(y^k, z^k)$ denotes $\overline{y^k} \cdot z^k$ (respectively, $y^k \cdot \overline{z^k}$ and $y^k \cdot z^k$) if variable x_i appears in

Table 1. Comparison among the proposed algorithms

| | Length $ g $ | Density $d(g)$ | Normalized Length $l^*(g) (= g /d(g))$ |
|----------------------------|------------------|-----------------|---|
| Random flipping | $ f t^2 \log n$ | $1/2^n$ | $ f 2^n t^2 \log n$ |
| Probabilistic local search | $ f (3n^3)^t$ | $(en/4t)^t/2^n$ | $ f 2^n (12tn^2/e)^t$ |
| Search space splitting | $ f 1.839^t$ | $2^t/2^n$ | $ f 2^n 0.920^t$ |

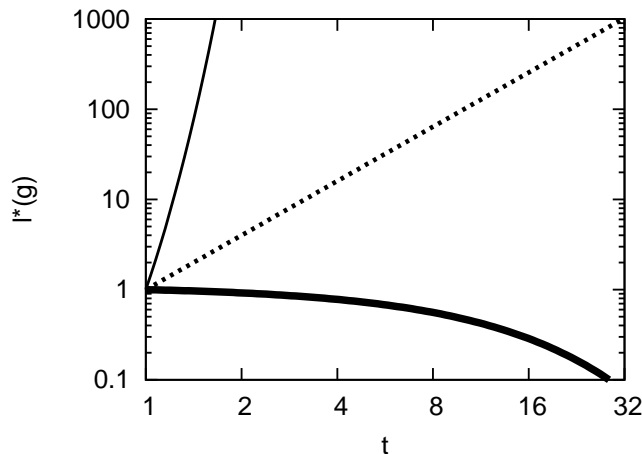


Fig. 1. Behaviors of the algorithms concerning $l^*(g)$

the first (respectively, the second and the third) position of clause C_ℓ . By substituting the variables in $f(x_1^t, \dots, x_n^t)$, we can obtain a formula with variables $x^0, u^1, u^2, \dots, u^t$ of length $O((3m)^t |f|)$.

The last one is as follows: (i) If a given formula f is 2-CNF, then solve it by the polynomial-time 2-SAT algorithm [1]. (ii) Otherwise, select an arbitrary clause $(x_i + x_j + x_k)$ in f . (iii) Recursively apply steps (i)–(iii) to formulas $f|_{x_i=1}$, $f|_{x_i=0, x_j=1}$, and $f|_{x_i=0, x_j=0, x_k=1}$. In our approach, we stop the recursion at the t -th depth, and output the disjunction of the intermediate formulas. Note that, if a satisfying assignment is found during the execution of steps (i)–(iii), we output the formula 1.

Evaluation of the Algorithms. Comparisons among the proposed three algorithms are shown in Table 1 and Fig. 1, where constant factors are ignored. The thick solid, thin solid, and dotted lines in Fig. 1 respectively correspond to the condensation algorithms based on a search space splitting, a probabilistic local search, and a random flipping.

Although the three algorithms use the iteration parameter t in different way, the comparison tells us some tendency on their behaviors. As for the DE problem, each of the three algorithms can arbitrarily condense the formula (i.e., obtain arbitrarily large $d(g)$) with proportional to the computation time. However, the condensation algorithm based on random flipping increase the density with only a small constant factor, while the length of the output formula grows proportionally to the computation time. Thus, the simple random flip does not work well. Contrary to the random flip algorithm, Schöning’s flip achieves an exponential increase of the density. On the other hand, the length of the output formula is $(3n^3)^t$ whose growth is even faster than that of the density. Therefore, as shown in Fig. 1, the search space splitting gives the best performance in terms of the measure $l^*(g)$, which decreases only by this algorithm. This algorithm can arbitrarily condense the formula (i.e., obtain arbitrarily small $l^*(g)$) with proportional to the computation time, while its length may become longer and longer.

Concluding Remarks. Our performance evaluation shows that better SAT algorithms do not always give us better condensation algorithms. Therefore, we need to construct an algorithm that is independent from existing SAT algorithms and is specific to the problems DE and NLR.

References

1. B. Aspvall, M. F. Plass, and R. E. Tarjan, "A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas", *Information Processing Letters*, 8(3), pp.121–123, 1979.
2. S. Baumer and R. Schuler, "Improving a Probabilistic 3-SAT Algorithm by Dynamic Search and Independent Clause Pairs", *Electronic Colloquium on Computational Complexity (ECCC)*, TR03-010, 2003.
3. E. Dantsin, et al., "A Deterministic $(2 - 2/(k + 1))^n$ Algorithm for k -SAT Based on Local Search", *Theoretical Computer Science*, 289(1), pp.69–83, 2002.
4. M.L. Fredman and L. Khachiyan, "On the Complexity of Dualization of Monotone Disjunctive Normal Forms", *J. Algorithms*, 21 (3), pp.618–628, 1996.
5. T. Hormeister, U. Schöning, R. Schuler, and O. Watanabe, "Probabilistic 3-SAT Algorithm Further Improved", *Proc. 10th Symposium on Theoretical Aspects of Computer Science*, LNCS 2285, pp.193–202, 2002.
6. K. Iwama, "Complexity of Finding Short Resolution Proofs", *Proc. 22nd Symposium on Mathematical Foundation of Computer Science*, LNCS 1295, pp.309–318, 1997.
7. M. Kummer, "Kolmogorov complexity and instance complexity of recursively enumerable sets", *SIAM J. Computing*, 25 (6), pp.1123–1143, 1996.
8. B. Monien and E. Speckenmeyer, "3-Satisfiability is Testable in $O(1.62^n)$ Steps", Technical Report Bericht Nr. 3/1979, Universität-Gesamthochschule-Paderborn, 1979. Reihe Theoretische Informatik.
9. P. Orponen, K.-I Ko, U. Schöning, and O. Watanabe, "Instance Complexity", *J. ACM*, 41 (1), pp.96–121, 1994.
10. C. H. Papadimitriou and D. Wolfe, "The Complexity of Facets Resolved", *J. Computers and System Sciences*, 37, pp.2–13, 1988.
11. U. Schöning, "A Probabilistic Algorithm for k -SAT and Constraint Satisfaction Problems", *Proc. 40th Annual Symposium on Foundations of Computer Science*, pp.410–414, 1999.