Test Instance Generation for MAX 2SAT (Extended Abstract)

Mistuo Motoki*

School of Information Science,
Japan Advanced Institute of Science and Technology,
1-1, Asahidai, Nomi, Ishikawa, 923-1292, Japan.
mmotoki@jaist.ac.jp

1 Introduction

Since MAX 2SAT is one of the famous NP-hard optimization problems, many heuristics and (polynomial-time) approximation algorithms have been proposed in the literature [1, 4–6]. To evaluate the performance of such algorithms, there are two possibilities; theoretical analysis and empirical study.

In theoretical analysis, an approximation ratio of the algorithm is often used as a measure. The approximation ratio is an upper bound on the ratio of an approximated cost to the optimal cost, and hence, this is a worst case measure. It is often difficult to analyze theoretically the performance of heuristics or hybrid algorithms.

On the other hand, empirical study can *estimate* the performance of approximation algorithms from various points of view.There is no difficulty in estimating the performance except generating a number of (random) input instances. Though it is obviously easy to generate test instances without the optimal solutions, we should also know the optimal solution for each test instance. While there exists a number of benchmark instances with the optimal solutions, we still do not have enough number of test instances. Hence, we would like to have a sure way of generating nontrivial test instances systematically, i.e., *instance generator*.

Our ideal goal is to design an algorithm that can randomly generate all possible test instances (i.e., whole 2CNF formulas) with the optimal solution where its running time is polynomial in the length of the output formula.[1] However, if there exists such an algorithm, the recognition of the pair of a test instance and its optimal solution is in NP. On the other hand, the complement problem, i.e., recognizing the pair of a test instance and its suboptimal solution, is also in NP, since random bits used to generate each formula in the algorithm become a witness for both problems. This concludes NP = co-NP and it is unlikely. Therefore we have to relax the problem.

We can consider two relaxations: (i) allow error in the output of the algorithm and (ii) restrict the class of instances generated. The former one gives Monte Carlo algorithms that output a feasible solution instead of the optimal solution with low error

* This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B), 1570008.

[1] If we accept exponential time, there exists a trivial algorithm; generate a formula at random, then find the optimal solution by exhaustive search.

probability (for example, [9, 10]). In this paper, we focus on the latter approach, i.e., an instance generator randomly outputs a test instance and the optimal solution with probability 1 where the set of instances generated is a strict subset of all possible instances (hereafter, we say such an instance generator is *exact*).

However, this approach creates new difficulties. If the instances generated this way are easy to solve, it is not appropriate to use them for empirical study. Hence, we have to theoretically analyze the hardness of solving the generated instances, for example, solving MAX 2SAT over the formulas generated is NP-hard.

In the literature, Dimitriou proposed an exact instance generator for MAX kSAT [3]. They experimentally showed that by appropriately choosing parameters one can control the hardness of the generated instances leading to an easy-hard-easy pattern. But there is no theoretical guarantee of hardness. Yamamoto also proposed an exact instance generator for MAX 2SAT [10]. To characterize the optimal solutions, this algorithm requires an expander graph, which is hard to randomly generate. Since they use an explicit expander graph construction algorithm, this is not truly a random instance generator.

Unfortunately, for any NP optimization problem U, the decision problem of U over the instances generated by any polynomial-time exact instance generator is in NP $\cap$ co-NP. Hence, it seems difficult to show the computational hardness of instances generated. Moreover, if a solver can efficiently recognize that input instances are generated by a specific instance generator, such instances might be easily solvable. Therefore, we investigate how hard it is to recognize our instances.

We propose an exact instance generator based on the concept of a linear-time algorithm for 2SAT. For the proposed generator, we show that the set of instances generated is NP-complete. From computational point of view, this means that finding an optimal solution for our instances is as hard as searching for a satisfying assignment for satisfiable 3CNF formulas. We also show that it is even NP-hard to approximately recognize our instances.

2 Our Instance Generator

We start with some notations. A *literal* over $X = \{x_1, \ldots, x_n\}$, the set of n Boolean variables, is $x \in X$ or its negation $\overline{x}$. A *k-clause* over X is a disjunction of exactly k literals over X whose underlying variables are distinct. Let *kCNF formula* over X be a collection of k-clauses over X. We allow any clause to appear more than once.A *truth assignment* over X is a map of X to $\{0, 1\}^n$. We use 1 and 0 to denote true and false respectively. A truth assignment t *satisfies* a clause c iff at least one literal in c has a value 1. Otherwise we say t *falsifies* c. MAX 2SAT is a problem to find an assignment that satisfies the maximum number of clauses for given 2CNF formula.

For any 2CNF formula F over X, the *implication graph* of F is a directed (multi) graph $G_F = (V, E)$, where V is the set of all possible literals, i.e., $X \cup \{\overline{x_i} \mid x_i \in X\}$, and $E = \left\{(v_i \to v_j) \mid v_i, v_j \in V \text{ and } (\overline{v_i} \vee v_j) \in F\right\}$. Note that if E contains an edge $(v_i \to v_j)$, then the edge $(\overline{v_j} \to \overline{v_i})$ also exists. We say that such an edge is a *complement edge* of the other. For any edge $(v_i \to v_j)$, if one assigns true to v_i, one has to set v_j true to satisfy the original clause $(\overline{v_i} \vee v_j)$; hence the name implication graph. Any 2CNF formula F is unsatisfiable iff G_F has a cycle that contains v and $\overline{v}$ simultaneously [2].

Strictly speaking, if there is one such cycle C, there must be another cycle that consists of all complement edges of the cycle C. Therefore we call such a pair of two cycles as a *contradictory bicycle*. For any truth assignment t over X, let $\mathcal{B}_t$ be the set of 2CNF formulas over X such that any $F \in \mathcal{B}_t$ satisfies the following conditions: (i) F has exactly one clause falsified by t, (ii) G_F has one contradictory bicycle, and (iii) if we remove any clause from F, the remaining formula is satisfiable, i.e., F is a minimal unsatisfiable formula. We also denote by C_t the set of 2-clauses over X satisfied by t.

It is easy to see that we can randomly generate any formula in $\mathcal{B}_t$ for an arbitrary truth assignment t. To illustrate this, w.l.o.g. we assume that $t = 1^n$. Let B be an arbitrary formula of $\mathcal{B}_t$. It is clear that every clause falsified by t consists of two negative literalsand B contains exactly one such clause. Such a clause is transformed into edges from a positive literal to a negative literal in the implication graph G_B. Furthermore, each cycle also has exactly one edge from a negative literal to a positive literal. We can divide each cycle into two paths, a path consisting of positive literals only and a path of negative literals only.We remark that there exists at least one common variable as a contradictory variable, in both paths. Also, since any 2-clause is complement-free, the last variable of each path is distinct from the first variable of the other path.Thus, we only need to generate two sequences of variables that have at least one common variable and the first variable of each sequence is distinct from the last variable of the other.

Here, for an arbitrary truth assignment t and a positive number k, we consider a 2CNF formula F that consists of (not necessary distinct) k formulas in $\mathcal{B}_t$ and some clauses in C_t. Obviously, any truth assignment falsifies at least k clauses of F since G_F has k contradictory bicycles. This means that an upper bound of the minimum number of unsatisfiable clauses is k. On the other hand, since there exist exact k formulas of B_t, F has just k clauses falsified by t, i.e., the lower bound is also k. Thus it is clear that t is the optimal solution of F and the minimum number of unsatisfiable clauses in F is k. Let $\mathcal{I}$ be a set of such formulas, i.e., $\mathcal{I} = \{F \mid \exists t \text{ s.t. } F \text{ consists of elements of } \mathcal{B}_t \text{ and } C_t\}$. It is easy to see that we can randomly generate an arbitrary formula in $\mathcal{I}$ and its optimal solution t by appropriate randomized algorithms (see Algorithm 1), e.g., first choose t at random, then construct a formula as a conjunction of some elements of $\mathcal{B}_t$ and C_t. Clearly, the running time of our instance generator is linear in the length of the instance generated. We remark that if the number of additional clauses from C_t is 0, the instance generated has at least two optimal solutions, t and $\bar{t}$, and hence, we add such clauses.

Algorithm 1: An example of generation algorithm

Input: the number of variables n
begin
  Let F be an empty formula;
  Choose $t \in \{0,1\}^n$ uniformly at random;
  Choose the minimum number of unsatisfiable clauses k (≥ 0);
  for $i = 1$ **to** k **do** Generate a 2CNF formula over X from $\mathcal{B}_t$ at random and add it to F;
  Add r clauses of C_t to F at random (where r is a random nonnegative integer);
  Output: F and t
end

3 The Hardness Results

In this section, we consider the hardness of the instances generated. As described in the introduction, it is not easy to show *computational* hardness of the instances generated. This means that if we can efficiently recognize instances generated, such instances cannot be hard. Therefore, we consider how hard it is to recognize our instances. First, we show the hardness of exact recognition.

Theorem 1. *The set $\mathcal{I}$ of generated instances is* NP*-complete.*

Proof. It is clear that $\mathcal{I}$ is in NP because of the witness, that is, a truth assignment t and a partition into k formulas of $\mathcal{B}_t$ and a subset of C_t for each instance.

Hereafter we show a reduction from 3SAT. Let $F_{3\mathrm{CNF}}$ be an arbitrary 3CNF formula over X with m 3-clauses $c_1, c_2, \ldots, c_m$. For any i, $1 \leq i \leq m$, we translate the ith 3-clause $c_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$ of $F_{3\mathrm{CNF}}$ ($l_{i,*}$ means an arbitrary literal over X) into the 2CNF formula over X and $Y_i = \{y_{i,1}, y_{i,2}\}$, $B_i = (\overline{l_{i,1}} \vee y_{i,1})(\overline{y_{i,1}} \vee l_{i,2})(\overline{l_{i,2}} \vee \overline{y_{i,2}})(y_{i,2} \vee l_{i,3})(\overline{l_{i,3}} \vee \overline{y_{i,1}})(y_{i,1} \vee y_{i,2})(\overline{y_{i,2}} \vee l_{i,1})(\overline{y_{i,2}} \vee l_{i,1})$. We remark that the variables in Y_i appear only in B_i. B_i contains exactly one contradictory bicycle, and thus any truth assignment over $X \cup Y_i$ falsifies at least one clause. Let a 2CNF formula $F_{2\mathrm{CNF}}$ be the conjunction of B_i for all i. Hence $F_{2\mathrm{CNF}}$ has $8m$ clauses over $n+2m$ variables, $X \cup (\bigcup_i Y_i)$. It is not difficult to show that, in $F_{2\mathrm{CNF}}$, the number of contradictory bicycles is equal to the minimum number of unsatisfiable clauses iff $F_{3\mathrm{CNF}}$ is satisfiable. □

This result directly means that finding an optimal solution of our instances is at least as hard as searching for a satisfying assignment of satisfiable 3CNF formulas. Thus, a polynomial-time algorithm that can obtain an optimal solution for any instance in $\mathcal{I}$ is unlikely. Moreover, in the above discussion we assume that we know the maximum contradictory bicycle packing. It, however, seems hard to obtain the maximum contradictory bicycle packing in general.

Next, we consider the hardness of approximate recognition.

Theorem 2. *For any constant $\varepsilon > 0$, it is* NP*-hard to distinguish any 2CNF formula in $\mathcal{I}$ from the 2CNF formulas in which the ratio of the minimum number of unsatisfiable clauses to the maximum number of contradictory bicycles is $9/8 - \varepsilon$.*

Proof. We consider the same reduction as in the proof of Theorem 1. We have already shown that any satisfiable $F_{3\mathrm{CNF}}$ with m clauses over X is transformed to $F_{2\mathrm{CNF}}$ with $8m$ clauses where the minimum number of unsatisfiable clauses is m.

Now we consider the case $F_{3\mathrm{CNF}}$ is unsatisfiable. For any truth assignment over X that falsifies the ith clause c_i of $F_{3\mathrm{CNF}}$, we can find a truth assignment over Y_i that falsifies exactly two clauses in B_i. We again remark that we can set a truth assignment over Y_i independently of a truth assignment over Y_j since any variable of Y_i does not appear in B_j for any $j \neq i$. Thus, if k clauses of $F_{3\mathrm{CNF}}$ are unsatisfiable, $m + k$ clauses of $F_{2\mathrm{CNF}}$ are unsatisfiable.

Now we focus on 3CNF formulas for which only a fraction $7/8 + \varepsilon$ of the clauses can be satisfied. Such 3CNF formulas are transformed into 2CNF formulas with m contradictory bicycles and $m + (1/8 - \varepsilon)m = (9/8 - \varepsilon)m$ unsatisfiable clauses. Since it is NP-hard to distinguish between such 3CNF formulas and satisfiable 3CNF formulas [7], we conclude the proof. □

Unfortunately, this result does not directly imply hardness of the instances generated for approximation algorithms. However, if there exists an approximation algorithm that approximates any instance of I within a fraction $\frac{8m-9m/8}{8m-m} = 55/56$, such an algorithm can distinguish satisfiable 3CNF formulas from unsatisfiable 3CNF formulas and it is unlikely. We remark that this ratio, $55/56 \approx 0.982$, is still much larger than $21/22 \approx 0.955$ which is the best known inapproximability upper bound for MAX 2SAT [7] (Khot et al. [8] recently improved this ratio to 0.944 under some unproven conjectures).

4 Concluding Remarks

We analyzed that it is hard to recognize instances by the proposed instance generator for MAX 2SAT. On the other hand, the generator is still naïve, that is, our generator is only generating positive instances of NP-complete problem. Hence it may generate a number of easy instances and it is important to eliminate such easy instances.

While we focused on theoretical hardness in this paper, we would like to experimentally check hardness against a number of MAX SAT solvers. Since the proposed instance generator uses many (exposed and hidden) parameters, such as the number of contradictory bicycle, the length of each bicycle, the total number of clauses and so on, we also have to determine an appropriate range of such parameters. We expect that there exists a phase transition phenomenon, and hence, an easy-hard-easy pattern on some parameters. Finally, it is better if we can generate some instances outside of $\mathcal{I}$. Since $\mathcal{I}$ is NP-complete, we may apply techniques to generate hard (or negative) instances for other NP-hard problems (e.g., SAT).

References

1. T. Asano and D. P. Williamson. Improved approximation algorithms for MAX SAT. *J. Algorithms*, Vol.42, pp.173–202, 2002.
2. B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inform. Process. Lett.*, Vol.8, pp.121–123, 1979.
3. T. Dimitriou. A wealth of SAT distributions with planted assignments. In *Proc. of CP 2003*, LNCS 2833, pp.274–287, 2003.
4. U. Feige and M. X. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proc. of ISTCS 1995*, pp.182–189, 1995
5. M. X. Goemans and D. P. Williamson. .879-approximation algorithms for MAX CUT and MAX 2SAT. In *Proc. of STOC 1994*, pp.422–431, 1994.
6. P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, Vol.44, pp.279–303, 1990.
7. J. Håstad. Some optimal inapproximability results. in *Proc. of STOC 1997*, pp.1–10, 1997.
8. S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for Max-Cut and other 2-variable CSPs? in *Proc. of FOCS 2004*, pp.146–154, 2004.
9. M. Motoki. Random instance generation for MAX 3SAT. In *Proc. of COCOON 2001*, LNCS2108, pp.502–508, 2001.
10. M. Yamamoto. On generating instances for MAX2SAT with optimal solutions. Dept. of Math. and Comp. Sciences Research Reports (Series C: Computer Science), C-191, `http://www.is.titech.ac.jp/research/research-report/C/C-191.ps.gz`, 2004.