PDMI PREPRINT — 19/1998

HARD FORMULAS FOR SAT LOCAL SEARCH ALGORITHMS

Edward A. HIRSCH

Russian Academy of Sciences
Steklov Mathematical Institute
St. Petersburg Department
27 Fontanka, St. Petersburg, 191011,
Russia
E-mail: hirsch@pdmi.ras.ru

September 6, 1998

Abstract

In 1992 B. Selman, H. Levesque and D. Mitchell proposed GSAT, a greedy local search algorithm for the Boolean satisfiability problem. Good performance of this algorithm and its modifications has been demonstrated by many experimental results. In 1993 I. P. Gent and T. Walsh proposed CSAT, a version of GSAT that almost does not use greediness. It has been recently proved that CSAT can find a satisfying assignment for a restricted class of formulas in the time c^n, where $c < 2$ is a constant. In this paper we prove a lower bound of the order 2^n for GSAT and CSAT. Namely, we construct formulas F_n of n variables, such that GSAT or CSAT finds a satisfying assignment for F_n only if this assignment or one of its n neighbours is chosen as the initial assignment for the search.

1 Introduction

In the past six years there has been an increased interest to local search algorithms for the Boolean satisfiability problem. Though this problem is $\mathcal{NP}$-complete (see e.g. [4]), B. Selman, H. Levesque and D. Mitchell have shown in [21] that an algorithm that uses local search can easily handle some of "hard" instances of SAT. Namely, they proposed a randomized greedy local search algorithm GSAT and presented experimental results showing its good performance for graph coloring and n-queens problem. Starting with a random assignment, GSAT changes at each step the value of one variable so that the number of satisfied clauses increases as much as possible (the increase can be non-positive). If the satisfying assignment is not found in a certain number of steps, GSAT starts with another random assignment. E. Kostsoupias and Ch. Papadimitriou [12] have proved that a similar algorithm finds a satisfying assignment in a polynomial time for "almost all" formulas. Later many experimental results concerning various local search algorithms for SAT were obtained [18, 20, 5, 6, 7, 8]. In particular, B. Selman and H. Kautz demonstrated in [19] that GSAT variants can solve large circuit synthesis and planning problems. Also some average-case upper bounds were proved [9, 3].

On the other hand, B. Selman, H. Levesque and D. Mitchell in [21] presented an example of a formula that can be hard for GSAT. They claimed that it could take exponential time for GSAT to find a satisfying assignment for such formulas. It is natural that the worst-case running time of an algorithm for SAT is exponential, since SAT is $\mathcal{NP}$-complete. However, starting from [14, 15] and [1, 2] many algorithms for SAT subproblems were constructed that have exponential upper bounds of the form c^n, where $c < 2$ is a constant and n is the number of variables. For example, in the recent paper of R. Paturi, P. Pudlak and F. Zane [16] it is proved that the satisfiability of a formula in k–CNF (the length of every clause is at most k) can be checked in a time of the order $2^{(1-1/k)n}$. Current upper bounds for checking the satisfiability of formulas in 3–CNF are approximately 1.5^n ([13, 17]). Also for SAT itself, the bounds were proved [10] with respect to the length of a formula and the number of clauses in it. However, none of these algorithms uses local search. Recently it was proved [11] that algorithm CSAT (a version of a local search algorithm for SAT proposed by I. P. Gent and T. Walsh [5, 6]) can check satisfiability of a formula in k–CNF–d (a formula in k–CNF in which each variable occurs at most d times) in a time of the order $2^{(1-1/(kd))n}$. The intriguing question is whether such an upper bound can be proved for formulas in CNF without restrictions, either for CSAT, GSAT, or another local search algorithm for SAT.

In this paper we prove that, unfortunately, such bounds cannot be proved for GSAT and CSAT. Namely, we construct a series of formulas F_n of n variables, such that GSAT (or CSAT) finds a satisfying assignment for F_n in expected time at least 2^n (up to a polynomial factor), i.e. GSAT and CSAT perform on these formulas no better than a trivial algorithm that just enumerates all assignments at random.

In Sect. 2 we give definitions and formulate precisely the algorithms we study. Section 3 contains the proof of the lower bound. In Sect. 4 we discuss further directions of the work.

2 Preliminaries

Let $\mathrm{Var} = \{x_1, x_2, \ldots\}$ be a set of Boolean variables (that can take values $True$ or $False$). Literals are variables and their negations (denoted by $\neg x_1, \neg x_2 \ldots$). A clause is a finite set of literals. In this section F is a formula in CNF (i.e. a finite set of clauses), I and J are truth assignments for the variables of F (i.e. finite sets of literals that correspond to the variables occuring in F). We consider only complete assignments, i.e. assignments that contain one of the literals v or $\neg v$ for each variable v occuring in F.

We say that I satisfies a clause C if $I \cap C \neq \emptyset$. We denote by $\mathrm{sat}(F, I)$ the number of clauses that I satisfies in F. If I satisfies all clauses of F, we say that it satisfies the formula F.

If I contains a literal $x \in \mathrm{Var}$, then we say that the variable x has the value $True$ in I. If it contains a literal $\neg x$, we say that the variable x has the value $False$ in I. We denote by $\mathrm{dist}(I, J)$ the number of variables that have different values in I and J.

The values of all variables in assignment $I^{(v)}$ coincide with the values in I except the variable v which has the opposite value. We say that we flip the value of v in I to obtain $I^{(v)}$.

We now describe the algorithms we study. Their input is a formula F in CNF. Both algorithms start with a random assignment and at each step change the value of exactly one variable. If a satisfying assignment for the input formula is not found in a certain number of steps (parameter `MAXFLIPS`), the algorithm chooses at random another initial assignment. The maximal number of assignments the algorithm examines, is determined by parameter `MAXTRIES`.

The difference between algorithms is in the choice of the variable whose value is changed. Algorithm GSAT (see Fig. 1) chooses the variable such that the number of clauses satisfied by the new assignment is the largest (however, this number can be less than the number of clauses satisfied by the current assignment). There can be several variables such that the new assignment satisfies the same number of clauses. However, it is not important for us in this paper whether one of them is chosen in a random way or deterministically.

Algorithm CSAT (see Fig. 2) has greater freedom of choice. It constructs a set that contains every variable such that flipping its value increases the number of satisfied clauses. Then it chooses one of these variables at random. If there are no such variables, then it chooses one at random from the set of variables that do not lead to the decrease of the number of satisfied clauses. If however all variables lead to such decrease, it chooses one at random from the set of all variables occuring in F.

3 The bound

In this section we construct formulas F_n that are "hard" for the algorithms GSAT and CSAT, and prove the desired lower bound. For an integer $n \geq 6$, formula F_n contains n Boolean variables and consists of $4n + 1$ clauses of three types (see Fig. 3). Clauses of the first type express that in every satisfying assignment for F_n the values of all variables should be equal. The clause $\{x_1, x_2\}$ implies that all these values are equal to $True$ (since

Algorithm GSAT.
Input:
A formula F in CNF, integers `MAXFLIPS` and `MAXTRIES`.
Output:
A truth assignment for the variables of F, or "No".

Method.

Repeat `MAXTRIES` times:

— Pick an assignment I at random.

— Repeat `MAXFLIPS` times:

— If I satisfies F, then output I.
— Choose a variable v in F such that $I^{(v)}$ satisfies the maximal possible number of clauses of F.
— Flip the value of v in I.

Output "No".

Figure 1: Algorithm GSAT

together with $\{x_1, \neg x_2\}$ it implies that x_1 must have the value $True$ in every satisfying assignment). Hence, the formula F_n has only one satisfying assignment, and this is the assignment $S_n = \{x_1, x_2, \ldots, x_n\}$ (it satisfies also all clauses of the third type).

Clauses of the third type are intended for "misleading" the algorithm. For each variable x_i, the formula F_n includes three clauses containing x_i without the negation. Each of these clauses also contains the negations of the next (modulo n) $n-4$ variables and the negations of two of the rest three variables. Every assignment that differs from the satisfying assignment by the values of at least three variables, satisfies all clauses of the third type. However, assignments that differ from the satisfying assignment by the values of only one variable, satisfy only $3n-3$ of these clauses. In the following lemmas we prove that if GSAT or CSAT encounters an assignment that differs from the satisfying assignment by the values of exactly two variables, then it chooses a wrong variable for flipping. Thus, in this case the satisfying assignment cannot be found unless the algorithm will start from another initial assignment. In the next two lemmas and in the theorem, S_n denotes the only satisfying assignment for the formula F_n.

Lemma 1 *Let I be an assignment for F_n, such that* $\mathrm{dist}(I, S_n) = 2$. *Then for every variable v such that* $\mathrm{dist}(I^{(v)}, S_n) = 1$, *the assignment $I^{(v)}$ satisfies strictly less clauses of F_n than I does.*

Proof. Let I differ from S_n by values of variables $v = x_i$ and x_j. Note that $I^{(v)}$ satisfies the

Algorithm CSAT.
Input:
A formula F in CNF, integers `MAXFLIPS` and `MAXTRIES`.
Output:
A truth assignment for the variables of F, or "No".

Method.

Repeat `MAXTRIES` times:

— Pick an assignment I at random.

— Repeat `MAXFLIPS` times:

— If I satisfies F, then output I.

— Choose at random a variable v in F such that $\text{sat}(F, I^{(v)}) > \text{sat}(F, I)$.
If there are no such variables, choose at random a variable v such that $\text{sat}(F, I^{(v)}) = \text{sat}(F, I)$.
If there are none of those, choose a variable v at random from the set of all variables occuring in F.

— Flip the value of v in I.

Output "No".

Figure 2: Algorithm CSAT

clause $\{x_1, x_2\}$, $3n-3$ clauses of the third type and $n-1$ clauses of the first type, i.e. $4n-3$ clauses in total. We consider three cases.
Case 1. $|(i-j) \bmod n| \neq 1$. I satisfies $n-2$ clauses of the first type and all clauses of other types, i.e. $4n-1$ clauses in total.
Case 2. $|(i-j) \bmod n| = 1$, but $\{i,j\} \neq \{1,2\}$. I satisfies $n-1$ clauses of the first type, $3n-1$ clauses of the third type and the clause $\{x_1, x_2\}$, i.e. $4n-1$ clauses in total.
Case 3. $\{i,j\} = \{1,2\}$. I satisfies $n-1$ clauses of the first type and $3n-1$ clauses of the third type, i.e. $4n-2$ clauses in total. □

Lemma 2 *Let I be an assignment for F_n ($n \geq 6$), such that $\text{dist}(I, S_n) = 2$. Then there exists a variable v such that $\text{dist}(I^{(v)}, S_n) = 3$ and $\text{sat}(F_n, I) = \text{sat}(F_n, I^{(v)})$.*

Proof. Let I differ from S_n by values of variables x_i and x_j. We choose v such that

(1) $v \neq x_1, x_2$;

(2) exactly one of the conditions $v = x_{j+1}$, $v = x_{j-1}$, $v = x_{i-1}$ or $v = x_{i+1}$ holds.

Type 1:

$$\{x_1, \neg x_2\}$$
$$\{x_2, \neg x_3\}$$
$$\dots$$
$$\{x_n, \neg x_1\}$$

Type 2:

$$\{x_1, x_2\}$$

Type 3:

$$\{x_1, \quad \neg x_2, \dots, \neg x_{n-3}, \quad \neg x_{n-2}, \neg x_{n-1}\}$$
$$\{x_1, \quad \neg x_2, \dots, \neg x_{n-3}, \quad \neg x_{n-1}, \neg x_n\}$$
$$\{x_1, \quad \neg x_2, \dots, \neg x_{n-3}, \quad \neg x_n, \neg x_{n-2}\}$$

$$\{x_2, \quad \neg x_3, \dots, \neg x_{n-2}, \quad \neg x_{n-1}, \neg x_n\}$$
$$\{x_2, \quad \neg x_3, \dots, \neg x_{n-2}, \quad \neg x_n, \neg x_1\}$$
$$\{x_2, \quad \neg x_3, \dots, \neg x_{n-2}, \quad \neg x_1, \neg x_{n-1}\}$$

$$\dots$$

$$\{x_n, \quad \neg x_1, \dots, \neg x_{n-4}, \quad \neg x_{n-3}, \neg x_{n-2}\}$$
$$\{x_n, \quad \neg x_1, \dots, \neg x_{n-4}, \quad \neg x_{n-2}, \neg x_{n-1}\}$$
$$\{x_n, \quad \neg x_1, \dots, \neg x_{n-4}, \quad \neg x_{n-1}, \neg x_{n-3}\}$$

Figure 3: Formula F_n

We can choose such v since $n \geq 6$. Then the assignment $I^{(v)}$ satisfies the same clauses of the second and the third types as I does. Also I and $I^{(v)}$ satisfy the same number of clauses of the first type. □

Theorem 1 *Let $n \geq 6$. The algorithms CSAT and GSAT cannot find a satisfying assignment for F_n in expected time less than $\Omega(2^n/(n+1))$.*

Proof. If CSAT or GSAT starts its search from an assignment that differs from S_n by values of at least two variables, then before outputing a satisfying assignment it must encounter an assignment that differs from S_n by values of exactly two variables. By Lemma 2, at the next step algorithm must flip the value of a variable such that the number of satisfied clauses will not decrease. It follows from Lemma 1 that the next assignment will differ from S_n by values of three variables.

Thus, the algorithm has a chance to succeed only if at some iteration it chooses S_n or one of its n neighbours as the initial assignment. □

4 Further work

It would be interesting to prove similar lower bounds for other local search algorithms for SAT (e.g., WSAT [20]). On the other hand, it would be interesting to show a good performance (or still a lower bound of the order 2^n) of these algorithms on such important classes of formulas as k–CNF.

References

[1] Dantsin, E.: Tautology proof systems based on the splitting method (in Russian). PhD thesis. Leningrad Division of Steklov Institute of Mathematics (LOMI) (1983)

[2] Dantsin, E., Kreinovich, V.: Exponential upper bounds for the propositional satisfiability problem (in Russian). In: Proceedings of the 9th National Conference on Mathematical Logic. Leningrad (1988) p. 47

[3] Franco, J., Swaminathan, R.: Average case results for satisfiability algorithms under the random clause model. Annals of Mathematics and Artificial Intelligence (to appear)

[4] Garey, M. R., Johnson, D. S.: Computers and intractability. A guide to the theory of NP-completeness. W. H. Freeman and Company, San Francisco (1979)

[5] Gent, I., Walsh, T.: The Enigma of Hill-climbing Procedures for SAT. Research Paper **605**, Department of Artificial Intelligence, University of Edinburgh (1992)

[6] Gent, I., Walsh, T.: Towards an Understanding of Hill-climbing Procedures for SAT. In: Proceedings of the 11th National Conference on Artificial Intelligence. Washington, DC, AAAI Press (1993) 28–33

[7] Gent, I., Walsh, T.: Unsatisfied Variables in Local Search. In: Hallam, J. (ed.): Hybrid Problems, Hybrid Solutions (AISB-95). IOS Press, Amsterdam (1995)

[8] Gu, J.: Efficient Local Search for Very Large-Scale Satisfiability Problem. ACM SIGART Bulletin. Vol. **3** (N**1**) (1992) 8–12

[9] Gu, J., Gu, Q.-P.: Average Time Complexity of The SAT1.2 Algorithm. In: Proceedings of the 5th Annual International Symposium on Algorithms and Computation (1994) 147–155

[10] Hirsch, E. A.: Two new upper bounds for SAT. In: Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (1998) 521–530

[11] Hirsch, E. A.: Local Search Algorithms for SAT: Worst-Case Analysis. In: Proceedings of the 6th Scandinavian Workshop on Algorithm Theory, LNCS **1432** (1998) 246–254

[12] Koutsoupias, E., Papadimitriou, C. H.: On the greedy algorithm for satisfiability. Information Processing Letters **43(1)** (1992) 53–55

[13] Kullmann, O.: Worst-case Analysis, 3-SAT Decision and Lower Bounds: Approaches for Improved SAT Algorithms. In: Du, D., Gu, J., Pardalos P. M. (eds.): DIMACS Proceedings SAT Workshop 1996. American Mathematical Society (1997)

[14] Monien, B., Speckenmeyer, E.: 3-satisfiability is testable in $O(1.62^r)$ steps. Bericht Nr. **3/1979**, Reihe Theoretische Informatik, Universität-Gesamthochschule-Paderborn

[15] Monien, B., Speckenmeyer, E.: Solving satisfiability in less than 2^n steps. Discrete Applied Mathematics **10** (1985) 287–295

[16] Paturi, R., Pudlak, P., Zane, F.: Satisfiability Coding Lemma. In: Proceedings of the 38th Annual Symposium on Foundations of Computer Science (1997) 566–574

[17] Schiermeyer, I.: Pure literal look ahead: An $O(1.497^n)$ 3-Satisfiability algorithm. In: Franco, J., Gallo, G., Kleine Büning, H., Speckenmeyer, E., Spera, C. (eds.): Workshop on the Satisfiability Problem, Technical Report. Siena, April, 29 – May, 3, 1996. University Köln, Report No. **96-230**

[18] Selman, B., Kautz, H.: An Empirical Study of Greedy Local Search for Satisfiability Testing. In: Proceedings of the 11th National Conference on Artificial Intelligence. Washington, DC, AAAI Press (1993)

[19] Selman, B., Kautz, H.: Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In: Proceedings of the 14th Conference on Artificial Intelligence (1996)

[20] Selman, B., Kautz, H., Cohen, B.: Noise strategies for improving local search. In: Proceedings of the 12th National Conference on Artificial Intelligence. Seattle, Washington, AAAI Press (1994) **1**:337–343

[21] Selman, B., Levesque, H., Mitchell, D.: A new method for solving hard satisfiability problems. In: Swartout, W. (ed.): Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, California, MIT Press (1992) 440–446