SAT Local Search Algorithms: Worst-Case Study

Edward A. Hirsch*

Abstract

Recent experiments demonstrated that local search algorithms (e.g. GSAT) are able to find satisfying assignments for many "hard" Boolean formulas. A wide experimental study of these algorithms demonstrated their good performance on some important classes of formulas as well as poor performance on some other ones. In contrast, theoretical knowledge of their worst-case behaviour is very limited. However, many worst-case upper and lower bounds of the form $2^{\alpha n}$ ($\alpha < 1$ is a constant) are known for other SAT algorithms, e.g. resolution-like algorithms. In the present paper we prove both upper and lower bounds of this form for local search algorithms. The class of *linear-size* formulas we consider for the upper bound covers most of the DIMACS benchmarks, the satisfiability problem for this class of formulas is $\mathcal{NP}$-complete.

1 Introduction

Recently 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. [17]), B. Selman, H. Levesque and D. Mitchell have shown in [23] that an algorithm that uses local search can easily handle some of "hard" instances of SAT. They proposed a randomized greedy local search procedure GSAT (see Fig. 1) for the Boolean satisfiability problem and presented reassuring experimental results 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. Later many experimental results concerning various modifications of GSAT were obtained [4, 20, 5, 22, 6, 21, 13]. In particular, B. Selman and H. Kautz demonstrated in [21] that GSAT variants can solve large circuit synthesis and planning problems. Also, some average-case upper bounds are known for local search algorithms [11, 7].

Starting from [15] (see also [16]) and [2], there are many theoretical results proving "less-than-2^N" upper bounds for SAT and its subproblems w.r.t. the number N of variables, the number K of clauses and the length L of a formula. However, most of them use splitting algorithms. For example, splitting algorithms give the upper bounds $O\left(2^{0.309K}\right)$ and $O\left(2^{0.103L}\right)$ [8] for SAT and the bounds near $O\left(2^{0.59N}\right)$ [12, 19] for the satisfiability problem for formulas in 3-CNF. On the other hand, a lot of exponential lower bounds for different proof systems (in particular, for regular resolution which is equivalent to splitting algorithms [1]) were proved ([24] is a good survey).

The only known worst-case bound for a local search algorithm is the upper bound $O\left(2^{(1-\frac{1}{kd})N}\right)$ for formulas in k–CNF–d (i.e. each of the variables occurs at most d times, and each clause contains

*Steklov Institute of Mathematics at St.Petersburg, 27 Fontanka, 191011 St.Petersburg, Russia. Email: hirsch@pdmi.ras.ru, URL: http://logic.pdmi.ras.ru/~hirsch/index.html. Supported in part by grants from INTAS and RFBR.

at most k literals) [9]. This bound was proved for a variant of GSAT that does not use greediness, namely, for algorithm CSAT ("cautious" SAT, see Fig. 2) suggested and studied by I. P. Gent and T. Walsh [4, 5, 6] who provided experimental evidence that greediness is not very important for GSAT efficiency. Given a formula F in CNF and a truth assignment I, we say that a variable v increases (decreases) the number of satisfied clauses, if the assignment that differs from I only by the value of v satisfies more (respectively, less) clauses of F than I does. CSAT chooses a variable at random from the set of variables increasing the number of satisfied clauses, if there are no such variables, it chooses one that does not decrease the number of satisfied clauses, if there are none of those, it chooses any variable.

In this paper we study local search algorithms CSAT and GSAT. We concentrate on worst-case upper and lower bounds of the form $2^{\alpha N}$ ($\alpha \leq 1$ is a positive constant). Namely, we

(1) improve the worst-case upper bound [9] for CSAT and extend this result to all *linear-size* formulas;

(2) prove the first worst-case lower bounds for CSAT and GSAT for formulas in CNF (the bound is $\Omega\left(2^N/p(N)\right)$, where p is a polynomial) and 3–CNF (the bound is $\Omega\left(2^{0.0817N}\right)$).

We consider CSAT and GSAT as Monte Carlo randomized algorithms that always output "no" if the input formula is unsatisfiable and output "yes" with probability greater than 50% if the input formula is satisfiable. In fact, in the latter case the algorithm finds a satisfying assignment.

Upper bounds. A Boolean formula is *linear-size* if the number of literal occurrences in it is linearly dependent on the number of variables, i.e., $L \leq \lambda N$ for a constant λ. For each $\lambda > 1$, the satisfiability problem is $\mathcal{NP}$-complete for formulas with $L \leq \lambda N$.

In addition, it is a common belief that there is a threshold $K/N \approx 4.25$ such that about 50% of formulas in 3–CNF located near it are satisfiable (see [3, 10] for lower and upper bounds for this threshold). Computational experiments show that random formulas picked from this region are probably the most "hard" (see, e.g., [14]). Clearly, these formulas are linear-size.

We prove that CSAT can solve the satisfiability problem for linear-size formulas in the time

$$2^{\left(1-\frac{1}{4(k-1)(\lambda-1)+2}\right)N}p(N)$$

with error probability $1/e$ independent on the input formula (p is a polynomial, $e = 2.71828\ldots$ is the base of the natural logarithm). We also prove a better bound for a very similar algorithm which combines CSAT with pure literal elimination. As a by-product, we improve the upper bound for CSAT on formulas in k–CNF–d to $2^{(1-\frac{1}{(k-1)d+1})N}$.

For simplicity, we reduce the power of CSAT by disabling downward moves (moves that decrease the number of satisfied clauses) and moves on plateaus (moves that do not change the number of satisfied clauses). We show that even in this form CSAT solves the satisfiability problem for linear-size formulas significantly better than a simple algorithm that just picks an assignment at random. Though the bound presented in this paper is worse than the recent bound for k-CNF which is due to R. Paturi, P. Pudlak and M. E. Saks and F. Zane [18] (their algorithm uses neither splitting nor local search), the result is nevertheless of particular interest because together with our conference paper [9] we present the first theoretically proved upper bound for experimentally well-studied local search algorithms for SAT.

Lower bounds. Many researchers claimed that some formulas are "hard" for local search algorithms. For example, B. Selman, H. Levesque and D. Mitchell [23] *claimed* (but not *proved*) that it could take exponential time for GSAT to find a satisfying assignment for the formula

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee \overline{x_4} \vee \overline{x_2}) \wedge (x_1 \vee x_5 \vee x_2) \wedge (x_1 \vee \overline{x_5} \vee x_2) \wedge$$
$$(\overline{x_1} \vee \overline{x_6} \vee x_7) \wedge (\overline{x_1} \vee \overline{x_7} \vee x_8) \wedge \ldots \wedge (\overline{x_1} \vee \overline{x_{N-1}} \vee x_N) \wedge (\overline{x_1} \vee \overline{x_N} \vee x_6). \qquad (1.1)$$

In this paper we concentrate on such "hard" examples and on *theoretical proofs* of their hardness for local search algorithms GSAT and CSAT.

All variables but x_1, x_2, x_5 occur in the formula (1.1) only twice. Our Theorem 3.1 gives the worst-case upper bound $2^{\frac{4}{5}N}p(L)$ for the running time of CSAT on such formulas[1] (p is a polynomial). On the other hand, in Sect. 4 we prove a lower bound of the order $2^{0.0817N}$ for formulas with at most three literals per clause. We use hard examples different from (1.1). These examples are very similar to hard examples presented in [17] (Exercise 11.5.6) for a random walk algorithm (and thus it is highly probable, but not proved here, that similar examples fail algorithms that combine local search with random walk [14]). As to the general case (formulas in CNF without any additional restrictions), we prove a lower bound of the order 2^N, which mean that CSAT and GSAT perform on these formulas not much better than a trivial algorithm. This bound is based on a different set of hard examples. The number of clauses in these examples is linearly dependent on the number of variables. Thus, we obtain the lower bound $2^{K/4}$ w.r.t. the number K of clauses in the input formula. Our lower bounds hold both for CSAT and GSAT.

Structure of the paper. In Sect. 2 we give definitions and formulate precisely the algorithms we study. In Sect. 3 we prove various worst-case upper bounds for CSAT and its simple modification. Section 4 contains the proofs of the lower bounds. In Sect. 5 we discuss the current situation with upper and lower bounds for SAT and further directions of the work.

2 Preliminaries

Let Var $= \{x_1, x_2, \ldots\}$ be a set of Boolean variables (that take the values $True$ and $False$). Literals are variables $x_1, x_2, \ldots$ and their negations $\overline{x_1}, \overline{x_2} \ldots$ 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 corresponding to the variables occurring in F). We consider only complete assignments, i.e. assignments that contain one of the literals v or $\overline{v}$ for each variable v occurring in F.

If a variable $v \in I$, we say that the value of the variable v in the assignment I is $True$, and write $v[I] = True$; if $\overline{v} \in I$, we write $v[I] = False$. 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 their 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 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.

A formula is in k–CNF if each of its clauses contains at most k literals. A formula is in k–CNF–d if each of its clauses contains at most k literals and every variable occurs in it at most d times.

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

[1] In fact, by a similar argument an upper bound $2^{\frac{2}{3}N}p(L)$ can be proved.

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 CSAT and GSAT 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 leading to the largest number of satisfied 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 all variables lead to such decrease, it chooses a variable at random from the set of all variables occurring in F.

To estimate the worst-case running time of CSAT, we reduce the power of CSAT by disabling downward moves (moves that decrease the number of satisfied clauses) and moves on plateaus (moves that do not change the number of satisfied clauses). However, our result holds for the original CSAT as well.

We first prove an upper bound for a "shortened" version called Algorithm SimpleCSAT (see Fig. 3) of this simplified algorithm. This version considers only one initial assignment (i.e., `MAXTRIES` $= 1$). Algorithm SimpleCSAT either outputs a satisfying assignment after having considered at most K assignments (K is the number of clauses in the input formula), or answers "No" (when it cannot choose a variable to change). Algorithm CSAT (with reduced power) then can be obtained by repeating Algorithm SimpleCSAT.

3 Worst-case upper bound for CSAT

In this section k and d are positive integers, e is the base of the natural logarithm, F is a satisfiable formula in k–CNF given as input to Algorithm SimpleCSAT, N is the number of variables in it. Let $\delta \cdot N$ be (some) lower bound for the number of variables occurring at most d times in the formula F.

Let I be a truth assignment for the variables of F. Let S be a satisfying assignment for F. We say that a variable v has the *S-correct* value in I, if $v[I] = v[S]$. Otherwise we say that v has the *S-incorrect* value in I (note that this value is uniquely defined since v is a Boolean variable). The variable v is *in a good position* w.r.t. formula F and assignment I, if it does not occur in any clause containing a variable which has the S-incorrect value in I (maybe except the variable v itself). Otherwise we say that v is *in a bad position*. We say that I is *changed locally* w.r.t. S if all variables that have the S-incorrect values in I, are in a good position and occur at most d times in the formula. We omit S in the above notation if its meaning is clear from the context.

Let i be an integer. We say that I satisfies the property $P_S(i)$ if $\mathrm{dist}(I, S) = i$ and I is changed locally w.r.t. S. Informally speaking, I satisfies the property $P_S(i)$ if it is obtained from S by changing the values of i variables such that

(1) each clause of F contains at most one of these variables;

(2) each of these variables occur at most d times in F.

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

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 $\mathrm{sat}(F, I^{(v)}) > \mathrm{sat}(F, I)$.
    If there are no such variables, choose at random a variable v such that $\mathrm{sat}(F, I^{(v)}) = \mathrm{sat}(F, I)$.
    If there are none of those, choose a variable v at random from the set of all variables occurring in F.
  - Flip the value of v in I.

Output "No".

Figure 2: Algorithm CSAT

Algorithm SimpleCSAT.
Input:
A formula F in CNF.
Output:
A truth assignment for the variables of F, or "No".

Method.

Pick an assignment I at random.

While I is not a satisfying assignment:

— Choose a variable v in F such that $\mathrm{sat}(F, I) < \mathrm{sat}(F, I^{(v)})$. If there are several possible choices, choose it at random. If there are no such variables, output "No".

— Assume $I = I^{(v)}$.

Output I.

Figure 3: Algorithm SimpleCSAT

We estimate the running time of Algorithm SimpleCSAT in two steps.

1. We prove that there are many assignments satisfying $P_S(i)$ and thus it is likely that Algorithm SimpleCSAT picks one.
2. We prove that it is likely that starting with an assignment satisfying $P_S(i)$, Algorithm SimpleCSAT stops in a polynomial time and outputs a satisfying assignment.

Remark 3.1 *S is the only assignment satisfying $P_S(0)$.*

Lemma 3.1 *Let S be a satisfying assignment for F. If an assignment I satisfies the property $P_S(i)$, then there are at most $id(k-1)$ variables v in a bad position that have the correct values in I.*

Proof. There are at most id clauses in F that contain variables having the incorrect values in I. These clauses contain at most $id(k-1)$ variables having the correct values in I. □

Lemma 3.2 *Let S be a satisfying assignment for F. If an assignment I satisfies the property $P_S(i-1)$ for $1 \le i \le 1 + \frac{\delta N}{(k-1)d+1}$, then there are at least $\delta N - (i-1)((k-1)d+1)$ variables v such that $I^{(v)}$ satisfies the property $P_S(i)$.*

Proof. There are at least $\delta N - (i-1)$ variables having the correct values in I and occurring at most d times in the formula. By Lemma 3.1 at most $(i-1)d(k-1)$ of them are in a bad position. The remaining $\delta N - (i-1) - (i-1)d(k-1)$ variables are exactly what we want. □

Lemma 3.3 *Let S be a satisfying assignment for F. There are at least*

$$\frac{\delta N(\delta N - ((k-1)d+1))(\delta N - 2((k-1)d+1)) \ldots (\delta N - (i-1)((k-1)d+1))}{i!}$$

assignments satisfying the property $P_S(i)$ for $1 \le i \le 1 + \frac{\delta N}{(k-1)d+1}$.

Proof. We prove this statement by the induction on i.
Base ($i = 1$). Trivial.
Step. Suppose the statement holds for $i-1$. By Lemma 3.2 we can map each assignment I satisfying $P_S(i-1)$ to at least $\delta N - (i-1)((k-1)d+1)$ assignments $I^{(v)}$ satisfying $P_S(i)$. Thus we have mapped each of

$$\frac{\delta N(\delta N - ((k-1)d+1))(\delta N - 2((k-1)d+1)) \dots (\delta N - (i-2)((k-1)d+1))}{(i-1)!}$$

assignments guaranteed by the induction hypothesis, to at least $\delta N-(i-1)((k-1)d+1)$ assignments. Since we have mapped at most i assignments into one assignment, the number of assignments satisfying $P_S(i)$ is at least

$$\frac{(\delta N - (i-1)((k-1)d+1))}{i} \cdot \cdot \frac{\delta N(\delta N - ((k-1)d+1)) \dots (\delta N - (i-2)((k-1)d+1))}{(i-1)!} = = \frac{\delta N(\delta N - ((k-1)d+1)) \dots (\delta N - (i-1)((k-1)d+1))}{i!}.$$

□

Lemma 3.4 *Let S be a satisfying assignment for F. If an assignment I satisfies the property $P_S(i)$ and a variable v has the incorrect value in S, then $I^{(v)}$ satisfies the property $P_S(i-1)$.*

Proof. All variables having the correct values in I, have the correct values in $I^{(v)}$. □

Lemma 3.5 *Let S be a satisfying assignment for F. If an assignment I is changed locally w.r.t. S, a variable v has the S-incorrect value in I, and $S^{(v)}$ is not a satisfying assignment for F, then* $\mathrm{sat}(F, I^{(v)}) > \mathrm{sat}(F, I)$.

Proof. W.l.o.g. we suppose $v[S] = True$. In the assignment I, the values of all variables in each of the clauses containing the literal $\overline{v}$, are correct. Thus, $I^{(v)}$ satisfies all these clauses, i.e. $\mathrm{sat}(F, I^{(v)}) \geq \mathrm{sat}(F, I)$.

Suppose $\mathrm{sat}(F, I^{(v)}) = \mathrm{sat}(F, I)$. Then I satisfies all clauses containing the literal v. Hence, the assignment $S^{(v)}$ satisfies the formula F, because I is changed locally w.r.t. S. □

Lemma 3.6 *Let S be a satisfying assignment for F. If Algorithm SimpleCSAT starts with an assignment I that satisfies the property $P_S(i)$, then the algorithm stops in a polynomial time and outputs a satisfying assignment with probability at least $((k-1)d+1)^{-i}$.*

Proof. There are i variables v having the S-incorrect values in I. By Lemma 3.5 for each such variable v, either $S^{(v)}$ is a satisfying assignment for F (suppose there are j such variables) or $\mathrm{sat}(F, I^{(v)}) > \mathrm{sat}(F, I)$. Now consider an assignment J that differs from S exactly in the values of the above mentioned j variables. Since I is changed locally, J is a satisfying assignment. For all $(i-j)$ variables v having the J-incorrect values in I, $\mathrm{sat}(F, I^{(v)}) > \mathrm{sat}(F, I)$.

Since I satisfies $P_J(i-j)$, by Lemma 3.1 there are at most $(i-j)d(k-1)$ variables u having the J-correct values in I, such that $\mathrm{sat}(F, I^{(u)}) > \mathrm{sat}(F, I)$. Thus, the algorithm chooses a variable having the J-incorrect value in I with probability at least $\frac{i-j}{i-j+(i-j)d(k-1)} = \frac{1}{(k-1)d+1}$. By Lemma 3.4 in this case the algorithm proceeds with processing an assignment having the property $P_J(i-j-1)$, etc. Thus, the algorithm stops after changing the values of at most i variables and outputs a satisfying assignment with probability at least $((k-1)d+1)^{-i}$. □

Lemma 3.7 *If a satisfiable formula F in k–CNF is given as input to Algorithm SimpleCSAT, and δN is the number of variables occurring at most d times in F, then Algorithm SimpleCSAT stops in a polynomial time and outputs a satisfying assignment with probability at least $2^{-(1-\frac{\delta}{(k-1)d+1})N}/p(N)$, where p is a polynomial.* □

Proof. Let S be a satisfying assignment for F. By Lemmas 3.3 and 3.6, the probability of outputing S after i flips is at least

$$\begin{aligned}
&\frac{\delta N(\delta N-((k-1)d+1))\ldots(\delta N-(i-1)((k-1)d+1))}{i!\cdot 2^N}((k-1)d+1)^{-i}\\
=\;&\frac{1}{2^N}\cdot\frac{\frac{\delta N}{(k-1)d+1}\cdot\frac{\delta N-((k-1)d+1)}{(k-1)d+1}\cdot\ldots\cdot\frac{\delta N-(i-1)((k-1)d+1)}{(k-1)d+1}}{i!}\\
\geq\;&\frac{\left\lfloor\frac{\delta N}{(k-1)d+1}\right\rfloor!}{2^N i!\left\lfloor\frac{\delta N}{(k-1)d+1}-i\right\rfloor!}.
\end{aligned}$$

We assume $i=\frac{1}{2}\left\lfloor\frac{\delta N}{(k-1)d+1}\right\rfloor$. Then, using the Stirling formula, the above probability is at least $2^{-(1-\frac{\delta}{(k-1)d+1})N}/p(N)$ for some polynomial p. □

By Lemma 3.7 the algorithm obtained by repeating Algorithm SimpleCSAT $2^{(1-\frac{\delta}{(k-1)d+1})N}p(N)$ times is a Monte Carlo algorithm for SAT with error probability

$$(1-2^{-\left(1-\frac{\delta}{(k-1)d+1}\right)N}/p(N))^{2^{\left(1-\frac{\delta}{(k-1)d+1}\right)N}p(N)}<1/e.$$

Thus we have proved the following theorem.

Theorem 3.1 *Given a formula in k–CNF, the Monte Carlo analog of Algorithm SimpleCSAT performs correctly with error probability at most $1/e$. Its worst-case running time is at most*

$$2^{\left(1-\frac{\delta}{(k-1)d+1}\right)N}q(N),$$

where N is the number of variables in the input formula, δN is the number of variables occurring at most d times in the input formula, q is a polynomial.

The result also holds for the original CSAT algorithm, since we may consider only the cases when it performs exactly as the Monte Carlo analog of Algorithm SimpleCSAT.

Corollary 3.1 *There exist polynomials p and q such that, given a formula in k–CNF with K clauses and N variables, δN of which occur at most d times, and values* `MAXFLIPS` $= K$ *and* `MAXTRIES` $= 2^{(1-\frac{\delta}{(k-1)d+1})N}p(N)$*, Algorithm CSAT returns a correct answer with probability at least $1-1/e$ independent on the input formula. Its worst-case running time is at most*

$$2^{\left(1-\frac{\delta}{(k-1)d+1}\right)N}q(N).$$

For formulas in k–CNF–d this gives us immediately the worst-case upper bound

$$2^{\left(1-\frac{1}{(k-1)d+1}\right)N}q(N).$$

(Approximate values of $1 - \frac{1}{(k-1)d+1}$ are given in Table 1, for example, CSAT finds a satisfying assignment for a formula in 3–CNF–3 in time $O\left(2^{0.858N}\right)$.) We cannot directly apply our arguments to arbitrary formulas in k–CNF. However, if the length of a formula is upper bounded by a linear function λN of the number N of its variables, a certain number of variables must occur in it rarely. Thus, by appropriate choice of δ, we can get a more or less good upper bound.

Let $\delta = \frac{1}{2}$. Since $\lambda \geq \delta + (d+1)(1-\delta)$, we have that $d \leq \frac{\lambda-1}{1-\delta} = 2(\lambda - 1)$ Therefore,

$$\frac{\delta}{(k-1)d+1} \geq \frac{1}{4(\lambda-1)(k-1)+2}$$

(approximate values of $1 - \frac{1}{4(\lambda-1)(k-1)+2}$ are given in Table 2). Thus, we have proved[2]

Corollary 3.2 *There exist polynomials p and q such that, given a formula in k–CNF of length at most λN, with K clauses and N variables, and values* `MAXFLIPS` $= K$, `MAXTRIES` $= 2^{(1-\frac{1}{4(k-1)(\lambda-1)+2})N} p(N)$, *Algorithm CSAT returns a correct answer with probability at least $1 - 1/e$ independent on the input formula. Its worst-case running time is at most*

$$2^{\left(1-\frac{1}{4(k-1)(\lambda-1)+2}\right)N} q(N).$$

However, if we augment Algorithm CSAT with pure literal elimination prior to processing the formula (we call this modified algorithm CSAT+P), then we can suppose that each literal occurs at least twice in the formula. In this case d can be estimated via δ from $\lambda \geq 2\delta + (d+1)(1-\delta)$, i.e. $d \geq \frac{\lambda-\delta-1}{1-\delta}$ (we assume $\delta < 1$). Let $M = (k-1)(\lambda-1)$. Assuming $\delta = \frac{M+1-\sqrt{(M+1)(M+1-k)}}{k}$, we have $\frac{\delta}{d} \geq \left(\frac{\sqrt{m+1}-\sqrt{m+1-k}}{k}\right)^2$. Approximate values of this bound for concrete values of k and λ are presented in Table 3; for example, CSAT+P solves the satisfiability problem for a formula with $L \leq 3N$ in time $O(2^{0.925N})$. More precisely, we have proved

Corollary 3.3 *Let λ be a positive number, k be a positive integer, $M = (k-1)(\lambda-1)$. There exist polynomials p and q such that, given a formula in k–CNF of length at most λN, with K clauses and N variables, and values* `MAXFLIPS` $= K$, `MAXTRIES` $= 2^{(1-\left(\frac{\sqrt{M+1}-\sqrt{M+1-k}}{k}\right)^2)N} p(N)$, *Algorithm CSAT returns a correct answer with probability at least $1 - 1/e$ independent on the input formula. Its worst-case running time is at most*

$$2^{\left(1-\left(\frac{\sqrt{M+1}-\sqrt{M+1-k}}{k}\right)^2\right)N} q(N).$$

4 Lower bounds for CSAT and GSAT

4.1 Formulas in general CNF

We now prove a lower bound for the running time of the algorithms CSAT and GSAT on formulas in CNF. We construct formulas F_N that are "hard" for the algorithms GSAT and CSAT. We then

[2] A slightly better bound can be proved by more accurate choice of δ and d. However, this bound looks very complicated and we do not present it here since

(1) our purpose is only to prove a less-than-2^n bound for CSAT, and

(2) below we prove a better bound for a very similar algorithm.

$k \backslash d$	3	6	9	12	15
3	0.858	0.924	0.948	0.960	0.968
4	0.900	0.948	0.965	0.973	0.979
5	0.924	0.960	0.973	0.980	0.984

Table 1: Approximate values of $1 - \frac{1}{(k-1)d+1}$.

$k \backslash \lambda$	3	6	9	12	15
3	0.945	0.977	0.985	0.989	0.992
4	0.962	0.984	0.990	0.993	0.995
5	0.971	0.988	0.993	0.995	0.996

Table 2: Approximate values of $1 - \frac{1}{4(\lambda-1)(k-1)+2}$.

$k \backslash \lambda$	3	6	9	12	15
3	0.925	0.974	0.984	0.989	0.991
4	0.948	0.983	0.990	0.993	0.994
5	0.961	0.987	0.992	0.995	0.996

Table 3: Approximate values of $1 - \left(\frac{\sqrt{m+1}-\sqrt{m+1-k}}{k}\right)^2$.

prove that it takes time of the order 2^N for CSAT or GSAT to find a satisfying assignment for F_N. For an integer $N \geq 6$, the formula F_N contains N Boolean variables and consists of $4N+1$ clauses of three types (see Fig. 4). 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 together with $\{x_1, \overline{x_2}\}$ it implies that x_1 must have the value $True$ in every satisfying assignment). Hence, the formula F_N has only one satisfying assignment $S_N = \{x_1, x_2, \ldots, x_N\}$ (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 of the third type containing non-negated x_i. Each of these clauses also contains the negations of the next (modulo N) $N-4$ variables and the negations of two of the remaining 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, the assignments that differ from the satisfying assignment by the values of exactly two variables, form an "insurmountable ring" around the satisfying assignment. Hence, in this case the satisfying assignment cannot be found until the algorithm restarts 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 4.1 *Let I be an assignment for F_N ($N \geq 6$), such that $\mathrm{dist}(I, S_N) = 2$. Then for every variable v if $\mathrm{dist}(I^{(v)}, S_N) = 1$, then $\mathrm{sat}(F_N, I^{(v)}) < \mathrm{sat}(F_N, I)$.*

Proof. Let I differ from S_N by the values of variables $v = x_i$ and x_j. Note that $I^{(v)}$ satisfies the 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: $|(i-j) \bmod N| \neq 1$.

In this case I satisfies $N-2$ clauses of the first type, the clause $\{x_1, x_2\}$ and $3N-1$ or $3N$ clauses of the third type, i.e. $4N-1$ or $4N-2$ clauses in total.

Case 2: $|(i-j) \bmod N| = 1$, but $\{i, j\} \neq \{1, 2\}$.

In this case 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: $\{i, j\} = \{1, 2\}$.

In this case 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 4.2 *Let I be an assignment for F_N ($N \geq 6$), such that $\mathrm{dist}(I, S_N) = 2$. Then there exists a variable v such that $\mathrm{dist}(I^{(v)}, S_N) = 3$ and $\mathrm{sat}(F_N, I) = \mathrm{sat}(F_N, I^{(v)})$.*

Proof. Let I differ from S_N by the 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.

(Such v exists 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. □

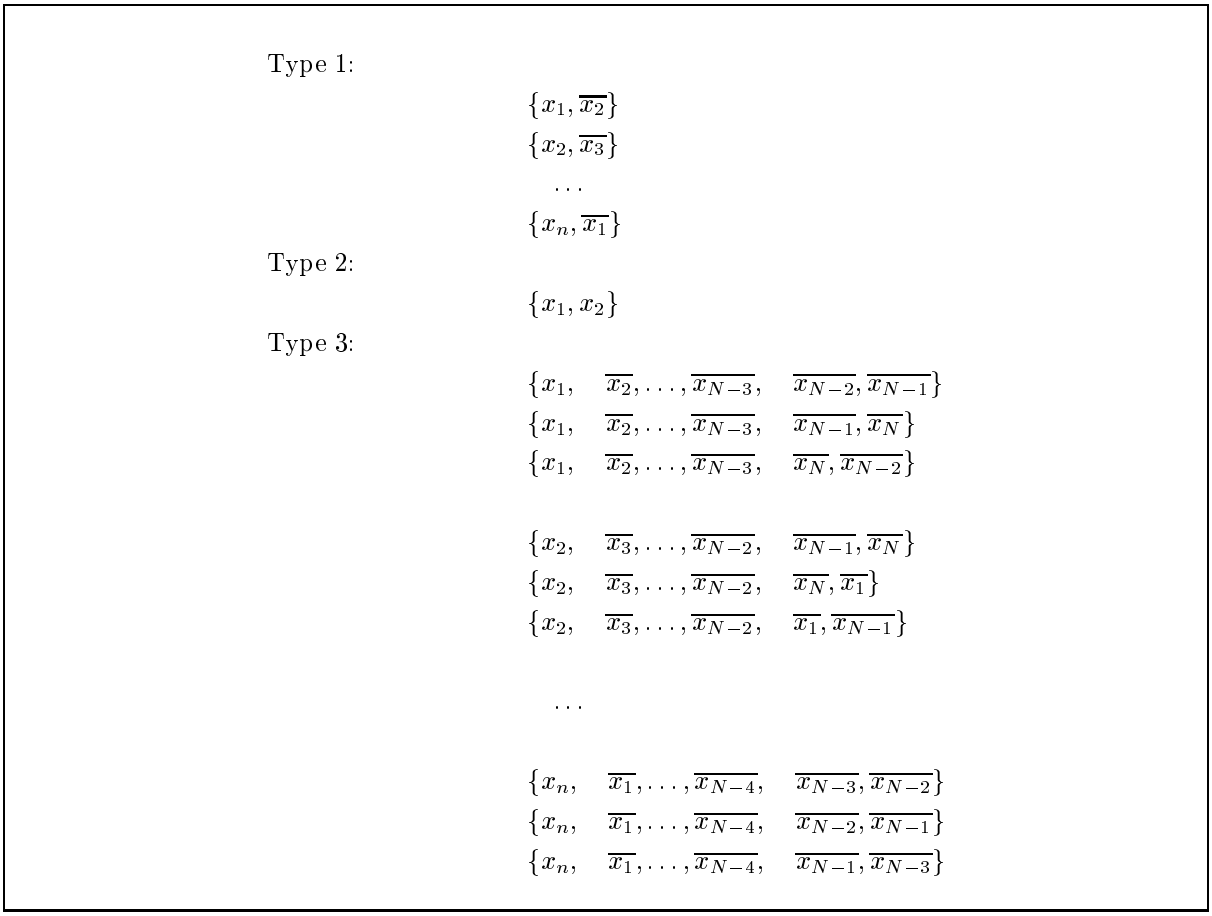

Figure 4: Formula F_N

Theorem 4.1 *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. Suppose that CSAT or GSAT starts its search from an initial assignment that differs from S_N by the values of *at least two* variables. We now prove that in this case the algorithm cannot find a satisfying assignment until it chooses another initial assignment.

Suppose that the algorithm outputs a satisfying assignment without choosing another initial assignment. Then the algorithm must encounter an assignment that differs from S_N by the values of *exactly two* variables. Let I be the last such assignment (i.e., after that the algorithm encounters only assignments that differ from S_N by the value of *exactly one* variable, and, finally, the satisfying assignment S_N).

By Lemma 4.2, at the next step the algorithm must flip the value of a variable v such that the number of satisfied clauses will *not decrease*, i.e. $\mathrm{sat}(F_N, I^{(v)}) \geq \mathrm{sat}(F_N, I)$. It follows that the assignment $I^{(v)}$ cannot differ from S_N by the value of *exactly one* variable since by Lemma 4.1 every such assignment satisfies strictly less clauses of F_N than I does.

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. The probability to choose one of these assignments is exactly $(N+1)/2^N$, i.e. the algorithm must perform $\Omega(2^N/(N+1))$ iterations to achieve a constant probability of error. □

4.2 Formulas in 3–CNF

We now prove a lower bound for the running time of the algorithms CSAT and GSAT on formulas in 3–CNF. The idea is the same as for formulas in general CNF, but the "insurmountable ring" around the satisfying assignment is not at the distance 2, but at the distance $\lceil \frac{N}{3} \rceil + 6$ from the satisfying assignment.

We now describe the "hard" formula G_N for an integer $N \geq 12$. It contains N variables and $\frac{N(N-1)(N-2)}{2} + N + 1$ clauses of three different types. Clauses of the first two types are the same as in the formula F_N described in the previous section. These clauses imply that G_N has the only satisfying assignment $S_N = \{x_1, x_2, \ldots, x_N\}$. Similarly to the previous section, clauses of the third types are satisfied by S_N and are intended for "misleading" the algorithm. However, these clauses differ from the ones in the formula F_N since we are limited to clauses containing at most three literals. The clauses of the third type are all the $\frac{N(N-1)(N-2)}{2}$ possible clauses $\{x_i, \overline{x_j}, \overline{x_k}\}$ consisting of a variable x_i and the negations of two other variables x_j and x_k.

Let I be an assignment such that $\mathrm{dist}(I, S_N) \geq \frac{N}{3} + 6$. Let I and S_N assign different values to variable v and identical values to variable w. Easy calculation shows that flipping the value of the variable v leads to decrease in the number of satisfied clauses, and flipping the value of the variable w leads to increase in the number of satisfied clauses. Hence, CSAT or GSAT can output a satisfying assignment only if it chooses an initial assignment at the distance at most $\frac{N}{3} + 5$ from the satisfying assignment S_N. There are at most $p(N)2^{H(\frac{1}{3})N}$ such assignments, where $H(\varepsilon) = -\varepsilon \log_2 \varepsilon - (1-\varepsilon)\log_2(1-\varepsilon)$ is the binary entropy function and p is some polynomial. Since $H(\frac{1}{3}) < 0.9183$, the lower bound follows:

Theorem 4.2 *Let $N \geq 12$. The algorithms CSAT and GSAT cannot find a satisfying assignment for G_N in expected time less than $\Omega(2^{0.0817N})$.*

5 Conclusion and further work

In this paper we studied the worst-case behaviour of the local search algorithms CSAT and GSAT. We proved a worst-case upper bound of the form $O\left(2^{\alpha N}\right)$ for the running time of CSAT on linear-size formulas in k–CNF (N is the number of variables, $0 < \alpha < 1$). On the other hand, we proved the lower bound of the order 2^N for formulas in CNF and the lower bound of the order $2^{0.0817N}$ for formulas in 3–CNF. In fact, the presented lower bound for formulas in 3–CNF holds also for GSAT and many other local search algorithms (e.g., for those that combine local search with random walk) and generalizes to k–CNF.

It would be interesting to obtain lower bounds for the running time of CSAT and GSAT on linear-size formulas, as well as to find more tight bounds for formulas in k–CNF of arbitrary size and for other interesting classes of formulas. Also, a wide variety of local search algorithms remains unstudied. It would be interesting to find classes of formulas on which these algorithms have better worst-case upper bounds than other (e.g., resolution-like) algorithms.

References

[1] Cook S., Reckhow R. On the lengths of proofs in the propositional calculus (preliminary version). In: Conference Record of Sixth Annual ACM Symposium on Theory of Computing (1974) 135–148

[2] Dantsin, E.: Two systems for proving tautologies, based on the split method. In: Journal of Soviet Mathematics **22** (1983) 1293–1305

[3] Frieze, A., Suen, S.: Analysis of two simple heuristics on a random instance of k–SAT Journal of Algorithms **20** (1996) 312–355

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

[5] 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

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

[7] 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

[8] Hirsch, E. A.: Two new upper bounds for SAT. In: Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (1998) 521–530. A journal version is submitted to this volume

[9] 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

[10] Kamath, A., Motwani, R., Palem, K., Spirakis, P.: Tail Bounds for Occupancy and the Satisfiability Threshold Conjecture. Random Structures and Algorithms **7** (1995) 59–80

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

[12] Kullmann, O.: New methods for 3–SAT decision and worst-case analysis. To appear in: Theoretical Computer Science (1998) (First version announced in Abstracts of contributed papers of the Logic Colloquium '93.)

[13] McAllester, D., Selman, B., Kautz, H.: Evidence for invariants in local search. In: Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference Providence, RI, AAAI Press (1997) 321–326

[14] Mitchell, D. G., Selman, B., Levesque, H.: Hard and Easy Distributions of SAT Problems. In: Proceedings of the Tenth National Conference on Artificial Intelligence, San Jose, California, AAAI Press/MIT Press (1992) 459-465

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

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

[17] Papadimitriou, Ch. H.: Computational Complexity. Addison-Wesley (1994)

[18] Paturi, R., Pudlak, P., Saks, M. E., Zane, F.: An Improved Exponential-time Algorithm for k–SAT. In: Proceedings of the 39th Annual Symposium on Foundations of Computer Science (1998) 628–637

[19] Schiermeyer, I.: Pure literal look ahead: An $O\left(1.497^n\right)$ 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**

[20] 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)

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

[22] 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

[23] 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

[24] Urquhart, A.: The Complexity of Propositional Proofs. Bulletin of Symbolic Logic. Vol. **1** (N**4**) (1995) 425–467