An Improved $\widetilde{\mathcal{O}}(1.234^m)$-Time Deterministic Algorithm for SAT

Masaki Yamamoto

Tokyo Institute of Technology
masaki.yamamoto@is.titech.ac.jp

Abstract. We improve an upper bound by Hirsch on a deterministic algorithm for solving general CNF satisfiability problem. With more detail analysis of Hirsch's algorithm, we give some improvements, by which we can prove an upper bound $\widetilde{\mathcal{O}}(1.234^m)$ w.r.t. the number m of input clauses, which improves Hirsch's bound $\widetilde{\mathcal{O}}(1.239^m)$.

1 Introduction

During the past decades, many algorithms for Boolean satisfiability problems have been proposed, and some of them were proved to improve the nontrivial worst-case upper bounds for the problems. Such worst-case analysis was initiated by Monien and Speckenmeyer [6] for k-SAT. Given a formula F in conjunctive normal form (in short, a CNF formula), the problem of deciding whether F is satisfiable is called SAT; for CNF formulas consisting of clauses at most k-literals (in short, k-CNF formulas), the problem is called k-SAT. Since the work of Monien and Speckenmeyer, k-SAT problems, in particular 3-SAT, have been studied intensively, and various interesting algorithms have been proposed. On the other hand, not so much improvements have been done for the general SAT problem, the satisfiability problem for general CNF formulas. Note that CNF formulas with no clause size restriction are useful for expressing some combinatorial problems such as graph problems. In this paper, we focus on such general CNF formulas and propose an improved algorithm for the general SAT. Throughout this paper, we denote by n and m, the number of variables and the number of clauses, respectively.

We briefly summarize a history of improving the bounds for SAT: randomized algorithms and deterministic algorithms, respectively. We below give only the exponential parts of the bounds, omitting polynomial factors.

1.1 Randomized Algorithms for SAT

The first nontrivial upper bound was given by Pudlák [7]: $2^{n-0.5\sqrt{n}}$. A slightly better bound was given by Dantsin et al. [1]: $2^{n-0.712\sqrt{n}}$. These are bounds with respect to the number of variables. Schuler gave an algorithm with respect to the number of variables and clauses [8]: $2^{n(1-1/\log 2m)}$. This bound is better than $2^{n-c\sqrt{n}}$ for any constant c when $m = o(2^n)$.

X. Deng and D. Du (Eds.): ISAAC 2005, LNCS 3827, pp. 644–653, 2005.

1.2 Deterministic Algorithms for SAT

The first nontrivial upper bound was given by Dantsin et al. [1]: $2^{n(1-2\sqrt{1/n \log m})}$. A better algorithm was given by Dantsin and Wolpert [2]: $2^{n(1-1/\log 2m)}$. This was obtained by derandomizing Schuler's algorithm [8]. These bounds are asymptotically 2^n as m gets large. Hirsch gave an algorithm of running time: $2^{0.30897m} \approx 1.239^m$. This works better than the above algorithms when $m < n/0.30897$.

In this paper, we improve on Hirsch's algorithm [5], and obtain a better algorithm with 1.234^m running time, which has not been improved for a few years. The basic approach of our algorithm is the same as Hirsch's; in fact, our algorithm is almost the same. By more careful analysis, we guarantees that this almost the same algorithm indeed achieves the desired upper bound. The advantage of our analysis is to guarantee some paths of the recursion tree of an execution reaching a trivial formula while the recursion tree by Hirsch's analysis is of depth only two. See Fig. 3.

2 Preliminaries

We give some basic notions and notations, and briefly review how we analyze the running time of splitting algorithms. Then we present Hirsch's algorithm [5], which we will improve in the next section.

2.1 Basic Notions and Notations

Let X be a finite set of Boolean variables. A *literal* is a variable $x \in X$ or its negation $\bar{x}$. A *clause* is a disjunction of literals. We alternatively regard a clause as a set of literals. The empty clause is interpreted as `false`. The size of a clause C (denoted by $|C|$) is the number of literals in C. A k-clause is a clause of size k. A k^+-clause and a k^--clause are clauses of size at least k and at most k, respectively. A *conjunctive normal form* (CNF for short) formula is a conjunction of clauses. Again, we alternatively regard a CNF formula as a set of clauses. The empty formula is interpreted as `true`. The size of a CNF formula F (denoted by $|F|$) is the number of clauses in F. A *truth assignment* (*assignment* for short) to X is a mapping from X to $\{\texttt{true}, \texttt{false}\}$. We denote `true` by 1 and `false` by 0. A clause is said to be *satisfied* by an assignment if at least one literal in the clause is assigned 1 by the assignment. A formula is said to be *satisfied* by an assignment if every clause in the formula is satisfied by the assignment. A formula is said to be *satisfiable* if there exists an assignment by which the formula is satisfied, otherwise, *unsatisfiable*. Given a CNF formula F, SAT is a problem of deciding whether F is satisfiable.

Let F be a CNF formula, and l be a literal in F. We denote by $F|_{l=1}$, a formula obtained from F by an assignment $l = 1$, that is, a formula transformed from F by eliminating clauses which contain l and by eliminating $\bar{l}$. Formula $F|_{l=0}$ is defined similarly. For any literal l, if it occurs positively (i.e., as l) occurs i times

and occurs negatively (i.e., as $\bar{l}$) j times in F, then we say that l is a (i,j)-*literal*. It is equivalent to saying that $\bar{l}$ is a (j,i)-*literal*. We sometimes call a (i,j)-literal a i-*literal* for short. Also by, e.g., "(i^+,j^-)-literal", we mean a (i',j')-literal for any $i' \geq i$ and $j' \leq j$.

As many heuristic algorithms for SAT, our algorithm (as well as Hirsch's algorithm) is based on "splitting" [4,3]: Given a CNF formula F. Choose an appropriate literal l (depending on the heuristics) in F, and split F on l, that is, obtain two sub-formulas $F|_{l=1}$ and $F|_{l=0}$. Then, we have that F is satisfiable iff one of $F|_{l=1}$ and $F|_{l=0}$ is satisfiable. Hence, the splitting algorithm can decide whether F is satisfiable by recursively calling this procedure above.

Sub-formulas produced by splitting can be simplified by simple transformation rules: "pure literal elimination", and "unit clause propagation" [4,3]. The pure literal elimination is to assign $l = 1$ for any *pure literal*, a literal l such that $\bar{l}$ never occurs in F. The unit clause propagation is to assign $l = 1$ if F has a clause consisting only this literal l. Another standard rule for simplifying formulas is "resolution". For any literal, let C and C' be clauses such that $l \in C$ and $\bar{l} \in C'$ [1]. Then, we call $(C \cup C') \backslash \{l, \bar{l}\}$ the *resolvent* by l of C' and C. Given a formula F and a literal l in F, the resolution on l is the following procedure: (1) add to F all resolvents by l, and (2) eliminate from F all clauses containing l or $\bar{l}$. Note that all these three operations do not change the satisfiability of a given formula F; that is, letting F' be a formula obtained by one of these operation to F, we have F is satisfiable iff F' is satisfiable [4].

The pure literal elimination and unit clause propagation always decrease the number of variables as well as the formula size, i.e., the number of clauses of a formula. On the other hand, the resolution does not necessarily decreases formula size while the number of variables does decrease. There are, however, we can guarantee some formula size reduction; when a resolution is made on some 1-literal, then the number of clauses gets decreased. Such a resolution is called *1-literal resolution*.

Hirsch's algorithm makes use of one more simplification rule, which is based on the *black-and-white literal principle* [**?**]. For a given formula F, suppose that all $(2,3)$-literals appear with some $(3,2)$-literals in F's clauses; that is, every clause containing a $(2,3)$-literal also has some $(3,2)$-literal. Then we can simply assign `false` to all $(2,3)$-literals, which does not change the satisfiability of F. This is because by this assignment, any clause containing $(2,3)$-literal is satisfied by some co-existing $(3,2)$-literal that is the negation of some $(2,3)$-literal and hence is assigned `true`. We call this simplification $(2,3)$-*literal elimination by the black-and-white literal principle*. Though the case where this simplification is applicable is rare, it removes one special case, which helps us to design a splitting algorithm.

2.2 Analysis of the Running Time of Splitting Algorithms

An execution of a splitting algorithm can be considered as a *branching tree*, whose nodes are labelled with formulas. That is, given a formula F and a variable x

[1] In the standard definition of resolution, it is also required that l is the only literal such that $l \in C$ and $\bar{l} \in C'$. But here we may remove this restriction.

which an execution of the splitting algorithm is split on. Then, in the branching tree, a node labelled with F has two children labelled with $F|_{x=1}$ and $F|_{x=0}$, respectively. We abuse a formula as a node labelled with the formula. Note that the running time of a splitting algorithm is a polynomial of input length times the number of leaves of the branching tree. Let F be a formula labelling a node of a branching tree, and $F_1, \cdots, F_s$ be its children. A *branching vector* of a node is an s-tuple $(t_1, \cdots, t_s)$, where t_i is a positive number bounded by $|F| - |F_i|$. The *characteristic polynomial* of a branching vector $\boldsymbol{t}$ is defined by $h_{\boldsymbol{t}}(x) = 1 - \sum_{i=1}^{s} x^{-t_i}$. Then, the equation $h_{\boldsymbol{t}} = 0$ has exactly one positive root, which means that it is the largest root of the equation. We denote this root by $\tau(\boldsymbol{t})$, and call it the *branching number* at the node of F. The branching number of a branching tree is the largest branching numbers among all its nodes, denoted by $\tau_{\max}$. The following lemma proved by Kullmann and Luckhardt allows us to estimate the number of leaves in a branching tree. (See [5] for the details.)

Lemma 1. Let $\tau_{\max}$ be the branching number of a branching tree representing an execution of a splitting algorithm which has taken as input a CNF formula F with m clauses. Then the number of leaves in the branching tree does not exceed $(\tau_{\max})^m$.

2.3 Hirsch's Algorithm

We now review Hirsch's algorithm [5], which we denote HIRSCH$(\cdot)$ in this paper. Hirsch's algorithm (see below for the description[2]) makes two types of splits. For both splits, split literals are chosen among all possible ones so that a branching number satisfies some specified bounds[3]. For each splitting (i.e., after fixing split literal value(s)), the simplification explained above is made on formulas. For any formula F, let REDUCE(F) be a formula obtained by applying to F one of the following operations until no further simplification is made: pure literal elimination, unit clause propagation, resolution such that the number of clauses doesn't increase, and $(2,3)$-literal elimination by the black-and-white literal principle.

Function HIRSCH(F)
If $F = \emptyset$ (meaning F is satisfiable), then return `true`.
If $\emptyset \in F$ (meaning F is unsatisfiable), then return `false`.

(S1) Splitting into two sub-problems.
For any literal l in F, consider the following split.

$$F_1 = \text{REDUCE}(F|_{l=1}) \quad \text{and} \quad F_0 = \text{REDUCE}(F|_{l=0}).$$

[2] For our later explanation, we state a description slightly different from the original one; but it is not so hard to check that they are equivalent.

[3] Intuitively, splitting is better if more clauses are removed with smaller branching variables, and the optimal one should be chosen. But for our target upper bound, we only have to choose one satisfying the specified bounds.

If there exists some literal l with branching number $\tau(|F| - |F_1|, |F| - |F_0|) \leq \tau(3,4)$, then execute this split, i.e., call HIRSCH(F_1) and HIRSCH(F_0) and return HIRSCH(F_1) $\vee$ HIRSCH(F_0).

(S2) (If the condition of (S1) fails, then) Splitting into four sub-problems. For any literal l in F, and any literals l' and l'' in $F|_{l=1}$ and $F|_{l=0}$ respectively, consider the following split.

$$F_{11} = \text{REDUCE}(F|_{l=1,l'=1}), \quad F_{10} = \text{REDUCE}(F|_{l=1,l'=0}), \quad \text{and}$$
$$F_{01} = \text{REDUCE}(F|_{l=0,l''=1}), \quad F_{01} = \text{REDUCE}(F|_{l=0,l''=0}).$$

Then, as it is shown in [5], there must be some split satisfying $\tau(|F| - |F_{11}|, |F| - |F_{10}|, |F| - |F_{01}|, |F| - |F_{00}|) \leq \tau(6,7,6,7)$. Choose one of such splits, call HIRSCH(F_{11}), HIRSCH(F_{10}), HIRSCH(F_{01}), and HIRSCH(F_{00}), and return `true` iff one of these calls yields `true`.

Note that $\tau(6,7,6,7) < \tau(3,4)$, and in [5], it is shown that branching number $\leq \tau(6,7,6,7)$ can be achieved at each split in the recursive execution HIRSCH(F) for any formula F. This proves the following bound.

Theorem 1. (Hirsch [5]) Given a CNF formula F with m clauses as input. The algorithm HIRSCH(F) decides whether F is satisfiable in time $\tau(6,7,6,7)^m = 2^{0.30897m} \approx 1.239^m$.

3 Improving on Hirsch's Algorithm

In this section, we give an algorithm and prove that it runs in time 1.234^m. Our algorithm is almost the same as Hirsch's, with some very minor modification on the function HIRSCH($\cdot$). What is important is that by careful analysis we prove that this almost the same algorithm indeed achieves the desired upper bound. Let us define some terminologies to give an overview of our analysis.

Definition 1. Let F be a CNF formula, and let l and l' be literals of F such that a clause of F contains l and l' both. We say that l and l' are *coincident* (alternatively say that l is *coincident* with l') if there is another clause in F containing l and l' both.

Definition 2. Let F be a CNF formula. We say that F is *normal* if there are only $(3^-, 3^-)$-literals in F, and there is no pair of coincident literals in F.

Consider any recursive execution of HIRSCH($\cdot$). Suppose that a formula F input to HIRSCH(F) consists of only $(3,3)$-literals. If there exists a pair (l, l') of coincident literals in F, we can proceed into step (S1) (not into (S2)), where we split F on l and resolve $F|_{l=1}$ on l' into a formula with size one fewer. Thus, the branching number at F is $\tau(3,4)$. Otherwise, i.e., if there exists no pair of coincident literals in F, that is, F is normal with no 2^--literals, then we proceed into step (S2), where we split F on an arbitrary literal l. and then we choose

an appropriate literal for each of $F_1 = F|_{l=1}$ and $F_0 = F|_{l=0}$, which has the branching number $\tau(3,4)$. Since each assignment $l = 1$ and $l = 0$ eliminates three clauses, we can obtain F has the branching number $\tau(6,7,6,7)$. This analysis of the worst case is exactly Hirsch's, and the recursion tree corresponding to the analysis is of depth two. See the tree enclosed with the dotted line in Fig. 3. We will show that in the worst case, for each root node of F_1 and F_0 there exists at least one path from the root to a leaf such that every node of the path has the branching number $\tau(3,4)$. See the recursion tree below in Fig 3. The following lemma guarantees such a bound.

Lemma 2. Given a CNF formula H with h clauses. Suppose that H is normal, and contains at least one 2^--literal. Then, one of the followings is satisfied: (a) H has a 1^--literal, (b) H has one of the following branching numbers: $\tau(4,4)$ and $\tau(3,5)$, and (c) we have sub-formulas H' with $|H'| = h - 3$ and H'' with $|H''| = h - 4$ (meaning H has the branching number $\tau(3,4)$) such that H' and H'' are two children of H and at least one of H' and H'' is normal with a 2^--literal.

This lemma works in our algorithm as follows. Consider the previous formulas F_1 and F_0, which are normal with a 2^--literal. Then, we can apply the lemma to F_1 and F_0, and split each of F_1 and F_0 as (a), (b), or (c) of the lemma. The precise algorithm of the splitting is stated in the proof of the lemma. (We denote our algorithm by HIRSCH$'(\cdot)$.) According to the lemma, if H only has the branching number $\tau(3,4)$, then at least one of the two children has the branching number $\tau(3,4)$. Furthermore, this recursively repeats to a leaf (meaning a trivial formula), or until a node with the branching number $(3,5)$ or $(4,4)$ (or even better) found. We call such a path $(3,4)$-*path*. It is not hard to see that the worst case of possible branching trees is a tree as shown below: a node with the branching number $\tau(3,4)$ continues to grow along the branch where four clauses are eliminated.

We call such a tree structure in Fig. 3 *the worst case branching tree*. In the branching tree, a black node represents a formula F at which the worst case branching tree resume, and a white node represents a formula of an inner node of the worst case branching tree. The number which an edge has in the figure is the number of eliminated clauses in the transformation from a parent node to its child node.

The proof of the lemma

We first assume that there is no 1^--literal in H. (We have excluded (a).) Thus, we only have the following types of literals in H: $(3,3)$-literals, $(2,3)$-literals, $(3,2)$-literals, and $(2,2)$-literals.

We first consider the case that there exists a $(2,3)$-literal in H. Let x be a $(2,3)$-literal. We further consider the following sub-cases:

(1) : x occurs with a $(3,3)$-literal in any clause
(2) : x occurs with a $(2,2^+)$-literal in a 2-clause
(3) : x occurs with only $(2,2^+)$-literals in a 3^+-clause

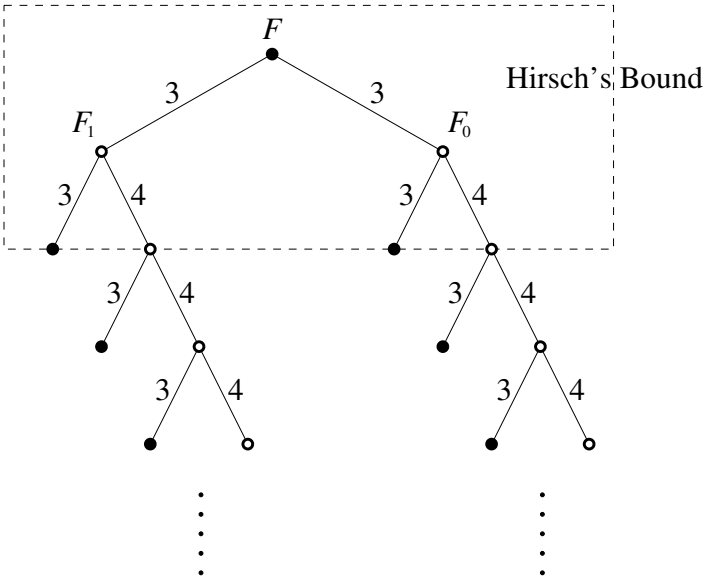

Fig. 1. The branching tree in the worst case

Let y be a $(3,3)$-literal of (1). Branching on y, the assignment $y = 1$ makes x become a 1-literal (since exactly one occurrence of x is eliminated because of no coincidence). Thus, that decreases at least one *additional* clause by the resolution on x, and this branch $y = 1$ totally decreases at least four clauses. Let $H'' = \mathrm{res}_x(H|_{y=1})$ where $\mathrm{res}_w(W)$ is a formula derived from W by the resolution on w. On the other hand, the assignment $y = 0$ eliminates three clauses containing $\bar{y}$, that results in $H' = H|_{y=0}$. It is clear that H' is normal (since H' is obtained from H just by eliminating literals and clauses). Suppose (on the contrary to (c)) that H' (as well as H'') has no 2^--literals, which means that H' consists of only $(3,3)$-literals. On this assumption, clauses of H containing $\bar{y}$ must have contained a 1-literal since any literal but $(1,1)$-literals, $(1,0)$-literals, and $(0,1)$-literals becomes a 2^--literal in H' (because of no coincidence). This is a contradiction to our assumption (which is there is no 1-literal in H). Thus, H' is normal with a 2^--literal. (This case satisfies (c).)

Let y be a $(2,2^+)$-literal of (2), i.e., the 2-clause C is $x \vee y$. Branching on x, the assignment $x = 1$ makes y become a 1-literal. Thus, this branch $x = 1$ totally decreases at least three clauses. (Let $H' = \mathrm{res}_y(H|_{x=1})$.) On the other hand, the assignment $x = 0$ makes C become a unit clause, i.e., $C = (y)$, forcing $y = 1$. Thus, the assignment $x = 0$ and $y = 1$ eliminates at least four clauses (three clauses containing $\bar{x}$ and C itself), that results in $H'' = H|_{x=0,y=1}$. It is clear that H'' is normal. If the other occurrence of y doesn't occur with $\bar{x}$ in H, then the assignment $x = 0$ and $y = 1$ eliminates five clauses ($|H''| = h - 5$), therefore, H has the branching number $\tau(3,5)$. Otherwise (i.e., the other occurrence of y occurs with $\bar{x}$ in H), the assignment $x = 0$ and $y = 1$ eliminates exactly four

clauses. Suppose (again on the contrary to (c)) that H'' (as well as H') has no 2^--literals. On this assumption, clauses of H containing $\bar{x}$ must have contained a 1-literal. This is a contradiction to our assumption. Thus, H'' is normal with a 2^--literal. (This case satisfies (c).)

Let y and z be those $(2, 2^+)$-literals of (3), i.e., the 3^+-clause C is $x \vee y \vee z \vee \cdots$. Branching on x, the assignment $x = 1$ makes y and z become 1-literals. Thus, that decrease at least additional two clauses by each resolution on y and z (because of no coincidence), and this branch $x = 1$ totally decreases at least four clauses. (Let $H'' = \mathrm{res}_{y,z}(H|_{x=1})$.) On the other hand, the assignment $x = 0$ eliminates three clauses containing $\bar{x}$, that results in $H' = H|_{x=0}$. By the same argument as the previous, we can conclude that H'' is normal with a 2^--literal. (This case satisfies (c).)

If there is no such $(2, 3)$-literal x as (1), (2), or (3) above, that means, each $(2, 3)$-literal occurs with some $(3, 2)$-literal, then we can assign true to all of $(3, 2)$-literals with no effect on satisfiability of H. (This is by the black-and-white literals principle.)

We next consider the case that there is no $(2, 3)$-literal in H, which means that there are only $(2, 2)$-literals and $(3, 3)$-literals in H. Let x be a $(2, 2)$-literal. If x occurs with a $(3, 3)$-literal in any clause, it is the same case as (1). Otherwise, that is, every $(2, 2)$-literal occurs with only $(2, 2)$-literals, let C_1, C_2 be clauses containing x, and let D_1, D_2 be clauses containing $\bar{x}$. We consider the following sub-cases:

(4) : for one of C_i, $|C_i| \geq 3$ and for one of D_i, $|D_i| \geq 3$
(5) : $|C_1| = |C_2| = 2$ and $|D_1| = |D_2| = 2$
(6) : either $|C_1| = |C_2| = 2$ or $|D_1| = |D_2| = 2$

For (4), let $|C_1| \geq 3$ and $|D_1| \geq 3$, i.e., $C_1 = (x \vee y \vee z \vee \cdots)$ for some literals y, z, and $D_1 = (\bar{x} \vee y' \vee z' \vee \cdots)$ for some literals y', z'. Branching on x, the assignment $x = 1$ makes y and z become 1-literals. Thus, this branch $x = 1$ totally decreases at least four clauses. The assignment $x = 0$ is the same as $x = 1$. Therefore, H has the branching number $\tau(4, 4)$.

For (5), let $C_1 = (x \vee y)$, $C_2 = (x \vee z)$ for some literals y, z, and $D_1 = (\bar{x} \vee y')$, $D_1 = (\bar{x} \vee z')$ for some literals y', z'. Branching on x, the assignment $x = 1$ makes D_1 and D_2 become unit clauses, i.e., $D_1 = (y')$ and $D_2 = (z')$, forcing $y' = 1$ and $z' = 1$. Thus, this branch $x = 1$ totally decreases at least four clauses. The assignment $x = 0$ is the same as $x = 1$. Therefore, H has the branching number $\tau(4, 4)$.

For (6), let $|C_1| = |C_2| = 2$, i.e., $C_1 = (x \vee y)$ and $C_2 = (x \vee z)$ for some literal y, z. Note that for at least one of D_1 and D_2, the size is at least three. Branching on x, the assignment $x = 1$ makes y and z become 1-literals. That only guarantees to decrease at least one additional clause by the resolution on y or z because the other occurrences of y and z could be in the same clause. Thus, this branch $x = 1$ totally decreases at least three clauses. On the other hand, the assignment $x = 0$ makes C_1 and C_2 become unit clauses, i.e., $C_1 = (y)$ and $C_2 = (z)$, forcing $y = 1$ and $z = 1$. If one of the other occurrences of y and z doesn't occur with $\bar{x}$, this branch $x = 0$ totally decreases at least five clauses.

Then, H has the branching number $\tau(3,5)$. Otherwise, i.e., the other occurrences of y and z both occur with $\bar{x}$, it only guarantees that this branch $x=0$ totally decreases at least four clauses. However, there should be another $(2,2)$-literal w occurring with $\bar{x}$ (since $|D_1| \geq 3$ or $|D_2| \geq 3$). The literal w becomes a 1-literal by $x=0$ and $y=z=1$. Thus, the assignment $x=0$ decreases at least one more additional clause by the resolution on w, and this branch $x=0$ totally decreases at least five clauses. Therefore, H has the branching number $\tau(3,5)$. ■

Remark 1. We should apply REDUCE$(\cdot)$ where the resolution which doesn't decrease the number of clauses is prohibited, to the transformation between formulas on the $(3,4)$-*path*. This is because otherwise, the resolution could spoil the coincidence of formulas while the number of clauses does not decrease by that resolution.

Lemma 3. The number of leaves in the worst case branching tree shown in Fig. 3 whose root is a formula with size m, is at most $O(1.234^m)$.

Proof. Let $T(m)$ be the number of leaves in the sub-tree whose root is a black node with m clauses. Let $S(m)$ be the number of leaves in the sub-tree whose root is a white node with m clauses. Then, we have recurrent equations:

$$\begin{aligned} T(m) &= 2S(m-3) \\ S(m-3) &= T(m-6) + S(m-7) \end{aligned}$$

From two equations above, we obtain $T(m) = O(1.234^m)$. (This comes from the following calculation: We first obtain $S(m) = S(m-4) + 2S(m-6)$ from the equations, and then the characteristic polynomial $1 - 1/x^4 - 2/x^6 = 0$ is derived from the recurrent equation. This polynomial corresponds to the branching number $\tau(4,6,6) \approx 1.234$, which means $S(m) = O(1.234^m)$.) ■

Theorem 2. Given a CNF formula F with m clauses. The upper bound of the running time of HIRSCH$'(F)$ is $\widetilde{\mathcal{O}}(1.234^m)$.

4 Conclusion

We have shown the bound 1.234^m for m clauses by improving on the case of formulas consisting of only $(3,3)$-clauses. We have so far obtained the branching number $\tau(3,4)$ except for such case. Thus, if we also obtained the same branching number for that case, we could improve our bound to $\tau(3,4)^m \approx 1.221^m$.

Acknowledgement

I thank Prof. Osamu Watanabe for all his support: giving valuable comments, discussing elaborately, and improving this paper. I also thank Prof. Kazuhisa Makino for discussing about what this paper concerns.

References

1. Dantsin E., Hirsch E. A., and Wolpert A., "Algorithms for SAT based on search in Hamming balls", Proc. of the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS04), 141-151, 2004.
2. Dantsin E. and Wolpert A., "Derandomization of Schuler's algorithms for SAT", Proc. of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT04), 69-75, 2004.
3. Davis M., Logemann G., and Loveland D., "'A machine program for theorem-proving", Comm. ACM(5), 394-397, 1962.
4. Davis M., and Putnam H., "A computing procedure for quantification theory", J. of ACM(7), 201-215, 1960.
5. Hirsch E. A., "New Worst-Case Upper Bounds for SAT", J. of Automated Reasoning, 24, 397-420, 2000. (It is also in Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA98), 521-530, 1998.)
6. Monien B, and Speckenmeyer E., "Solving satisfiability in less than 2^n steps", Discrete Appl. Math.(10), 287-295, 1985.
7. Pudlák P., "Satisfiability - algorithm and logic", Proc. of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS98), 129-141, 1998.
8. Schuler R., "An algorithm for the satisfiability problem of formulas in conjunctive normal form", J. of Algorithms, 54 40-44, 2005.