ISSN 1342-2812

Research Reports on Mathematical and Computing Sciences

Another proof for an upper bound of a local search algorithm for 3-SAT

Masaki Yamamoto

masaki.yamamoto@is.titech.ac.jp

September 2004, C–197

Department of
Mathematical and
Computing Sciences
Tokyo Institute of Technology

SERIES C: Computer Science

Abstract: We present another proof for an upper bound of a deterministic local search algorithm [3] for 3-SAT. This bound $O(1.481^n)$ is currently no longer the best (where n is the number of variables) while the best bound is slightly better: $O(1.473^n)$ due to [1].

1 Introduction

There have been a lot of algorithms for the satisfiability problem (SAT) proposed and the upper bound improved. This was initiated by [4]. They analyzed a well-known procedure called Davis-Putnam procedure, and especially showed that the algorithm deterministically solves 3-SAT in time $O(1.619^n)$.

Local search is another major procedure for SAT. It is [5] at the first time that gives a randomized analysis for local search for k-SAT. The randomized algorithm, given a k-CNF formula F over n variables, randomly chooses exponentially many initial assignments, and runs local search for each of them. Namely, if the assignment doesn't satisfy F, the algorithm chooses an arbitrary unsatisfied clause, chooses a literal at random from that clause, and flips its value. If any satisfying assignment is not found in $3n$ steps, the algorithm starts local search with another initial assignment randomly chosen.

The deterministic algorithm [3] for k-SAT can be viewed as a derandomized version of the randomized algorithm explained above. It derandomizes the following two factors: (1) the choice of initial assignments, and (2) the choice of variables in local search. To derandomize these, we first cover the space of all possible 2^n assignments by *balls* of Hamming distance r. Then, for each ball, we run a deterministic version of local search to check whether there is a satisfying assignment inside the ball. The value of radius r is chosen optimal so that the overall running time is minimal. It is shown that the algorithm solves k-SAT in $O((2 - 2/(k+1))^n)$. It is also shown that the bound is decreased to $O(1.481^n)$ for $k = 3$ by using a more sophisticated version of local search.

In this paper, we present another version of local search, resulting in the same bound as [3] for 3-SAT. This bound is currently no longer the best while the best bound is $O(1.473^n)$ [1]. Before describing our local search and the analysis of the upper bound, we review the basic idea of [3] in the next section. Then, we present our local search and analyze the upper bound in section 3. In conclusion section, we indicate a possibility to achieve the currently best bound $O(1.481^n)$.

2 Deterministic Local Search Algorithm for k-SAT

In this section, we look over a deterministic local search algorithm [3] for k-SAT which solves in a polynomial factor of $(2 - 2/(k+1))^n$.

We here deal with *k-CNF* formula ($k \geq 3$) over n variables. A *literal* is a variable or a negated variable. A *clause* is disjunction of literals. The *size* of clause is the number of literals of clause. A *k-CNF* (standing for Conjunctive Normal Form) formula is conjunction of clauses such that the size of each clause is at most k.

For an assignment t and a literal l, we denote by $t|_{l=1}$ the assignment obtained from t by setting the value of l to 1. Given a formula F, we also denote by $F|_{l=1}$ the formula obtained from F by assigning 1 to l, i.e., the clauses containing the literal l are deleted from F, and the literal $\bar{l}$ is deleted from the clauses containing $\bar{l}$.

Local Search: Search(F, t, r)

step 1: If $r = 0$ or F contains an empty clause, then return *unsatisfiable*.

step 2: If all clauses of F are true under t, then output *satisfiable*.

step 3: Pick arbitrarily an unsatisfied clause C under t. For each literal l of C, call **Search**$(F|_{l=1}, t, r-1)$.

step 4: Return *unsatisfiable*.

Figure 1: Local Search

We note some notions and notations used in [3]. We here identify assignments to variables with binary words. The set of these words of length n is $\{0,1\}^n$, denoted by H_n. Thus, any assignment is regarded as an element of H_n. The *ball* of radius r around an assignment t is, denoted by $B_r(t)$, the set of all assignments whose Hamming distance from t is at most r. A *covering code* for H_n is a set $\mathcal{C}$ such that $H_n = \bigcup_{t \in \mathcal{C}} B_r(t)$.

Local Search:

We see how to derandomize the choice of variables in the randomized local search. Given a formula F and an assignment $t \in H_n$ which is the center of a ball of radius r, we will check whether there exists a satisfying assignment for F inside the ball $B_r(t)$. The efficiency of the derandomized local search relies on the following important observation.

Lemma 2.1 Let F be a formula, and t be an assignment such that F is unsatisfied by t. Then, for any unsatisfied clause C, F has a satisfying assignment in $B_r(t)$ if and only if

$$\exists l \in C \ \left[F|_{l=1} \text{ has a satisfying assignment in } B_{r-1}(t)\right]$$

Proof. It is clear that $F|_{l=1}$ has a satisfying assignment $t' \in B_{r-1}(t)$ for some $l \in C$ implies F has a satisfying assignment in $B_r(t)$ because $t'|_{l=1}$ satisfies F and $t'|_{l=1}$ is in $B_r(t)$.

Suppose that F has a satisfying assignment t_0 such that $t_0 \in B_r(t)$. Let $X = \{x_1, \cdots, x_n\}$ be a set of variables and X' be a subset of those variables whose values differ between t_0 and t. We may assume w.l.o.g. $X' = \{x_1, \cdots, x_r\}$. Then, for any unsatisfied clause C, there exists a literal l in C such that $l \in X'$ or $\bar{l} \in X'$ because otherwise C cannot be satisfied by t_0. Thus, F has a satisfying assignment in $B_{r-1}(t|_{l=1})$ for some $l \in C$, that is, $F|_{l=1}$ has a satisfying assignment in $B_{r-1}(t)$. ■

The recursive procedure **Search**(F, t, r) as shown in Figure 1 is the derandomized local search, which takes a formula F, an assignment t, and radius r. The lemma above confirms that it is not necessary to search through all assignments of $B_r(t)$. That is, the procedure below finds a satisfying assignment if and only if F has a satisfying assignment in $B_r(t)$. It is easily proven by induction on r: Suppose that **Search**$(F, t, r-1)$ output *satisfiable* if and only if F has a satisfying assignment in $B_{r-1}(t)$. Then, by Lemma 2.1, it also holds for r.

The recursion depth of **Search**(F, t, r) is at most r. If the input formula F is in k-CNF, the number of recursive calls done at step 3 is at most k. Thus, it runs in time $O(k^{rn})$.

Lemma 2.2 The procedure above finds a satisfying assignment if and only if F has a satisfying assignment in $B_r(t)$. It runs in time $O(\alpha^{rn})$ where $\alpha = k$.

Covering Codes:

We see how to derandomize the choice of initial assignments. Give an assignment $t \in H_n$. Consider a ball $B_r(t)$ of radius r. We define the *volume* $V(n,r)$ of the ball as the number of assignments in this ball; that is,

$$V(n,r) \;=\; \sum_{i=0}^{r} \binom{n}{i}.$$

Using the normalized covering radius $\rho = r/n$ satisfying $0 \le \rho \le 1/2$, the volume $V(n,r)$ can be estimated as follows:

$$\beta(n)^{-1} \cdot 2^{h(\rho)n} \;\le\; V(n,r) \;\le\; 2^{h(\rho)n},$$

where $h(\rho) = -\rho \log \rho - (1-\rho)\log(1-\rho)$ is the binary entropy function, and $\beta(n) = \sqrt{8n\rho(1-\rho)}$. (See Lemma 2.4.4 of [2].) Thus, for any covering code $\mathcal{C}$ of radius r,

$$|\mathcal{C}| \;\ge\; \frac{2^n}{V(n,r)} \;\ge\; 2^{(1-h(\rho))n}.$$

On the other hand, for any integer $n \ge 1$ and $r \ge 0$, there exists a covering code $\mathcal{C}$ of length n, covering radius at most r, and size at most $\lceil n \cdot 2^n / V(n,r) \rceil$. This can be shown by the probabilistic argument. (See Theorem 12.1.2 of [2].) By applying the lower bound of $V(n,r)$, we obtain the following lemma.

Lemma 2.3 For any n there exists a covering code of length n, radius at most ρn, and size at most $n\beta(n) \cdot 2^{(1-h(\rho))n}$.

This lemma only guarantees the existence of such a covering code. Now we describe a deterministic algorithm for constructing such a covering code. This problem is considered as Minimum Set Cover problem: given a finite set S and a collection C of subsets of S, then find a minimum set cover for S, i.e., a minimum subset C' of C such that every element of S belongs to at least one member of C'. It is known that there exists a $(\ln|S| + 1)$-approximate algorithm for this problem, which runs in time $O(|S| \cdot \sum_{C' \in C} |C'|)$. We apply this approximate algorithm to construct such a covering code by regarding H_n as S, and $B_r(t)$ for each $t \in H_n$ as a member of C. Since $|S| = |H_n| = 2^n$, and $\sum_{C' \in C} |C'| \le |C| \cdot 2^n = 2^{2n}$, we can obtain a covering code in time $O(2^{3n})$ whose approximate ratio is $\ln 2^n + 1 = O(n)$. Note that since the minimum size of covering codes of radius ρn for H_n is at most $n\beta(n) \cdot 2^{(1-h(\rho))n}$ by lemma 2.3, the approximate algorithm finds a covering code of size at most $c \cdot n^2 \beta(n) \cdot 2^{(1-h(\rho))n}$ for some constant c.

Lemma 2.4 Let $n \ge 1$, $0 < \rho < 1/2$, and $\beta(n) = \sqrt{8n\rho(1-\rho)}$. Then a covering code of length n, radius at most ρn, and size at most $c \cdot n^2\beta(n) \cdot 2^{(1-h(\rho))n}$ for some constant c can be constructed in time $O(2^{3n})$.

For aiming at the whole time complexity less than $O(2^n)$, this running time is too large. To get around this problem, we partition the n bits word into the n/d bits words and apply the approximate algorithm to each n/d bits word, i.e., for each of d words, we construct a covering code $\mathcal{C}'$ of radius $\rho n/d$ for $H_{n/d}$. We then define $\mathcal{C}$ for H_n to be the set of all concatenations of d words from $\mathcal{C}'$.

Lemma 2.5 Let $d \ge 2$ be a divisor of $n \ge 1$, and $0 < \rho < 1/2$. Then there is a polynomial $q_d(n)$ such that a covering code of length n, radius at most ρn, and size at most $q_d(n) \cdot 2^{(1-h(\rho))n}$ can be constructed in time $q_d(n) \cdot (2^{3n/d} + 2^{(1-h(\rho))n})$.

Proof. First of all, we show that $\mathcal{C}$ is a covering code of radius at most ρn for H_n. We consider an arbitrary word $t \in H_n$ and the partition $t_1, \cdots, t_d$ of the word t with length n/d for each. It is clear that for any t_i there exists a word $w_i \in \mathcal{C}'$ such that $t_i \in B_{\rho n/d}(w_i)$. Since the concatenation w of $w_1, \cdots, w_d$ is in $\mathcal{C}$ and the radius of $\mathcal{C}'$ is at most $\rho n/d$, t is in $B_{\rho n}(w)$. Thus, $\mathcal{C}$ is a covering code of radius at most ρn for H_n.

The size of $\mathcal{C}$ is at most $(c \cdot n^2\beta(n) \cdot 2^{(1-h(\rho))n/d})^d$, which is $q_d(n) \cdot 2^{(1-h(\rho))n}$ for some polynomial $q_d(n)$. The time taken to construct $\mathcal{C}$ is d multiplied by the time taken to construct $\mathcal{C}'$, plus the size of $\mathcal{C}$, that is, $q_d(n) \cdot 2^{3n/d} + q_d(n) \cdot 2^{(1-h(\rho))n}$ for some polynomial $q_d(n)$. ■

Time complexity:

Summing up above all, it takes $q_d(n) \cdot 2^{(1-h(\rho))n} \cdot \alpha^{\rho n}$ for local search since there are $q_d(n) \cdot 2^{(1-h(\rho))n}$ initial assignments, and for each initial assignment t, it takes $O(\alpha^{\rho n})$ to check whether there is a satisfying assignment in $B_{\rho n}(t)$. Moreover, it takes $q_d(n) \cdot (2^{3n/d} + 2^{(1-h(\rho))n})$ to construct a covering code for H_n. Thus, the total time is:

$$q_d(n) \cdot (2^{3n/d} + 2^{(1-h(\rho))n}) \ + \ q_d(n) \cdot 2^{(1-h(\rho))n} \cdot \alpha^{\rho n}.$$

The first term above can be made smaller than the second by choosing a suitable d, say, $d = 6$. With this value, we obtain the total time:

$$\text{poly}(n) \cdot 2^{(1-h(\rho))n} \cdot \alpha^{\rho n} \ = \ O\big(\big(2(\alpha\rho)^\rho(1-\rho)^{1-\rho}\big)^n \big).$$

Taking $\alpha = k$ and $\rho = 1/(k+1)$, we have $(2 - 2/(k+1))^n$ for k-SAT. This bound is what is shown in [3] for general case. In the next section, we focus on 3-SAT, and reduce the value of α less than the trivial value $\alpha = k$.

3 A Different Version of Local Search

In this section, we give another proof for the upper bound [3] for 3-SAT. That is, we present a different version of local search from [3], which have the same bound.

In local search, given a formula F, an initial assignment t, and radius r, we arbitrarily pick an unsatisfied clause C, and then for each $l \in C$ we search for a satisfying assignment inside the ball $B_{r-1}(t|_{l=1})$. We regard this branched procedure as a tree structure: Consider that we make branches on a clause $C = l \vee l' \vee l''$. Then we suppose the following tree.

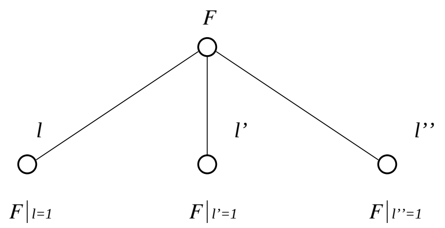

Figure 2: Tree Structure

Consider a 3-CNF formula F over X and an assignment t to X. Let $C^\ominus$ be a set of unsatisfied clauses of F by t, and $X^\ominus = \{x \in X : \exists C \in C^\ominus[x \in C \text{ or } \bar{x} \in C]\}$. We define as $C^\oplus$ a set of satisfied clauses C by t such that every true literal of C is a literal of some variable of $X^{\ominus}$. We divide $C^{\oplus}$ into three sets C^{i+} of clauses for $i = 1, 2$, and 3: a set of clauses containing i true literals, respectively. In our local search, we use a notion "conflict": for any clause $C \in C^{\ominus}$, if there exists a literal $l \in C$ and a clause $C' \in C^{\oplus}$ such that C' contains $\bar{l}$, we say that C *conflicts* with C' w.r.t. l.

We present a branching rule in Figure 3 applied to our local search. Let S_r be the number of leaves of the search tree constructed by applying our branching procedure. We'll estimate the upper bound of S_r for each case.

Branching Procedure

if (a literal appears in different clauses C and C' of $C^{\ominus}$)
  then make branches on C and C' as Figure 4
else if ($\exists C \in C^{\ominus}$ and $C' \in C^{1+}$ such that C conflicts with C')
  then make branches on C and C' as Figure 5
else if ($\exists C \in C^{\ominus}$ and $C' \in C^{2+}$ such that C conflicts with C')
  then make branches on C and C' as Figure 6
else if ($\exists C \in C^{\ominus}$ and $C' \in C^{3+}$ such that C conflicts with C')
  then make branches on C and C' as Figure 6
else
  return F is satisfiable

Figure 3: Branching Procedure

At first, we deal with two clauses C and C' of $C^{\ominus}$ such that a literal l appears commonly in C and C', say, $C = l \vee a \vee b$ and $C' = l \vee a' \vee b'$. We make branches on those clauses C and C' as shown in Figure 4.

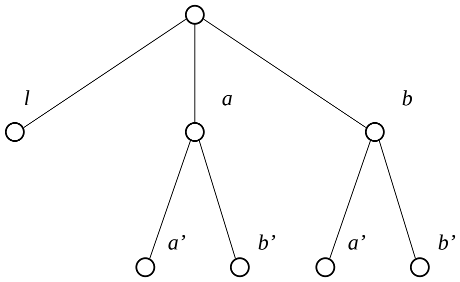

Figure 4: Branching Procedure 1

Lemma 3.1 For two clauses C and C' of $C^{\ominus}$ such that a literal appears commonly in C and C', the branching on C and C' as Figure 4 is valid. Thus, in this case we obtain the following recurrence for S_r:

$$S_r \ \leq \ S_{r-1} \ + \ 4S_{r-2}.$$

which is for some constant c,

$$S_r \ \leq \ c \cdot \left(\frac{1+\sqrt{17}}{2}\right)^r.$$

Proof. Let l be a literal which appears in C and C', and let $C = (l \vee a \vee b)$ and $C' = (l \vee a' \vee b')$. By branching on C first, we have the search tree as Figure 4 because in the branches $a = 1$ and $b = 1$ on C, the search tree branched on C' doesn't need to have the branch $l = 1$. ■

In what follows, we suppose that there are no such clauses as the lemma above. We next deal with any clause $C = (l \vee a \vee b)$ of $C^{\ominus}$ such that C conflicts with some $C' \in C^{1+}$ w.r.t. l, say, $C' = \bar{l} \vee a' \vee b'$. We make branches on those clauses C and C' as shown in Figure 5.

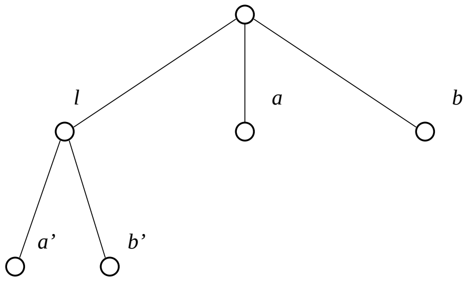

Figure 5: Branching Procedure 2

Lemma 3.2 For any clause C of $C^{\ominus}$ such that C conflicts with some $C' \in C^{1+}$, the branching on C and C' as Figure 5 is valid. Thus, in this case we obtain the following recurrence for S_r:

$$S_r \;\leq\; 2S_{r-1} \;+\; 2S_{r-2}.$$

which is for some constant c,

$$S_r \;\leq\; c \cdot \left(1 + \sqrt{3}\right)^r.$$

Proof. Let $C = (l \vee a \vee b)$ and $C' = \bar{l} \vee a' \vee b'$. Suppose that we make branches on $C = (l \vee a \vee b)$. Then, flipping the value of l makes C' falsified. Thus, in this branch $l = 1$ we can make branches on C' pruning the branch $l = 1$. ■

Note that given a clause $C = (l_1 \vee l_2 \vee l_3)$ of $C^{\ominus}$ such that C conflicts with $C' \in C^{1+}$ w.r.t., say l_1, if C again conflicts with $C'' \in C^{1+}$ w.r.t. another literal, say l_2, then we can obtain the better ratio for S_r. (It is the best when C conflicts w.r.t. all three literals of C.)

Now, we suppose that $C^{1+} = \phi$. We deal with any clause of $C^{\ominus}$ conflicting with some clause of $C^{\oplus}$ which has multiple true literals. We first take a look into the case that we can see the formula is satisfiable.

Lemma 3.3 Given a 3-CNF formula F over X and an assignment t. If for each clause C of $C^{\ominus}$, there exists a literal l of C such that C doesn't conflict with any clause of $C^{\oplus}$ w.r.t. l, then F is satisfiable.

Proof. Let Y be a set of literals obtained by choosing one literal l from each $C \in C^{\ominus}$ such that C doesn't conflict with any clause of $C^{\oplus}$ w.r.t. l. We'll show that the assignment t' obtained from t by flipping the value of each literal of Y satisfies F.

It suffices to show that t' satisfies every clause of $C^{\oplus}$. (It is obvious that t' satisfies each clause of $F \backslash C^{\oplus}$.) Note that by definition, $C = l_1 \vee l_2 \vee l_3$ doesn't conflict with any clause of $C^{\oplus}$ w.r.t., say l_1 if and only if

1. $\overline{l_1}$ doesn't appear in any clause of $C^{\oplus}$, or
2. if $\overline{l_1}$ appears in a clause $C' = (\overline{l_1} \vee \overline{l} \vee l')$ of C^{2+}, C contains l, or
3. if $\overline{l_1}$ appears in a clause $C' = (\overline{l_1} \vee \overline{l} \vee \overline{l'})$ of C^{3+}, some clause (not necessarily C) of $C^{\ominus}$ contains at least two literals of l_1, l, and l'.

It is obvious in the case 1 that t' satisfies each clause of $C^{\oplus}$. In the cases 2 and 3, t' still satisfies each clause of $C^{\oplus}$ because we change the value of only one literal for each clause of $C^{\ominus}$, and thus for each clause of $C^{\oplus}$, at least one true literal remains unchanged. ■

If it is not the case of the lemma above, that is, for each $C \in C^{\ominus}$, C conflicts some clause C' of C^{2+} or C^{3+} w.r.t. each literal of C, we make branches on such a clause C. In Figure 6, we show the search tree where $C = a \vee b \vee c$ conflicts with $C_a = a \vee a' \vee a''$ w.r.t. a, $C_b = b \vee b' \vee b''$ w.r.t. b, and $C_c = c \vee c' \vee c''$ w.r.t. c, and the clauses containing a'', b'', and c'' are $a'' \vee l_a \vee l'_a$, $b'' \vee l_b \vee l'_b$, and $c'' \vee l_c \vee l'_c$, respectively.

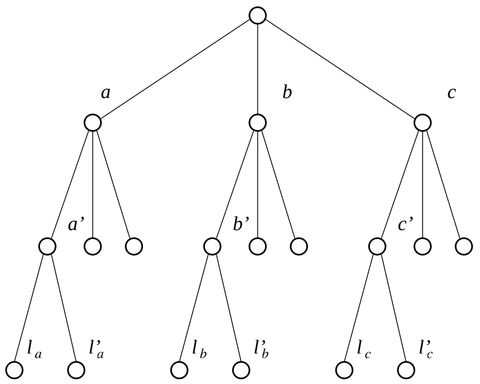

Figure 6: Branching Procedure 3

Lemma 3.4 For such a clause $C = a \vee b \vee c$ above, the branching on C as Figure 6 is valid. Thus, in this case, we obtain the following recurrence for S_r:

$$S_r \ \leq\ 6S_{r-2}\ +\ 6S_{r-3}.$$

which is for some constant c,

$$S_r \ \leq\ c \cdot \alpha^r,$$

where c is a constant, and α is the largest root of $\alpha^3 - 6\alpha - 6 = 0$.

Proof. Let C, C', and C'' be clauses of $C^{\ominus}$ containing a, a', and a'', respectively as above. Consider that we make branches on C, C', and C'' in this order. Since $t|_{a=1,a'=1,a''=1}$ falsifies C_a, the search tree branched on C'' doesn't need to have the branch $a'' = 1$. It is the same for b and c. ■

Note that the ratio for S_r of the lemma above is the worst among all the cases that for each $C \in C^{\ominus}$, C conflicts some clause C' of C^{2+} or C^{3+} w.r.t. each literal of C. The upper bound of S_r is the worst among cases shown in the three figures. Thus, we finally have:

Theorem 3.5 For our local search applying the branching procedure, we have

$$S_r \le c \cdot \alpha^r.$$

where α is the largest root of $\alpha^3 - 6\alpha - 6 = 0$, which is $\alpha = 4^{1/3} + 2^{1/3} \approx 2.848$.

4 Conclusion

We have shown another version of a deterministic local search for 3-SAT, which has the same upper bound as [3]. This bound $O(1.481^n)$ is currently no longer the best while the best bound is slightly better: $O(1.473^n)$ due to [1].

We end with indicating a possibility to achieve the best bound $O(1.473^n)$. Recall the worst case which gives the bound $O(1.481^n)$. That is, the case as shown in Figure 6 which have the recurrence $\alpha^3 - \alpha - 6 = 0$. This case is rather particular than the others: F is partitioned into $C^{\ominus}$, $C^{\oplus}$, and $F' = F \backslash (C^{\ominus} \cup C^{\oplus})$ such that $C^{\oplus}$ is composed of only C^{3+}. Moreover, F is satisfiable if and only if $F \backslash F'$ is satisfiable. From this observation, we are concerned with the following problem:

Monotone Negative Once 3-SAT (MNO 3-SAT)

Instance: A 3-CNF formula F such that each clause of F has either only positive literals or only negative literals, and for each variable, its negative literal occurs exactly once.

Question: Is there a satisfying assignment for F?

If we could find a good algorithm for solving this problem, our local search procedure would achieve the currently best bound. More precisely, given a 3-CNF formula F over n variables for MNO 3-SAT, we are subject to finding a deterministic algorithm in time $O(\alpha^{n/3})$ where $\alpha = (1 + \sqrt{21})/2$, which is the largest root of recurrence $\alpha^3 - 5\alpha - 6 = 0$.

Acknowledgement

I thank Professor Osamu Watanabe for all his supports: giving valuable comments, discussing, and improving this report.

References

[1] T. Brueggemann, W. Kern, "An improved local search algorithm for 3-SAT", Theoretical Computer Science, to appear.

[2] G. Cohen, I. Honkala, S. Litsyn, A. Lobstein, Covering Codes, Mathematical Library, vol.54, Elsevier, Amsterdam, 1997.

[3] E. Dantsin, A. Goerdt, E.A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, O. Raghavan, U. Schöning, "A deterministic $(2 - 2/(k + 1))^n$ algorithm for k-SAT based on local search", Theoretical Computer Science 289, 69-83, 2002.

[4] B. Monien, E. Speckenmeyer, "Solving Satisfiability in less than 2^n Steps", Discrete Applied Mathematics 10, 287-295, 1985.

[5] U. Schöning, "A Probabilistic Algorithms for k-SAT Based on Limited Local Search and Restart", Algorithmica 32, 615-623, 2002.