Improved Algorithms for CNF Satisfiability Problems

Suguru Tamaki

Abstract

Improved Algorithms for CNF Satisfiability Problems

Suguru Tamaki

Kyoto University, Japan

2006

In this thesis we study the computational complexity of CNF Satisfiability problem (SAT for short) in an algorithmic point of view. SAT problem is, given a Conjunctive-Normal-Form Boolean formula, to determine whether there is an assignment to the variables that satisfies all clauses of the formula. This problem is one of the most studied problems among all NP-complete problems and has been studied in a lot of research fields, including artificial intelligence, logic design, and scheduling, without saying computational complexity.

One of the central problem in SAT is to design algorithms for n variable formulas that run in time c^n ($c < 2$). We would like to reduce the constant c in the exponent as small as possible. Such results are very important in both theoretical and practical sense. In this thesis we give several new algorithms and analyses of them for 3SAT, where clause length of input formulas is restricted to three.

In Chapter 3, we improve Schöning's randomized local search algorithm using partial knowledge on satisfying assignments. Our main idea is to use the bias in the number of 0's and 1's of a satisfying assignment. Actually we take the the number of 0's and 1's as a parameter and design an algorithm optimized for each parameter value. Though our algorithm do not improve the general case, we can apply our algorithm to some natural combinatorial problems like 3DM matching, where the number of 0's and 1's in satisfying assignments are often inbalanced. Then we give some experimental results that shows our algorithm is faster for several instances in practical sense.

In Chapter 4, we present new worst-case upper bounds for 3SAT. The previous best algorithm for general 3SAT is an improved version of Schöning's algorithm and

for unique 3SAT is PPSZ's randomized splitting algorithm. Our result is based on the following observation: if an input formula has few satisfying assignments, PPSZ's performs well, and if it has many satisfying assignments, Schöning's does well. We give the analysis of two algorithms using a combinatorial structure that is closely related to the number of satisfying assignments of formulas. We obtain an improved upper bounds by selecting two algorithms for the parameter value used in the analysis.

We also present an improvement on Schöning's randomized local search algorithm. In the original Schöning's algorithm we use an uniformly generated assignment as starting point of search. Hofmeister et al. gave a better way to obtain a good starting point with higher probability by using independent clause set and improved the algorithm. We extend the definition of independent clause set, namely, we introduce independent clause pair set and analyze it. As a result, we obtain a further better way to obtain a good starting point of search and improved the previous algorithm.

In Chapter 5, we propose a new algorithm that improves the current best worst-case upper bounds of 3SAT shown in Chapter 4. It is known that PPSZ's algorithm can find a satisfying assignment in sub-exponential time when given an assignment that agrees with a satisfying assignment in a large fraction (say, 2/5 fraction) of variables. We say such an assignment a good assignment. In the current best algorithm we try to find a good assignment by guessing uniformly at random. Our basic idea is to use Schöning's algorithm for obtaining a good assignment with higher probability. We analyze this new algorithm under some assumption and improves the current best worst-case upper bounds of 3SAT. Under the same assumption, we also improves unique 3SAT case, which seems more difficult than to improve that of the general case.

Improved Algorithms for CNF Satisfiability Problems

A Dissertation
Presented to the Graduate School of Informatics
Kyoto University
in Candidacy for the Degree of
Doctor of Philosophy

by
Suguru Tamaki
Kyoto University, Japan

Supervisor: Prof. Kazuo Iwama

Contents

List of Figures

Acknowledgments

I would like to express my sincere appreciation to my supervisor, Prof. Kazuo Iwama. He allowed me to study at his laboratory and also guided me strongly and enthusiastically throughout my research activities. This thesis would not be completed without his great advice. His precious words would have a favorable influence on my future activities.

I would like to express my sincere appreciation to Prof. Hiro Ito and Prof. Shuichi Miyazaki. They taught me several important theoretical principles with their accurate comments and suggestions on my research. Several enthusiastic discussions with them were quite exciting experiences for me.

I am deeply grateful to Professor Takashi Horiyama for supporting my research activities with his exceeding contribution to Iwama laboratory.

I would like to thank all the members of Iwama laboratory for giving me helpful advice and encouragements for my research work. I would like to thank all of my friends for having a meaningful and enjoyable time together.

I am very thankful to Prof. Taiichi Yuasa and Prof. Hidetoshi Onodera for their valuable comments on this thesis.

Finally, I would like to express my supreme gratitude to my family, for encouraging and supporting me at all times.

This work is financially supported in part by the 21st Century COE Program for Research and Education of Fundamental Technologies in Electrical and Electronics Engineering.

Chapter 1

Introduction

1.1 Background

Satisfiability problem (SAT for short) is, given a Boolean formula F, to determine whether there is an assignment to the variables that satisfies the formula. In this thesis we study the computational complexity of satisfiability problems in an algorithmic point of view. SAT is a typical NP problem, where NP stands for the class of problems solvable in nondeterministic polynomial time. If we are given a satisfying assignment of an input formula, we can verify it efficiently. However, it is hard to find satisfying assignments in general case. Actually Cook proved that SAT is NP-complete. That means we can reduce every NP problem into SAT in polynomial time. It is widely believed that NP-complete problems can not be solved in polynomial time, or even in sub-exponential time. Thus typical time complexity of SAT is of the form 2^{cn} where $c < 1$ is some constant. Our main purpose of this thesis is to reduce the constant c as small as possible. The importance of the constant c is apparent. Consider 2^n and $2^{0.6n}$. The former is greater than the latter by $2^{0.4n}$ factor. When $n = 500$, the difference becomes $2^{200} \sim 10^{20}$, that is an astronomically large number. Thus improving $c = 1$ to $c = 0.6$ is very meaningful when we treat an exponential function.

In this thesis we try to design and improve the algorithms for SAT exploiting several properties of CNF formulas. Our typical approach is as follows: First we define the property of formula and parameterize it. For example, we take the prop-

erty that a formula has many satisfying assignments or not and define the number of satisfying assignments as a parameter. Second we design an algorithm optimized for formulas with a specific range of the parameter. Finally we may have a set of algorithms optimized for every range of the parameter. Of course, when we see the history, several improvements seem to follow this approach implicitly. However, we use this idea more explicitly. In general, to improve an algorithm we must find "hard instances" against the algorithm and then design a new algorithm for them. The main difference is we do not need to find hard instances since we first parameterize some property and optimize algorithms for each parameter value. Hard instances can be naturally found in optimization steps. This seems very useful approach and clarify the situation of improvements because we can try improving algorithms automatically in some sense. Now the task of improving algorithms becomes as follows: There are a lot of natural properties of formula and there are also a lot of algorithms for SAT. We select one property and one algorithm, and do optimization. Repeat this as many as possible for pairs of a property and an algorithm. Then we have a list of algorithms optimized according to the parameter. If we obtain an improvement on a set of algorithms optimized for every range of one fixed parameter, we are done. Even if we cannot obtain an improvement for some range, the result may have practical importance because some practical problems described by SAT formula appears with some limited range of parameter.

Importance of Satisfiability Problems Why SAT plays a special role among several NP-complete problems like max clique, min vertex cover and min coloring? One explanation is its generality of the problem. SAT instance consists of a boolean formula. This means we can easily express a mathematical statement as a SAT instance. By the definition of NP-completeness, we can do the same for other NP-complete problems. However it is often not convenient to express propositional logic statements as graphs or set systems. A lot of NP-complete problems can be easily reduced to SAT, but the reduction of opposite direction is often very complicated, like a typical proof of NP-completeness using carefully designed gadgets. Thus if we study one problem from NP-complete problems, SAT seems the most natural

one. Another explanation is its practical importance. We know several problems that appear in artificial intelligence, logic circuit design, scheduling are naturally expressed as SAT problems. Designing fast algorithms for SAT directly provides fast solutions for these problems.

Here we give a list of references around SAT: approximation algorithms [ACG+99, Vaz01], average case analyses of SAT algorithms [Che03, Fla03, Gen98, KP92, KV06], experiment and efficient implementation of SAT algorithms [CI95, CI96, CIKM97, HK05, IKM+00, Mor93, SK93, SLBH05, SLM92], lower bounds for SAT [FvM00, Wil05], circuit complexity [All96, BS90, Hås89, HJP95, IM02, IPZ01, PSZ00, Weg87], proof complexity [BP98, Urq95], exact algorithms for NP-complete problems [AS03, BE05, Bei99, Bys04, Epp03, FG04, FGK05, FGK06, FKT04, GHNR03, NR03, Woe03, Woe04], see also proceedings of recent SAT conference [SAT03, SAT04, SAT05].

1.2 Overview of the Thesis

In this thesis we treat the k-satisfiability problem (k-SAT for short). In k-SAT problem, a Boolean formula F is given by a set of clauses each of which has length at most k. The problem is to determine whether there is an assignment to the variables that satisfies the formula. This problem has been studied by many researchers, and various algorithms have been proposed. Some algorithms indeed have a worst-case time complexity much better than the trivial exhaustive search algorithm; see, e.g.,[MS85,PPSZ98,Sch99,Kul99,DGHS00,Hir00a].

In Chapter 3, we improve Schöning's randomized walk algorithm using partial knowledge on solutions. In [Sch99], Schöning gave the celebrated randomized algorithm for 3SAT, which runs in an expected time of 1.334^n. The algorithm is a simple local search. We give a generalization of the analysis of [Sch99], which says that if we have some knowledge on the value in a satisfying assignment, we can increase the success probability. For example, suppose that we know for some reason, 90% of the odd-indexed variables $x_1, x_3, x_5 \ldots$ take value 1 in a satisfying assignment. Then our analysis shows that we would be able to obtain solution in roughly 1.196^n steps, which is much better than the original 1.334^n. As a concrete example of such partial

knowledge on solutions, we consider an imbalance between the number of 0's and 1's in the satisfying assignment. Suppose that we know the satisfying assignment includes $p_0 n$ 0's and $p_1 n$ 1's ($0 \leq p_0 \leq 1$ and $p_1 = 1 - p_0$). Then we can obtain optimal algorithm by improving Schöning's algorithm. We look a combinatorial problem, i.e., 3-Dimensional Matching (3DM), as a concrete example where such an imbalance appears. If we reduce 3DM instances to SAT formulas, the resulting formulas do have the imbalance whose degree is represented by $p_1 = 1/k$. We also show preliminary experimental results.

In Chapter 4, we present new worst-case upper bounds for 3SAT. Our bound is 1.3225^n that improves the previous best 1.328^n by [Rol03]. The basic idea is to combine two existing algorithms, the one by Paturi, Pudlák, Saks and Zane [PPSZ98] and the other by Schöning [Sch99]. It should be noted, however, that simply running the two algorithms independently does not seem to work. The algorithm of [PPSZ98] is called **PPSZ**, which is based on a randomized Davis-Putnam combined with bounded resolution. This algorithm has the unique feature that it achieves a quite nice performance, $O(1.3071^n)$, for a unique 3-CNF formula, i.e., a formula which has only one satisfying assignment. As the number m of satisfying assignments grows, the bound, denoted by $T_{\text{PPSZ}}(m)$, degenerates, i.e., $T_{\text{PPSZ}}(m)$ is an increasing function. In contrast, the algorithm of [Sch99] is based on the standard local search for which the above intuition is obviously true. Namely its running time $T_{\text{SCH}}(m)$ is the worst when $m = 1$ and then decreases. Recall that $T_{\text{PPSZ}}(1) < T_{\text{SCH}}(1) = O(1.334^n)$. So, if we run the two algorithms in parallel, then the running time is bounded by $\min\{T_{\text{PPSZ}}(m), T_{\text{SCH}}(m)\}$ which becomes maximum ($= T_{\text{PPSZ}}(m_0) = T_{\text{SCH}}(m_0)$) at $m = m_0$. Obviously $T_{\text{SCH}}(m_0) < T_{\text{SCH}}(1)$. Although $T_{\text{SCH}}(1)$ is not the currently best, there is a lot of hope of breaking it since $T_{\text{PPSZ}}(1)$ is much better than the current best. Unfortunately, this approach has an obstacle. We know the value of $T_{\text{PPSZ}}(m)$ but we do not know that of $T_{\text{SCH}}(m)$ for the following reason. To obtain $T_{\text{SCH}}(m)$, it appears that we need to know the Hamming distance between the (randomly chosen) initial assignment and its closest satisfying assignment. However, there is no obvious way of doing so, since it is quite hard to analyze how (multi) satisfying assignments of a 3-CNF formula can distribute in the whole space of 2^n assignments.

To overcome this difficulty, we analyze two algorithms simultaneously, that is, we give a lower bound on the success probability that at least one algorithm returns a satisfying assignment given a same initial assignment.

We also present an improvement on Schöning's randomized local search algorithm extending the techniques developed by [SSW01, HSSW02]. Our result gives $O(1.329917^n)$ upper bounds for 3SAT. This bound is the first result that achieves better bounds than $O(1.33^n)$ at the moment after the result of [SSW01, HSSW02] appeared. This result is a generalization of the algorithm of [HSSW02]. The main idea is to change the probability distribution for the initial assignment depending on the input clauses. In the original Schöning's algorithm, each variable is set to 0 with probability 1/2. The information given by the clauses is completely ignored. Although a clause $C = x_1 \vee x_2 \vee x_3$ tells us that not all those three variables should be set to zero simultaneously, the original initialization phase selects such an assignment with probability 1/8. In order to exploit such information, [HSSW02] introduced the notion of maximal independent clause set. Our improvement is based on extending the definition of the maximal independent clause set. We introduce the notion of a good pair for clauses and allow maximal independent clause set contains not only clauses but good pairs. This enables us to exploit more information from clauses.

In Chapter 5, we propose a new algorithm that improves the current best worst-case upper bounds of 3SAT under assumption. Our main idea is to use Schöning's **RandomWalk** for obtaining better initial assignments for **PPSZ**. The result also improves the current best worst-case upper bounds of Unique3SAT, which seems more difficult than to improve that of the general case. The basic idea of **PPSZ** is as follows: Suppose that a given formula $G(x_1, \ldots, x_n)$ has exactly one satisfying assignment $z = z_1 z_2 \cdots z_n \in \{0,1\}^n$ (can be extended to the general case). Also let π be a permutation of $\{1, 2, \ldots, n\}$. Then if we assign each value of z into $\{x_1, \ldots, x_n\}$ in the order of π, (i.e., $z_{\pi(1)} \rightarrow x_{\pi(1)}$ in Step 1, $z_{\pi(2)} \rightarrow x_{\pi(2)}$ in Step 2, and so on), a certain number of variables $\subseteq \{x_1, \ldots, x_n\}$ are *forced*. Here, we say that a variable x is forced in the above course of sequential assignment with respect to π and z, if x becomes a unit clause in Step k for some $k \geq 1$. [PPSZ98] shows that the number N of such forced variables can be made quite large by adding clauses by

resolution. For a randomly chosen π, they proved that the expected value of N is at least $(2\ln 2 - 1)n \approx 0.613n$. This implies that if we know the correct values of the unforced variables, ("correct" means the same value as z), then we can retrieve the whole values of z by the above process. Roughly speaking it is enough to know the correct values of only $0.387n$ variables to obtain the satisfying assignment. Of course there is no obvious ways of getting the correct values of $0.387n$ variables. Our idea is to use **RandomWalk** for this purpose. If we guess the values $0.387n$ variables uniformly at random, the probability that all variables become correct is $2^{-0.387n}$. We can observe that **RandomWalk** generates better distribution than guessing uniformly at random if we assume **RandomWalk** has some property. In fact we can obtain an upper bound of $O(1.2991^n)$ for unique 3SAT and $O(1.308^n)$ for general 3SAT by this approach.

Chapter 2

Algorithms for CNF Satisfiability Problems

2.1 Preliminaries

In this section, we introduce some definitions and notations we will use throughout this thesis. We also state some basic facts.

For any finite set S, $|S|$ denote the cardinality of S. Let $[n] \equiv \{1, 2, \ldots, n\}$ and $\mathbf{P}_n$ denote the set of all permutations over $[n]$. An *alphabet* is any non-empty, finite set. Typically Σ is $\{0, 1\}$ or $[d]$. Given an alphabet Σ, a *string* over Σ is a sequence of symbols from Σ. The *length* of a string a is the number of symbols a consists of. Given an alphabet Σ, we denote by Σ^n the set of all strings over Σ with length n. Given a string $a \in \Sigma^n$, we denote by a_i the symbol appears in ith place of a, and in addition given $I \in [n]$, denote by a_I the substring consists of $\{a_i\}_{i \in I}$, that is, substring obtained by restricting a to the index set I. Given two strings $a, b \in \Sigma^n$, $d(a, b) \equiv \#\{i | a_i \neq b_i\}$ denote the *Hamming distance* between two strings a and b.

The *Hamming ball* of radius d around $x \in \{0, 1\}^n$ is $\mathbf{B}(x, d) \equiv \{y \in \{0, 1\}^n \mid d(x, y) \leq d\}$. The *subcube* with respect to $x \in \{0, 1\}^n$ and $I \subseteq [n]$ is $\mathbf{C}(x, I) \equiv \{y \in \{0, 1\}^n \mid y_I = x_I\}$. We call a variable x_i is *definig variable* of $\mathbf{C}(x, I)$ if $i \in I$ and *non-definig variable* if $i \notin I$. We will use the following lemma about subcube:

Lemma 2.1 *Given a nonempty $S \in \{0, 1\}^n$, $\{0, 1\}^n$ can be partitioned into a family*

$\{\mathbf{C}(z, I_z) \mid z \in S\}$ *of disjoint subcubes so that* $\mathbf{C}(z, I_z)$ *contains* $z \in S$ *but no other* $z' \in S - \{z\}$.

The proof of this lemma is easy, see [PPSZ05]. We call such a partition *subcube partition* of $\{0,1\}^n$ with respect to S.

We write $\log x$ as $\log_2 x$ and $\ln x$ as $\log_e x$. Binary entropy function is

$$h(x) = x \log \frac{1}{x} + (1-x) \log \frac{1}{1-x}.$$

Now we recall some basic definitions concerning k-SAT. Given n Boolean variables $x_1, \ldots, x_n$, an assignment to those variables is a vector $a = (a_1, \ldots, a_n) \in \{0,1\}^n$. A *clause* C of length k is a disjunction $C = l_1 \vee l_2 \vee \cdots l_k$ of literals where a literal is either a variable x_i or its negation $\bar{x}_i$ $(1 \leq i \leq n)$. For some constant k, k-CNF formula is a conjunction of clauses of length k. Sometimes we think k-CNF formula as a set of clauses. The problem k-SAT is defined as follows:

Input: k-CNF formula $F = C_1 \wedge \cdots \wedge C_m$. Each C_i is a clause of length at most k.

Problem: there an assignment that makes F true, i.e., a satisfying assignment for F?

Throughout this thesis, we will use n for the number of variables and m for the number of clauses in the given input formula. It can be assumed without loss of generality (and we will do so in the rest of the thesis) that no variable appears twice in the same clause. Furthermore, we may assume that no clause appears twice in the input formula, hence, the number of clauses is bounded by $O(n^k)$ and it makes sense to estimate the running time in terms of n.

We use and analyze probabilistic algorithms that find satisfying assignments with positive success probability. The success probability of one repeat-iteration, i.e., the probability of finding a satisfying assignment (if one exists) during one execution of the repeat loop is at least $p(n)$ for some fuction of n.

Lemma 2.2 *Let A be an probabilistic alogrithm runs in time $t(n)$ with success probability $p(n)$. Then there exists an algorithm A' runs in time $N \cdot t(n)/p(n)$ with success probability at least $1 - e^{-N}$.*

Proof. If we run A independently t times, then the probablity that A succeeds at least once is $1-(1-p(n))^t$. Recall that $(1-1/x)^x < 1/e$ holds for $x > 1$. Let A' be an algorithm that runs A independently $t = N/p(n)$ times. Then its running time is $N \cdot t(n)/p(n)$ and success probability at least $1 - e^{-N}$. □

2.2 Randomized Local Search Algorithms

In this section, we describe a randomized local search algorithm **SCH** and its major properties. The algorithm consists of its main routine **SCH** and **RandomWalk** as follows:

RandomWalk(CNF formula G, assignment y);
  $y' = y$;
  for $3n$ times
    if y' satisfies G
      then return y'; exit;
    $C \leftarrow$ a clause of G that is not satisfied by y';
    Modify y' as follows:
      select one literal of C uniformly at random and
      flip the assignment to this literal;
  end
  return y'

SCH(CNF-formula F, integer T)
  repeat T times
    $y =$ uniformly random vector $\in \{0,1\}^n$
    $z =$ **RandomWalk**(F, y);
    if z satisfies F
      then output(z); exit;
  end
  output('Unsatisfiable');

Lemma 2.3 ([Sch02]) *Let F be a kCNF formula and a^* be a satisfying assignment for F. For each assignment a, the probability that a satisfying assignment is found by* **RandomWalk**(F, a) *is at least* $\left(\frac{1}{k-1}\right)^{d(a,a^*)}$.

The following lemma states the bound for kSAT obtained in [Sch02] using the above lemma.

Theorem 2.1 ([Sch02]) *For any satisfiable formula F on n variables, the success probability of one repeat-iteration of* **SCH** *is at least* $\left\{\frac{1}{2} + \left(\frac{1}{k-1}\right)\right\}^n$.

2.3 Randomized Splitting Algorithms

In this subsection, we describe **PPSZ** and its major properties. The algorithm consists of the following four procedures, **PPSZ** (main routine), **Modify**, **Search** and **Resolve**.

Modify(CNF formula G, permutation π of $\mathbf{P}_n$, assignment y)

$G_0 = G$.
for $i = 1$ **to** n **do**
  if G_{i-1} contains unit clause $x_{\pi(i)}$
    then $z_{\pi(i)} = 1$
  else if G_{i-1} contains unit clause $\bar{x}_{\pi(i)}$
    then $z_{\pi(i)} = 0$
  else $z_{\pi(i)} = y_{\pi(i)}$
  $G_i = G_{i-1}$ with $x_{\pi(i)} = z_{\pi(i)}$
end
return z

Search(CNF-formula F, integer T)

repeat T times
  π = uniformly random permutation of $\mathbf{P}_n$
  y = uniformly random vector $\in \{0,1\}^n$
  $z = \mathbf{Modify}(F, \pi, y)$;

---

if z satisfies F
  then output(z); exit;
end
output('Unsatisfiable');

Consider a single run of **Modify**(G, π, y). Recall that each variable x_i is assigned so as to satisfy some unit clause, or is set to y_i. A variable whose assignment is determined by a unit clause is said to be forced. Let $I(G, \pi, y)$ denote the set of indices of variables that are not forced with respect to π and y. The following is one of the key lemmas to analyze **Modify**:

Lemma 2.4 ([PPZ99]) *Let $\tau(G, z)$ denote the probability with respect to random π and y, that* **Modify**(G, π, y) *returns z. Then*

$$\tau(G, z) = \mathbf{E}_\pi[\mathbf{Pr}_y[y \in \mathbf{C}(z, I(G, \pi, z))]].$$

A good bound for $\tau(G, z)$ is shown in [PPSZ98, PPSZ05]. We need some definitions on Resolution. Clauses C_1 and C_2 are said to *conflict* on variable v if one of them contains v and the other $\bar{v}$. C_1 and C_2 is a *resolvable pair* if they conflict on exactly one variable v. For such a pair, its *resolvant*, denote by $R(C_1, C_2)$ is the clause $C = D_1 \wedge D_2$ where D_1 and D_2 are obtained by deleting v and $\bar{v}$ from C_1 and C_2. If C_1 and C_2 are in the formula F, then adding $R(C_1, C_2)$ does not change the satisfying assignments of F. We say that the resolvable pair C_1 and C_2 is *s-bounded* if $|R(C_1, C_2)| \leq s$.

Resolve(CNF-formula F, integer s)
  $F_s = F$.
  while F_s has an s-bounded resolvable pair C_1, C_2
    with $R(C_1, C_2) \notin F_s$
  $F_s = F_s \wedge R(C_1, C_2)$.
  return(F_s).

PPSZ(CNF-formula F, integer s, integer T)

$F_s =$ **Resolve**(F, s).

Search(F_s, T).

Let $P(v, G, z)$ denote the probability with respect to a random π and a fixed assignment z, that the variable v is forced in **Modify**(G, π, z). It is easy to see $\mathbf{E}_\pi[|I(G, \pi, z)|] = n - \sum_v P(v, G, z)$. We say z is a d-isolated satisfying assignment of F if none of the assignments in $\mathbf{B}(z, d)$ satisfies F. Define

$$\mu_k = \sum_{j=1}^{\infty} \frac{1}{j\left(j + \frac{1}{k-1}\right)}$$

and

$$\varepsilon_k^{(d)} = \frac{3}{(d-1)(k-1)+2}.$$

Lemma 2.5 ([PPSZ98, PPSZ05]) *Let F be a k-CNF formula and z be a d-isolated satisfying assignment of F. If $s \geq k^d$, then for any variable v,*

$$P(v, F_s, z) \geq \frac{\mu_k}{k-1} - \varepsilon_k^{(d)}.$$

Using the above lemmas, we can bound $\tau(G, z)$.

Theorem 2.2 ([PPSZ98, PPSZ05]) *Let F be a k-CNF formula and z be a d-isolated satisfying assignment of F. If $s \geq k^d$, then for any variable v,*

$$\tau(F_s, z) \geq 2^{-\left(1 - \frac{\mu_k}{k-1}\right)}.$$

In general case, F may not have a d-isolated satisfying assignment. We state the result in terms of subcube. Now, given formula F with a set of satisfying assignments S and the subcube partition $\{\mathbf{C}(z, I_z) \mid z \in S\}$, $\tau(F_s, z|\mathbf{C}(z, I_z))$ is defined as the probability (averaged over y) that a single execution of **Modify** finds the assignment z under the condition that the initial assignment $y \in \mathbf{C}(z, I_z)$.

Lemma 2.6 ([PPSZ98, PPSZ05]) *For any satisfiable k-CNF formula F and any partition $\{\mathbf{C}(z, I_z)\}$, if $y \in \mathbf{C}(z, I_z)$ is chosen uniformly at random, then the value $\tau(F_s, z|\mathbf{C}(z, I_z))$ is bounded as follows:*

$$\tau(F_s, z|\mathbf{C}(z, I_z)) \geq 2^{-\left(1-\frac{\mu_k}{k-1}\right)} \left\{ \frac{(k-2)2^{-1-\frac{\mu_k}{k-1}}}{k-1} \right\}^{|I_z|}.$$

Chapter 3

Exploiting Partial Knowledge of Satisfying Assignments

3.1 Introduction

In the k-satisfiability problem (k-SAT for short), a Boolean formula F is given by a set of clauses each of which has length at most k. The problem is to determine whether there is an assignment to the variables that satisfies the formula. This problem has been studied by many researchers, and various algorithms have been proposed. Some algorithms indeed have a worst-case time complexity much better than the trivial exhaustive search algorithm; see, e.g.,[MS85,PPSZ98,Sch99,Kul99,DGHS00,Hir00a].

In [Sch99], Schöning gave the celebrated randomized algorithm for the CNF Satisfiability Problem (SAT), which runs in an expected time of 1.334^n (multiplied by a polynomial). The algorithm is a simple local search, i.e., (i) selecting an initial assignment at random, (ii) selecting an arbitrary unsatisfied clause and flipping one of the variables in it, and (iii) repeat step (ii) $3n$ times. He proved that the possibility p of successfully finding a satisfying assignment by this procedure is

$$p \geq \left(\frac{1}{2}\left(1+\frac{1}{k-1}\right)\right)^n, \tag{3.1}$$

where k is the maximum number of literals in each clause. In the case of 3SAT, the value of the right hand size is $(3/4)^n$. In other words, we can find a satisfying

assignment with high probability by repeating the above procedure roughly $(4/3)^n$ times (multiplied by a polynomial).

In this chapter, we first give a generalization of the equation 3.1, namely, we prove that

$$p \geq \prod_{i=1}^{n} \left(t_i + \frac{f_i}{k-1} \right),$$

where t_i ($f_i = 1 - t_i$, resp.) is the probability that variable x_i is assigned a correct (incorrect, resp.) value at the initialization step. (If $t_i = f_i = 1/2$, then (2) is the same as (1).) This equation says that if we have some knowledge on the value of x_i in a satisfying assignment, we can increase the success probability. For example, suppose that we know for some reason, 90% of the odd-indexed variables $x_1, x_3, x_5 \ldots$ take value 1 in a satisfying assignment. Then our best strategy is to select 1 initially for all the odd-indexed variables and to select 0 or 1 at random for the even-indexed variables. Then the success probability (when $k = 3$) calculated from (2) is

$$p \geq 1^{0.45n} \left(\frac{1}{2}\right)^{0.05n} \left(\frac{3}{4}\right)^{0.5n} \cong 0.836^n.$$

This means that we would be able to obtain solution in roughly $(1/0.836)^n = 1.196^n$ steps , which is much better than the original 1.334^n.

As a concrete example of such partial knowledge on solutions, we consider an imbalance between the number of 0's and 1's in the satisfying assignment. Suppose that we know the satisfying assignment includes $p_0 n$ 0's and $p_1 n$ 1's ($0 \leq p_0 \leq 1$ and $p_1 = 1 - p_0$). Then we can obtain optimal probabilities q_0 and q_1 ($= 1 - q_0$) by using (2), such that we should assign 0 to each variable with probability q_0 and 1 with q_1 at the beginning. Our result shows that the expected time complexity when we use this optimal initial-assignment is

$$T = \begin{cases} \left(\frac{1}{p_0}\right)^{p_0 n} \left(\frac{1}{p_1}\right)^{p_1 n} \left(\frac{k-1}{k}\right)^{n} & \text{for } \frac{1}{k} \leq p_0 \leq \frac{k-1}{k}, \\ (k-1)^{\min\{p_0 n, p_1 n\}} & \text{for } p_0 < \frac{1}{k} \text{ or } p_0 > \frac{k-1}{k}. \end{cases}$$

For example, when $p_0 = 2/3$, $T = 1.260^n$ and when $p_0 = 0.9$, we have $T = 1.072^n$. Such an imbalance of 0's and 1's often appears in instances encoded from other problems. F or example, SAT-instances encoded from the class-schedule problem [CIKM97, IKM+00] have the property that solutions must have very few 1's. Also, let us remember the famous result by Cook [Coo71] where SAT is first proved to be NP-complete. One can see that his reduction also has the same property.

In this chapter, we take a more combinatorial problem, i.e., 3-Dimensional Matching (3DM), as a concrete example of such an imbalance. An instance of 3DM is given as (W, X, Y, M) where $|W| = |X| = |Y| = q$ and $M \subseteq W \times X \times Y$. If each element in $W \cup X \cup Y$ appears in M evenly, i.e., roughly k times, then our reduction gives a kSAT instance using kq variables. Our reduction also assures that any satisfying assignment has exactly q 1's against the kq variables. In other words, the resulting formulas do have the imbalance whose degree is represented by $p_1 = 1/k$. Note that this reduction is quite natural and it appears hard to come up with another reduction (whether or not it creates the imbalance) which provides reasonably simple formulas.

We also show preliminary experimental results. Our instances are those encoded from 3DM and from prime factorization. It is clearly demonstrated that our approach is faster than the original Schöning's, especially for the second set of instances. Note that the second instances are harder than the other since the number of satisfying assignments is few.

Our result is not a general improvement, but we believe that there are many cases for which our approach is useful.

3.2 Selection of Initial Assignments

In this section, we first generalize the equation 3.1 which is then used to derive improved bounds for kCNF-formulas having the imbalance in their solutions. From now on, when we say algorithm A_s, it means **SCH** which repeats **RandomWalk** exactly one time.

Lemma 3.1 . *For some satisfying assignment a^*, let t_i $(1 \leq i \leq n)$ be the probability that variable x_i receives the same (correct) initial assignment as a^*. Also let $f_i =$*

$1 - t_i$. *Then the probability* p *that* A_s *is successful is*

$$p \geq \prod_{i=1}^{n} \left(t_i + \frac{f_i}{k-1} \right).$$

Proof. Suppose that X (X', resp.) is a random variable such that X (X', resp.) variables among $x_1, \ldots, x_n$ (among $x_2, \ldots, x_n$, resp.) receive incorrect values in the initial assignment. Then by Lemma 1, the probability p can be written as

$$\begin{aligned} p &\geq \sum_{j=0}^{n} Pr\{X = j\} \left(\frac{1}{k-1} \right)^j \\ &= t_1 \sum_{j=0}^{n-1} Pr\{X' = j\} \left(\frac{1}{k-1} \right)^j + f_1 \sum_{j=0}^{n-1} Pr\{X' = j\} \left(\frac{1}{k-1} \right)^{j+1} \\ &= \left(t_1 + \frac{f_1}{k-1} \right) \sum_{j=0}^{n-1} Pr\{X' = j\} \left(\frac{1}{k-1} \right)^j \end{aligned}$$

By applying a similar reduction to the summation term $n - 1$ times, we can obtain the inequality in the lemma. □

Now we consider kCNF-formulas having the imbalance in their solutions. Suppose that a given formula f has a satisfying assignment a^* which has l 0's and $(n - l)$ 1's. Let $p_0 = l/n$ and $p_1 = (n - l)/n$. Our new algorithm **IT** is the following:

IT(CNF-formula F, real p_0, integer T)
  repeat T times
    y = randomly generated vector $\in \{0,1\}^n$ satisfies the following:
      $\mathbf{Pr}[y_i = 0] = q_0(p_0), \mathbf{Pr}[y_i = 1] = 1 - q_0(p_0)$
    $z = \mathbf{RandomWalk}(F, y)$;
    if z satisfies F
      then output(z); exit;
  end
  output('Unsatisfiable');

Similarly, We denoted by $A_s(p_0)$, **IT** which repeats **RandomWalk** exactly one time. The difference from A_s only in selecting initial assignments: Namely, each variable x_i is assigned 0 with probability q_0 and is assigned 1 with probability $1-q_0$, where the value of q_0 is given by the following theorem.

Theorem 3.1 *Let p be the probability that Algorithm $A_s(p_0)$ is successful. Then p becomes maximum when the probability q_0 with which each variable is assigned 0 initially is given as*

$$q_0(p_0) = \begin{cases} 1 & \text{for } p_0 < \dfrac{1}{k}, \\ \dfrac{kp_0 - 1}{k-2} & \text{for } \dfrac{1}{k} \le p_0 \le \dfrac{k-1}{k}, \\ 0 & \text{for } p_0 > \dfrac{k-1}{k}, \end{cases}$$

and the value of p for this optimal q_0 is

$$p(p_0) \ge \begin{cases} p_0^{p_0 n} p_1^{p_1 n} \left(\dfrac{k}{k-1}\right)^n & \text{for } \dfrac{1}{k} \le p_0 \le \dfrac{k-1}{k}, \\ \left(\dfrac{1}{k-1}\right)^{\min\{p_0 n, p_1 n\}} & \text{for } p_0 < \dfrac{1}{k} \text{ or } p_0 > \dfrac{k-1}{k}. \end{cases}$$

Proof. By Lemma 2, the probability p can be written as

$$p(p_0) \ge \prod_{i=0}^{n} \left(t_i + \frac{f_i}{k-1}\right) = \left(q_0 + \frac{q_1}{k-1}\right)^{p_0 n} \left(q_1 + \frac{q_0}{k-1}\right)^{p_1 n}.$$

To decide the value of q_0 that maximizes p, we consider the following function

$$\begin{aligned} \sigma(q_0) &= \log\left\{\left(q_0 + \frac{q_1}{k-1}\right)^{p_0} \left(q_1 + \frac{q_0}{k-1}\right)^{p_1}\right\} \\ &= p_0 \log\left\{\left(1 - \frac{1}{k-1}\right) q_0 + \frac{1}{k-1}\right\} + (1-p_0)\log\left\{-\left(1 - \frac{1}{k-1}\right) q_0 + 1\right\}. \end{aligned}$$

σ is convex in $[0,1]$, so it takes maximum value where its derivative is 0 or at either

end of the interval $[0, 1]$. Since

$$\sigma'(q_0) = p_0 \frac{\left(1 - \frac{1}{k-1}\right)}{\left(1 - \frac{1}{k-1}\right) q_0 + \frac{1}{k-1}} + (1 - p_0) \frac{-\left(1 - \frac{1}{k-1}\right)}{-\left(1 - \frac{1}{k-1}\right) q_0 + 1},$$

$\sigma'(q_0) = 0$ implies $q_0 = (kp_0 - 1)/(k - 2)$. By substituting this optimal q_0, or substituting $q_0 = 0$ or $q_0 = 1$ if $(kp_0 - 1)/(k - 2)$ is less than 0 or greater than 1, respectively, we obtain the theorem. □

Remark *The value of q_0 is quite different from the value of p_0. For example, if $p_0 = 0.6$ and $k = 3$, the value of q_0 is 0.8, and if $p_0 \geq 2/3$, then $q_0 = 1.0$. Namely, the imbalance should be expanded in the initial assignment.*

Fig. 3.1 shows numerical examples of Theorem 3.1 for $k = 3, 4, 5$, and 6. The horizontal axis shows the value of $0 \leq p_0 \leq 1$ and the vertical axis shows the value of c supposing that the optimal bound of Theorem 3.1 is represented as c^n. Note that the time complexity is roughly bounded by $1/p$.

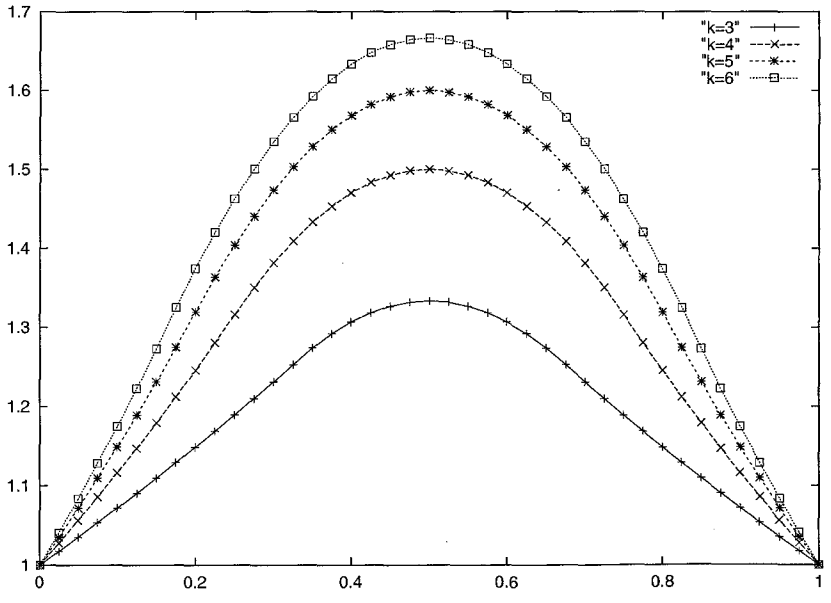

Figure 3.1: Numerical examples of Theorem 1

3.3 Three Dimensional Matching

An instance of 3DM is given as (W, X, Y, M) where W, X and Y are disjoint sets of size q and M ($|M| = n$) is a subset of $W \times X \times Y$. Its question is whether or not there is a subset $M' \subseteq M$ such that $|M'| = q$ and all elements in W, X, Y appear (exactly once) in M'. For an integer k, k3DM is a restricted version of 3DM, namely, each element in W, X, Y appears at most k times in M (and therefore $n \leq kq$). k3DM can be reduced to kSAT as follows: For given (W, X, Y, M), we construct a formula F such that: (i) F uses n variables $z_1, z_2, \ldots, z_n$ (z_i corresponds to the ith triple in M). (ii) Suppose that an element $w \in W$ appears in the i_1th, i_2th, ..., i_kth triples in M. Then we prepare a CNF-formula $U^w(z_{i_1}, z_{i_2}, \ldots, z_{i_k})$ such that it becomes 1 if and only if exactly one of $z_{i_1}, z_{i_2}, \ldots, z_{i_k}$ is 1. When $k = 3$, for example, $U^w(z_{i_1}, z_{i_2}, z_{i_3})$ can be written as

$$\begin{aligned}(\neg z_{i_1} \vee \neg z_{i_2} \vee \neg z_{i_3}) \quad &\wedge \quad (\neg z_{i_1} \vee \neg z_{i_2} \vee z_{i_3}) \wedge (z_{i_1} \vee \neg z_{i_2} \vee \neg z_{i_3}) \\ &\wedge \quad (\neg z_{i_1} \vee z_{i_2} \vee \neg z_{i_3}) \wedge (z_{i_1} \vee z_{i_2} \vee z_{i_3}).\end{aligned}$$

(iii) The entire formula F is a conjunction of U^w for all $w \in W$, U^x for all $x \in X$ and U^y for all $y \in Y$.

We can easily see that (i) F is satisfiable iff the original (W, X, Y, M) has a matching, and (ii) if F is satisfiable, then any solution has q 1's, i.e., an imbalanced satisfying assignment. For example, if $k = 3$, then $p_1 = 1/3$ and $A_s(2/3)$ finds a solution in time 1.260^n and if $k = 4$, then $A_s(1/4)$ for 4SAT does so in time 1.317^n.

As a comparison, let us consider a naive method of solving k3DM directly. Since each element in W appears in M at most k times, there are at most k^q different ways of selecting q (or less) triples from M which cover all elements in W. One can compute whether or not these q triples constitute a matching in polynomial time. Thus the time complexity of this algorithm can be written as $k^{n/k}$. This is 1.443^n for $k = 3$ and 1.588^n for $k = 4$. In both cases our bounds of $A_s(1/k)$ are much better. Note that it is hard to find other reductions which are reasonably simple, whether or not their satisfying assignments are balanced. Figure 3.2 shows the running time of a naive algorithm. The horizontal axis shows the value of α where $q = \alpha n$ and

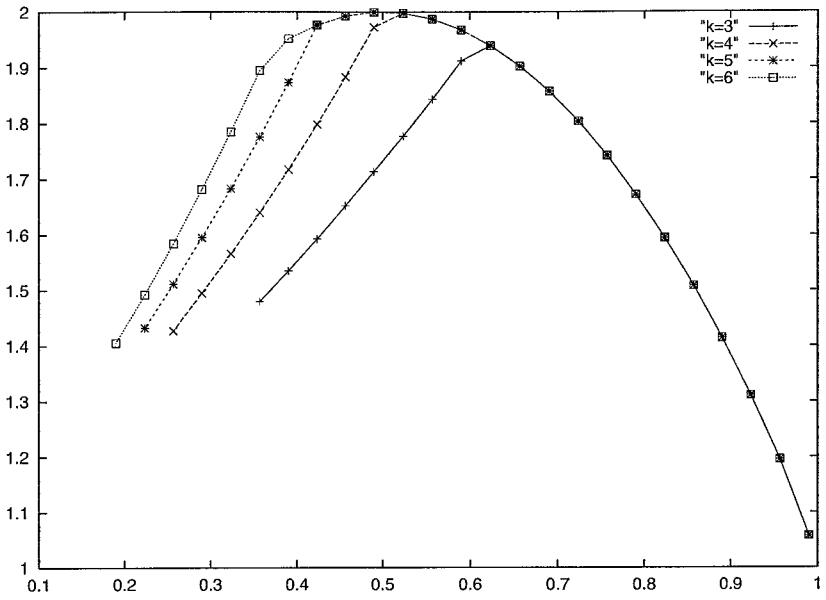

Figure 3.2: Running time of naive algorithms

the vertical axis shows the running time for each $k = 3, 4, 5, 6$. Figure 3.3 shows the comparison of the running time of a naive algorithm, original Schöning's and ours.

3.4 Experiments

Experiments were conducted for CNF-formulas reduced from 3DM and from prime factorization. For the 3DM formulas, we first obtain a random 3DM instance by generating n triples which are to be in M. This generation is basically random but (i) to assure that the instance has a matching, we first generate an artificial matching (of q triples) and then (ii) add $n - q$ triples so that each element in $W \times X \times Y$ appears exactly three times. This 3DM instance is reduced to a 3SAT instance as described above.

We have generated 15 different formulas for each of $n = 250, 500, 750, 1000$ and 1250. We tested six different local-search algorithms:

(1) Pure GSAT [SLM92].

(2) Weighting [CI95, CI96, Mor93, SK93].

(3) Weighting + 50% Random Walk.

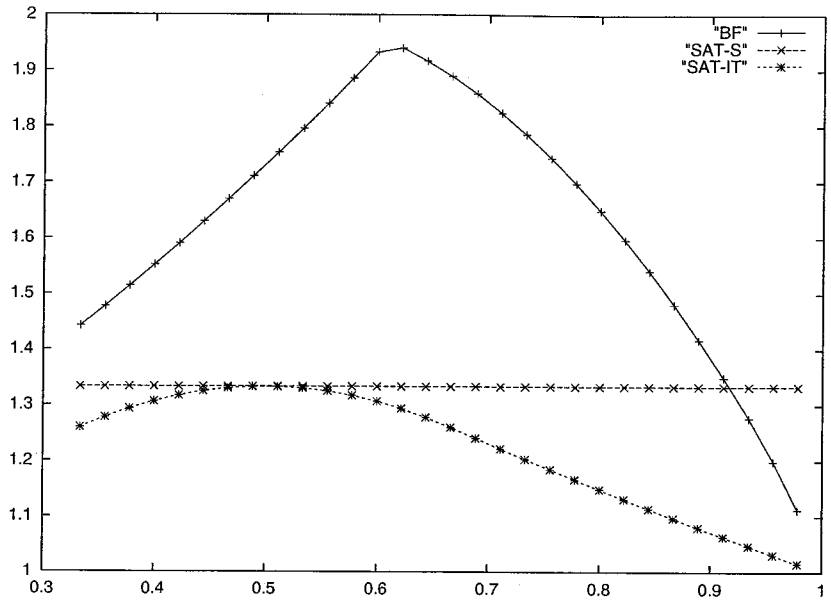

Figure 3.3: Comparison of the running time for $k = 3$

(4) GSAT + 50% Random Walk [SK93].

(5) Schöning.

(6) Our modified Schöning.

Each algorithm is run 100 times (using different random numbers) for each single instance. Since (5) and (6) execute $3n$ flips in a single try, we also execute the same number of flips in other algorithms. (Since each algorithm has its own recommended value for the number of flips in a single try, this setting might not be too fair.) The result is given in Table 1, which shows the ratio of successful tries (average values over 15 instances). Algorithms (1) through (6) are denoted by g, gw, gwn, gn, A_s and $A_s(2/3)$, respectively. We also tested algorithms g, gw, gwn and gn for the imbalanced initial assignments as $A_s(2/3)$, but we did not find clear differences.

One can immediately see that the absolute success ratio of our algorithm is very high compared to the analysis. Obvious reason is that each instance has a lot of satisfying assignments; there is a good chance that one of them happens to be quite close to the initial assignment chosen by the algorithm. Although this nature certainly discourages the effort of selecting initial assignments cleverly, our $A_s(2/3)$ is clearly better than others.

# of variables	g	gw	gwn	gn	A_s	$A_s(2/3)$
250	0.129	0.176	0.528	0.475	0.468	0.613
500	0.002	0.058	0.4355	0.41	0.405	0.581
750	0.00	0.019	0.368	0.388	0.428	0.599
1000	0.00	0.015	0.37	0.385	0.357	0.585
1250	0.00	0.007	0.345	0.354	0.367	0.631

Table 3.1: 3DM Instances

q_0	0.5	0.6	0.7	0.8	0.9	1.0
	0.356	0.475	0.576	0.634	0.648	0.598

Table 3.2: Effect of the value of q_0

Note that $A_s(2/3)$ initially assigns 0 to all the variables (i.e., $q_0 = 1.0$) by Theorem 1. Table 2 shows how the success ratio changes according to the value of q_0 by using five 3DM formulas of 1250 variables. Our algorithm is run 5000 times for each instance for $q_0 = 0.5, 0.6, 0.7, 0.8, 0.9$ and 1.0 and the table shows the average success ratio of five instances. As mentioned above, $A_s(2/3)$ becomes optimal for $q_0 = 1.0$. However, the experiments suggest the optimal point exists around $q_0 = 0.9$ probably for the same reason mentioned before.

Another benchmark is a reduction from prime factorization. Again instances are 3CNF-formulas, which are denoted by P_n. If integer n can be represented by $n = n_1 \times n_2$ with integers n_1 and $n_2 > 1$, then P_n has a single satisfying assignment corresponding to this pair of n_1 and n_2. Hence, if n is a product of two prime numbers, P_n has only one satisfying assignment. P_n is constructed by simulating the usual multiplication procedure using many auxiliary variables other than those used for binary representations of n_1 and n_2. Our experiment has used P_{129} which uses 136 variables and contains 337 clauses. Since $129 = 3 \times 43$, P_{129} has only one satisfying assignment. To make the imbalanced situation, we flip the polarities of the variables appropriately so that the satisfying assignment has 50%, 60%, 70%, 80% and 90% 0's.

Table 3 shows how many times $A_s(q_0)$ succeeds out of 100,000 tries. Each column corresponds to the imbalance described above (50% for the first column and 90% for

the last one). Note that the optimal q_0 for the 60% imbalance is 0.8 due to Theorem 1. Note that P_{129} includes a lot of clauses which contains only one or two literals, which obviously makes it easy to solve.

p_0	q_0					
	0.5	0.6	0.7	0.8	0.9	1.0
0.5	5	5	5	11	4	8
0.6	4	11	8	14	16	8
0.7	6	6	15	20	28	50
0.8	6	15	28	74	179	520
0.9	9	4	61	235	994	5063

Table 3.3: Prime-factorization instances

Our third benchmark was taken from the DIMACS benchmark set. We tested only one instance called aim-50-1_6-yes1-1.cnf, which is basically a random 3SAT instance (each clause includes exactly three literals) and has 50 variables and 80 clauses. Also, it has only one satisfying assignment. Thus this instance appears the hardest among what we used in the experiments. Like Table 3, Table 4 shows how many times $A_s(q_0)$ succeeds out of 500,000 tries. (Note that the number of tries is five times as many as Table 3.)

p_0	q_0					
	0.5	0.6	0.7	0.8	0.9	1.0
0.5	8	4	6	6	5	7
0.6	13	10	9	17	25	25
0.7	7	20	30	58	139	318
0.8	8	22	54	187	585	1449
0.9	10	24	154	778	5356	25274

Table 3.4: DIMACS instances

3.5 Constraint Satisfaction Problems

Schöning shows in [Sch99] that a similar local search algorithm is also efficient for Constraint Satisfaction Problem (CSP). An instance of CSP is a set of constraints

$C_1, C_2, \ldots, C_m$ and each constraint $C_i(x_1, x_2, \ldots, x_n)$ is a function from $\{0, 1, \ldots, d-1\}$ into $\{0, 1\}$. Each variable x_i takes one of the d different values. If each constrain depends on at most l variables, we call the problem (d, l)-CSP. Slightly changing **RandomWalk** algorihtm, the followings hold:

Lemma 3.2 ([Sch02]) *Let $d \geq 3$ and F be a (d, l)-CSP instance and a^* be a satisfying assignment for F. For each assignment a, the probability that a satisfying assignment is found by* **RandomWalk**(F, a) *is at least $\left(\frac{1}{h-1}\right)^{d(a,a^*)}$ where $h = (l-1)(d-1)+1$.*

Theorem 3.2 ([Sch02]) *For any satisfiable (d, l)-CSP instance F on n variables and $d \geq 3$, the success probability of one repeat-iteration of* **SCH** *is at least $d^{-n}\left\{1 + \left(\frac{d-1}{h)}\right)\right\}^n$ where $h = (l-1)(d-1)+1$.*

Suppose that there is a similar imbalance in a solution, such that $p_0 n$ variables take value 0 (and the other $(1-p_0)n$ ones take 1 through $d-1$). Then by changing the initial assignment exactly as before, we can improve the expected time complexity T as follows.

$$T = \begin{cases} \left(\dfrac{d-1}{h-1}\right)^{-p_0 n}\left(1+\dfrac{d-2}{h-1}\right)^{-p_1 n} & \text{for } p_0 < \dfrac{1}{h+d-2}, \\ \left(q_0+\dfrac{(d-1)q_1}{h-1}\right)^{-p_0 n}\left(q_1+\dfrac{q_0+(d-2)q_1}{h-1}\right)^{-p_1 n} & \text{for } \dfrac{1}{h+d-2} \leq p_0 \leq \dfrac{h-1}{h+d-2}, \\ \left(\dfrac{d-1}{h-1}\right)^{-p_0 n} & \text{for } p_0 > \dfrac{h-1}{h+d-2}, \end{cases}$$

where $p_1 = 1 - p_0$, $q_0 = \dfrac{(h+d-2)p_0 - 1}{h-2}$, $q_1 = 1 - \dfrac{q_0}{d-1}$ and $h = (l-1)(d-1)+1$.

Proof. The success probability is estimated by Lemma 3.2 as follows:

$$p(p_0) \geq \left(p_0 + \frac{(d-1)p_1}{h-1}\right)^{p_0 n}\left(p_1 + \frac{p_0+(d-2)p_1}{h-1}\right)^{p_1 n}$$

We would like to maximize the value of

$$\begin{aligned}
&\left(p_0 + \frac{(d-1)p_1}{h-1}\right)^{p_0} \left(p_1 + \frac{p_0 + (d-2)p_1}{h-1}\right)^{p_1} \\
= &\left\{\left(1 - \frac{1}{h-1}\right)y + \frac{1}{d}\left(1 + \frac{d-1}{h-1}\right)\right\}^{p_0} \\
\times &\left\{\left(-\frac{1}{d-1} - \frac{h-2}{(d-1)(l-1)} + \frac{1}{h-1}\right)y + \left(\frac{1}{d} + \frac{d-2}{d(d-1)} + \frac{1}{d(h-1)}\right)\right\}^{p_1}.
\end{aligned}$$

Define

$$f(y) = (ay + b)^{p_0}(cy + d)^{p_1},$$

then this value becomes maximum at

$$y = -\frac{p_0 ad + p_1 cb}{(p_0 + p_1)ac}.$$

Substituting approapriate values, it is shown that $p(p_0)$ becomes maximum at

$$y = \frac{h + d - 2}{d(h-2)} \cdot \frac{(d-1)p_0 - p_1}{p_0 + p_1}.$$

□

Fig. 3.4 shows numerical examples of for $d = 4, l = 3, 4, 5, 6$ and Fig. 3.5 shows numerical examples of for $d = 3, 4, 5, 6, l = 4$. The horizontal axis shows the value of $0 \leq p_0 \leq 1$ and the vertical axis shows the value of c supposing that the optimal time bound represented as c^n.

3.6 Deterministic Algorithms for Large bias

In this section, we discuss the derandomization of our result. Dantsin et. al. [DGH+02] have shown that Schöning's algorithm can be derandomized. Namely, they gave a deterministic algorithm similar to Schöning's. Thus it is natural to try to derandomize our algorithm. The following algorithm is a deterministic analogue of **RandomWalk** shown in [HSSW02].

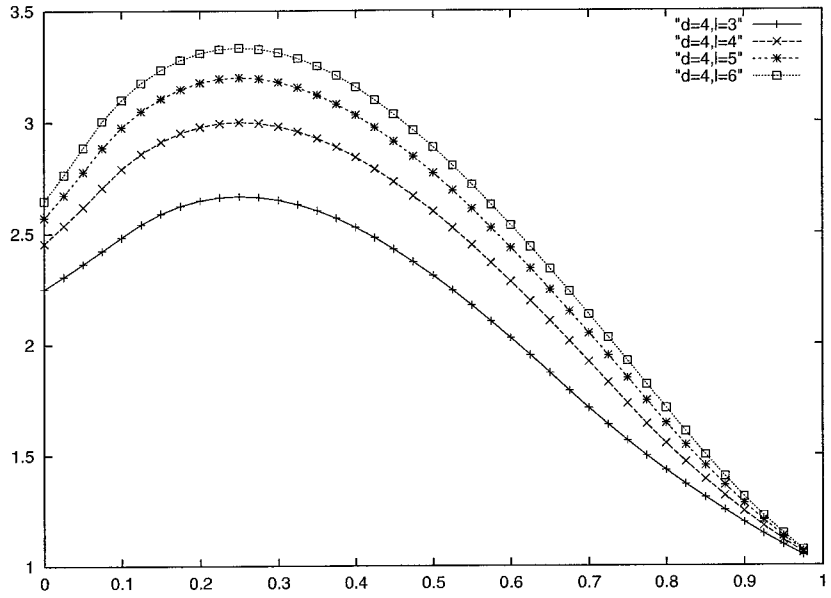

Figure 3.4: Running time for different l

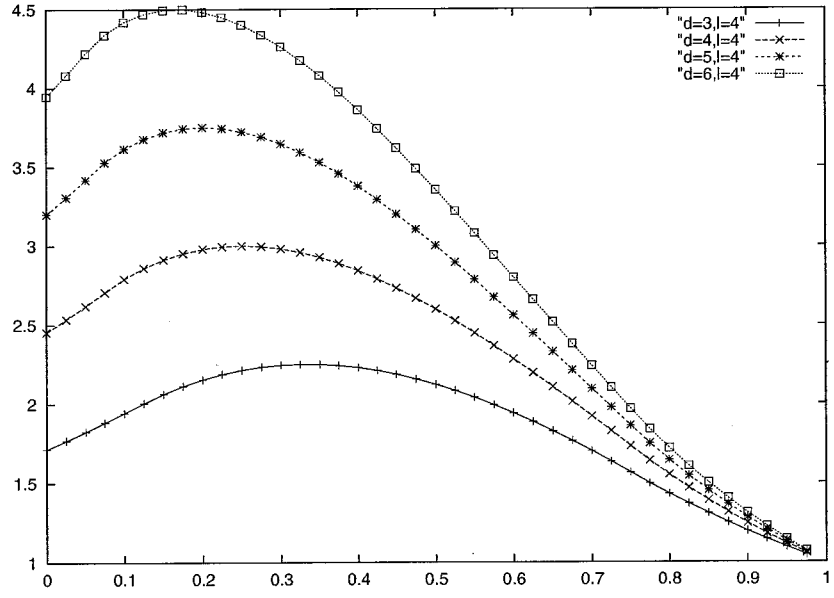

Figure 3.5: Running time for different d

LocalSearch(CNF formula G, assignment y, integer r);
  if y satisfies G
    then return y
  else if $r < 0$
    then return null
  else
    Pick a clause C false under y
    for each literal $l \in C$
      do flip l's value in y
      if **LocalSearch**$(G|_{l=1}, y, r-1)$ returns
        a satisfying assignment y
        then return y
  return null
  end

LocalSearch and its valiant is analyzed in [DGH+02, BK04] and the running time is estimated as follows:

Lemma 3.3 ([DGH+02, BK04]) *For any satisfiable k-CNF formula F,* **LocalSearch**(F, y, r) *finds a satisfying assignment a s.t. $d(y, a) \leq r$ in time at most k^r. For $k = 3$, the running time can be reduced to 2.792^r.*

This lemma yields derandomization of our result immediately. Let F be a k-CNF formula and assume its satisfying assignment has at most $p_0 n$. Then if we run **LocalSearch**(F, y, r) with $y = 000\cdots0, r = p_0 n$, we can find a satisfying assignment. The running time is greatly improved when the bias is large, but we get no improvement for small bias case. Figure 3.6 shows the running time of derandomized algorithm.

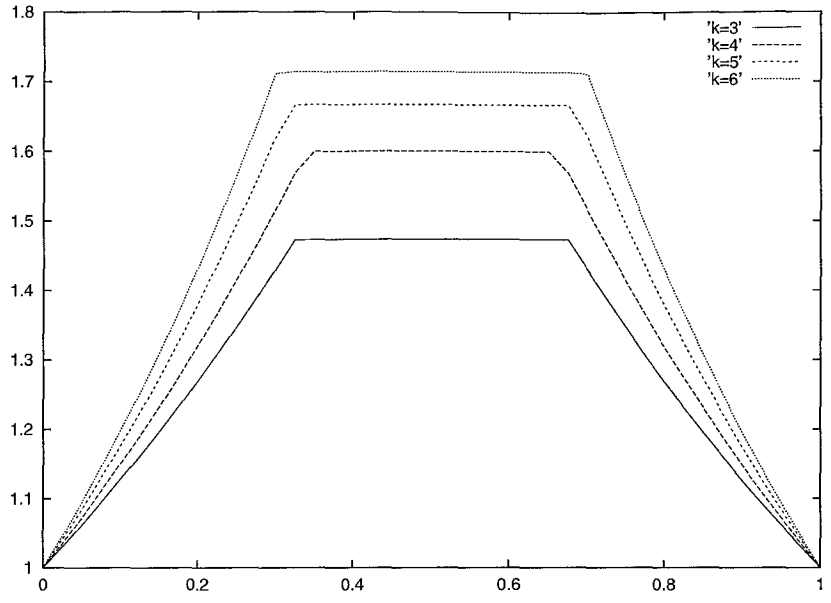

Figure 3.6: Running time of deterministic algorithm

3.7 Concluding Remarks

In this chapter, we improve Schöning's randomized walk algorithm using partial knowledge on solutions. Actually we improve the algorithm for the formula with a satisfying assignment in which the number of 0's and 1's are unbalanced. Though our algorithm do not improve the general case, we can apply our algorithm to some natural combinatorial problems like 3DM matching, where the number of 0's and 1's in satisfying assignments are often unbalanced. Then we give some experimental results that shows our algorithm is faster for several instances in practical sense.

Appendix

Table 3.5 shows the application of our result to several NP-complete problems. Reduction to CNF formula is similar to 3-Dimensional Matching case and we obtain non-trivial algorithms for these problems.

Problem	Brute Force	SAT	p_1
3-Dimensional Matching	$\min\left\{\binom{n}{\alpha n}, k^{\alpha n}\right\}$	k-SAT	α
d-Dimensional Matching	$\min\left\{\binom{n}{\alpha n}, k^{\alpha n}\right\}$	k-SAT	α
EXACT COVER BY 3-SETS	$\min\left\{\binom{n}{\alpha n}, k^{\alpha n}\right\}$	k-SAT	α
MONOCHROMATIC TRIANGLE	$\binom{n}{\alpha n/2}$	3-SAT	α
SET SPLITTING	$\binom{n}{\alpha n/2}$	k-SAT	α

Table 3.5: Reduction to CNF formula

Chapter 4

Improved Upper Bounds for 3-SAT

4.1 Introduction

4.1.1 New wrost-case upper bounds for 3SAT

Worst-case upper bounds for 3SAT (k-SAT in general) have been one of the most well-studied topics in theoretical computer science. For small k's, especially for $k = 3$, there exists a lot of algorithms which run significantly faster than the trivial 2^n bound. Roughly speaking most algorithms are based on Davis-Putnam. [Sch99] is the first local search algorithm which gives a guaranteed performance for general instances and [DGH+02], [HSSW02], [BS03] and [Rol03] follow up this Schöning's approach. The following Figure 4.1 summarizes those algorithms where a constant c means that the algorithm runs in time $O(c^n)$.

Our new bounds are denoted by [*] in the above list, namely we prove:

Theorem 4.1 *For any satisfiable n-variable 3-CNF (4-CNF) formula F, there exists a randomized algorithm that finds a satisfying assignment of F in expected running time* $O(1.3225^n)$ $(O(1.4705^n))$.

The basic idea is to combine two existing algorithms, the one by Paturi, Pudlák, Saks and Zane [PPSZ98] and the other by Schöning [Sch99]. It should be noted, however, that simply running the two algorithms independently does not seem to work. Also, our approach can escape one of the most complicated portions in the

3SAT	4SAT	5SAT	6SAT	type	ref.
1.839	-	-	-	det.	[MS79]
1.782	1.835	1.867	1.888	det.	[PPZ99]
1.769	-	-	-	det.	[Dan83]
1.618	1.839	1.928	1.966	det.	[Luc84, MS85]
1.588	1.682	1.742	1.782	prob.	[PPZ99]
1.579	-	-	-	det.	[Sch92]
1.505	-	-	-	det.	[Kul99]
1.481	1.6	1.667	1.75	det.	[DGH+02]
1.474	-	-	-	det.	[BK04]
1.362	1.476	1.569	1.637	prob.	[PPSZ98]
1.334	1.5	1.6	1.667	prob.	[Sch02]
1.3302	-	-	-	prob.	[HSSW02]
1.3300	-	-	-	prob.	[**]
1.3290	-	-	-	prob.	[BS03]
1.3280	-	-	-	prob.	[Rol03]
1.3225	1.4705	-	-	prob.	[*]

Figure 4.1: Worst-case upper bounds for k-SAT

analysis of [PPSZ98]. In this chapter we focus on the 3-SAT case; the 4-SAT case is very similar and may be omitted. The same approach does not improve the bounds for 5-SAT or more.

The algorithm of [PPSZ98] is called ResolveSat, which is based on a randomized Davis-Putnam combined with bounded resolution. ResolveSat behaves like local search algorithms, that is, it takes an random assignment and a random variable ordering as an input and tries to modify initial assignment to be satisfiable one by using Davis-Putnam procedure. This algorithm has the unique feature that it achieves a quite nice performance, $O(1.3071^n)$, for a unique 3-CNF formula, i.e., a formula which has only one satisfying assignment. As the number m of satisfying assignments grows, the bound, denoted by $T_{\text{PPSZ}}(m)$, degenerates, i.e., $T_{\text{PPSZ}}(m)$ is an increasing function. [PPSZ98] needed a lot of effort to stop this degeneration by formalizing the intuition that if the formula has many satisfying assignments, then finding one should be easy.

In contrast, the algorithm of [Sch99] is based on the standard local search for which the above intuition is obviously true. Namely its running time $T_{\text{SCH}}(m)$ is the

worst when $m = 1$ and then decreases. Recall that $T_{\text{PPSZ}}(1) < T_{\text{SCH}}(1) = O(1.334^n)$. So, if we run the two algorithms in parallel, then the running time is bounded by $\min\{T_{\text{PPSZ}}(m), T_{\text{SCH}}(m)\}$ which becomes maximum $(= T_{\text{PPSZ}}(m_0) = T_{\text{SCH}}(m_0))$ at $m = m_0$. Obviously $T_{\text{SCH}}(m_0) < T_{\text{SCH}}(1)$. Although $T_{\text{SCH}}(1)$ is not the currently best, there is a lot of hope of breaking it since $T_{\text{PPSZ}}(1)$ is much better than the current best.

Unfortunately, this approach has an obstacle. We know the value of $T_{\text{PPSZ}}(m)$ but we do not know that of $T_{\text{SCH}}(m)$ for the following reason. To obtain $T_{\text{SCH}}(m)$, it appears that we need to know the Hamming distance between the (randomly chosen) initial assignment and its closest satisfying assignment. However, there is no obvious way of doing so, since it is quite hard to analyze how (multi) satisfying assignments of a 3-CNF formula can distribute in the whole space of 2^n assignments. To overcome this difficulty, we analyze two algorithms simultaneously, that is, we give a lower bound on the success probability that at least one algorithm returns a satisfying assignment given a same initial assignment.

4.1.2 Improving Randomized Local Search Algorithms

As seen in the previous chapters, Schöning's randomized local search algorithm is very simple and fast. Thus to improve this algorithm was thoght to be very difficult. First breakthrough was achived by [SSW01, HSSW02] and upper bounds for 3SAT is reduced to 1.3303^n from $(4/3)^n$. The authors of [SSW01, HSSW02] observed that there is a better way to obtain an initial assignment than simply generating uniformly at random. They introduece the notion of *independent clauses* of CNF formula and show that if the number of independent clauses is sufficeintly large, we can obtain a good initial assignment. However, it is not necessary that the number of independent clauses is large. To deal such a case, they use another simple algorithm that runs very fast for a formula with small number of independent clauses. In this chapter, we present an improved algorithm of [HSSW02] extending the notion of independent clauses.

Organization of chapter In section 2, we present the analysis of our new algorithm that achieves new worst case upper bounds for 3SAT. In section 3, slightly refined analysis of our algorithm is given. In section 4, we review the basic idea of [HSSW02]. In secton 5, we present our improvement.

4.2 Combination of Two Algorithms

As mentioned previously, we cannot analyze a simple repetition of **SCH** and **PPSZ**. Our solution is to use the same random assignment for each execution of **SCH** and **PPSZ**. Namely, our algorithm is:

IT(CNF-formula F, integer s, integer T)
  $F_s =$ **Resolve**(F, s).
  repeat T times
    $y =$ uniformly random vector $\in \{0,1\}^n$
    $\pi =$ uniformly random permutation of $1, 2, \ldots, n$
    $z =$ **Modify**(F_s, π, y);
    $z' =$ **RandomWalk**(F, y);
    if z satisfies F
      then output(z); exit;
    else if z' satisfies F
      then output(z'); exit;
  end
  output('Unsatisfiable');

Now we present the analysis of our new algorithm. Let p_0 be the probability that the above single try finds a satisfying assignment if the given formula is satisfiable. To obtain p_0, there are two key lemmas, for which we recall some definitions. Let sat(F) be the set of satisfying assignments of the formula F. A set of assignments, $C \subseteq \{0,1\}^n$, is called a *subcube*, if C is determined by fixing a certain number of variables. For example, $\{0000, 0001, 0010, 0011\}$ is a subcube obtained by fixing $x_1 = x_2 = 0$. Now it turns out that the whole space, $\{0,1\}^n$, can always be partitioned

into a family $\{C_z \mid z \in \mathrm{sat}(F)\}$ of disjoint subcubes so that C_z contains $z \in \mathrm{sat}(F)$ but no other $z' \in \mathrm{sat}(F) - \{z\}$ (see chapter 2). Note that the existence (not explicit construction) of such partition suffices for our purpose.

Now, given formula F with a set of satisfying assignments S and the subcube partition $\{\mathbf{C}(z, I_z) \mid z \in S\}$, $\tau(F, z|\mathbf{C}(z, I_z))$ ($\sigma(F, z|\mathbf{C}(z, I_z))$, resp.) is defined as the probability (averaged over y and π) that a single execution of **Modify** (**RandomWalk**, resp.) finds the assignment z under the condition that the initial assignment $y \in \mathbf{C}(z, I_z)$. We state the lower bounds of $\tau(F, z|\mathbf{C}(z, I_z))$ and $\sigma(F, z|\mathbf{C}(z, I_z))$.

Lemma 4.1 *For any satisfiable 3CNF formula F and any partition $\mathbf{C}(z, I_z)$, if $y \in \mathbf{C}(z, I_z)$ is chosen uniformly at random, then the value $\tau(F_s, z|\mathbf{C}(z, I_z))$ is bounded as follows:*

$$\tau(F_s, z|\mathbf{C}(z, I_z)) \geq 2^{-(1-\gamma_3)n-\gamma_3|I_z|},$$

where $\gamma_3 = 2 - 2\ln 2$.

Lemma 4.2 *For any satisfiable 3CNF formula F and any partition $\mathbf{C}(z, I_z)$, if $y \in \mathbf{C}(z, I_z)$ is chosen uniformly at random, then the value $\sigma(F, z|\mathbf{C}(z, I_z))$ is bounded as follows:*

$$\sigma(F, z|\mathbf{C}(z, I_z)) \geq \left(\frac{3}{4}\right)^{n-|I_z|}.$$

Proof. Omitted. □

Now our success probability of a single run is at least the probability of Lemmas 4.1 and 4.2, i.e.,

$$\begin{aligned} p_0 &\geq \sum_{z \in \mathrm{sat}(F)} \max\{\tau(F, z|\mathbf{C}(z, I_z)), \sigma(F, z|\mathbf{C}(z, I_z))\}\mathbf{Pr}[z \in \mathbf{C}(z, I_z)] \\ &\geq \min_{z \in \mathrm{sat}(F)} \max\{\tau(F, z|\mathbf{C}(z, I_z)), \sigma(F, z|\mathbf{C}(z, I_z))\} \end{aligned}$$

since

$$\sum_{z \in \mathrm{sat}(F)} \mathbf{Pr}[z \in \mathbf{C}(z, I_z)] = 1.$$

$$p_0 \geq \max\{\tau(F, z|\mathbf{C}(z, I_z)), \sigma(F, z|\mathbf{C}(z, I_z))\},$$

which becomes minimum $(= \Omega(1.3227^{-n}))$ when

$|I_z| = 0.027940n$. Now the standard probabilistic argument allows us to claim that our algorithm finds a satisfying assignment with high probability for $I = O(1.3227^n)$. Recall that I is the number of repetitions.

Similar result holds for 4SAT using following lemmas.

Lemma 4.3 *For any satisfiable 4CNF formula F and any partition $\mathbf{C}(z, I_z)$, if $y \in \mathbf{C}(z, I_z)$ is chosen uniformly at random, then the value $\tau(F_s, z|\mathbf{C}(z, I_z))$ is bounded as follows:*

$$\tau(F_s, z|\mathbf{C}(z, I_z)) \geq 2^{-(1-\gamma_4)(n-|I_z|)}(2/3)^{|I_z|},$$

where $\gamma_4 = 0.4451818849$.

Lemma 4.4 *For any satisfiable 4CNF formula F and any partition $\mathbf{C}(z, I_z)$, if $y \in \mathbf{C}(z, I_z)$ is chosen uniformly at random, then the value $\sigma(F, z|\mathbf{C}(z, I_z))$ is bounded as follows:*

$$\sigma(F, z|\mathbf{C}(z, I_z)) \geq \left(\frac{2}{3}\right)^{n-|I_z|}.$$

Now

$$p_0 \geq \max\{\tau(F, z|\mathbf{C}(z, I_z)), \sigma(F, z|\mathbf{C}(z, I_z))\},$$

which becomes minimum $(= \Omega(1.4705^{-n}))$ when $|I_z| = 0.049007n$.

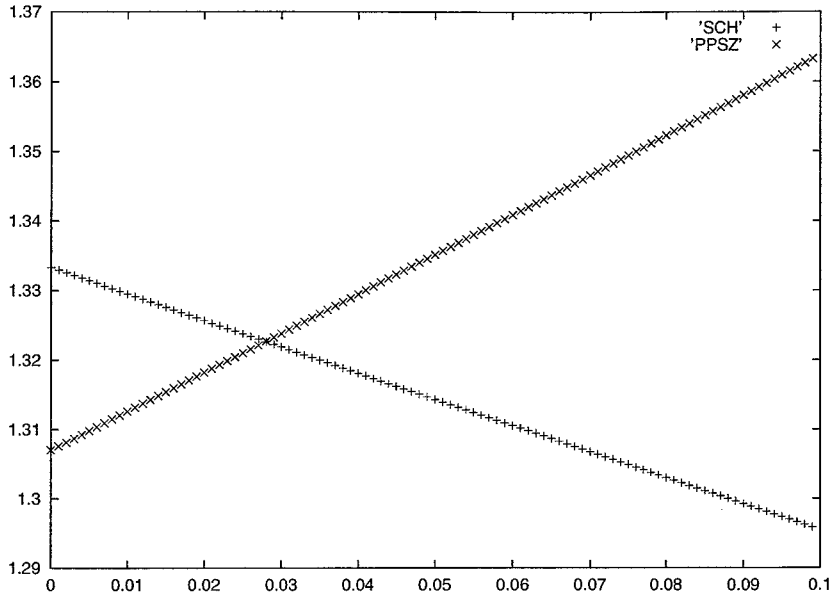

Figure 4.2: Trade-offs of PPSZ and SCH

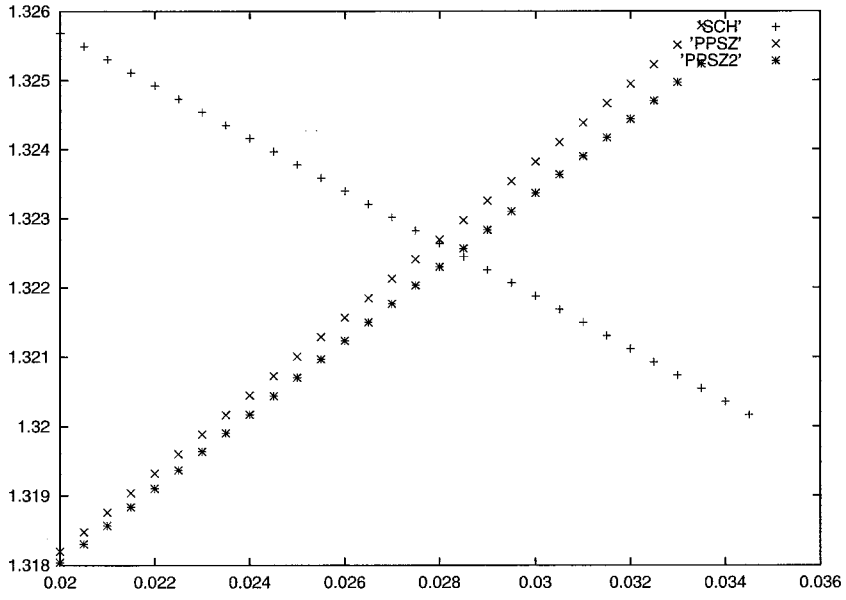

Figure 4.3: Refined analysis of PPSZ

4.3 Further Improvements

In this section, we slightly improve the running time of our algorihm by more refined analysis. We used lemma 4.1 for lower bounding the success probability of **PPSZ** algorithm. However, lemma 4.1 is obtained by the analysis optimized for unique 3SAT. We can obtain a different bound that yields better trade-offs as following:

Lemma 4.5 *For any satisfiable 3CNF formula F and any partition $\mathbf{C}(z, I_z)$ described above, if $y \in \mathbf{C}(z, I_z)$ is chosen uniformly at random, then the value $\tau(F_s, z|\mathbf{C}(z, I_z))$ is bounded as follows:*

$$\tau(F_s, z|\mathbf{C}(z, I_z)) \geq 2^{-(1-\gamma(\alpha))(n-|I_z|)}\alpha^{|I_z|},$$

where

$$\gamma(\alpha) = \frac{6\alpha - 6\alpha^2 \ln(1/\alpha) - 4\alpha^3 - 1}{3\alpha^2}, \frac{1}{2} \leq \alpha \leq 1.$$

The improvement is shown in Figure 4.3. Using this lemma, we obtain 1.3225^n upper bounds, which slightly improves previous 1.3227^n.

4.4 Improving Randomized Local Search Algorithms

Independent Clauses We begin by introducing some notions. Two clauses C and C' are called independent if they have no variables in common. E.g., $C = x_1 \vee x_2 \vee x_3$ and $C' = \bar{x}_1 \vee x_5 \vee x_6$ are not independent. For a formula F, a maximal independent clause set $\mathcal{C}$ is a subset of the clauses of F such that all clauses in $\mathcal{C}$ are (mutually) independent and no clause of F can be added to $\mathcal{C}$ without destroying this property the dependency. If $\mathcal{C}$ is a maximal independent clause set for a formula F then every clause C in F contains at least one variable that occurs in (some clause of) $\mathcal{C}$. Otherwise, $\mathcal{C}$ would not be a maximal independent clause set. This gives the following consequence: if we assign constants to all the variables contained in the independent clauses, then - after the usual simplifications consisting of removing constants - we obtain a 2-CNF formula F^*, since every clause in F^* has at most two

literals. It is well known that there is a polynomial-time algorithm for checking the satisfiability of a 2-SAT formula [APT79] and finding a satisfying assignment if one exists. Hence, given a satisfiable formula F and a maximal independent clause set with $\hat{m}$ independent clauses $C_1, \ldots, C_{\hat{m}}$, we are able to find a satisfying assignment in time $poly(n) \cdot 7^{\hat{m}}$. Namely for each of the $\hat{m}$ independent clauses, we can run through all seven (of the eight) assignments that satisfy the clause. The remaining 2-SAT formula is tested in polynomial time for satisfiability.

For a better understanding of what follows, we remark that this algorithm could also be replaced by a randomized algorithm: set each of the independent clauses with probability 1/7 to one of the seven satisfying assignments. Then, solve the remaining 2-SAT problem. The success probability of this randomized algorithm is at least $(1/7)^{\hat{m}}$ which is large if $\hat{m}$ is small.

Basic Idea The basic idea behind the improvement of [HSSW02] is as follows: If we are given an instance of 3-SAT where we find a maximal independent clause set of $\hat{m}$ clauses where $\hat{m}$ is small, we get a small running time (or large success probability) by the algorithm just described. On the other hand, $\hat{m}$ could be as large as $n/3$. Here we are able to improve the first phase of Schöning's original algorithm. In the original algorithm, each variable is set to 0 with probability 1/2. The information given by the clauses is completely ignored. Although a clause $C = x_1 \vee x_2 \vee x_3$ tells us that not all those three variables should be set to zero simultaneously, the original initialization phase selects such an assignment with probability 1/8. Their improvement is to initialize the three variables in every independent clause so that assignment $(0, 0, 0)$ should be avoided. The computation shows that the success probability of this modified version of Schöning's algorithm increases with $\hat{m}$.

Underlying Algorithms The algorithm starts by computing a maximal independent clause set. It is clear that this can be done in polynomial time by a greedy algorithm which selects independent clauses until no more independent clauses can be added. Let $C_1, \ldots, C_{\hat{m}}$ be the independent clauses thus chosen. By renaming variables and exchanging the roles of x_i and $\bar{x}_i$ if necessary we may assume that $C_1 = x_1 \vee x_2 \vee x_3$, $C_2 = x_4 \vee x_5 \vee x_6$ etc. Hence, the variables $x_1, \ldots, x_{3\hat{m}}$ are those

Subroutine Ind-Clauses-Assign(p_1, p_2, p_3);
  for $C \in \{C_1, \ldots, C_m\}$
    Assume that $C = x_i \vee x_j \vee x_k$
    Set the variables x_i, x_j, x_k randomly in such a way
    that for $a = (x_i, x_j, x_k)$ the following holds:
      $\Pr\{a = (0,0,1)\} = \Pr\{a = (0,1,0)\} = \Pr\{a = (1,0,0)\} = p_1$
      $\Pr\{a = (0,1,1)\} = \Pr\{a = (1,0,1)\} = \Pr\{a = (1,1,0)\} = p_2$
      $\Pr\{a = (1,1,1)\} = p_3$
  end for;

Figure 4.4: Subroutine Ind-Clauses-Assign

contained in the independent clauses and $x_{3\hat{m}+1}, \ldots, x_n$ are the remaining variables. For assigning constants to the variables in the independent clauses, we apply a randomized procedure called Ind-Clauses-Assign which depends on three parameters p_1, p_2, and p_3. to these three, we choose one of the seven assignments for the three variables of one clause. The details of the procedure are explained in Figure 3. After Ind-Clauses-Assign is called, all variables $x_1, \ldots, x_{3\hat{m}}$ are assigned constants.

Assume in the following that a^* is an arbitrary but satisfying assignment to the input formula. For each independent clause C_i, we can count the number of the variables in C_i that are set to 1 by a^*. Since a^* is a satisfying assignment for all clauses, we have that either one, two or three of the variables in C_i are set to 1, for each i. For our analysis, let m_1 (m_2 and m_3 respectively) be the number of clauses in $\{C_1, \ldots, C_{\hat{m}}\}$ in which exactly one variable is set to 1 by a^* (two and three variables, respectively). Let us also abbreviate $\alpha_i := m_i/\hat{m}$. We have $\hat{m} = m_1 + m_2 + m_3$ hence $\alpha_1 + \alpha_2 + \alpha_3 = 1$. We are now ready to describe two randomized algorithms which play key roles in the improved 3-SAT algorithm [HSSW02].

Algorithm Red2 and Its Success Probability Algorithm Red2 is the generalization of the algorithm which checks all $7^{\hat{m}}$ assignments to the variables in the maximal independent clause set. It reduces a 3-SAT formula to a 2-SAT formula and is successful if the 2-SAT formula is satisfiable. The algorithm is described in

Algorithm Red2(q_1, q_2, q_3); $\#q_i$ are probabilities with $3q_1 + 3q_2 + q_3 = 1\#$
Ind-Clauses-Assign(q_1, q_2, q_3);
Simplify the resulting formula and
start the polynomial-time 2-SAT algorithm;

Figure 4.5: Algorithm Red2

Figure 4.

Algorithm Red2 finds a satisfying assignment when the partial assignment to the variables $x_1, \ldots, x_{3\hat{m}}$ (which satisfies the clauses $C_1, \ldots, C_{\hat{m}}$) can be extended to a complete satisfying assignment. This is the case, e.g., if the partial assignment agrees with a^* on $x_1, \ldots, x_{3\hat{m}}$. The probability of this event is exactly $q_1^{m_1} \cdot q_2^{m_2} \cdot q_3^{m_3}$, as we show now: Let C_j be one of the $\hat{m}$ clauses and let a^* have exactly i 1's for the variables in C_j. The probability that algorithm Red2 assigns values to the three variables in C_j that agree with a^* is q_i. By multiplying over all clauses, we obtain the above probability. The following theorem states the bound with the parameters we need later on:

Theorem 4.2 *[HSSW02]. Algorithm Red2 has success probability at least*

$$\left(\frac{\alpha_1}{3}\right)^{m_1} \cdot \left(\frac{\alpha_2}{3}\right)^{m_2} \cdot \alpha_3^{m_3} = \left[\left(\frac{\alpha_1}{3}\right)^{\alpha_1} \cdot \left(\frac{\alpha_2}{3}\right)^{\alpha_2} \cdot \alpha_3^{\alpha_3}\right]^{\hat{m}}.$$

Algorithm RW and Its Success Probability Algorithm RW improves the first phase of Schöning's random walk (RW) algorithm in which an initial assignment is chosen. Instead of initializing each variable x_i with probability 1/2 to 1, variables $x_1, \ldots, x_{3\hat{m}}$ in the independent clauses are assigned by Subroutine Ind-Clauses-Assign to the probability distributions p_1, p_2 and p_3. Algorithm RW is described in Figure 4 and its success probability is analyzed in the following theorem.

Theorem 4.3 *[HSSW02]. The success probability of algorithm RW is at least* $P_{RW} :=$

$$\left(\frac{3}{4}\right)^{n-3\hat{m}} \cdot \left(\frac{3p_1}{2} + \frac{9p_2}{8} + \frac{p_3}{4}\right)^{m_1} \cdot \left(\frac{9p_1}{8} + \frac{3p_2}{2} + \frac{p_3}{2}\right)^{m_2} \cdot \left(\frac{3p_1}{4} + \frac{3p_2}{2} + p_3\right)^{m_3}.$$

Algorithm RW(p_1, p_2, p_3); $\#p_i$ are probabilities with $3p_1 + 3p_2 + p_3 = 1\#$
Ind-Clauses-Assign(p_1, p_2, p_3);
Set the variables $x_{3\hat{m}+1}, \ldots, x_n$ independently of each other to 0 or 1, each with probability 1/2.
To the assignment a obtained in this way, apply RandomWalk(a).

Figure 4.6: Algorithm RW

Proof. By Lemma 1, the success probability of RandomWalk(a), where a has Hamming distance d from the satisfying assignment $a^* = (a_1^*, \ldots, a_n^*)$ is at least $(1/2)^d/p(n)$. As in the proof of Theorem 1, we obtain for the success probability:

$$Pr\{\text{success}\} \geq E[(\frac{1}{2})^{d(a,a^*)}]/p(n).$$

Here, it does not hold that all variables are fixed independently of each other. But the only dependence is between variables which are in the same clause C_i, where $1 \leq d \leq \hat{m}$. Define $X_{1,2,3}$ to be the random variable which is the Hamming distance between (a_1, a_2, a_3) and (a_1^*, a_2^*, a_3^*). Define $X_{4,5,6}$ etc. similarly and let $X_i = d(a_i, a_i^*)$ for $i \geq 3\hat{m} + 1$. We have

$$d(a, a^*) = X_{1,2,3} + X_{4,5,6} + \cdots + X_{3\hat{m}-2,3\hat{m}-1,3\hat{m}} + X_{3\hat{m}+1} + X_{3\hat{m}+2} + \cdots + X_n,$$

hence

$$\begin{aligned} Pr\{\text{success}\} &\geq \frac{1}{p(n)} \cdot E[(\frac{1}{2})^{X_{1,2,3}+X_{4,5,6}+\cdots+X_{3\hat{m}-2,3\hat{m}-1,3\hat{m}}+X_{3\hat{m}+1}+X_{3\hat{m}+2}+\cdots+X_n}] \\ &= \frac{1}{p(n)} \cdot E[(\frac{1}{2})^{X_{1,2,3}}] \cdot E[(\frac{1}{2})^{X_{4,5,6}}] \cdots E[(\frac{1}{2})^{X_{3\hat{m}-2,3\hat{m}-1,3\hat{m}}}] \cdot \prod_{i=3\hat{m}+1}^{n} E[(\frac{1}{2})^{X_i}]. \end{aligned}$$

We show how to analyze $E[(\frac{1}{2})^{X_{1,2,3}}]$, the other terms $E[(\frac{1}{2})^{X_{4,5,6}}]$ etc. are analyzed in the same way. It turns out that $E[(\frac{1}{2})^{X_{1,2,3}}]$ depends on how many ones (a_1^*, a_2^*, a_3^*) contains. We have to analyze the three possible cases:

Case $a_1^* + a_2^* + a_3^* = 3$:

This implies $(a_1^*, a_2^*, a_3^*) = (1,1,1)$. The algorithm chooses $(x_1, x_2, x_3) = (1,1,1)$ with probability p_3 and then the Hamming distance from (a_1^*, a_2^*, a_3^*) is zero. Similarly,, the algorithm sets (x_1, x_2, x_3) with probability $3p_2$ to one of (0 ,1 ,1), (1,0 ,1) and (1,1,0) which leads to Hamming distance 1, etc. Thus, by definition of the expected value, we obtain

$$\begin{aligned} E[(\frac{1}{2})^{X_{1,2,3}}] &= (\frac{1}{2})^0 \cdot p_3 + (\frac{1}{2})^1 \cdot 3p_2 + (\frac{1}{2})^2 \cdot 3p_1 \\ &= \frac{3}{4}p_1 + \frac{3}{2}p_2 + p_3. \end{aligned}$$

In the similar manner, we can analyze the other two cases:

Case $a_1^* + a_2^* + a_3^* = 1$:

$$\begin{aligned} E[(\frac{1}{2})^{X_{1,2,3}}] &= (\frac{1}{2})^0 \cdot p_1 + (\frac{1}{2})^1 \cdot 2p_2 + (\frac{1}{2})^2 \cdot (2p_1 + p_3) + (\frac{1}{2})^3 \cdot p_2 \\ &= \frac{3}{2}p_1 + \frac{9}{8}p_2 + \frac{p_3}{4}. \end{aligned}$$

Case $a_1^* + a_2^* + a_3^* = 2$:

$$\begin{aligned} E[(\frac{1}{2})^{X_{1,2,3}}] &= (\frac{1}{2})^0 \cdot p_2 + (\frac{1}{2})^1 \cdot (2p_1 + p_3) + (\frac{1}{2})^2 \cdot 2p_2 + (\frac{1}{2})^3 \cdot p_1 \\ &= \frac{9}{8}p_1 + \frac{3}{2}p_2 + \frac{p_3}{2}. \end{aligned}$$

The values m_1, m_2, m_3 count for how many clauses which of the three cases holds. Hence, we obtain the bound on Pr {ssuccess} stated in the theorem. □

Combining the Algorithms The simplest way to improve upon the $O(poly(n) \cdot (4/3)^n)$ bound of Schöning's 3-SAT algorithm is as follows: We first call Red2(1/7,1/7,1/7). And then call RW(4/21,2/21,3/21). The success probability of this combined algorithm is at least $\Omega(1.330258^{-n})$, which we will prove now: First, observe that the success probability of algorithm Red2(1/7,1/7,1/7) is at least $(1/7)^{\hat{m}} \geq (1/7)^{\hat{m}}/p(n)$. On the other hand, by Theorem 3, with the chosen parameters, the success proba-

bility of algorithm RW is at least

$$\left(\frac{3}{4}\right)^{n-3\hat{m}} \cdot \left(\frac{3}{7}\right)^{\hat{m}} \cdot \frac{1}{p(n)} = \left(\frac{3}{4}\right)^{n} \cdot \left(\frac{64}{63}\right)^{\hat{m}} \cdot \frac{1}{p(n)}.$$

The combined algorithm is successful if one of the two randomized algorithms is successful and thus the success probability of the combined algorithm is at least as large as the maximum of the two success probabilities. We observe that the bound on the success probability of algorithm Red2 decreases with $\hat{m}$ while the bound on the success probability of algorithm RW increases with $\hat{m}$ hence it suffices to compute the $\hat{m}$ where both are equal. This is the case for

$$\frac{n}{\hat{m}} = \frac{\log 9/64}{\log 3/4} \approx 6.8188417, \text{i.e.}, \hat{m} \approx 0.1466525 \cdot n.$$

This leads to a success probability of at least $\Omega(1330258^{-n})$ and a randomized algorithm for 3-SAT with expected running time $O(1.330258^n)$.

4.5 Our Improvements

Good pair Our Improvement is obtained by extending the technique of [HSSW02]. Without loss of generality, we can assume that, given 3-SAT formula F, if a literal x_i appears in a clause C, then there exists a clause C' such that $\bar{x}_i$ appears in C'. (Otherwise we eliminate such a literal by assigning the value that makes the literal true.) We call C and C' are a *good pair* if they have the same variable and its sign is different in each clause between them. By renaming variables and exchanging the roles of x_i and $\bar{x}_i$, *good pair* is classified into 6 types:

- type 1: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_4 \vee x_5)$
- type 2: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$
- type 3: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_4)$
- type 4: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$
- type 5: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$

Subroutine Construct Ind-Clauses-Set{**input** F};
Compute *maximal independent good pair set* $\mathcal{C}'$
Compute *maximal independent clause set* $\mathcal{C}$ by adding independent clauses to $\mathcal{C}'$.
return $\mathcal{C}$.

Figure 4.7: Subroutine Construct Ind-Clauses-Set

- type 6: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$

For every type, we can see that exactly 1/4 of all possible assignments do not satisfy both of a good pair at the same time. E.g., for $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_4)$, $(x_1, x_2, x_3, x_4) = (0,0,0,0), (0,0,0,1), (1,1,0,0)$ and $(1,1,1,0)$ are unsatisfying assignments among 16 possible assignments. To exploit good pair for the improvement of the first phase of Schöning's algorithm, we modify some notions. Two good pairs $C \wedge C'$ and $C'' \wedge C'''$ are called independent if they have no variable in common. A good pair $C \wedge C'$ and a clause C'' are also called independent if they have no variable in common. Recall that, in the original algorithm, *maximal independent clause set* can be constructed arbitrary. For our improvement, we construct *maximal independent clause set* as described as Figure 6.

Probabilistic analysis We begin by showing some lemmas.

Lemma 4.6 *For a type 1 good pair* $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_4 \vee x_5)$, *we can obtain*

$$E[(\frac{1}{2})^{X_{1,2,3,4,5}}] = (\frac{45}{182})$$

by Ind-Clauses-Assign-1 $(\frac{8}{91}, \frac{2}{91}, \frac{6}{91}, \frac{2}{91}, \frac{3}{91}, \frac{3}{182})$.

Proof. Figure 4.8 shows Ind-Clauses-Assign-1$(p_1, p_2, p_3, p_4, p_5, p_6)$ where p_i's have to satisfy

$$4p_1 + 8p_2 + 4p_3 + 2p_4 + 4p_5 + 2p_6 = 1$$

Similarly to the analysis of Ind-Clauses-Assign, we can obtain the following ten cases.

Subroutine Ind-Clauses-Assign-1 $(p_1, p_2, p_3, p_4, p_5, p_6)$;
Assume that $C = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_4 \vee x_5)$
Set the variables x_1, x_2, x_3, x_4, x_5 randomly in such a way
that for $a = (x_1, x_2, x_3, x_4, x_5)$ the following holds:
Pr{a=(0,0,1,0,0)}=Pr{a=(0,1,0,0,0)}= Pr{a=(1,0,0,0,1)}=Pr{a=(1,0,0,1,0)}=p_1
Pr{a=(0,0,1,0,1)}=Pr{a=(0,0,1,1,0)}= Pr{a=(0,1,0,0,1)}=Pr{a=(0,1,0,1,0)}=
Pr{a=(1,0,1,0,1)}=Pr{a=(1,0,1,1,0)}= Pr{a=(1,1,0,0,1)}=Pr{a=(1,1,0,1,0)}=p_2
Pr{a=(0,0,1,1,1)}=Pr{a=(0,1,0,1,1)}= Pr{a=(1,1,1,0,1)}=Pr{a=(1,1,1,1,0)}=p_3
Pr{a=(0,1,1,0,0)}=Pr{a=(1,0,0,1,1)}=p_4
Pr{a=(0,1,1,0,1)}=Pr{a=(0,1,1,1,0)}= Pr{a=(1,0,1,1,1)}=Pr{a=(1,1,0,1,1)}=p_5
Pr{a=(0,1,1,1,1)}=Pr{a=(1,1,1,1,1)}=p_6

Figure 4.8: Subroutine Ind-Clauses-Assign-1

Case $d((a_1^*, a_2^*, a_3^*), (0,0,0)) = 1$ and $d((a_1^*, a_4^*, a_5^*), (1,0,0)) = 1$:

$$E[(\frac{1}{2})^{X_{1,2,3,4,5}}] = \frac{3}{2}p_1 + \frac{15}{8}p_2 + \frac{9}{16}p_3 + \frac{9}{16}p_4 + \frac{21}{32}p_5 + \frac{3}{16}p_6$$

Case $d((a_1^*, a_2^*, a_3^*), (0,0,0)) = 1$ and $d((a_1^*, a_4^*, a_5^*), (1,0,0)) = 2$ and $a_1^* = 0$:

and

Case $d((a_1^*, a_2^*, a_3^*), (0,0,0)) = 2$ and $d((a_1^*, a_4^*, a_5^*), (1,0,0)) = 1$ and $a_1^* = 1$:

$$E[(\frac{1}{2})^{X_{1,2,3,4,5}}] = \frac{15}{16}p_1 + \frac{75}{32}p_2 + \frac{15}{16}p_3 + \frac{3}{8}p_4 + \frac{15}{16}p_5 + \frac{3}{4}p_6$$

Case $d((a_1^*, a_2^*, a_3^*), (0,0,0)) = 1$ and $d((a_1^*, a_4^*, a_5^*), (1,0,0)) = 3$:

and

Case $d((a_1^*, a_2^*, a_3^*), (0,0,0)) = 3$ and $d((a_1^*, a_4^*, a_5^*), (1,0,0)) = 1$:

$$E[(\frac{1}{2})^{X_{1,2,3,4,5}}] = \frac{9}{16}p_1 + \frac{15}{8}p_2 + \frac{3}{2}p_3 + \frac{3}{8}p_4 + \frac{9}{8}p_5 + \frac{3}{4}p_6$$

Case $d((a_1^*, a_2^*, a_3^*), (0,0,0)) = 1$ and $d((a_1^*, a_4^*, a_5^*), (1,0,0)) = 2$ and $a_1^* = 1$:

and

Case $d((a_1^*, a_2^*, a_3^*), (0,0,0)) = 2$ and $d((a_1^*, a_4^*, a_5^*), (1,0,0)) = 1$ and $a_1^* = 0$:

$$E[(\frac{1}{2})^{X_{1,2,3,4,5}}] = \frac{9}{8}p_1 + \frac{3}{2}p_2 + \frac{3}{4}p_3 + \frac{33}{32}p_4 + \frac{9}{8}p_5 + \frac{3}{8}p_6$$

Case $d((a_1^*, a_2^*, a_3^*), (0,0,0)) = 2$ and $d((a_1^*, a_4^*, a_5^*), (1,0,0)) = 2$:

$$E[(\frac{1}{2})^{X_{1,2,3,4,5}}] = \frac{21}{32}p_1 + \frac{15}{8}p_2 + \frac{9}{8}p_3 + \frac{9}{16}p_4 + \frac{3}{2}p_5 + \frac{3}{4}p_6$$

Case $d((a_1^*, a_2^*, a_3^*), (0,0,0)) = 2$ and $d((a_1^*, a_4^*, a_5^*), (1,0,0)) = 3$:

and

Case $d((a_1^*, a_2^*, a_3^*), (0,0,0)) = 3$ and $d((a_1^*, a_4^*, a_5^*), (1,0,0)) = 2$:

$$E[(\frac{1}{2})^{X_{1,2,3,4,5}}] = \frac{3}{8}p_1 + \frac{3}{2}p_2 + \frac{3}{2}p_3 + \frac{3}{8}p_4 + \frac{3}{2}p_5 + \frac{3}{2}p_6$$

Since $(p_1, p_2, p_3, p_4, p_5, p_6) = (\frac{8}{91}, \frac{2}{91}, \frac{6}{91}, \frac{2}{91}, \frac{3}{91}, \frac{3}{182})$, we obtain $E[(\frac{1}{2})^{X_{1,2,3,4,5}}] = (\frac{45}{182})$. □

We can construct and analyse Ind-Clauses-Assign-i for each type i and obtain similar bounds as follows.

- type 2: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$

$$E[(\frac{1}{2})^{X_{1,2,3,4}}] = (\frac{27}{82})$$

- type 3: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_4)$

$$E[(\frac{1}{2})^{X_{1,2,3,4}}] = (\frac{15}{46})$$

- type 4: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$

$$E[(\frac{1}{2})^{X_{1,2,3}}] = (\frac{9}{20})$$

- type 5: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$

$$E[(\frac{1}{2})^{X_{1,2,3}}] = (\frac{45}{105})$$

- type 6: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$

$$E[(\frac{1}{2})^{X_{1,2,3}}] = (\frac{7}{16})$$

Now we compute the success probability of the modified RW from the analysis of the independent clause set $\mathcal{C}$ computed by Construct Ind-Clauses-Set (with the parameters described as above).

Theorem 4.4 *The success probability of the modified algorithm RW is at least* $P_{RW} :=$

$$\left(\frac{3}{4}\right)^n \cdot \left(\frac{64}{63}\right)^{m_0} \cdot \left(\frac{2560}{2457}\right)^{m_1} \cdot \left(\frac{128}{123}\right)^{m_2} \cdot \left(\frac{640}{621}\right)^{m_3} \cdot \left(\frac{16}{15}\right)^{m_4} \cdot \left(\frac{40}{39}\right)^{m_5} \cdot \left(\frac{28}{27}\right)^{m_6} \cdot \frac{1}{p(n)},$$

where m_i is the number of good pairs of type i in $\mathcal{C}$ and m_0 is the number of single clauses in $\mathcal{C}$.

The success probability of modified Red2 is at least $(1/7)^{m_0} \cdot (1/24)^{m_1} \cdot (1/12)^{m_2+m_3} \cdot (1/6)^{m_4+m_5+m_6}$. However, in the worst case, combining the algorithms does not improve the upper bound. It is the case when $\mathcal{C}$ contains small number of good pairs, e.g., $m_1, m_2, \ldots, m_6 = o(n)$, we have no advantage of good pairs.

To deal with such case, following observation is useful. Let $\mathcal{C}$ be a fixed maximal independent clause set, let $\mathcal{C}''$ be a set of good pairs in $\mathcal{C}$ and let $\mathcal{C}'$ be a set of single clauses in $\mathcal{C}$. Our problem occurs when $\mathcal{C}''$ is small and $\mathcal{C}'$ is large to some extent. The reason why a clause $C \in \mathcal{C}'$ cannot constitute a good pair is that any candidate clause for a good pair with C is not independent to the pairs in $\mathcal{C}''$ So, if $\mathcal{C}''$ is removed, $C \in \mathcal{C}'$ may become good pair.

Subroutine Construct Ind-Clauses-Set{**input** F};
Compute *maximal independent set of type 1'-3' good pair*C'
Compute *maximal independent clause set* C by adding
independent clauses or good pair to C'.
return C.

Figure 4.9: Subroutine Construct Ind-Clauses-Set

For a fixed formula F and its maximal independent set C, let V be a set of clauses that are not independent to C''. If we assign constants to all the variables contained in C'' and apply usual simplification technique to F, we obtain a formula F^* that satisfies the properties:

(1) the clauses contained in V are eliminated or have at most two literals.

(2) if we make a good pair by using clause in C', another clause of good pair is contained in V.

In F^*, a good pair consits of a clause in V and in C' is classified into three types.

- type 1': $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_4)$
- type 2': $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2)$
- type 3': $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2)$

Similarly to the analysis of Ind-Clauses-Assign-i, we have the following bounds for the above three types.

- type 1': $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_4)$

$$E[(\frac{1}{2})^{X_{1,2,3,4}}] = (\frac{9}{26})$$

- type 2': $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2)$

$$E[(\frac{1}{2})^{X_{1,2,3}}] = (\frac{9}{19})$$

- type 3': $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2)$

$$E[(\frac{1}{2})^{X_{1,2,3}}] = (\frac{5}{11})$$

Now we compute the success probability of the modified RW on F^* from the analysis of the independent clause set $\mathcal{C}$ computed by Construct Ind-Clauses-Set (with the parameters described as above).

Theorem 4.5 *The success probability of algorithm RW on F^* is at least $P_{RW} :=$*

$$\left(\frac{4}{3}\right)^{n-5m_1-4(m_2+m_3)-3(m_4+m_5+m_6)} \left(\frac{63}{64}\right)^{m_{0'}} \left(\frac{119}{128}\right)^{m_{1'}} \left(\frac{57}{64}\right)^{m_{2'}} \left(\frac{297}{320}\right)^{m_{3'}}$$

where $m_{i'}$ is the number of good pairs of type i' in $\mathcal{C}$ and $m_{0'}$ is the number of single clauses in $\mathcal{C}$.

The success probability of modified Red2 is at least $(1/7)^{m_{0'}}\ (1/10)^{m_{1'}}\ (1/5)^{m_{2'}+m_{3'}}$ Here again, we have no advantage of good pairs when $m_{i'}$ is small. In such case, we assign constant to all variables contained in the good pairs of type 1'-3' and apply usual simplification technique to F^*. As a result, we obtain a formula F^{**} such that clauses contained in V are eliminated or have at most one literals. Thus, we can eliminate all variables in C' in F^{**} because variables contained in C' appear as unit clauses or pure literals. (We call a literal l_i is a pure literal in F iff a literal $\bar{l}_i$ does not appear in F.)

Upper bounds Now, we consider the upper bound of our improved algorithm. From theorem 5 and its counter part algorithm, we obtain two upper bounds

$$\left(\frac{4}{3}\right)^{n} \left(\frac{63}{64}\right)^{m_0} \left(\frac{2457}{2560}\right)^{m_1} \left(\frac{123}{128}\right)^{m_2} \left(\frac{621}{640}\right)^{m_3} \left(\frac{15}{16}\right)^{m_4} \left(\frac{39}{40}\right)^{m_5} \left(\frac{27}{28}\right)^{m_6}$$

$$7^{m_0}\ 24^{m_1}\ 12^{m_2+m_3}\ 6^{m_4+m_5+m_6}$$

where $0 \leq 5m_1 + 4(m_2 + m_3) + 3(m_4 + m_5 + m_6) \leq n$.

From theorem 6 and its counter part algorithm, we obtain two upper bounds

$$24^{m_1}\ 12^{m_2+m_3}\ 6^{m_4+m_5+m_6}\ \left(\frac{4}{3}\right)^{n-5m_1-4(m_2+m_3)-3(m_4+m_5+m_6)}$$

$$\times \left(\frac{63}{64}\right)^{m_{0'}} \left(\frac{119}{128}\right)^{m_{1'}} \left(\frac{57}{64}\right)^{m_{2'}} \left(\frac{297}{320}\right)^{m_{3'}}$$

$$24^{m_1}\ 12^{m_2+m_3}\ 6^{m_4+m_5+m_6}\ 7^{m_{0'}}\ 10^{m_{1'}}\ 5^{m_{2'}+m_{3'}}$$

where $0 \leq 4m_{1'} + 3(m_{2'} + m_{3'}) \leq n - 5m_1 - 4(m_2 + m_3) - 3(m_4 + m_5 + m_6)$.
From the observation in the last part of the analysis, we obtain the following upper bound

$$24^{m_1}\ 10^{m_{1'}}\ 5^{m_{2'}+m_{3'}}\ T(n - 5m_1 - 4(m_2 + m_3) - 3(m_4 + m_5 + m_6) - m_{1'} - 3m_{0'})$$

where $T(n)$ is the best upper bound for 3-SAT with n variables. Define
$T'(m_0, m_1, m_2, m_3, m_4, m_5, m_6) =: \max_{m_{0'},m_{1'},m_{2'},m_{3'}} \min\{(3), (4), (5)\}$,
then we have $T(n) = \max_{m_0,m_1,m_2,m_3,m_4,m_5,m_6} \min\{(1), (2), T'\}$. By numerical analysis we obtain $T(n) = O(1.329917^n)$.

4.6 Concluding Remarks

In this chapter we present new worst-case upper bounds for 3SAT. Our algorithm make use of trade-off between two algorithms and achieves current best running time. Recently we learn that Rolf gave more refined analysis of **PPSZ** that yields even better trade-offs as follows:

Lemma 4.7 ([Rol05b]) *For any satisfiable 3CNF formula F and any partition $\mathbf{C}(z, I_z)$ described above, if $y \in \mathbf{C}(z, I_z)$ is chosen uniformly at random, then the value $\tau(F_s, z|\mathbf{C}(z, I_z))$ is bounded as follows:*

$$\tau(F_s, z|\mathbf{C}(z, I_z)) \geq 2^{-(1-\gamma')(n-|I_z|)-\beta'|I_z|},$$

where $\gamma' = 0.61229, \beta' = 0.90625$.

This further improves the upper bound for 3SAT into 1.32216^n.

We also present an impovend on Schöning's randomized local search algorithm extending the techniques develped by [SSW01, HSSW02]. Our result gives $O(1.329917^n)$ upper bounds for 3SAT. This bound is the first result that achives better bounds than $O(1.33^n)$ after [SSW01, HSSW02]. Independent of our work, [BS03] gives $O(1.3280^n)$ upper bounds using similar method and additional ideas. Following the result of [BS03], [Rol03] gives $O(1.3270^n)$ upper bounds by more refined analysis. We do not know this approach, that is, the combination of local search and independent clause, yields new worst-case upper bounds for 3SAT that beats the current best bound $O(1.3225^n)$. However, if we can analyse the running time of these kind of algorithms when combined with **PPSZ**, we may obtain further improvements.

Chapter 5

Increasing the Success Probability of PPSZ-type Algorithms

5.1 Introduction

Worst-case upper bounds for 3SAT (k-SAT in general) have been one of the most well-studied topics in theoretical computer science. Figure 4.1 in previous chapter shows its rich history of improvements. The current best bound is $O(1.3225^n)$ by as shown in the previous section, which is based on a tricky combination of the two existing algorithms, Schöning's Local Search ([Sch02], denoted by **SCH**) and Randomized Davis Putnam by Paturi, Pudlák, Saks and Zane ([PPSZ98], denoted by **PPSZ**). This chapter stays on the same line, namely a combination of the two brilliant algorithms, but uses a completely different approach.

The basic idea of **PPSZ** is as follows: Suppose that a given formula $G(x_1, \ldots, x_n)$ has exactly one satisfying assignment $z = z_1 z_2 \cdots z_n \in \{0,1\}^n$ (can be extended to the general case). Also let π be a permutation of $\{1, 2, \ldots, n\}$. Then if we assign each value of z into $\{x_1, \ldots, x_n\}$ in the order of π, (i.e., $z_{\pi(1)} \to x_{\pi(1)}$ in Step 1, $z_{\pi(2)} \to x_{\pi(2)}$ in Step 2, and so on), a certain number of variables $\subseteq \{x_1, \ldots, x_n\}$ are *forced*. Here, we say that a variable x is forced in the above course of sequential assignment with respect to π and z, if x becomes a unit clause in Step k for some $k \geq 1$. [PPSZ98] shows that the number N of such forced variables can be made quite large by adding clauses by resolution. For a randomly chosen π, they proved

that the expected value of N is at least $(2\ln 2 - 1)n \approx 0.613n$. This implies that if we know the correct values of the unforced variables, ("correct" means the same value as z), then we can retrieve the whole values of z by the above process. Roughly speaking it is enough to know the correct values of only $0.387n$ variables to obtain the satisfying assignment.

Our Contribution. Of course there is no obvious ways of getting the correct values of $0.387n$ variables. Our idea is to use **SCH** for this purpose: Recall that local search, in general, starts with a random assignment y and gradually approaches to the final goal, i.e., the satisfying assignment z. The Hamming distance $d(y, z)$ between y and z is expected to be $n/2$, which becomes 0 when we get to the goal. In the case of **SCH**, we can always decrease the Hamming distance by one with probability at least $1/3$ in each step.

Suppose that the current assignment a^* is close to z, say $d(a^*, z) = 0.05n$. Also suppose that we use this a^* (and a randomly selected π) for the retrieval process of **PPSZ**. If π is fixed then the set, U, of unforced variables is also fixed, and recall that if all the values of the variables in U are correct then we can successfully get the satisfying assignment z. Since $|U| = 0.387n$ and the $0.95n$ variables ($=S$) have correct values in y^*, it appears that the above success probability (=probability that U are included in S) is not too small. In fact we can obtain an upper bound of $O(1.2991^n)$ for unique 3SAT and $O(1.3231^n)$ for general 3SAT by this approach. (Of course we do not know if the current assignment is sufficiently close to z. So we try the above **PPSZ** retrieval process in each step of **SCH**; if successful we are done, otherwise we just flip one variable by the **SCH** rule.)

One drawback of this approach is that the intermediate assignment a^* of **SCH** is not uniformly at random. Hence we need some assumption to prove the upper bounds previously mentioned. However, we can also give some observation which claims that the above non-uniformity is probably not too serious. Apparently it is obviously important future work to give a formal solution to this question.

Related Work. The idea of using **PPSZ** in each step of local search is not new; UnitWalk by Hirsch and Kojevnikov [HK05] is the following procedure: In each step, we apply the same retrieval process using a^* and (a randomly selected) π, and if

there are unit clauses then we fix their values so as to satisfy the unit clauses and *modify* (simplify) the formula by this assignment. If there is no unit clause, then we select one variable *at random* and flip its value. Thus there are two important differences: (i) They change the formula in the course of the local search (we do not). (ii) They select the variable to be flipped at random (we use the **SCH**). Apparently these two differences make their analysis hard and they give only experimental data for the performance, which are quite nice especially for "hard" benchmarks.

Since **SCH** is a very simple algorithm (and its analysis is also simple and beautiful), many researches tried to improve it by adding heuristics. However, improvement appears to be harder than it looks, and there are only a few successful attempts so far [HSSW02, BS03, Rol03]. Furthermore, the basic ideas of them are quite simpler, i.e., using a biased initial assignment rather than the one selected uniformly at random. We proposed a completely different approach in previous chapter where **SCH** and **PPSZ** are run in parallel and an answer of the faster one is taken. However, a simple implementation of this idea fails and they escape this problem by using the same initial assignment for both **SCH** and **PPSZ**. [GNR04] also depends on a diffident idea. They define an *independent set* of formula, and prove that if the size of the independent set is sufficiently large, then both **SCH** and **PPSZ** can be improved.

5.2 Preliminaries

In this section, we review the basic property of Schöning's algorithm.

Lemma 5.1 ([Sch02]) *Let F be a 3CNF formula and a^* be a satisfying assignment for F. For each assignment a, the probability that a satisfying assignment is found by* **RandomWalk**(F, a) *is at least* $(1/2)^{d(a,a^*)}$.

The following lemma states the bound for 3SAT obtained in [Sch02] using the above lemma.

Lemma 5.2 ([Sch02]) *For any satisfiable formula F on n variables, the success probability of one repeat-iteration of* **SCH** *is at least* $(3/4)^n$.

We can generalize lemmas 5.1 and 5.2 by analyzing **RandomWalk** in a way similar to [Sch02]

Lemma 5.3 *Let F be a 3CNF formula and a^* be a satisfying assignment for F. For each assignment a and $0 \leq d \leq d(a, a^*)$, the probability that* **RandomWalk**(F, a) *reaches some $b \in \mathbf{B}(a^*, d)$, $(= \sum_b \mathbf{Pr}[$it reaches $b \in \mathbf{B}(a^*, d)])$ is at least $(1/2)^{d(a,a^*)-d}$. Furthermore, if a is chosen uniformly at random and $0 \leq d \leq n/3$, then the probability that* **RandomWalk**(F, a) *reaches some $b \in \mathbf{B}(a^*, d)$ is at least $(3/4)^n(1/2)^{-d}$.*

Now, given formula F with a set of satisfying assignments S and the subcube partition $\{\mathbf{C}(z, I_z) \mid z \in S\}$, $\sigma(F, z|\mathbf{C}(z, I_z))$ is defined as the probability (averaged over y) that a single execution of **RandomWalk** finds the assignment z under the condition that the initial assignment $y \in \mathbf{C}(z, I_z)$.

Lemma 5.4 *For any satisfiable 3CNF formula F and any partition $\mathbf{C}(z, I_z)$ described above, if $y \in \mathbf{C}(z, I_z)$ is chosen uniformly at random, then the value $\sigma(F, z|\mathbf{C}(z, I_z))$ is bounded as follows:*

$$\sigma(F, z|\mathbf{C}(z, I_z)) \geq \left(\frac{3}{4}\right)^{n-|I_z|}.$$

5.3 New Algorithm

In this section, we present our new algorithm and its analysis. As shown below, **IT** is similar as the previous section, i.e., it runs the two algorithms in parallel. Our key routine is **Walk&Modify**, which tries to retrieve all the bits of the satisfying assignment from the current assignment.

---

c **Walk&Modify**(CNF formula G_1, G_2, permutation π, assignment y);
$y' = y$;
for $3n$ times
  if y' satisfies G_1
    then return y'; exit;

---

else if $y'' = \mathbf{Modify}(G_2, \pi, y')$ satisfies G_1
then output(y''); exit;
$C \leftarrow$ a clause of G_1 that is not satisfied by y';
Modify y' as follows:
select one literal of C uniformly at random and
flip the assignment to this literal;
end
return y'

IT(CNF-formula F, integer s, integer T)
$F_s = \mathbf{Resolve}(F, s)$.
repeat T times
$y =$ uniformly random vector $\in \{0,1\}^n$
$\pi =$ uniformly random permutation of $1, 2, \ldots, n$
$z = \mathbf{Walk\&Modify}(F, F_s, \pi, y)$;
if z satisfies F
then output(z); exit;
else if $z' = \mathbf{Modify}(F_s, \pi, y)$ satisfies F
then output(z'); exit;
end
output('Unsatisfiable');

5.3.1 Uniformity assumption

Lemma 5.3 for unique 3SAT says that **RandomWalk** reaches some $b \in \mathbf{B}(a^*, d)$ with probability at least $(3/4)^n(1/2)^{-d}$. Since $|\mathbf{B}(a^*, d)| \approx \binom{n}{d}$, if we assume that **RandomWalk** reaches each of $\mathbf{B}(a^*, d)$ with the same probability, then that probability is

$$P_0 = \left(\frac{3}{4}\right)^n \left(\frac{1}{2}\right)^{-d} / \binom{n}{d}.$$

If this assumption is true (actually it is enough that the assumption is true for a some specific value of d which is given later), then it implies that each bit is independently correct with probability at least $P_0 \times \binom{n-1}{d}$, which makes the analysis much easier.

Unfortunately, it is unlikely that the assumption is true. So, we introduce a bit weaker assumption, namely;

Assumption 1. RandomWalk reaches each a, such that $d(a, a^*) = d$ with probability at least $P_0/p(n)$ for some $p(n) = e^{o(n)}$.

Since we are ignoring a sub-exponential factor, this assumption can obviously replace the stronger one. Now let us consider how reasonable **Assumption 1** is. One bad scenario is that some variable has an exponentially smaller number of chances of being flipped. If this would be the case, then such a variable almost always has the same value as its initial value. Since the initial value is correct with probability 1/2, the value of this variable in the current assignment a^* is also correct with probability 1/2 although the expected number of correct values in a is, say, $0.95n$. Fortunately, such an extreme imbalance in the number of flips does not happen. To see this, recall that a formula G for unique SAT has at least one critical clause for each variable. Let $(x_i + x_j + x_k)$ be such a critical clause for x_k. Then, if the initial assignment to x_k is wrong, then this clause becomes false with probability 1/4. If there are many such clauses, the probability increases. However, since there are only a polynomial number of clauses and the number of steps in each run is only $3n$, the difference among such probabilities is within polynomial. It appears that this observation implies **Assumption 1**, but we cannot prove it at this moment.

5.3.2 Upper bounds of the complexity

If we use **Assumption 1**, then the following analysis is not so hard:

Theorem 5.1 *Under* **Assumption1**, *for any 3CNF formula F and its unique satisfying assignment z, the success probability of one repeat-iteration of* **IT** *is at least* $\rho(F_s, z) \geq 1.2991^{-n}$.

Proof. Fix 3CNF formula F and its unique satisfying assignment z. Then,

$$\rho(F_s, z) = \mathbf{E}_\pi[\mathbf{Pr}_y[y \in \mathbf{C}(z, I(F_s, \pi, z))]]$$

By **Assumption 1**, **RandomWalk** reaches each a in $\mathbf{S} = \{a \mid d(a,z) = d\}$ with probability at least $P_0/p(n)$ and the size of the "good" subset of $\mathbf{S}$, the number of assignments that are contained in $\mathbf{C}(z, I(F_s, \pi, z))$ with Hamming distance d from z is $\binom{n - |I(F_s,\pi,z)|}{d}$. Hence,

$$\rho(F_s, z) \geq \mathbf{E}_\pi \left[\max_{0 \leq d \leq n/3} \left\{ \binom{n - |I(F_s,\pi,z)|}{d} \times P_0/p(n) \right\} \right].$$

Using lemma ?? and the convexity of the term in expectation, we can assume $|I(F_s, \pi, z)| = (1-\gamma)n$ for any π. Optimizing d, we obtain $\rho(F_s, z) \geq 1.2991^{-n}$ when $d = 0.227$. □

Theorem 5.2 *Under* **Assumption1**, *for any satisfiable 3CNF formula F, the success probability of one repeat-iteration of* **IT** *is at least*

$$\rho(F_s, z) = \sum_{z \in S} \mathbf{Pr}[z \in \mathbf{C}(z, I_z)] \rho(F_s, z | \mathbf{C}(z, I_z)) \geq 1.308^{-n}.$$

Proof. Let S be a set of satisfying assignments of F and fix the subcube partition $\{\mathbf{C}(z, I_z) \mid z \in S\}$. Note that the number of assignments that are contained in $\mathbf{C}(z, I(F_s, \pi, z)) \cap \mathbf{C}(z, I_z)$ with Hamming distance d from z is $\binom{n - |I_z| - |I(F_s,\pi,z)|}{d}$. Using lemma 2.6 under the assumption, we can bound

$$\begin{aligned} \rho(F_s, z | \mathbf{C}(z, I_z)) &\geq \mathbf{E}_\pi[\mathbf{Pr}_{y \in \mathbf{C}(z, I_z)}[y \in \mathbf{C}(z, I(F_s, \pi, z))]] \\ &\geq \mathbf{E}_\pi \left[\max_{0 \leq d \leq n/3} \left\{ \binom{n - |I_z| - |I(F_s,\pi,z)|}{d} \times \left(\frac{3}{4}\right)^{n-|I_z|} \left(\frac{1}{2}\right)^{-d} / \binom{n}{d} \right\} \right] \end{aligned}$$

Since this bound depends on the size of the cube $|I_z|$, we use lemma 2.6 as previous chapter when $|I_z|$ is small. The convexity of the term in expectation, we can assume $|I(F_s, \pi, z)| = (1-\gamma)(n - |I_z|$ for any π. Optimizing d, we obtain $\rho(F_s, z) \geq 1.308^{-n}$, when $|I_z| = 0.0423$. □

5.4 On the Possibility of Derandomization

In this section, we consider the possibility of derandomizing our algorithm that may beat the current best deterministic algorithm for 3SAT. The current best deterministic algorithm [BK04] is a derandomized version of Schöning's Random Walk algorithm and that achieves 1.473^n upper bounds for 3SAT. Although we do not know whether PPSZ algorithm can be derandomized, [Rol05b] gave a derandomized PPSZ algorithm for Unique SAT. The following is the Rolf's deterministic algorithm that do not use randomly generated initial inputs.

Modify'(CNF formula G, permutation π, assignment y, integer t)

$G_0 = G$, $j = 1$.
for $i = 1$ **to** n **do**
  if G_{i-1} contains unit clause $x_{\pi(i)}$
    then $z_{\pi(i)} = 1$
  else if G_{i-1} contains unit clause $\bar{x}_{\pi(i)}$
    then $z_{\pi(i)} = 0$
  else if $Gj \leq t$
    then $z_{\pi(i)} = y_{\pi(i)}$
  else return null.
  $G_i = G_{i-1}$ with $x_{\pi(i)} = z_{\pi(i)}$
end
return z

Rolf(CNF-formula F, set of permutations P, integer s, integer t)

$F_s =$ **Resolve**(F, s).
foreach $\pi \in P$ and $y \in \{0,1\}^t$
  $z =$ **Modify'**(F_s, π, y, t);
  if z satisfies F
    then output(z); exit;
end
output('Unsatisfiable');

Theorem 5.3 ([Rol05b]) *For any uniquely satisfiable 3CNF formula F and appropriately chosen s, t and P,* **Rolf** *finds a satisfying assignment of F in time $2^{(1-\gamma)n}$ where $\gamma = 2 - 2\ln 2$.*

We can extend this theorem in terms of subcube partition.

Lemma 5.5 *For any satisfiable 3CNF formula F and any partition $\mathbf{C}(z, I_z)$ and appropriately chosen s, t and P,* **Rolf** *finds $z \in \mathbf{C}(z, I_z)$ in time*

$$2^{(1-\gamma(\alpha))(n-|I_z|)}(2/\alpha)^{|I_z|},$$

where

$$\gamma(\alpha) = \frac{6\alpha - 6\alpha^2 \log_e(1/\alpha) - 4\alpha^3 - 1}{3\alpha^2}$$

and any constant α s.t. $\frac{1}{2} \leq \alpha \leq 1$.

This estimate of the running time is almost as same as the probabilistic case, but worse by $2^{|I_z|}$ factor.

Now we have a fast algorithm for formulas which have satisfying assignments with small number of defining variables (small $|I_z|$). Thus if we could show a fast algorithm for the case of large number of defining variables, we would get desired upper bounds. However, it is very hard to show such an algorithm. Recall when we analyze probabilistic algorithms using subcube partition, we only need to estimate the success probability averaged over initial assignments drawn from one specific subcube. That means we can fix one specific subcube and assume that defining variables of the cube are assigned correct value. In deterministic case, the situation becomes different. If we want to analyze the algorithm on one fixed subcube, we must explicitly choose the subcube by assigning correct values to defining variables. Intuitively this assigning process costs $2^{|I_z|}$ factor and we cannot expect that derandomized version of Schöning's works well for the cube with large defining variables.

Subcube partition may be not useful for analyzing deterministic algorithm, we can use simpler property of formulas. Now we take the number of satisfying assignment as a parameter, that is the original motivation of our improvement, and observe the running time of two algorithms in terms of this parameter. Let F be a 3CNF formula with 2^{dn} satisfying assignments. It is easy to see that for any subsube partition $\{\mathbf{C}(z, I_z)\}$ from 2^{dn} satisfying assignments, there exists one subcube with $|I_z| \leq dn$. Thus the running time of **Rolf** is estimated as follows:

Lemma 5.6 *For any 3CNF formula F with at most 2^{dn} satisfying assignments and appropriately chosen s, t and P,* **Rolf** *finds $z \in \mathbf{C}(z, I_z)$ in time*

$$2^{(1-\gamma(\alpha))(1-d)n}(2/\alpha)^{dn},$$

where

$$\gamma(\alpha) = \frac{6\alpha - 6\alpha^2 \ln(1/\alpha) - 4\alpha^3 - 1}{3\alpha^2}$$

and any constant α s.t. $\frac{1}{2} \leq \alpha \leq 1$.

The above lemma is useful for formulas with few satisfying assignments. For formulas with many satisfying assignment, Hirsh[Hir98] gave the following result:

Theorem 5.4 ([Hir98]) *For any 3CNF formula F with at least 2^{dn} satisfying assignments, there exists a deterministic algorithm that finds a satisfying assignment of F in time $2^{7.37(1-d)n}$.*

Combining these algorithms, we improve the 1.473^n upper bound for 3SAT for formulas with 2^{dn} satisfying assignments when d is sufficiently small or large. Figure 5.1 shows the running time of three algorithms.

5.5 Exploiting bias in satisfying assignments

In this section, we consider 3CNF-formulas having the bias in their solutions or many satisfying assignments.

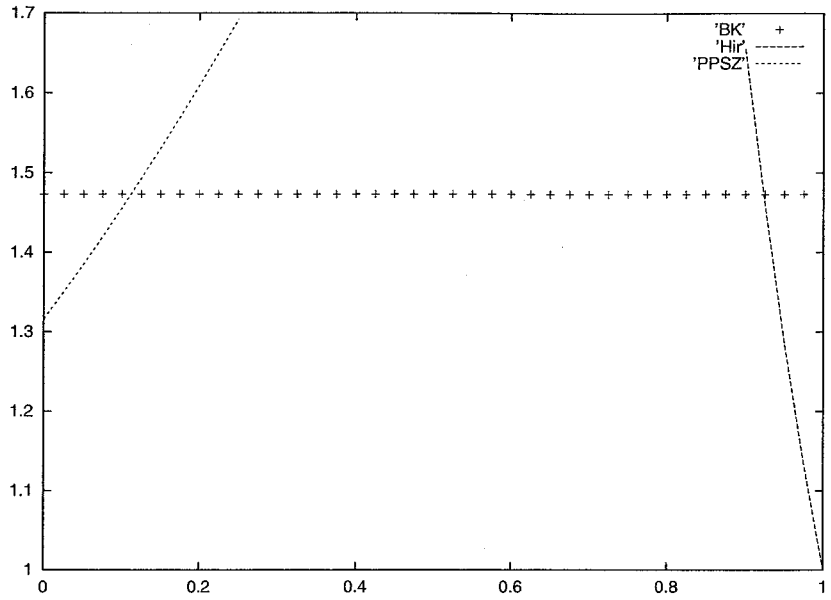

Figure 5.1: Running time of deterministic algorithms

Bias in a satisfying assignment Suppose that a given formula F has a satisfying assignment a^* which has l 0's and $(n-l)$ 1's. Let $p_0 = l/n$ and $p_1 = (n-l)/n$. We have shown how to modify the initial assignment of **SCH** to obtain a better success probability, i.e., each variable x_i is assigned 0 with probability q_0 and is assigned 1 with probability q_1, where the value of q_0 is appropriately determined. If we choose optimal q_0 with respect to p_0, the following holds:

$$\sigma(F,z) \geq \begin{cases} \left\{ p_0^{p_0}(1-p_0)^{1-p_0} \left(\frac{3}{2}\right) \right\}^n & \text{for } \frac{1}{2} \leq p_0 \leq \frac{2}{3}, \\ \left(\frac{1}{2}\right)^{(1-p_0)n} & \text{for } p_0 > \frac{2}{3}. \end{cases}$$

Now we consider similar situation in **PPSZ**. Given a uniquely satisfiable formula F, the following quantity measures the *strong bias* in a satisfying assignment z.

$$p_0 = p_0(F) \equiv \min_{\pi} \left\{ \frac{|\{i \in I(F,\pi,z) \mid x_i = 0\}|}{|I(F,\pi,z)|} \right\}$$

If we assign 0 with probability p_0 and 1 with probability $1 - p_1$ to each variable x_i,

the following holds: If $p_0 \geq 1/2$,

$$\begin{aligned}\tau(G,z) &= \mathbf{E}_\pi[\mathbf{Pr}_y[y \in \mathbf{C}(z, I(G,\pi,z))]] \\ &\geq \mathbf{E}_\pi[\{p_0^{p_0}(1-p_0)^{1-p_0}\}^{-|I(G,\pi,z)|}] \\ &\geq \{p_0^{p_0}(1-p_0)^{1-p_0}\}^{-(1-\gamma)n}.\end{aligned}$$

The success probability is shown in Fig. 5.2. Compared to **SCH**, **PPSZ** is less sensitive to the bias of assignments.

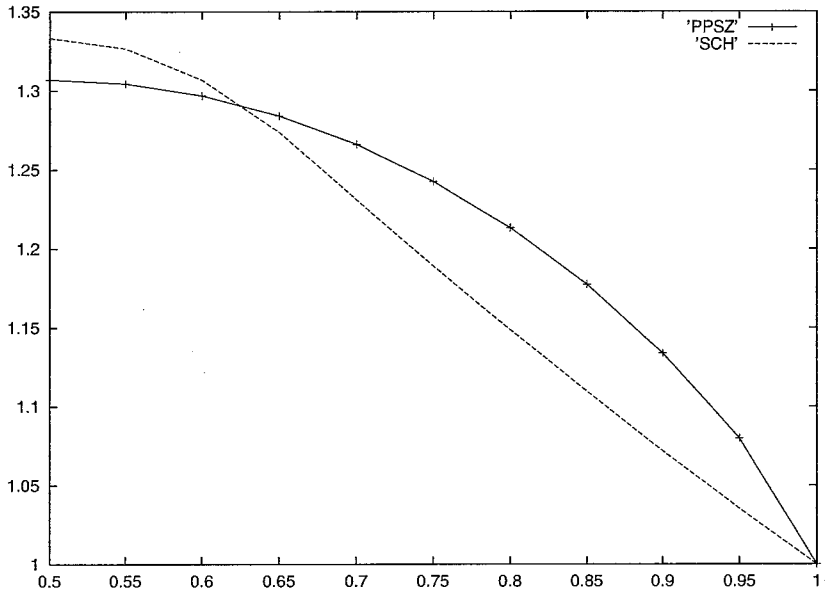

Figure 5.2: Bias in a satisfying assignment and time complexity

Number of satisfying assignments Figure 5.3 shows the running time of algorithms for 3CNF formulas with 2^{dn} satisfying assignments. This result follows from following lemmas.

Theorem 5.5 ([CIKP03]) *For n variables k-CNF F with s solutions, success probability of* **PPSZ** *is at least* $1/(2^n/s)^{1-1/k}$.

Lemma 5.7 *For n variables k-CNF F with* 2^{tn} *solutions, success probability of*

PPSZ *is at least*

$$\tau(F_s, z|\mathbf{C}(z, I_z)) \geq 2^{-\left(1-\frac{\mu_k}{k-1}\right)} \left\{ \frac{(k-2)2^{-1-\frac{\mu_k}{k-1}}}{k-1} \right\}^{tn}.$$

Theorem 5.6 *For n variables k-CNF F with $2^{h(\alpha)n}$ solutions, success probability of* **SCH** *is at least*

$$\begin{cases} \left(\frac{k}{2(k-1)}\right)^n \cdot (k-1)^{\alpha n}, & 0 \leq \alpha \leq \frac{1}{k} \\ 2^{h(\alpha)n}/2^n, & \frac{1}{k} \leq \alpha \leq \frac{1}{2} \end{cases}$$

This theorem can be proved by lemma 2.3 and the following isoperimetric inequality:

Lemma 5.8 ([Har66], (see [AS00], p.104)) *Let $B(x)$ be a Hamming ball of radius x and $B(A, x)$ be a set of points within Hamming distance at most x from some point in a set A. Then,*

$$|S| \geq |B(a)| \Rightarrow |B(S, d)| \geq B|(a + d)|.$$

We can see that **SCH** is not so fast as **PPSZ** for formulas with very large number of satisfying assignments. This is an interesting phenomena compared to the analysis based on subcube partition, where **SCH** is better for assignments with large number of defining variables.

5.6 Concluding Remarks

In this chapter, we propose a new algorithm that improves the current best worst-case upper bounds of 3SAT under assumption. Our main idea is to use **RandomWalk** for obtaining better initial assignments for **PPSZ**. The result also improves the current best worst-case upper bounds of Unique3SAT, which seems more difficult than to improve that of the general case. Of course, the obvious future work is to verify the assumption used here. We do not know our result can be derandomized since we do not have a fast deterministic algorithm for a formula with many satisfying

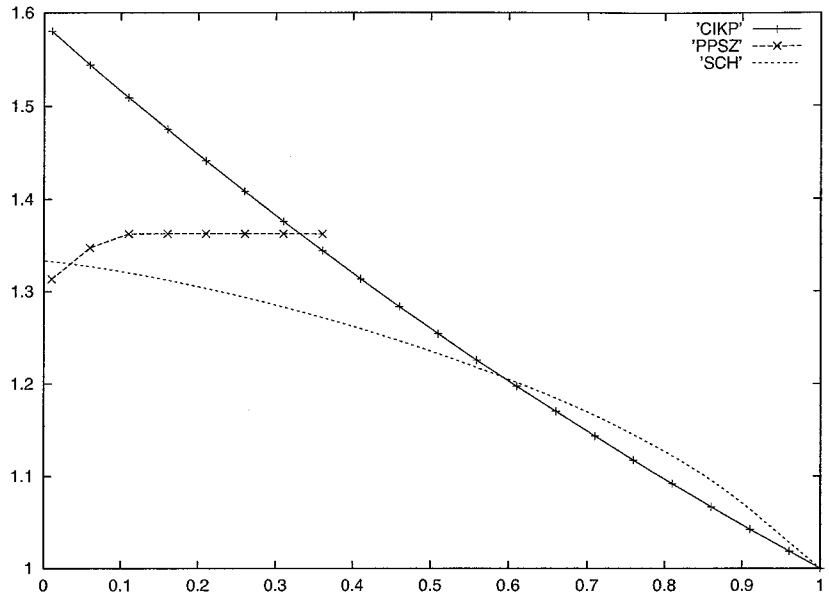

Figure 5.3: Number of satisfying assignments and time complexity

assignments.

Chapter 6

Conclusion

In this thesis we study the computational complexity of CNF Satisfiability problem (SAT for short) in an algorithmic point of view. One of the central problem in SAT is to design algorithms for n variable formulas that run in time c^n ($c < 2$). We would like to reduce the constant c in the exponent as small as possible. In this thesis we give several new algorithms and analyses of them for 3SAT, where clause length of input formulas is restricted to three.

In Chapter 3, we improve Schöning's randomized local search algorithm using partial knowledge on satisfying assignments. Our main idea is to use the bias in the number of 0's and 1's of a satisfying assignment. Actually we take the the number of 0's and 1's as a parameter and design an algorithm optimized for each parameter value. Though our algorithm do not improve the general case, we can apply our algorithm to some natural combinatorial problems like 3DM matching, where the number of 0's and 1's in satisfying assignments are often inbalanced. Then we give some experimental results that shows our algorithm is faster for several instances in practical sense.

In Chapter 4, we present new worst-case upper bounds for 3SAT. The previous best algorithm for general 3SAT is an improved version of Schöning's algorithm and for unique 3SAT is PPSZ's randomized splitting algorithm. Our result is based on the following observation: if an input formula has few satisfying assignments, PPSZ's performs well, and if it has many satisfying assignments, Schöning's does well. We give the analysis of two algorithms using a combinatorial structure that is

closely related to the number of satisfying assignments of formulas. We obtain an improved upper bounds by selecting two algorithms for the parameter value used in the analysis.

We also present an improvement on Schöning's randomized local search algorithm. In the original Schöning's algorithm we use an uniformly generated assignment as starting point of search. Hofmeister et al. gave a better way to obtain a good starting point with higher probability by using independent clause set and improved the algorithm. We extend the definition of independent clause set, namely, we introduce independent clause pair set and analyze it. As a result, we obtain a further better way to obtain a good starting point of search and improved the previous algorithm.

In Chapter 5, we propose a new algorithm that improves the current best worst-case upper bounds of 3SAT shown in Chapter 4. It is known that PPSZ's algorithm can find a satisfying assignment in sub-exponential time when given an assignment that agrees with a satisfying assignment in a large fraction (say, 2/5 fraction) of variables. We say such an assignment a good assignment. In the current best algorithm we try to find a good assignment by guessing uniformly at random. Our basic idea is to use Schöning's algorithm for obtaining a good assignment with higher probability. We analyze this new algorithm under some assumption and improves the current best worst-case upper bounds of 3SAT. Under the same assumption, we also improves unique 3SAT case, which seems more difficult than to improve that of the general case.

Throughout this thesis, we follow a general framework to design new algorithms as follows: First we define the property of formula and parameterize it. Second we design an algorithm optimized for formulas with a specific range of the parameter. Finally if we have a set of algorithms optimized for every range of the parameter. Our framework is promising for further improvements. A good start point is to find another natural and useful properties of CNF formulas.

Bibliography

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. M. Spaccamela, and M. Protasi. Complexity and Approximation. Combinatorial Optimization Problems and their Approximability Properties. Springer-Verlag, 1999.

[All96] E. Allender. Circuit complexity before the dawn of the new millennium. In *Proceedings 16th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS '96)*, Lecture Notes in Computer Science 1180, pp. 1–18, 1996.

[APT79] B. Aspvall, M. F. Plass and R. E. Tarjan. A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. *Information Processing Letters*, 8(3):121–123, 1979.

[AS00] N. Alon and J. H. Spencer. The probabilistic method (second edition). John Wiley & Sons, Inc., 2000.

[AS03] V. Arvind and R. Schuler. The Quantum query complexity of 0-1 knapsack and associated claw problems. In *Proc. ISAAC 2003*, LNCS 2906, 168–177, 2003.

[BE05] R. Beigel and D. Eppstein. 3-coloring in time $O(1.3289^n)$. *J. Algorithms*, 54(2):168–204, 2005.

[Bei99] R. Beigel. Finding Maximum Independent Sets in Sparse and General Graphs. In *the proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[BK04] T. Brueggemann and W. Kern. An improved local search algorithm for 3-SAT. *Theoretical Computer Science*, 329(1-3):303–313, 2004.

[BP98] P. Beame and T. Pitassi. Propositional Proof Complexity: Past, Present, and Future. *Bulletin of the European Association for Theoretical Computer Science*, 65:66–89, 1998.

[BS90] R. Boppana and M. Sipser. The complexity of finite functions. Handbook of Theoretical Computer Science, Vol.A:Algorithm and Complexity, Elsevier, 1990.

[BS03] S. Baumer and R. Schuler. Improving a probabilistic 3-SAT Algorithm by Dynamic Search and Independent Clause Pairs. *Selected Revised Papers from 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, LNCS 2919:150–161, 2003.

[Bys04] Jesper Makholm Byskov. Enumerating Maximal Independent Sets with Applications to Graph Coloring *Operations Research Letters*, 32(6):547–556, 2004.

[Che03] H. Chen. An Algorithm for SAT Above the Threshold. In *Proc. SAT 2003*, LNCS 2919, pp. 14–24, 2003.

[CI95] B. Cha and K. Iwama. Performance test of local search algorithms using new types of random CNF formulas. *Proc. IJCAI95*, pp. 304–310, 1995.

[CI96] B. Cha and K. Iwama. Adding new clauses for faster local search. *Proc. AAAI 1996*, pp. 332–337, 1996.

[CIKM97] B. Cha, K. Iwama, Y. Kambayashi, and S. Miyazaki. Local search algorithms for partial MAXSAT. *Proc. AAAI 1997*, pp. 263–268, 1997.

[CIKP03] C. Calabro, R. Impagliazzo, V. Kabanets and R. Paturi. The complexity of unique k-SAT: an isolation lemma for k-CNFs. In *Proceedings of IEEE Conference on Computational Complexity*, pp. 135–141, 2003.

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, pp. 151–158, 1971.

[Dan83] E. Dantsin. Two systems for proving tautologies, based on the split method. *Journal of Soviet Mathematics*, 22:1293–1305, 1983.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[DGH+02] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $2 - \frac{2}{k+1}$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002.

[DHIV03] E. Dantsin, E. A. Hirsch, S. Ivanov, M. Vsemirnov. Algorithms for SAT and Upper Bounds on Their Complexity. *Journal of Mathematical Sciences*, 118(2):4948–4962, 2003.

[DHW04] E. Dantsin, E. A. Hirsch, A. Wolpert, Algorithms for SAT based on search in Hamming balls. In *Proceedings of STACS 2004*, LNCS 2996, pp. 141–151, 2004.

[DHW05] E. Dantsin, E. A. Hirsch and A. Wolpert. Clause shortening combined with pruning yields a new upper bound for deterministic SAT algorithms. ECCC Report TR05-102, 2005.

[DKW05] Evgeny Dantsin, Vladik Kreinovich, Alexander Wolpert. On quantum versions of record-breaking algorithms for SAT. *ACM SIGACT News*, 36(4):103–108, 2005.

[DP60] A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[Epp03] D. Eppstein. The traveling salesman problem for cubic graphs. *8th Worksh. Algorithms and Data Structures*, Lecture Notes in Comp. Sci. 2748, pp. 307–318, 2003.

[FG04] Jorg Flum and Martin Grohe. Parametrized Complexity and Subexponential Time. *Bulletin of the European Association for Theoretical Computer Science*, 84, 2004.

[FGK05] Fedor V. Fomin, Fabrizio Grandoni and Dieter Kratsch. Measure and Conquer: Domination - A Case Study. *Proc. ICALP 2005*, pp. 191–203, 2005.

[FGK06] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Measure and Conquer: A Simple $O(2^{0.288n})$ Independent Set Algorithm. *Proc. SODA 2006*, to appear.

[FKT04] Fedor V. Fomin, Dieter Kratsch, Ioan Todinca: Exact (Exponential) Algorithms for Treewidth and Minimum Fill-In. *Proc. ICALP 2004*, pp. 568–580, 2004.

[Fla03] A. Flaxman. A spectral technique for random satisfiable 3CNF formulas. In *Proc. SODA 2003*, pp. 357–363, 2003.

[FvM00] L. Fortnow and D. van Melkebeek. Time-space tradeoffs for nondeterministic computation *Proceedings of the 15th IEEE Conference on Computational Complexity*, pp.2–13, 2000.

[Gen98] I. P. Gent. On the stupid algorithm for satisfiability. APES Technical Report 03-1998.

[GHNR03] J. Gramm, E. A. Hirsch, R. Niedermeier and P. Rossmanith. New Worst-Case Upper Bounds for MAX-2-SAT with Application to MAX-CUT. *Discrete Applied Mathematics*, 130(2):139–155, 2003.

[GNR04] R. Gummadi, N. S. Narayanaswanny and V. Ramaswamy. Algorithms for satisfiability using independent sets of variables. *The Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, LNCS 3542, pp. 133-144, 2004.

[Har66] L. H. Harper. Optimal numberings and isoperimetric problems on graphs. *Journal of Combinatorial Theory*, 1:385–394, 1966.

[Har67] L. H. Harper. A necessary condition on minimal cube numberings. *Journal of Applied Probability*, 4:397–401, 1967.

[Hås89] J. Håstad. Almost optimal lower bounds for small depth circuits. Randomness and Computation, Advances in Computing Research 5, pp. 143–170, 1989.

[Hir98] E. A. Hirsch. A Fast Deterministic Algorithm for Formulas That Have Many Satisfying Assignments. *Logic Journal of the IGPL*, 6(1):59–71, 1998.

[Hir00] E. A. Hirsch. New Worst-Case Upper Bounds for SAT. *Journal of Automated Reasoning*, 24(4):397–420, 2000.

[Hir03] E. A. Hirsch. Worst-case study of local search for MAX-k-SAT. *Discrete Applied Mathematics*, 130(2):173–184, 2003.

[HJP95] J. Håstad, S. Jukna and P. Pudlák. Top-down lower bounds for depth-three circuits. *Computational Complexity*, 5:99–112, 1995.

[HK05] E. A. Hirsch, A. Kojevnikov. UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. *Annals of Mathematics and Artificial Intelligence*, 43(1–4):91–111, 2005.

[HSSW02] T. Hofmeister, U. Schöning, R. Schuler and O. Watanabe. Probabilistic 3-SAT Algorithm Further Improved. *Proceedings 19th Symposium on Theoretical Aspects of Computer Science*, LNCS 2285: 193–202, 2002.

[IKM+00] K. Iwama, D. Kawai, S. Miyazaki, Y. Okabe, and J. Umemoto. Parallelizing local search for CNF satisfiability using vectorization and PVM. *Proc. Workshop on Algorithm Engineering (WAE 2000)* pp. 123–134, 2000.

[IM02] K. Iwama and H. Morizumi. An explicit lower bound of $5n - o(n)$ for boolean circuits. In *Proceedings 27th Conference on Mathematical Foundation of Computer Science*, LNCS 2420, pp. 353–364, 2002.

[IP99] R. Impagliazzo and R. Paturi. Complexity of k-SAT. In *Proceedings of IEEE Conference on Computational Complexity*, 1999.

[IPZ01] R. Impagliazzo, R. Paturi and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[KP92] E. Koutsoupias, C. H. Papadimitriou. On the Greedy Algorithm for Satisfiability. *Inf. Process. Lett.* 43(1): 53-55, 1992.

[Kul99] O. Kullmann, New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223(1-2):1–72, 1999.

[KV06] M. Krivelevich and D. Vilenchik. Solving random satisfiable 3CNF formulas in expected polynomial time. *Proc. SODA 2006*, to appear, 2006.

[Luc84] H. Luckhardt. Obere Komplexitätsschranken für TAUT-Entscheidungen. In *Frege Conference 1984*, pp. 331-337, 1984.

[MS79] B. Monien and E. Speckenmeyer. 3-satisfiability is testable in $O(1.62^r)$. Technical Report Beicht Nr. 3/1979, Universität-Gesamthochschule-Paderborn, 1979.

[MS85] B. Monien and E. Speckenmeyer. Solving satisfiability less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985.

[Mor93] P. Morris. The breakout method for escaping from local search minima. *Proc. AAAI 1993*, pp. 40–45, 1993.

[NR03] R. Niedermeier and P. Rossmanith. An efficient fixed parameter algorithm for 3-Hitting Set. *Journal of Discrete Algorithms*, 1:89–102, 2003.

[Pap91] C. H. Papadimitriou. On selecting a satisfying assignment. In *Proccedings of FOCS 1991*, pp. 163–169, 1991.

[Pap94] C. H. Papadimitriou. Computational Complexity. Addison-Wesley, 1994.

[PPSZ98] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. *Proceedings 39th Annual Symposium on Foundations of Computer Science*, pp. 628–637, 1998.

[PPSZ05] R. Paturi, P. Pudlák, M. E. Saks and F. Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, 2005.

[PPZ99] R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. *Chicago Journal of Theoretical Computer Science*, 1999.

[PSZ00] R. Paturi, M. E. Saks, and F. Zane. Exponential lower bounds for depth 3 Boolean circuits. *Computational Complexity*, 9(1):1–15, 2000.

[Pud98] P. Pudlák. Satisfiability – algorithms and logic. In *Proc. MFCS 1998*, LNCS 1450, pp. 129–141, 1998.

[PZ98] R. Paturi and F. Zane. Dimension of projections in boolean functions. *SIAM J. Discrete Math.*, 11(4):624–632, 1998.

[Rol03] D. Rolf. 3-SAT $\in$ $RTIME(O(1.32793^n))$. ECCC TR03-054, 2003.

[Rol05a] D. Rolf. Derandomization of PPSZ for Unique-k-SAT. In *Proc. SAT2005*, LNCS 3569, pp. 216–225, 2005.

[Rol05b] D. Rolf. Improved Bound for the PPSZ/Schöning-Algorithm for 3-SAT. ECCC TR05-159, 2005.

[SAT03] Enrico Giunchiglia, Armando Tacchella (Eds.). Selected Revised Papers from 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), Lecture Notes in Computer Science 2919, 2004.

[SAT04] Holger H. Hoos, David G. Mitchell (Eds.). Revised Selected Papers from 7th International Conference on Theory and Applications of Satisfiability Testing (SAT 2004), Lecture Notes in Computer Science 3542, 2005.

[SAT05] Fahiem Bacchus, Toby Walsh (Eds.). Proceedings 8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005), Lecture Notes in Computer Science 3569, 2005.

[Sch92] I. Schiermeyer. Solving 3-Satisfiability in less than 1.579^n steps. *CSL 92*, LNCS 702, pp. 379–394, 1992.

[Sch99] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. *Proceedings 40th Annual Symposium on Foundations of Computer Science*, pp. 410–414, 1999.

[Sch02] U. Schöning. A probabilistic algorithm for k-SAT based on limited local search and restart. *Algorithmica*, 32(4):615–623, 2002.

[Sch05] R. Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *Journal of algorithms*, 54(1):40–44, 2005.

[SK93] B. Selman and H. Kautz. Local search strategy for satisfiability testing. 2nd DIMACS challenge Workshop, 1993.

[SLBH05] L. Simon, D. Le Berre, E. A. Hirsch, The SAT2002 Competition. *Annals of Mathematics and Artificial Intelligence*, 43(1-4):307–342, 2005.

[SLM92] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. *Proc. AAAI 1992*, pp. 440–446, 1992.

[SSW01] R. Schuler, U. Schöning, and O. Watanabe. An improved randomized algorithm for 3-SAT. LA Symposium 2000 (Winter), 2001.

[Tre04] Luca Trevisan. A Note on Approximate Counting for k-DNF. In *Proc. APPROX-RANDOM 2004*, pp. 417–426, 2004.

[Urq95] Alasdair Urquhart. The Complexity of Propositional Proofs. In Current Trends in Theoretical Computer Science, 1995.

[Val77] L. Valiant. Graph-theoretic arguments in low-level complexity. In *Proceedings 6th Conference on Mathematical Foundation of Computer Science*, LNCS 53, pp. 162–176, 1977.

[Vaz01] V. V. Vazirani. Approximation Algorithms. Springer-Verlag, 2001.

[Weg87] I. Wegener. The Complexity of Boolean Functions. Wiley, 1987.

[Wil04] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, to appear. An earlier version appeared in ICALP 2004.

[Wil05] R. Williams. Better Time-Space Lower Bounds for SAT and Related Problems. In *Proceedings IEEE Conference on Computational Complexity (CCC 2005)*, pp. 40–49, 2005.

[Woe03] Gerhard J. Woeginger. Exact Algorithms for NP-Hard Problems: A Survey. *Revised Papers from 5th International Workshop on Combinatorial Optimization*, Lecture Notes in Computer Science 2570, pp. 185–208, 2003.

[Woe04] Gerhard J. Woeginger. Space and Time Complexity of Exact Algorithms: Some Open Problems (Invited Talk). *IWPEC 2004*, LNCS 3162, pp. 281–290, 2004

[Yam05] Masaki Yamamoto. An Improved $O(1.234^m)$-Time Deterministic Algorithm for SAT. *Proc. ISAAC 2005*, pp. 644–653, 2005.

[Zan98] F. Zane. Circuits, CNFs, and Satisfiability. PhD thesis, UCSD, 1998.

Publication List

- Kazuo Iwama and Suguru Tamaki. Exploiting Partial Knowledge of Satisfying Assignments. *Discrete Applied Mathematics*, invited and accepted for presentation in a special issue on the theory and applications of satisfiability testing.

- Kazuo Iwama and Suguru Tamaki. Improved Upper Bounds for 3-SAT. In *Proceedings of 15th annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2004), pp. 328–329, New Orleans, USA, January, 2004.

- Kazuo Iwama and Suguru Tamaki. Exploiting Partial Knowledge of Satisfying Assignments. In *Proceedings of 5th Workshop on Algorithm Engineering* (WAE 2001), Lecture Notes in Computer Science 2141, pp. 118–128, Aarhus, Denmark, August, 2001.

- Kazuo Iwama and Suguru Tamaki. Exploiting Partial Knowledge of Satisfying Assignments. Working Notes of LICS2001 Workshop on Theory and Applications of Satisfiability Testing, Boston, USA, pp. 106–114, June, 2001.

- Kazuo Iwama and Suguru Tamaki. Increasing the Success Probability of PPSZ-type Satisfiability Testing. *IEICE Technical Committee on Computation* (COMP), Tokyo, Japan, March, 2006, to appear.

- Kazuo Iwama and Suguru Tamaki. Improved Upper Bounds for 3-SAT. Electronic Colloquium on Computational Complexity, Report TR03-053, pp. 1–3, 2003.