5-2010

Parameterized Algorithm for 3-SAT

Yi Gao
University of Texas-Pan American

PARAMETERIZED ALGORITHM FOR 3-SAT

A Thesis

by

YI GAO

Submitted to the Graduate School of the
University of Texas-Pan American
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2010

Major Subject: Computer Science

PARAMETERIZED ALGORITHM FOR 3-SAT

A Thesis
by
YI GAO

COMMITTEE MEMBERS

Dr. Yang Liu
Chair of Committee

Dr. Zhixiang Chen
Committee Member

Dr. Bin Fu
Committee Member

May 2010

ABSTRACT

Gao, Yi, Parameterized Algorithm for 3-SAT. Master of Science (MS), May, 2010, 29 pp., 3 tables, 7 figures, and 35 references.

The SAT problem is the classical NP-complete problem. In the past, many methods have been proposed for solving this problem. We investigated a new method for 3-SAT problem, which is a fixed parameterized algorithm proposed in this paper first. This method uses a fixed parameter k, where k is the number of true values in an assignment for checking whether the formula is satisfied or not. The complexity of our algorithm is $O(3^k)$, which is exponentially independent of the number of variables. Theoretical analysis shows that when k is small, this method has smaller search space and higher speed.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

The SAT problem is one of the typically classical NP-complete problems in computer science [1]. It is to determine whether the variables of a formula can be assigned in such a way that the formula evaluates to be true. In recent years, SAT problem is a central problem that has applications in numerous areas, such as artificial intelligence, mathematical logic, and computing theory with wide range of computer aided design applications [2-18]. Modern SAT-solvers can be generally used to solve many important and practical problems. Therefore, it has attracted the researchers' interest and attention.

P and NP-complete

In computational complexity theory, NP is the set of all decision problems for which the "yes"-answers can be verified to be correct or not in polynomial time by a non-deterministic Turing Machine. In an equivalent formal definition, NP is the set of decision problems solvable in polynomial time by a non-deterministic Turing Machine, including P problems, NP- complete problems and etc.

The relationship between the complexity classes P and NP is the famous open problem in theoretical computer science. See Figure 1 for their relationship.

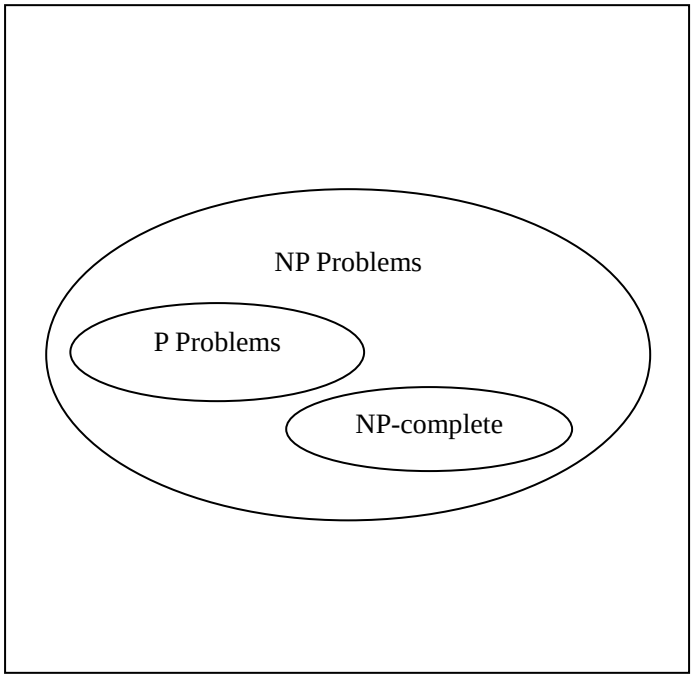

Figure 1 Diagram of P and NP-complete

Problem Description

SAT problem is a decision problem. It tests whether a given Boolean formula, usually represented in conjunctive normal form, is satisfiable or not by searching for an assignment of true or false values to variables that makes all the clauses of the formula evaluate to true.

A Boolean formula is said to be in CNF, conjunctive normal form, if it is a conjunction ("and") of disjunctions ("ors") of literals. A literal is either x, or its negation $\overline{x}$, for a Boolean variable x. According to Boolean logic, a literal has two different types; one is positive, the other is negative. For example, x_5 is a positive literal and $\overline{x_5}$ is a negative literal. Disjunctions are called clauses. Propositional formulas that

are not in CNF can be transformed into CNF in a standard way [29] and [30], and this process is called clausification. Determining the satisfiability of a formula in which each clause is limited to at most three literals is NP complete. An assignment for a certain formula can be checked in polynomial time. We explain it by using the example below:

$$\left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(x_2 \vee \overline{x_3} \vee x_4\right) \wedge \left(\overline{x_1} \vee x_3 \vee x_5\right) \wedge \left(x_2 \vee \overline{x_4} \vee x_5\right) \wedge \left(x_3 \vee x_4 \vee \overline{x_5}\right)$$

This formula has five clauses and five literals, and there are exactly three literals in each clause. Suppose there is such an assignment, x_1 = TRUE, x_2 = TRUE, x_3 = TRUE, x_4 = TRUE, x_5 = TRUE. The answer to the instance is “YES”, because each clause is TRUE. Actually any set of assignments that includes x_1 = TRUE, x_2 = TRUE, x_3 = TRUE is satisfiable. If there is no such assignment, the answer to the instance is “NO” and the formula is not satisfiable.

Application

Up to now, SAT problem is not only studied in theoretical computer science but also intensively used in hardware design, electronic design automation (EDA) [2], delay-fault testing, equivalence checking, circuit delay computation, logic synthesis and functional vector generation [3], among other applications. (See [2, 4, 5, 6, 7, and 8]). Moreover, SAT problem can especially help researchers to solve instances of binate covering problems for those in which the constraints are hard to satisfy [9, 10, 11, 12, and 13]. In addition, its application involves to other domains, such as Artificial Intelligence [14, 15] and Operations Research [16].

Recently we have seen dramatic improvements in SAT algorithms, which have been thoroughly validated in different application areas. With respect to applications of SAT in EDA, in most cases, the original problem formulation starts from a circuit description, for

which a given (circuit) property needs to be validated for at least one primary input vector. The resulting circuit formulation, which may only be implicitly specified, is then mapped into an instance of SAT, in most cases, using Conjunctive Normal Form (CNF) formulas.

SAT solving algorithms involve compute-intensive, logic bit-level, highly parallelizable operations, which makes reconfigurable computing appealing [17]. Many methodologies have been proposed to accelerate SAT solving using reconfigurable computing by either migrating the whole problem to hardware or partitioning the problem into hardware and software parts [18, 19]. SAT solving is hard and in no way precludes its use in solving the particular SAT instances that arise in real problems. Recent progress in practical applications of SAT has built upon two bases: improved SAT-solving engines and innovative ways to encode real problems in ways that can exploit those engines. Recent SAT-solvers have been developed in a scientific community that has greatly practical applicability, and the development of the solvers has, in turn, spurred work on new ways to exploit such solvers. The resulting positive spiral has led, for instance, to the development of commercial hardware verification tools in which SAT-solvers play an important role.

The utilization of CNF models and SAT algorithms has important advantages:

- Existing and extensively validated SAT algorithms can be used instead of dedicated circuit algorithms.
- New improvements and SAT algorithms can be easily applied to each target application.
- As observed in [23], the structural information of the circuit can be ignored.

- In many electronic design automation problems, a large number of instances of SAT have to be solved for each circuit. Hence, mapping a given problem description into SAT can represent a significant percentage of the overall running time [24].

The general SAT problem is NP complete and is hard to solve when input size is large. However, there are some special cases of SAT problem, which can be solved efficiently, have significant application. One of the most important restrictions of SAT problem is HORNSAT, where the formula is a conjunction of Horn clauses [20]. This problem is solved by the polynomial-time Horn-satisfiability algorithm [21], and is in fact P-complete [22]. It can be seen as P's version of the Boolean satisfiability problem. A Horn clause is a clause with, at most, one positive literal, called the head of the clause, and any number of negative literals, forming the body of the clause. A Horn formula is a propositional formula formed by conjunction of Horn clauses.

K-SAT Problem

K-SAT problem is a special case of SAT problem, where clauses have at most k literals. If an assignment of variables that satisfies the formula exists, the formula is said to be satisfiable, otherwise it is unsatisfiable. A clause is satisfied if one of its literals is bound to true, unsatisfied if all its literals are bound to false.

2-SAT problem can be solved in polynomial time. It is to determine the satisfiability of a formula in which each clause is limited to at most two literals.

3-SAT is a special case of k-SAT. It is a well known NP complete problem and has many practical applications, like consistency check in expert system knowledge bases and asynchronous circuit synthesis [3]. 3-SAT problem remains NP-complete even if all

expressions are written in conjunctive normal form with exactly three literals per clause. Each literal is a variable or a negation of a variable and each variable can appear multiple times in the expression. Table 1 briefly introduces some well-known algorithms and their best upper bounds of 3-SAT.

Previous Works for K-SAT

The k-SAT problem is to judge whether a k-CNF G has a satisfying assignment. If $NP \neq P$, for $k > 2$, it has no polynomial time algorithm for the k-SAT.

In [25], the first well-known algorithm for 3-SAT problem was proposed by Monien and Speckenmeyer, it has a bound of $O(1.618^n)$.

In [26], a well known randomized algorithm bound for 3-SAT problem was given by Paturi, Pudlak, Saks, and Zane in 1998, this algorithm is called PPSZ. To find satisfying assignments of Boolean formulas in conjunctive normal form, they proposed and analyzed ResolveSat, which is a simple randomized algorithm. The algorithm consists of two stages: a preprocessing stage in which resolution is applied to enlarge the set of clauses of the formula, followed by a search stage that uses a simple randomized greedy procedure to look for a satisfying assignment. For each value of k, the algorithm is faster than any currently known algorithm.

In [27], Schöning proposed a simple randomized algorithm for solving the k-SAT problem. In the case of 3-SAT, the algorithm has an expected running time $O(1.334^n)$ when given a formula F on n variables. This was the best running time known for an algorithm solving 3-SAT. To solve k-SAT and constraint satisfaction problems, they present a simple probabilistic algorithm. This algorithm follows a simple local-search paradigm: randomly guess an initial assignment and then, guided by those clauses that are

not satisfied, by successively choosing a random literal from such a clause and flipping the corresponding bit, try to find a satisfying assignment. If no satisfying assignment is found after O(n) steps, start over again. Their analysis shows that for any satisfiable k-CNF formula with n variables this process has to be repeated only t times, on the average, to find a satisfying assignment, where t is within a polynomial factor of $(2(1-1/k))^n$.

In [28], Iwama and Tamaki improved the upper bound for 3-SAT to O (1.324^n) in 2003. They focused on the 3-SAT problem. The basic idea is to combine two existing algorithms, the one by Paturi, Pudl´ak, Saks and Zane and the other by Schöning. It should be noted, however, that simply running the two algorithms independently does not seem to work. Also, their approach can escape one of the most complicated portions in the analysis.

There are many other famous algorithms proposed in the past, a well known deterministic algorithm was proposed by Dantsin et al in [34]. A randomized algorithm was proposed by Iwama and Tamaki in [28].

Table 1 briefly introduces some well-known algorithms and their upper bounds of k-SAT, where $k \in \{3,4,5,6\}$.

Table1 Complexities of Selected Algorithms for the Satisfiability Problem [35]

Authors	Algorithm Type	3-SAT	4-SAT	5-SAT	6-SAT
Backtracking Algorithm	deterministic	$O(1.91^n)$	$O(1.96^n)$	$O(1.98^n)$	$O(1.99^n)$
Monien [25]	deterministic	$O(1.61^n)$	$O(1.83^n)$	$O(1.92^n)$	$O(1.96^n)$
Dantsin et al. [34]	deterministic	$O(1.48^n)$	$O(1.60^n)$	$O(1.66^n)$	$O(1.71^n)$
Paturi et al. [26]	randomized	$O(1.36^n)$	$O(1.47^n)$	$O(1.56^n)$	$O(1.63^n)$
Schöning [27]	randomized	$O(1.33^n)$	$O(1.50^n)$	$O(1.60^n)$	$O(1.66^n)$
Iwama and Tamaki [28]	randomized	$O(1.32^n)$	$O(1.47^n)$	N/A	N/A

CHAPTER II

PARAMETERIZED ALGORITHM FOR 3-SAT

Introduction of Parameterized Algorithm

According to the common belief that $P \neq NP$, NP-complete, or otherwise NP-hard, problems require time that is exponential in input size. Therefore, if the input size is large, it is unfeasible to find solutions to those problems such as SAT problem.

In real world, applications of NP-complete may have some small parameters which can be used to find solutions efficiently. Some problems with certain parameter fixed can be solved by algorithms that are exponential only in the size of the fixed parameters while polynomial in the size of the input size. Such an algorithm is called a fixed-parameter tractable algorithm. A parameterized problem that allows for such a fixed-parameter tractable algorithm is said to be a fixed-parameter tractable problem and belongs to the class FPT. It seems a good supplement of the theory of NP-completeness. The problem in FPT can be solved efficiently for small values of the fixed parameters.

For example, the vertex cover problem is in FPT. This problem is that given a graph G, to find k number of vertices in G such that every edge of G is incident to at least one of those vertices. It is a NP-complete problem which has been applied in many areas such as network optimization and bioinformatics.

An exhaustive search algorithm can solve the problem in time $2^{O(k)}n^{O(1)}$. Vertex cover is therefore a fixed-parameter tractable problem, and there may only need a vertex cover of a few vertices in some applications. For those applications of small vertex cover (i.e. k is small), we can solve the problem efficiently. After many researches, many fixed parameterized algorithm for this problem have been developed. A well known algorithm for this problem has a running time $O(1.286^k+kn)$ in [32].

However, some problems are not believed to be in FPT. An example is deciding whether an n-vertex graph contains an independent set of cardinality k or not. The complement of a maximum independent set is the set of vertices not belonging to the independent set, forms a minimum vertex cover, which is a fixed-parameter tractable problem. There is an algorithm which can solve the independent set of cardinality k with an upper bound of $O(n^{0.792k})$ in [31]. So far no algorithm with a running time of the form $f(k)n^{O(1)}$ is known.

Fixed Parameterized Algorithm for 3-SAT

We proposed a fixed-parameter tractable algorithm for 3-SAT problem in this paper. In other words, our algorithm has a fixed parameter k to solve the 3-SAT problem and here k is the number of true values for variables in an assignment. Therefore, our algorithm can determine if there is one assignment with exactly k number of variables true to make a given formula satisfiable. We select the variables in the assignment for true by finding the clauses whose literals are all positive. To make a clause of all positive literals true, we must set at least one variable to true. The assignment is unsatisfied unless it can make all the clauses true. Therefore, for all positive clauses, we must select one literal and set its variable in the assignment to true. In the next section, we proved that the

complexity of our algorithm is 3^k, and it is exponentially independent of the number of variables.

The pseudo codes and details of our fixed parameterized algorithm are in Figure 2, Figure 3 and Figure 5. Figure 2 is the main function of our algorithm. It runs two recursive functions alternatively. Figure 3 is the one recursive function to check if the given formula can be satisfied by assignment with exactly k number of true values and Figure 5 is the other recursive function to check if the given formula can be satisfied by assignment with exactly k number of false values.

```
Algorithm K_Try(F)

Input: an instance F of CNF.
Output: True or False.

1. Initialize two assignments A1 and A2;
2. Assign all the variables in A1 False, all the
   variables in A2 True;
3. Let n denote the number of varibles in F;
4. For k = 1 to n/2:
5.    If(P1(F,A1,k) is True or P2(F,A2,k) is True)
6.       return True;
7.    EndIf
8. EndFor
9. return False;
```

Figure 2 Our Algorithm for 3-SAT Problem

The Algorithm in Figure 2 calls the recursive function in Figure 3 and Figure 5 to find assignments with exactly k number of variables true and false separately. The recursive function P1 and P2 need a fixed parameter k to search a 3-tree structure T by

using Deep First Search algorithm. The 3-tree T has exactly k layers. Each node of T is a clause whose literals are all positive or negative, and each line of the 3-tree denotes a choice of setting one of the three variables in a clause to true or false. Figure 4 shows an example of the process of our algorithm in Figure 3. In Figure 3, the output of the function p1 is “True” or “False”. Output “True” means a satisfied assignment with k number of true values. The recursive function p2 in Figure 5 has the same logic of p1.

```
Recursive Function 1 P1(F,A,k)

Input: an instance F of CNF, an assignment A, and an
integer k.
Output: True if F can be satisfied by an assignment of at
most k variables of true value; False otherwise;

1.  If (k==0)
2.      If F is satisfied by the assignment A
3.          return True;
4.      Else
5.          return False;
7.  Find a clause C of all positive literals;
8.  For each literal L in C
9.      Let F'= F;
10.     Let A'= A;
11.     Set the value of L in A' to true;
12      Remove all clauses containing L, and all literals L
        in F';
13.     If (P1(F',A',k-1))
14.         return True;
15. EndFor
16. return False;
```

Figure 3 Recursive Function 1

In this example, we suppose k = 2, and select variables by finding the clause with all

positive literals. The process stop and return true when a satisfied assignment is found; it return false if there is no satisfied assignment has been found until the last leaf of this 3-tree has been tested.

1. Find some clauses with all positive literals (suppose we find the clause $(x_1 \vee x_2 \vee x_3)$).
2. Set x_1 = true.
3. Find another all positive clause, suppose we find the clause $(x_4 \vee x_5 \vee x_6)$.
4. Set x_4 = true, x_5 = true, x_6 = true respectively, and test the assignment is satisfied or not. If unsatisfied, reset them false (suppose not satisfied).
5. Set x_1 = false, x_2 = true; repeat step 3 and step 4.
6. Set x_2 = false, x_3 = true; repeat step 3 and step 4.
7. If not satisfiable, return No.

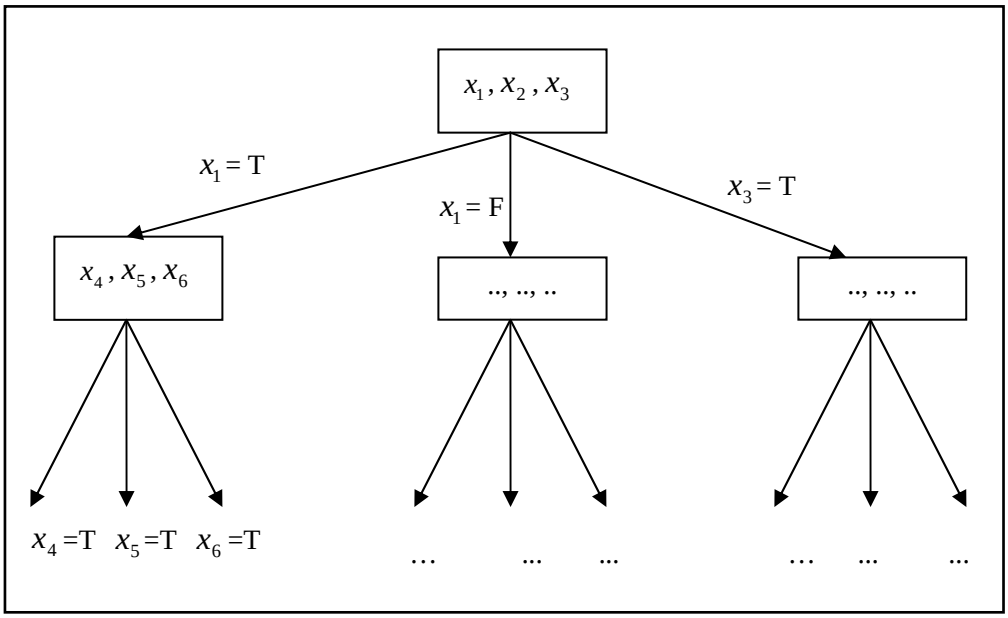

Figure 4 Example of Our Fixed Parameterized Algorithm

```
Recursive Function 2 P2(F,A,k)

Input: an instance F of CNF, an assignment A, and an
integer k.
Output: True if F can be satisfied by an assignment of at
most k variables of false value; False otherwise;

1.  If (k==0)
2.      If F is satisfied by the assignment A
3.          return True;
4.      Else
5.          return False;
7.  Find a clause C of all negative literals;
8.  For each literal L in C
9.      Let F'= F;
10.     Let A'= A;
11.     Set the value of L in A' to false;
12      Remove all clauses containing L, and all literals L
        in F';
13.     If (P1(F',A',k-1))
14.         return True;
15. EndFor
16. return False;
```

Figure 5 Recursive Function 2

The Correctness of Our Fixed Parameterized Algorithm

In this section, we show our algorithm solves the 3-SAT correctly. Given a 3-CNF instance F, and an integer parameter k, it is obvious that if the result of our algorithm to the instance F is “TRUE”, F can be satisfied by an assignment of less or equal than k true variables.

We will prove if the result of our algorithm to F is “FALSE”, there is no assignment with less or equal k true variables exists to make F satisfied.

Suppose, there is one such assignment A and the result of our algorithm to F is "FALSE".

First of all, the assignment A can make all the clauses true, thus it makes the first selected clause true, the first selected clause means the root of the 3-tree based on our method. Therefore, at least one variable of this clause in the assignment A is true. In addition, A also makes the clause of corresponding child node true, thus at least one variable of this child node in the assignment A is true. This process can be repeated until it reach a leaf of the 3-tree, otherwise Assignment A is unsatisfied. It is clear that the assignment generated by that leaf should be the same assignment as A. It is that the two assignments share k number of true variables and both of them have exactly k number of true variables.

The result of our algorithm to F is "FALSE", which means the assignment generated by that leaf is not satisfied, however A is satisfied, which is a contradiction. Thus we obtain the following theorem.

Theorem 1 Given a 3-CNF instance F, there is an assignment with less or equal than k number of true variables satisfied if and only if the result of our parameterized algorithm to F is "TRUE".

The Complexity of Our Fixed Parameterized Algorithm

Suppose the operation of test an assignment is O(1), our algorithm need $O(3^k)$ time to check whether an satisfied assignment exists with less or equal than k true values.

This is a recursive process which can be shown by a 3- tree structure in Figure 4. Without loss of generality, assume k>=1. To determine k number of variables, it first finds a clause with all positive literals $(x_i \vee x_j \vee x_l)$, and set x_i, x_j and x_l to true

respectively. After one variable in the clause has been selected and set to true, it continue find other clauses of all positive literals and set one variable to true per time until k number of true variables have been selected. After that, it checks whether the assignment is satisfied or not. If not, it set the true variable back to false; and set another variable in the clause to true to do the satisfiability test again. If all the variables in the clause have been selected and no assignments satisfied, it has to go back to the previous clause, do the above steps again. There are two conditions make this recursive process stop. The one is that a satisfied assignment is found; the other is that there is no satisfied assignment has been found until the last leaf of this 3-tree has been tested

Suppose that the complexity of the recursive function is f(x), where x is the number of true values. Based on the recursive function in Figure 3 or Figure 5, f(x) = 3f(x-1). We suppose that f (0) is O(1), therefore the complexity of f (k) = 3f (k-1) = 3^2f(k-2) = … = O(3^k). It is just the same as the number of nodes n in the 3-tree.

Furthermore, the common 3-SAT problem can be solved by our method in O($3^{n/2}$) time. To solve a general 3-SAT problem, we can try our method with k = 1, 2, 3, …, n/2 for the number of variables of true value, if there is no assignment satisfied, we then try our method with k = 1, 2, 3, …, n/2 for the number of variables of false value. The complexity is O(3^1) + O(3^2) + … + O($3^{n/2}$), which is O($3^{n/2}$). The formula is unsatisfiable if there is no solution after that. If there is an assignment makes the formula satisfiable, at least n/2 numbers of its variables are true; otherwise, at least n/2 number of its variables are false.

Thus we obtain the following theorems.

Theorem 2 Given a 3-CNF instance F and an integer k, our algorithm need no more

than the $O(3^k)$ time to check if there exists an assignment with equal or less than k true variables make F satisfied.

Theorem 3 Given a 3-CNF instance F with n variables, our algorithm need no more than $O(3^{n/2})$ time to check its satisfiability.

CHAPTER III

EXPERIMENTAL RESULTS

Algorithm Implementation

There are two algorithms for the 3-SAT problem we implemented in this paper. The first algorithm is a fixed parameterized algorithm proposed by us firstly. The second one is the algorithm proposed by Schöning in [27] in this paper.

Schöning proposed a simple randomized algorithm for solving the K-SAT problem. In the case of 3-SAT, the algorithm has an expected running time $O(1.334^n)$ when given a formula F on n variables. To solve k-SAT and constraint satisfaction problems, they present a simple probabilistic algorithm. This algorithm follows a simple local-search paradigm: randomly guess an initial assignment and then, guided by those clauses that are not satisfied, by successively choosing a random literal from such a clause and flipping the corresponding bit, try to find a satisfying assignment. If no satisfying assignment is found after $3n$ steps, start over again.

This method is a simple randomized algorithm and it is not only very efficient but also easy to implement. Therefore, we first implement this method as the beginning of our research.

```
Schoning Algorithm P_Try(F)

Input: an instance F of CNF.
Output: True or False.

1.    Guess an initial assignment A
2.    Let n denote the number of varibles;
3.    Repeat 3n times:
4.       If (the formula is satisfied by the
         assignment A)
5.          Stop and return True;
6.       Else
7.          Find all the clauses not being satisfied
            by the assignment A;
8.          Let C be one clause unsatisfied;
9.          Pick one of the k literals in the clause C
            uniformly at random and change its value
            in the assignment A;
10.      EndIf
11.   EndRepeat
12.   return False;
```

Figure 6 Schöning's Algorithm

Figure 6 is the core function in Schöning's algorithm. The algorithm is a randomized algorithm which tries the function in Figure 2 at most $O(1.334^n)$ times. Therefore, its complexity is $O(1.334^n)$. If no satisfied assignment can be found after $O(1.334^n)$ time, It says the formula is unsatisfiable.

The Complexities of Our Fixed Parameterized Algorithm

If the input formula is unsatisfiable, the result of running the above two algorithms will always be rejected. However, if it is satisfiable, we suppose that the probability of getting a satisfied assignment is p. It is clear that p is related with the input size n. Then, if we try the procedure m times and the probability that we do not find a satisfying

assignment after m repetitions with independent random bits is $(1-p)^m$, which is equal or less than e^{-pm}. Therefore, to achieve an acceptable error probability of e^{-20} one needs to choose $m = 20 \times p^{-1}$ independent repetitions of try. It was shown in [27] that p is equal or greater than $(k/2(k-1))^n$ for k-SAT. Therefore, for 3-SAT problem, if we try the above algorithm $20 \times (4/3)^n$ times, the complexity of the algorithm for 3-SAT is $O(1.334^n)$ and achieve an error probability of no more than e^{-20}.

Schöning's algorithm is a simple algorithm and it is easy to implement. When the input size is small, the algorithm is very fast. However, It is obvious that the complexity of this algorithm is exponential related with the number of variables n in a formula, which means when n is larger; its speed is very slow. In this paper, we proposed a new algorithm to solve the 3-SAT problem; it is a fixed-parameter tractable algorithm. In other words, our algorithm needs a fixed parameter k to solve the 3-SAT problem, where k is the number of true values for variables in an assignment. Therefore, our algorithm can determine if there is one assignment with exactly k numbers of variables true to make a given formula satisfiable. The complexity of our algorithm is 3^k, and it is exponentially independent of the number of input variables.

Experimental Results

The solvers are implemented by c++. The experiments were performed on a computer with an Intel Pentium 2.8 GHz processor and 1,024 megabytes of RAM memory. We use the generator mkcnf.c for the test cases. The program "mkcnf.c" [33] is loosely based on "mwff.c", which was contributed to the Dimacs Challenge by Bart Selman of AT&T.

We first test the performance of our fixed parameterized algorithm by using different

ranges of k with test data 1. 1000 test cases are generated by the program “mkcnf.c”. Each case has 50 variables and 120 clauses. In addition, we guarantee that each of them is satisfiable.

Table 2 lists the experimental results. The Total Running Time is the running time for checking the whole cases in test data 1, and the Number of Satisfied Cases is how many cases are satisfied in test data 1 according to the result of our algorithm when k is smaller than 5, 10, 15, 20 and 25.

If k is equal or less than 5, the average running time is 0.00134 seconds, and 23 instances have solutions. If k is enlarged to 15, all the instances are satisfied and the average running time is 0.07376 seconds. This experiment shows if k is small, this fixed parameterized algorithm has smaller search space and higher speed.

Table 2 Running Times of Our Fixed Parameterized Algorithm

	k<=5 (n=50)	k<=10 (n=50)	k<=15 (n=50)	k<=20 (n=50)	k<=25 (n=50)
Total Running Time (sec)	1.34	52.27	73.76	73.75	73.76
Average Running Time (sec)	0.00134	0.05227	0.07376	0.07375	0.0.7376
The Number of Satisfied Cases	23	846	1000	1000	1000
Accuracy (satisfied number / total number)	2.3%	84.6%	100%	100%	100%

Figure 7 shows the detail of the first experiment, when k is becoming bigger, the number of satisfied cases is increasing exponentially, however the running is also becoming longer. Therefore, the experimental results show that our fixed parameterized algorithm is very suitable for the 3-SAT problems that need assignments with limit number of true values.

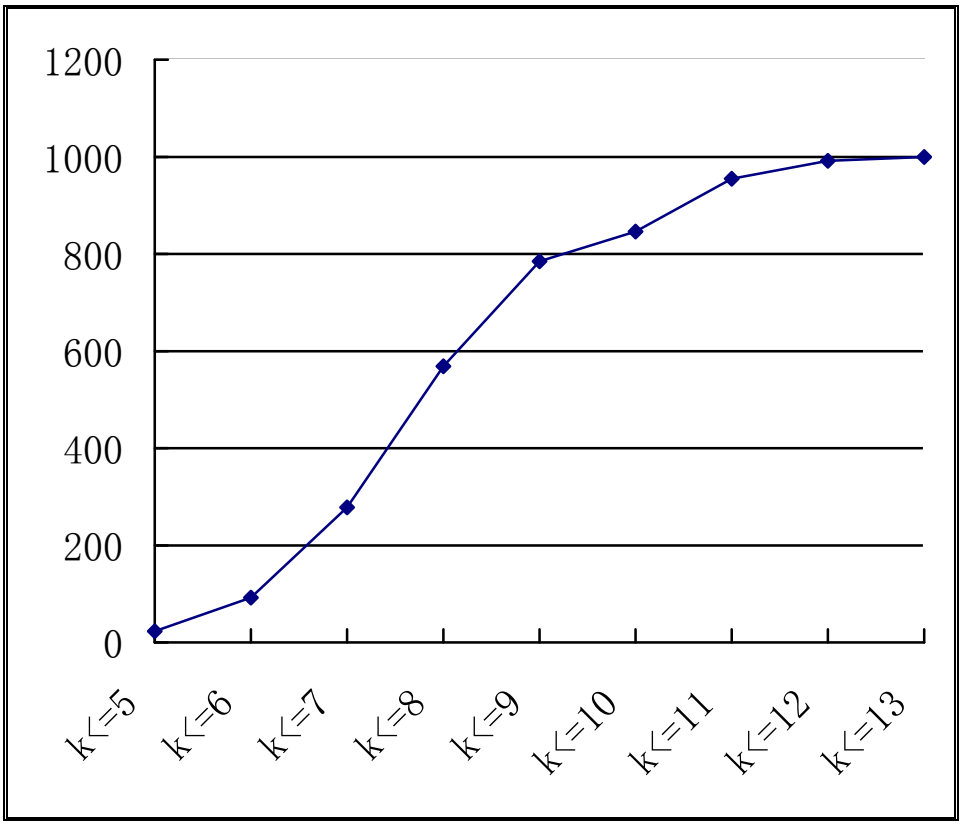

Figure 7 Plots of Our Fixed Parameterized Algorithm for Different k

To further test the time performance of the algorithm, we compare it with Schöning's algorithm by using test data 2. We generated 1000 test cases for this experiment. Each of them has 30 variables and 100 clauses.

The test cases of the second experiment are special and different with the previous test data 1. The test cases in test data 1 are produced by the program "mkcnf.c". The test cases in test data 2 were generated by our won. For each test case, we guarantee that at least one assignment with less or equal than 5 variables true can make it satisfiable. To make sure that each test case has a satisfiable solution with less or equal variable true, we did the following steps when generate the test data 2.

1. Generate the 120 clauses one by one; each clause has exactly 3 literals.
2. Make sure the 3 literals in a clause are different variables.

3. Make sure the clauses generated are different.
4. For each literal in a clause, randomly select a variable x_i from the variable set $\{x_1, \ldots, x_i, \ldots, x_{30}\}$.
5. If the selected i is less or equal than 5, make the literal positive.
6. If a clause doesn't contain variables i less or equal than 5, randomly make one of its literals negative.

Test cases generated by the above method are satisfied by the assignment that x_1, x_2, x_3, x_4 and x_5 are true, and from x_6 to x_{30} are false. Therefore, the assignment has exactly 5 true values and it is clear that all the test cases in test data 2 are satisfied by it.

We check the satisfiability of the test cases by the SAT solver implemented by Schöning's algorithm and by our fixed parameterized algorithm, and compare their running time.

The data in Table 3 shows that the running time of our algorithm is shorter than that of Schöning's Algorithm. It means that when k is smaller than 5, the SAT solver implemented with our fixed parameterized algorithm has smaller search space and higher speed than the well known SAT solver by Schöning.

It is reasonable because the parameter k is no more than 5. Suppose the operation of testing the assignment is O(1), the running time of our algorithm is $O(3^5)$, and the running time of Schöning's is 20×1.33^n, where n is the number of variables. In the experiment, the number of variables is 30. It is obviously that our algorithm is faster than that of Schöning.

Table 3 Comparisons of Two Algorithms

	Schöning's Algorithm	Fixed Parameterized Algorithm
Total Running Time (sec)	48.72	21.01
Average Running Time (sec)	0.04872	0.02101
The Number of Test Cases	1000	1000

The experimental results show that our algorithm can help to improve efficiency on resolve a 3-SAT problem when the input size n is large and the parameter k is relative small. It is better than Schöning's Algorithm for solving 3-SAT when k is less or equal than five. Our theoretical analysis proves that, when parameter k is a small value, the SAT solver implemented with our method has smaller search space and higher speed.

CHAPTER IV

SUMMARY AND CONCLUSION

We investigated algorithms for the 3-SAT problem and. First we study the well known SAT solver proposed by Schöning. Next we develop a fixed parameterized algorithm for 3-SAT problem. Our algorithm needs a fixed parameter k to solve the 3-SAT problem and k is the number of true values for variables in an assignment. The theoretical analysis shows that the complexity of our algorithm is $O(3^k)$, which is exponentially irrelevant with the number of variables. We test its performance by comparing it with Schöning's algorithm. Our experimental results show that when k is smaller or equal than 5, its speed is fast than that of Schöning's. Therefore, we hold that the SAT problem can be solved efficiently for small values of the fixed parameters.

In the future, we would like to further study the fixed parameterized algorithm for K-SAT problem.

REFERENCES

[1] S. Cook, “The complexity of theorem-proving procedures,” *in Proc. 3rd ACM Symp. on the Theory of Computing. ACM Press*, 1971.

[2] T. Larrabee, “Test Pattern Generation Using Boolean Satisfiability,” *IEEE Trans. on Computer-Aided Design*, vol. 11, no. 1, pp. 4-15, January 1992.

[3] F. Fallah, S. Devadas and K. Keutzer, “Functional Vector Generation For HDL Models Using Linear Programming and 3-Satisfiability,” *in Proc. of the Design Automation Conf.*, pp. 528-533, June 1998.

[4] C.-A. Chen and S. K. Gupta, “A Satisfiability-Based Test Generator for Path Delay Faults in Combinational Circuits,” *in Proc. of the Design Automation Conf.*, pp. 209-214, June 1996.

[5] S. Devadas, K. Keutzer and S. Malik, “Computation of Floating Mode Delay in Combinational Circuits: Practice and Implementation,” *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 12 no. 12, pp. 1923-1936, December 1993.

[6] J. Marques-Silva and K. A. Sakallah, “Robust Search Algorithms for Test Pattern Generation,” *in Proc. of the Int’l Symp. on Fault- Tolerant Computing*, pp. 152-161, June 1997.

[7] P. McGeer, A. Saldanha, P. R. Stephan, R. K. Brayton and A. L. Sangiovanni-Vincentelli, “Timing Analysis and Delay-Test Generation Using Path Recursive Functions,” *in Proc. of the Int’l Conf. on Computer-Aided Design*, pp. 180-183, November 1991.

[8] P. Tafertshofer, A. Ganz and M. Henftling, “A SAT-Based Implication Engine for Efficient ATPG, Equivalence Checking, and Optimization of Netlists,” *in Proc. of the Int’l Conf. on Computer-Aided Design*, pp. 648-657, November 1997.

[9] O. Coudert, “On Solving Covering Problems,” *in Proc. of the Design Automation Conf.*, June 1996.

[10] F. Ferrandi et al., “Symbolic Algorithms for Layout-Oriented Synthesis of Pass Transistor Logic Circuits,” *in Proc. of the Int’l Conf.e on Computer-Aided Design*, November 1998.

[11] P. Flores, H. Neto and J. Marques-Silva, “An Exact Solution to the

Minimum-Size Test Pattern Problem" *in Proc. of the Int'l Conference on Computer Design*, October 1998.

[12] R. Fuhrer and S. Nowick, "Exact Optimal State Minimization for 2- Level Output Logic," *in Int'l Workshop on Logic Synthesis*, June 1998.

[13] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publishers, 1996.

[14] R. Bayardo Jr. and R. Schrag, "Using CSP Look-Back Techniques to Solve Real-World SAT Instances," *in Proc. of the Nat'l Conf. on Artificial Intelligence*, pp. 203-208, July 1997.

[15] H. Zhang, "SATO: An Efficient Propositional Prover," *in Proc. Of Int'l Conf. on Automated Deduction*, pp. 272-275, July 1997.

[16] P. Barth, "A Davis-Putnam Enumeration Algorithm for Linear pseudo-Boolean Optimization," *Technical Report MPI-I-95-2-003, Max Planck Institute for Computer Science*, 1995.

[17] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM computing Surveys*, vol. 34, no. 2, pp. 171-210, June 2002.

[18] I. Skliarova and A. B. Ferrari, "Reconfigurable Hardware SAT Solvers: A Survey of Systems," *IEEE Trans. on Computers*, vol. 53, no. 11, pp. 1449-1461, November 2004.

[19] J. P. Marques-Silva and L. Guerra e Silva, "Solving Satisfiability in Combinational Circuits," *IEEE Design and Test of Computers*, pp. 16-21, July-August 2003.

[20] H. J. Keisler, "Reduced products and Horn classes", *Transactions of the American Mathematical Society*, vol. 117, pp. 307–328, 1965.

[21] Chandru, Vijaya; Collette R. Coullard, Peter L. Hammer, Miguel Montañez, and Xiaorong Sun, "On renamable Horn and generalized Horn functions," *Annals of Mathematics and Artificial Intelligence*, pp. 33–47, 2005.

[22] Cai, Jin-Yi; Sivakumar, D., "Sparse hard sets for P: resolution of a conjecture of Hartmanis", *Journal of Computer and System Sciences*, vol. 58, no.2, pp. 280–296, 1999.

[23] P. Tafertshofer, A. Ganz and M. Henftling, "A SAT-Based Implication Engine for Efficient ATPG, Equivalence Checking, and Optimization of Netlists," *in Proc. of the Int'l Conf. on Computer-Aided Design*, pp. 648-657, November 1997.

[24] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Trans. on Computer-Aided Design*, vol. 11, no. 1, pp. 4-15, January 1992.

[25] B. Monien, E. Speckenmeyer, “Solving satisfiability in less than 2^n steps,” *Discrete Applied Mathematics*, vol. 10, pp. 287–295, 1985.

[26] R. Paturi, P. Pudlak, M.E. Saks, and F. Zane, “An improved exponential-time algorithm for k-SAT,” *in Proc. of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pp.628-637, 1998.

[27] U. Schöning, “A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems,” *in Proc. of the 40th Annual Symposium on Foundations of Computer Science*, pp.410, October 17-18, 1999.

[28] Kazuo Iwama, Suguru Tamaki, “Improved upper bounds for 3-SAT,” *in Proc. of the 15th annual ACM-SIAM symposium on Discrete algorithms*, January 11-14, 2004.

[29] G. Tseitin, “On the complexity of derivation in propositional calculus,” *Studies in Constr. Math. and Math. Logic*, 1968.

[30] J. Marques-Silva, “Practical Applications of Boolean Satisfiability,” *in Int. Workshop on Discrete Event Systems, WODES08*, 2008.

[31] Papadimitriou C H, Yannakakis M, “On the complexity of database queries,”*Journal of Computer and System Sciences*, vol. 58, pp. 407-427, 1999.

[32] Fellows M. Blow-ups, “win/win’s, and crown rules: Some new directions in FPT,” *Lecture Notes in Computer Science* (WG’03), pp.1-12, 2003.

[33] Van Gelder, A, Problem generator mkcnf.c. *In Proc. DIMACS.* Challenge archive, 1993.

[34] Evgeny Dantsin, Andreas Goerdt, Edward A Hirsch, Ravi Kannan, Jon Kleinberg, Christos Papadimitriou, Prabhakar Raghavan, Üwe Schoning, “A deterministic (2- 2/(k+ 1))^n algorithm for k-SAT based on local search,” *Theoretical Computer Science*, vol. 289, pp. 69-83, 2002,.

[35] Jörg Rothe, “Komplexitätstheorie und Kryptologie: Eine Einführung in Kryptokomplexität,” pp. 295, Springer, 2008

BIOGRAPHICAL SKETCH

Yi Gao was born in Shanghai, PRC. She got a Bachelor Degree of Accounting in University of Shanghai for Science and Technology. Before going to study in USA, she was a research assistant in Haitong Securities Co. Ltd in Shanghai and later became a Finance Accountant in the same company. She graduated from University of Texas-Pan American with Master of Science in Computer Science in May, 2010.