The Analysis and Research on Computational Complexity

Qiang Gao[1], Xinhe Xu[2]
1.College of Information Science and Engineering, Northeastern University, Shenyang,110004
E-mail: tommy_06@163.com
2.College of Information Science and Engineering, Northeastern University, Shenyang,110004
E-mail: xuxinhe@mail.neu.edu.cn

Abstract：Computational complexity is a branch of the theory of computation. It is used to measure how hard a problem is solved and the common measures include time and space. The classes of time complexity generally include: P, NP, NP-hard, NP-complete and EXPTIME; the classes of space complexity generally include: PSPACE, NPSPACE, PSPACE-hard and PSPACE-complete. Researching computational complexity of a problem can make it explicit whether there is an effective solving algorithm of the problem or not. This paper introduces and analyzes some fundamental concepts of computational complexity, and discusses complete problems of time complexity and space complexity by examples; What's more, the relation among complexity classes is analyzed in detail.

Key words：Computational complexity, Turing machine, NP-complete, PSPACE-complete

1 Introduction

Theory is relevant to practice. It provides conceptual tools that practitioners use in computer engineering[1]. Computational complexity is an important branch of the theory of computation, it aims to tell us how many resources (times or spaces) a problem is solved with. Some classical hard problems it involves include: traveling salesman problem[2]; SAT (satisfiability problem, it involves chip testing and computer design to image analysis and software engineering etc[2]); circuit complexity[3]; computer game[1] and so on. The research on computational complexity could help us understand capabilities and limitations of the computer much better. If a problem is proved to be solved with great difficulty, people need not spend much effort on looking for efficient solving algorithm of the problem[1] and then make a rational choice. This paper introduces some fundamental concepts of computational complexity, summarizes current research status and discusses complete problems of time complexity and space complexity by examples; What's more, the relation among complexity classes is analyzed in detail.

2 Computational complexity

Computational complexity is an important branch of the theory of computation. It is used to measure how many resources (e.g. times or spaces) a problem is solved with[1]. Some fundamental concepts of computational complexity will be given before time complexity and space complexity are introduced.

2.1 Decidable problem

Decidable problem refers to a problem whose algorithm is solvable, i.e. it could enter its accept state or reject state and the computation is terminated (i.e. halting state[4]) when it is implemented on computational models (e.g. finite automaton or Turing machine). Undecidable problem refers to a problem whose algorithm is unsolvable, i.e. it could not enter halting state (i.e. doesn't halt[4]) when it is implemented on computational models.

2.2 Mapping reducibility

Problem A is mapping reducible to problem B, if there is a computational function f, where for every w, $w \in A \Leftrightarrow f(w) \in B$, written $A \leqslant m\ B$. The function f is called the reduction of A to B[1].

It refers to convert one problem to another problem, and then a solution to the second problem is used to solve the first problem. If problem A is mapping reducible to problem B, the solution to problem B could solve problem A[3]. For example, if you want to find your way around a new city, you can solve it by referring to the map of the city. So finding your way around the city may be reduced to the problem of finding a map of the city.

2.3 DTM and NTM

2.3.1 DTM—Deterministic Turing Machine

Turing machine (Fig.1) is a model of a general purpose computer[5], first proposed by Alan Turing in 1936[1], and it can do everything that a real computer can do. Turing machine uses an infinite tape as its unlimited memory. It has a tape head that can read and write symbols and move around on the tape. Initially the tape contains only the input string and it is blank everywhere else. If the machine needs to store information, it may write this information on the tape. To read the information that it has written, the machine can move its head back over it. The machine continues computing until it decides to produce an output. The outputs accept and reject are obtained by entering designated accepting and rejecting states. If it can not enter either of them, it will go on forever, never halting[1].

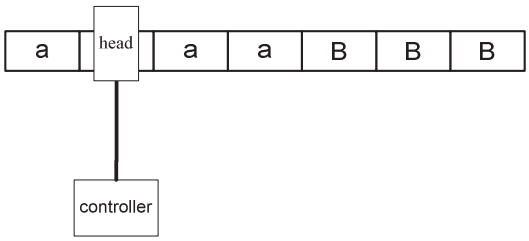

Fig.1 The picture of Turing machine

Turing machine has an infinite tape that can write and read, and its head can be moved back over it, so its capability of describing solving algorithm of problem is

Supported by National Natural Science Foundation of China, ID61370153

better than finite automaton[1] (Its memory is limited) or pushdown automaton[1] (Its stack is a "last in, first out storage device). For example, DTM can decide $A = \{0^{2^n} \mid n \geq 0\}$, the language (Language is a formal description of problem) consisting of all strings of 0s whose length is a power of 2,the description (DTM M_2) of its algorithm[1] will be showed as follows,

M_2 = "On input string w:

1) Sweep left to right across the tape, crossing off every other 0.

2) If in stage 1 the tape contained a single 0, accept.

3) If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, reject.

4) Return the head to the left-hand end of the tape.

5) Go back to stage 1."

Each iteration of stage 1 cuts the number of 0s in half. If the number of 0s on the tape is odd and great than 1 after stage 1, the original number of 0s on the tape could not have been a power of 2. Therefore the machine rejects immediately[1].It is obvious that M_2 decides the language. Finite automaton can not describe such algorithm because of its limited memory. Although pushdown automaton has an unlimited storage stack, it can not achieve crossing off every other 0 because its stack is "last in, first out" device. Above all, DTM's capability of describing solving algorithm of problem is better.

2.3.2 NTM—Nondeterministic Turing Machine[5]

During the process of computation of DTM, there is only one possibility in its next action; NTM's next action may have several possibilities, so the computation of NTM is more like a tree and every branching point in the tree corresponds to a point in the computation[5]. If any branching point enters accept state, the machine accepts the input.

NTM doesn't correspond to any practical computing device and it is a hypothetical device[7]. However, NTM is a useful mathematic definition because it is more suitable to describe some problems such as computer game. According to the rule of computer games, that a player usually has many choices to move is same to the computation of NTM.

Every NTM has an equivalent DTM[1]. In other words, the capability of the two TMs is equivalent. A theorem was given in [7]: If a DTM simulates a NTM which runs in $t(n)$ time, the DTM will complete the simulation in $2^{O(t(n))}$ time.

3 Time complexity

Time complexity denotes that how much time the solution of a problem requires. Time that the solution of a problem with fixed size requires is a constant; for a problem with arbitrary size, the complexity of the solution is measured by a convenient form of estimation called asymptotic notation[1]. For example, the function $f(n) = n^3+n^2+1$, the asymptotic notation for describing this function is $O(n^3)$ (the big-O denotes a constant) because the highest order term dominates the other terms. Moreover, classifying time complexity of some problem can illustrate how hard it is solved.

3.1 P（Deterministic Polynomial Time）

P is the set of problems that can be solved in polynomial time on DTM[6]. The polynomial time algorithms are fast enough for many purposes, but exponential time algorithms are rarely used[1]. If a problem is in P, it is considered to be easy to solve[8].

The example of P: the problem of testing whether two numbers are relatively prime, i.e. RELPRIME = {<a, b> | a and b are relatively prime} [1]; If 1 is the largest integer that can evenly divide two numbers both, the two numbers are relatively prime. For example 9 and 10 are relatively prime. Euclidean Algorithm[9] is a fast solution that can judge whether two numbers are relatively prime or not. The algorithm is used to compute the greatest common divisor of natural number a and natural number b, written gcd (x, y)[10], it is the largest integer that evenly divides two numbers both, e.g. gcd (12,15) = 3. It is obvious that x and y are relatively prime if gcd (x, y) = 1.Euclidean Algorithm is described[9] with pseudocode as follows,

```
function gcd (a, b)
  while b ≠ 0
    t ← b （"←" denotes assignment）
    b ← a mod b ("mod" is the remainder after the integer division of a by b)
    a ← t
  return a
```

The output value of the algorithm is the greatest common divisor of a and b.

Algorithm R[1] solves RELPRIME, using E as a subrountine.

R = "On input <a, b>, where a and b are natural numbers:

1) Run E on <a, b>.

2) If the result is 1, accept. Otherwise, reject."

The analysis on complexity of R: Obviously, the complexity of R is determined by Euclidean Algorithm.

The complexity of Euclidean Algorithm have been thoroughly researched and is h^2 (h is digit of the smallest number both two numbers)[11]. Hence the complexity of algorithm R is h^2 and then indicates that R is in P, i.e. polynomial time algorithm

3.2 NP（Nondeterministic Polynomial Time）

NP is the set of problems that can be solved in polynomial time on NTM[6]. As previously mentioned, if a DTM simulates a NTM which runs in $t(n)$ time, the DTM will complete the simulation in $2^{O(t(n))}$ time. So the difference of time between P and NP is exponential and NP is suspected infeasible[8]. It is unclear so far whether the problems that are in NP have efficient polynomial time algorithms[8]. However, to verify whether certain problem is true or not is implemented in polynomial time[1]. For example, a polynomial verifiable problem is compositeness. Recall that a natural number is composite

if it is the product of two integers greater than 1. Although a polynomial solution to it hasn't been found, it is easy to verify whether a number is composite (all that is needed is a divisor of that number)[1] or not, e.g. 24 is composite because it is the product of 4 and 6. Above all, NP is the class of languages that have polynomial time verifiers[1].

3.3 EXPTIME（Exponential Time）

EXPTIME is the set of all decision problems solvable by a deterministic Turing machine in $O^{(2p(n))}$ time, where *p(n)* is a polynomial function of n. EXPTIME is not more feasible than other complexities that include P, NP, PSPACE and NPSPACE[3].

An example of EXPTIME is given: G_3[12],every position in G_3 is a 4-tuple (τ, W-LOSE(X,Y), B-LOSE(X,Y), α)，where τ denotes the player whose turn it is to play from position, W-LOSE=$C_{11}\vee C_{12}\vee C_{13}\vee\ldots\vee C_{1p}$ and B-LOSE=$C_{21}\vee C_{22}\vee C_{23}\vee\ldots\vee C_{2q}$ are Boolean formulas in 12DNF[13], that is, each C_{1i} or C_{2j} is a conjunction of at most 12 literals($1\leqslant i\leqslant p$) ($1\leqslant j\leqslant q$); Each literal is a variable in X(Y), e.g. z or ~z (the negation of z); α is an assignment (0 or 1) of values to the set of variables $X\cup Y$, e.g. X={x1, x2, x3}, Y={y1, y2, y3}, then $X\cup Y$ ={ x1, x2, x3, y1, y2, y3}. The players play alternately. Player W (B) moves by changing the value of precisely one variable in X (Y). In particular, passing is not permitted. Precisely, W can move from (W, W-LOSE, B-LOSE, α) to (B, W-LOSE(X,Y), B-LOSE(X,Y), α') iff B-LOSE is false under the assignment α, and α and α' differ in the assignment of exactly one variable in X. W(B) loses if the formula is true after some move of player W (B).[12] proved that this game is solved in $O(2^{n/\log n})$ time, where n is the length of input w. i.e. the game is in EXPTIME.

4 Space complexity

Space complexity denotes that how much space the solution of a problem requires. The problem that belongs to space complexity is much harder than the problem that belongs to time complexity because a machine that runs quickly cannot use a great deal of space. If a machine accesses at most a memory unit at each step of its computation, any machine that operates in *t(n)* time can use at most *t(n)* space[1]. Like time complexity, space complexity of the solution to a problem is estimated by asymptotic notation. Moreover, classifying space complexity of some problem can illustrate how hard it is solved.

4.1 PSPACE (Deterministic Polynomial Space)

PSPACE is the set of problems that can be solved in polynomial space on DTM[14]. A problem is given as follows, TQBF[6] (True Qualified Boolean Formula), e.g. $\forall x\exists y(x\wedge y)$, the function means that, for every value for the variable x, some value for the variable y makes x ∧y true. TQBF denotes that every variable that appears in the function is bound to the quantifier[6]. The TQBF problem is to determine whether a fully quantified Boolean formula is true or false and it is in PSPACE.

4. 2 NPSPACE (Non-deterministic Polynomial Space)

NPSPACE is the set of problems that can be solved in polynomial space on NTM. Space is different from time because space can be used repeatedly. According to Savitch's theorem[15], any NTM that uses *f(n)* space can be converted to a DTM that uses only $f^2(n)$ space. The theorem can draw a conclusion that PSPACE = NPSPACE because the square of any polynomial is still a polynomial[1].

5 The complete problem of computational complexity

Complete problems are considered to be the most difficult in a complexity class[2] because any other problem in the class is easily reduced to it. The followings are two common complete problems.

5.1 NP-complete

A language B is NP-complete if it satisfies two conditions:

(1) B is in NP, and

(2) every A in NP is polynomial time reducible to B[1].

NP-complete problem is the most difficult in NP class[2].

Cook[16] proved the first problem that is NP-complete[17], i.e. SAT[16](Satisfiability problem). A problem is proved to be NP-complete by mapping reducibility. If a problem is NP-complete, the polynomial time solution to it doesn't exist. So people need not spend much effort on looking for efficient solving algorithm of the problem[1]. According to the definition of NP-complete, if B only satisfies the second condition, B isn't in NP, then B is called NP-hard[1]. NP-hard could be more difficult than NP-complete.

An example of NP-complete: the vertex cover problem[18]

A vertex cover of undirected graph G is the subset of the nodes, called V, where every edge of G touches one of those nodes[19]. The vertex cover problem determines whether a graph contains a vertex cover of a specified size:

VERTEX-COVER = {<G, k> | G is an undirected graph that has a k-node vertex cover}[1]. It is NP-complete[19].

For the proof of NP-complete problem A, the definition of NP-complete must be used. The steps of the proof are as follows:

1) Prove that A is in NP.
2) Find a problem B that has been proved to be NP-complete.
3) Prove that B can be reducible to A.
4) Prove that the reducibility is implemented in polynomial time.

The third step is the most difficult in these steps, and the construction of a reasonable reducibility needs

some ingenious ideas. [1] used 3SAT (It is a conjunctive formula whose clause contains three literals. It is one of Karp's 21 NP-complete problems[17]) that had been proved to be NP-complete. Let 3SAT formula be $(x \vee x \vee y) \wedge (\bar{x} \vee \bar{y} \vee \bar{y}) \wedge (\bar{x} \vee y \vee \bar{y})$ and construct a reducibility(see Fig.2).

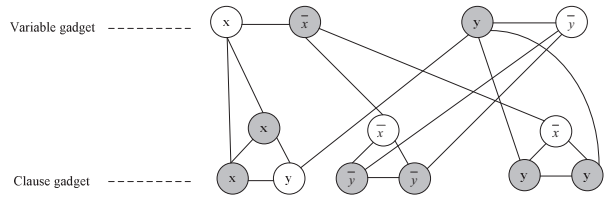

Fig.2 The structure of reducibility

The figure includes a variable gadget and a clause gadget. The variable gadget decides the assignments of variables and the clause gadget corresponds to three clauses of the 3SAT formula. There are a total of $2m + 3l$ (m denotes the number of variables in formula, l denotes the number of clauses) nodes. Let $k = m + l$, according to the 3SAT formula, the value of k is 8. So the condition that satisfies the 3SAT formula is that the undirected graph has a 8-node vertex cover. The rules[1] are as follows: Firstly, choose assignments of variables that satisfies the formula, i.e. setting x to be false corresponds to node ~x of the variable gadget and setting y to be true corresponds to node y of the variable gadget. Secondly, choose two nodes that are not true in every clause gadget and the nodes chosen become the nodes of vertex cover. Because every variable of the third clause is true, one variable that doesn't help the 8-node vertex cover is removed and another two nodes become the nodes of vertex cover. Like those shaded nodes shown in Fig. 2, these nodes cover all edges of the undirected graph. Hence it is proved that the vertex cover of the graph satisfies the 3SAT formula (i.e. make the formula be true), and then shows that 3SAT is reducible to the vertex cover. According to the definition of NP-complete, the vertex cover problem is NP-complete (Another steps of the proof are omitted).

5.2 PSPACE-complete[1]

B language is PSPACE-complete if it satisfies two conditions:

(1) B is in PSPACE, and

(2) every language in PSPACE is polynomial time reducible to B.

PSPACE-complete problem is the most difficult in PSPACE class. According to the definition of PSPACE-complete, if B only satisfies the second condition, B isn't in PSPACE, then B is called PSPACE-hard[1]. PSPACE-hard could be more difficult than PSPACE-complete. The common problems that are PSPACE-complete include QBF[20] (Qualified Boolean Formula), Formula Game[1], Chess Game[1] and so on.

An example of PSPACE-complete is given: the Generalized Geography Game[1] (hereinafter referred to as the GG). It is a geographic game, each player takes turns naming cities from anywhere in the world. Each city chosen must begin with the same letter that ended the previous city's name. Repetition isn't permitted. The game starts with some designated city and ends when some player loses because he or she is unable to continue. The process of proof on PSPACE-complete problem is similar to NP-complete, at first proves that the problem is in PSPACE, a polynomial space algorithm was given by [1] and proved that GG problem can be solved in polynomial space. Then make use of Formula Game (In fact, it is exactly TQBF) that had been proved to be PSPACE-complete and construct a reducibility skillfully (see Fig.3).

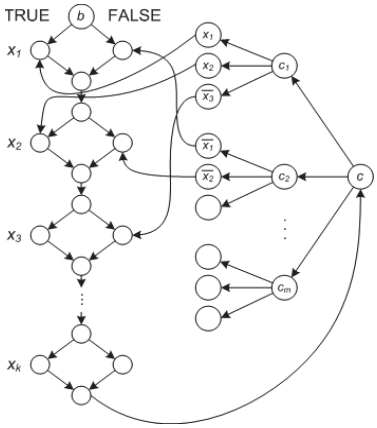

Fig. 3 The full structure of the reducibility[1]

This is a directed graph of GG, and the formula game will be simulated on this graph. Let the formula be $\exists x_1 \forall x_2 \exists x_3 \forall x_4 .... \exists x_k [(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \cdots) \wedge \cdots \wedge ()]$.

Play starts at b. Player I and II assign value to variables (i.e. choose nodes. Suppose that nodes on the left of every diamond (e.g. x_1, x_2, …, x_k) is 1 and those on the right is 0), hence certain player that can not choose node lose (According to the rule of GG, every nodes in the graph is chosen only one time). The reducibility guarantees that if formula is true, player I wins the formula game, and if formula is false, player II wins. And then the reducibility illustrates that the formula game can be simulate in GG and be solved. Hence the formula game can be reducible to GG. According to the definition of PSPACE-complete, GG problem is PSPACE-complete (Another steps of the proof are omitted).

6 The relations of some complexity classes

Some common time complexity class and space complexity class are introduced in the above chapters, and now this paper is going to discuss which of them is relatively easier, and which of them is relatively more difficult.

6.1 The relation of P and NP

P is the set of problems that can be solved in polynomial time on DTM; NP is the set of problems that can be solved in polynomial time on NTM. If a NP problem is solved in DTM, the DTM must simulate the process of NTM and the simulation is implemented in exponential time. Hence it is generally believed that NP is more difficult than P. i.e. $P \subseteq NP$ [3]. But P=NP? A lot of researchers have tried proving the question. It is one of the greatest unsolved problems in theoretical computer science and contemporary mathematics. If P = NP, all of problems that is in NP would have polynomial time algorithms, in other words, many hard problems will become easy to solve. But researchers has not succeeded to date[1]. Most researchers believe that the two classes are not equal. Cook[16] proved the first problem that is NP-complete, i.e. SAT[16](Satisfiability problem). In a sense all problems become one problem by mapping reducibility[2]. It provides convenience for researchers who try to prove whether P is equal to NP. Because if even one of them has a polynomial time algorithm, then every problem in NP has a polynomial time algorithm[2].

6.2 The relation of time complexity (P and NP) and space complexity (PSPACE and NPSPACE)

As previously mentioned, a machine that runs quickly cannot use a great deal of space. If a machine accesses at most a memory unit at each step of its computation, any machine that operates in $t(n)$ time can use at most $t(n)$ space[1]. Hence the problem that belongs to space complexity (PSPACE and NPSPACE) is much harder than the problem that belongs to time complexity (P and NP).

6.3 The relation of space complexity (PSPACE and NPSPACE） and EXPTIME

A lemma was given by [1]: let M be an LBA with q states and g symbols in the tape alphabet. There are exactly qng^n distinct configurations of M for a tape of length n. According to the lemma, a TM that uses $f(n)$ space can have at most $f(n)\times 2^{O(f(n))}$ (asymptotic notation) different configurations. Let every configuration need a time unit, then in the worst case (TM enters accept state or reject state when the $f(n)\times 2^{O(f(n))}$ th configuration is computed, i.e. TM enters halting state), the problem is solved in $f(n)\times 2^{O(f(n))}$ time. i.e. the PSPACE problems must be solved in $f(n) \times 2^{O(f(n))}$ time. Hence $PSPACE \subseteq EXPTIME$[3]. Above all, the relations of these complexity classes are

$P \subseteq NP \subseteq PSPACE=NPSPACE \subseteq EXPTIME$[1] ,as shown in Fig.4.

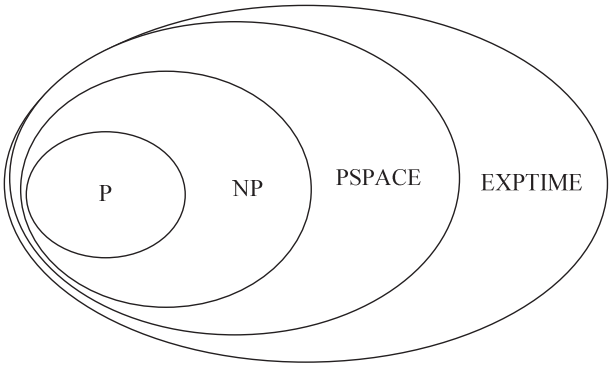

Fig.4 The relations of some complexity classes

7 Summaries

Computational complexity is a branch of the theory of computation. It is a computer science problem that is very theoretical and challenging. Researching on computational complexity could make us understand capabilities and limitations of the computer much better. If a problem is proved to be solved very hard (e.g. NP-complete, PSPACE-complete), people need not spend much effort on looking for efficient solving algorithm of the problem and then improve work efficiency. This paper introduces and analyzes some fundamental concepts of computational complexity, and discusses complete problems of time complexity and space complexity by examples; What's more, the relation among complexity classes is analyzed in detail. In this field, there are some problems that have not been solved yet. E.g. P=NP? , are the relations of these complexity classes absolutely right? These are very important topics concerned by scholars at home and abroad.

References

[1] Michael Sipser. Introduction to the Theory of Computation(Second Edition), China Machine Press, 2006.

[2] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, Algorithm, McGraw-Hill Science/Engineering/Math; 1 edition (September 13, 2006).

[3] Christos Papadimitriou, Computational Complexity, Addison-Wesley, ISBN 0-201-53082-1. Section 20.1, page 491.

[4] 朱一清，可计算性与计算复杂性，国防工业出版社，2006年4月.

[5] 堵丁柱，计算复杂性导论，高等教育出版社，2000.

[6] Sanjeev Arora, Boaz Barak. Computational Complexity, Cambridge University Press, 2007.

[7] 陈志平、徐宗本，计算机数学_计算复杂性理论与NPC、NP难问题的求解，科学出版社，2001.

[8] http://zh.wikipedia.org/wiki/NP

[9] http://zh.wikipedia.org/wiki/COPRIME

[10] Graham, R. L.; Knuth, D. E.; Patashnik, Concrete Mathematics, Addison-Wesley, 1989: 107-110.

[11] Honsberger R. Mathematical Gems II. The Mathematical Association of America. 1976.

[12] L. J. STOCKMEYER AND A. K. CHANDRA, Provably difficult combinatorial games, SIAM J. Compuf. 8 (1979), 151-174.

[13](美)罗森著、袁崇义译，离散数学及其应用，机械工业出版社，2011.

[14]http://zh.wikipedia.org/wiki/PSPACE

[15] Savitch, Walter J. (1970), Relationships between nondeterministic and deterministic tape complexities, Journal of Computer and System Sciences 4 (2): 177–19

[16] Cook, Stephen. The complexity of theorem proving procedures. Proceedings of the Third Annual ACM Symposium on Theory of Computing. 1971: 151 – 158.

[17]KARP, R.M. Reducibility among combinatorial problems, In Complexity of Computer Computations (1972), R.E. Miler and J.W. Thatcher, Eds., Plenum Press, pp. 85-103.

[18] Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms, MIT Press 2009 (2nd edition).

[19]http://zh.wikipedia.org/wiki/VERTEX COVER

[20] Lintao Zhang, SEARCHING FOR TRUTH: TECHNIQUES FOR SATISFIABILITY OF BOOLEAN FORMULAS, Princeton University, June 2003.