

A Probabilistic Study of 3-SATISFIABILITY

Tevfik Aytemiz

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Industrial and Systems Engineering

Sheldon H. Jacobson, Co-Chair

C. Patrick Koelling, Co-Chair

Ebru K. Bish

Russell D. Meller

Keying Ye

June 2001

Blacksburg, Virginia

Keywords: Satisfiability, 3-Satisfiability, Max 3-Satisfiability, Threshold Conjecture, Tabu Search, Probabilistic Analysis of Max 3-Satisfiability.

Copyright 2001, Tevfik Aytemiz

A Probabilistic Study of 3-SATISFIABILITY

Tevfik Aytemiz

(ABSTRACT)

Discrete optimization problems are defined by a finite set of solutions together with an objective function value assigned to each solution. Local search algorithms provide useful tools for addressing a wide variety of intractable discrete optimization problems. Each such algorithm offers a distinct set of rules to intelligently exploit the solution space with the hope of finding an optimal/near optimal solution using a reasonable amount of computing time.

This research studies and analyses randomly generated instances of 3-SATISFIABILITY to gain insights into the structure of the underlying solution space. Two random variables are defined and analyzed to assess the probability that a fixed solution will be assigned a particular objective function value in a randomly generated instance of 3-SATISFIABILITY. Then, a random vector is defined and analyzed to investigate how the solutions in the solution space are distributed over their objective function values. These results are then used to define a stopping criterion for local search algorithms applied to MAX 3-SATISFIABILITY.

This research also analyses and compares the effectiveness of two local search algorithms, tabu search and random restart local search, on MAX 3-SATISFIABILITY. Computational results with tabu search and random restart local search on randomly generated instances of 3-SATISFIABILITY are reported. These results suggest that, given a limited computing budget, tabu search offers an effective alternative to random restart local search. On the other hand, these two algorithms yield similar results in terms of the best solution found. The computational results also suggest that for randomly generated instances of 3-SATISFIABILITY (of the same size), the globally optimal solution objective function values are typically concentrated over a narrow range.

Acknowledgements

I would like to first acknowledge and thank my advisors Dr. Sheldon H. Jacobson and Dr. C. Patrick Koelling. Their guidance, time and continuing support through my doctoral dissertation studies made this research possible. I especially would like to thank Dr. Sheldon H. Jacobson who has patiently assisted in my development both as a researcher and as a person. I also would like to thank my continuing and former committee members Dr. Ebru Bish, Dr. Russell D. Meller, Dr. Keying Ye, Dr. John E. Kobza and Dr. Joel A. Nachlas for their time and guidance. In addition, I would like to thank Dr. Marvin Nakayama of the New Jersey Institute of Technology for his guidance and suggestions on the research presented in this dissertation.

I would also like to thank the “GHC” team : Dr. Sheldon H. Jacobson, Lieutenant Colonel (Dr.) Alan W. Johnson, Dr. Kelly A. Sullivan, Dr. Diane E. Vaughan, Lieutenant Colonel (Dr.) Darrall Henderson, Derek E. Armstrong, Jon M. Bowman, Jeffrey E. Orosz, Julie E. Virta, Hemanshu Kaul and Daniel W. Cranston. I especially would like to thank Lieutenant Colonel (Dr.) Darrall Henderson and Dr. Diane E. Vaughan for their time, help and support when I needed the most. I would also like to acknowledge the National Science Foundation (DMI-9907980) and the Air Force Office of Scientific Research (F49620-98-1-0111, F49620-01-1-0007) for their support of research that was used in this dissertation.

I would like to thank the Industrial and Systems Engineering Department faculty and staff for their support of education. I especially would like to thank Ms. Lovedia Cole for her dedication to students.

I would like to thank my roommates and friends Hamza Yeşilyurt, Ümit Seyhan, Samet Seyhan, Doğan Gürsoy and Ahmet Demirci for their help and moral support through my doctoral dissertation studies. Last but not least, I would like to thank Mersin University for their financial support, which made this research possible.

Dedication

I dedicate this research in memories of my father, İbrahim Aytemiz, and my mother, Meryem Aytemiz. I also dedicate this research to my sisters Alev Akbulak, Nuran Bayır and Gamze Aytemiz.

Table of Contents

Chapter 1: Introduction and Background	1
1.1. Discrete Optimization Problems	1
1.2. Local Search Algorithms	2
1.3. Research Goals	3
Chapter 2: The SATISFIABILITY Problem.....	5
2.1. Definitions	5
2.2. Application Areas	6
2.2.1. Combinational Circuits.....	7
2.2.2. Machine Shop Scheduling.....	10
2.3. Algorithms	14
2.3.1. Splitting and Resolution	14
2.3.2. Integer Programming	16
2.3.3. Local Search Algorithms	17
Chapter 3: A Probabilistic Study of Randomly Generated	
3-SATISFIABILITY Instances.....	19
3.1. Definitions	20
3.2. Analysis for a Given Solution of 3-SATISFIABILITY	20
3.3. Analysis for the Solution Space, Ω	29
3.4. The Threshold Conjecture	36
3.5. Concluding Remarks	40
Chapter 4: Tabu Search and Random Restart Local Search Algorithms	
for MAX 3-SATISFIABILITY	41
4.1. Tabu Search and Random Restart Local Search Framework.....	41
4.1.1. Tabu Search Framework.....	41
4.1.2. Random Restart Local Search Framework	43
4.2. Algorithm Implementations for MAX 3-SATISFIABILITY	44
4.2.1. Definitions.....	44
4.2.2. Tabu List	44
4.2.3. Tabu Search Implementation.....	45
4.2.4. Random Restart Local Search Implementation	46

Chapter 5: Computational Results	48
5.1. Complete Enumeration Results.....	48
5.2. Tabu Search Results	52
5.3. Random Restart Local Search Results.....	54
5.4. Stopping Criterion for Local Search Algorithms.....	55
Chapter 6: Conclusion and Future Research	61
Appendix A	63
Appendix B	94
Appendix C	103
Appendix D	105

List of Tables

Table 2.1: Six Basic Types of Logic Gates	7
Table 2.2: Consistency Condition for the Operation i (where $d = 4$)	11
Table 3.1: Generic Form of Matrix A	30
Table 3.2: Matrix A for an Instance of 3-SATISFIABILITY	31
Table 3.3: β_l and α_l Values Compared to Dubois's Estimates for c_l^* Values.....	39
Table 5.1: Values for z^*	57
Table 5.2: Values for k^*	58
Tables A.1.1 - A.1.20: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses.....	63
Tables A.2.1 - A.2.2: Maximum Number of Satisfied Clauses	82
Table A.3: The Threshold Conjecture	84
Tables A.4.1 – A.4.3 : Geary's Test Results for Normality.....	85
Table B.1: Tabu Search versus Complete Enumeration.....	94
Tables B.2.1 - B.2.6: Tabu Search Results I.....	96
Table B.3: Tabu Search Results II	100
Tables C.1.1 - C.1.4: Random Restart Local Search Results	103

List of Figures

Figure 2.1: A Combinational Circuit.....	7
Figure 3.1: Paths on the Matrix A	35
Figure A.1: The Threshold Conjecture.....	84
Figures A.2.1 – A.2.6: Distribution of the Solution Space over the Number of Satisfied Clauses.....	91
Figures D.1 - D.4: z Values for Tabu Search	105

Chapter 1 :

Introduction and Background

1.1. Discrete Optimization Problems

Discrete optimization problems are defined by a finite set of solutions together with an objective function value assigned to each solution (Garey and Johnson 1979). The goal when addressing such problems is to find solutions that globally optimize the objective function. Unless otherwise noted, assume that discrete optimization problems are maximization problems.

Computational complexity theory provides a well-defined framework to assess the difficulty of these problems. Garey and Johnson (1979) formally defines the classes P (Polynomial), NP (Non-deterministic Polynomial) and NP -complete. These classes are defined to apply only to decision problems (i.e., problems whose solution yields either “yes” or “no” response). The class P contains decision problems that can be solved in polynomial time in the size of the problem instance. The class NP contains decision problems for which it can be checked in polynomial time (in the size of the problem instance) whether a given potential solution solves the problem. The relationship between the classes P and NP (i.e., the classic question of whether $P = NP$) is a difficult open research question in theoretical computer science. Garey and Johnson (1979) explicitly state that every decision problem contained in the class P is also contained in the class NP (i.e., $P \subseteq NP$). On the other hand, it is still unknown if this inclusion is proper (i.e., $P \neq NP$).

The main reason why many scientists believe that $P \neq NP$ is the existence of the class of NP -complete problems. A decision problem is said to be NP -complete if it belongs to the class NP and another NP -complete problem can be reduced to it in polynomial time. This means that every NP -complete problem can be reduced to at least one other NP -complete problem in polynomial time. Moreover, if any NP -complete problem can be solved in polynomial time, then every problem in the class NP can be solved in polynomial time (i.e., $P = NP$). To date, no NP -complete problems have been shown to be solvable in polynomial time. By definition, NP -complete problems are said to be the hardest problems in the class NP . The seminal paper by Cook (1971) proves the first decision problem, SATISFIABILITY, to be NP -complete. Over the past thirty years, a significant amount of research has been done in this area, with numerous problems proven to be NP -complete. In addition, special cases of NP -complete problems that can

be solved in polynomial time have been identified (e.g., see Coullard et al. 1998 for certain matching problems and Aspvall et al. 1979 for the 2-SATISFIABILITY problem). See Garey and Johnson (1979) for an in-depth discussion on the theory of *NP-completeness*, including formal definitions, proofs and an extensive list of *NP-complete* problems.

For convenience, the classes *P* and *NP-complete* restrict attention only to decision problems that belong to the class *NP*, hence exclude discrete optimization problems. On the other hand, many discrete optimization problems can also be formulated as decision problems (e.g., this can be done using objective function value bounding techniques). Therefore, the classes *P*, *NP* and *NP-complete* can be extended to categorize optimization problems. Garey and Johnson (1979) formally define the classes *NP-hard*, *NP-easy* and *NP-equivalent* for search and optimization problems. The class *NP-easy* contains search and optimization problems that can be reduced (in a polynomial time) to a decision problem in the class *NP*. A search or optimization problem is said to be *NP-hard* if an *NP-complete* problem can be reduced to it in polynomial time (in the size of the problem instance). The class *NP-equivalent* contains search and optimization problems that are both *NP-easy* and *NP-hard*.

1.2. Local Search Algorithms

Numerous techniques and algorithms have been developed to find approximate (near-optimal) solutions to *NP-complete* and *NP-hard* optimization problems using a reasonable amount of computing time (e.g., see Ceria et al. 1998 for set covering problems). It is a formidable challenge to construct an algorithm that finds near-optimal solutions for all instances of a particular problem. Some algorithms are *problem-specific*, focusing on certain characteristics of a particular problem (e.g., Hochbaum and Pathria 1997 for p-center problems, Kumar et al. 2000 for flexible assembly system design problems). Other algorithms tend to be *problem-independent*, offering a means to find reasonable solutions to a wide variety of problems. Local search algorithms such as *simulated annealing* (Eglese 1990, Fleischer 1995), *genetic algorithms* (Liepins and Hillard 1989, Muhlenbein 1997), *tabu search* (Glover and Laguna 1997), *threshold accepting* (Dueck and Scheuer 1990) and the *noising method* (Charon and Hudry 1993), each use a distinct search rule to intelligently explore the solution space with the hope of finding optimal/near-optimal solutions. Simulated annealing mimics the annealing process for crystalline solids, where a solid is cooled very slowly from an elevated temperature, with the hope of relaxing towards a low energy state. Genetic algorithms emulate the evolutionary behavior of biological systems to create subsequent generations that guide the search

towards near-optimal solutions. Tabu search uses memory to guide the search towards near-optimal solutions by dynamically managing a list of forbidden (tabu) solutions. The common theme among all these approaches is that they are designed to efficiently search for globally optimal solutions for hard discrete optimization problems.

1.3. Research Goals

Local search algorithms are designed to address a wide variety of discrete optimization problems. Therefore, implementation of such algorithms for a specific problem often requires additional decisions to be made (e.g., neighboring functions, stopping conditions). Such decisions may affect the performance of an algorithm and require knowledge on the problem being studied. In particular, knowledge of the structure of the solution space (i.e., how the solutions are distributed over the different objective function values) may suggest ways to improve the performance of an algorithm.

The purpose of this research is to explore this observation for 3-SATISFIABILITY. This research is classified into two main parts. The objective of the first part is to probabilistically study and analyze randomly generated instances of 3-SATISFIABILITY and gain insights into the structure of the solution space. The objective of the second part is to test and compare the effectiveness of tabu search and random restart local search algorithms on MAX 3-SATISFIABILITY, and to suggest strategies to define a stopping criterion for local search algorithms using the results obtained from the first part.

This research is organized as follows: Chapter 2 formally introduces SATISFIABILITY and MAX SATISFIABILITY. A summary of the application areas for SATISFIABILITY is also provided. Chapter 3 presents a probability study for randomly generated instances of 3-SATISFIABILITY. This is done by first defining two random variables $f_m(\omega)$ and $h_m(\omega)$. Using the definition of discrete optimization problems, these random variables can be seen as random functions that assign a particular objective function value to a given solution (i.e., an element from the set of solutions). The characteristics and asymptotic behavior of these random variables are analyzed under the assumption that a 3-SATISFIABILITY solution is given. This analysis is then extended to include all solutions in the solution space (i.e., the set of all possible solutions). A matrix structure and a random vector $\mathbf{S}(\mathbf{m})$ are defined to analyze the distribution of the solution space over the different objective function values. In addition, a phenomenon, termed *the threshold conjecture*, is described and analyzed using the random variables $f_m(\omega)$ and

$h_m(\omega)$. Chapter 4 presents and outlines (in pseudo-code form) tabu search and random restart local search implementations for MAX 3-SATISFIABILITY. Chapter 5 reports the results from several computational experiments executed on randomly generated instances of 3-SATISFIABILITY. These experiments are designed to illustrate the results presented in Chapter 3, as well as to test and compare the effectiveness of tabu search and random restart local search algorithms on MAX 3-SATISFIABILITY. In addition, using results in Chapter 3, several strategies are suggested to define a stopping criterion for local search algorithms. Chapter 6 contains concluding comments and directions for future research.

Chapter 2:

The SATISFIABILITY Problem

This chapter formally introduces SATISFIABILITY. Section 2.1 formally defines the SATISFIABILITY problem. Section 2.2 provides a summary of real-world applications of SATISFIABILITY. The chapter concludes with a brief summary of algorithms used to address SATISFIABILITY.

2.1. Definitions

To describe the SATISFIABILITY problem, several definitions are needed. A *clause* is a combination of *literals*, where a literal is a Boolean variable (X_i) or its negation (\bar{X}_i). The *solution space*, $\Omega = \{0, 1\}^n$, is the set of all possible solutions (i.e., $|\Omega| = 2^n$), where n is the number of Boolean variables. A solution, $\omega \in \Omega$, is a Boolean vector of size n . Given a solution, $\omega \in \Omega$, a clause is satisfied if at least one literal in this clause takes on the value one. Using these definitions, SATISFIABILITY can now be formally described.

Given an instance of SATISFIABILITY (i.e., a collection of m clauses defined by n Boolean variables), is there a solution, (i.e., a set of values for the n Boolean variables) that satisfies all m clauses?

If the answer to this instance of SATISFIABILITY is yes (no), then this instance is said to be *satisfiable* (*unsatisfiable*). Several variations of SATISFIABILITY have been studied, based on the number of literals assigned to each clause. For example, if the number of literals is two or three for all clauses, then SATISFIABILITY is referred to as 2-SATISFIABILITY or 3-SATISFIABILITY, respectively.

The following example illustrates the 3-SATISFIABILITY problem. Consider an instance of SATISFIABILITY containing 4 clauses involving 5 Boolean variables, where each clause has exactly 3 literals.

$$\text{Clause \#1: } X_1 \vee \bar{X}_2 \vee X_4$$

$$\text{Clause \#2: } X_1 \vee X_3 \vee \bar{X}_5$$

$$\text{Clause \#3: } \bar{X}_1 \vee \bar{X}_2 \vee \bar{X}_3$$

$$\text{Clause \#4: } X_2 \vee X_3 \vee X_4$$

If all Boolean variables are set to one, then all the clauses are satisfied except Clause #3. Note that all the clauses can be satisfied by setting $X_3 = 0$ and $X_1 = X_2 = X_4 = X_5 = 1$.

SATISFIABILITY is defined as a decision problem, since the solution yields either a “yes” or “no” response (i.e., all m clauses can or cannot be satisfied). Any instance of SATISFIABILITY can be formulated as an optimization problem (termed MAX SATISFIABILITY) by introducing the objective function $f = \sum_{j=1}^m C_j(\omega)$, where the goal is to maximize f over the solution space Ω ,

where

$$C_j(\omega) = \begin{cases} 1 & \text{if clause } j \text{ is satisfied for solution } \omega \\ 0 & \text{otherwise} \end{cases}.$$

SATISFIABILITY was the first problem proven to be *NP-Complete* (Cook 1971). SATISFIABILITY is also important for solving numerous real-world problems, ranging from computer chip design to robotics (e.g., see Gu et al. 1997 for an extensive list of application areas). Such applications have been the driving force behind much of the research done to address the problem. In addition, there exist two special cases of SATISFIABILITY (i.e., 2-SATISFIABILITY and Horn-SATISFIABILITY) that are solvable in polynomial time (in the size of the problem instance) (see Aspvall et al. 1979, Dowling and Gallier 1984).

2.2. Application Areas

Applications of SATISFIABILITY include several other well-known *NP-complete* problems such as vertex cover, graph k -colorability, and Hamiltonian circuit. Creignou (1993) has shown that several *NP-complete* problems can be reduced to SATISFIABILITY in polynomial time (in the size of the problem instance). SATISFIABILITY also has numerous real-world applications. One direct application of SATISFIABILITY is in the design of combinational circuits used in computers. Gu et al. (1997) lists several real-world applications of SATISFIABILITY, including image matching problems, VLSI design, optical character recognition, task planning, task scheduling, the constraint satisfaction problem, the n queens problem, and cryptography. This section describes (in greater detail) two of these problems.

2.2.1. Combinational Circuits

A combinational circuit consists of one or more combinational elements (i.e., logic gates) interconnected in an acyclic way (i.e., contains no cycles). Interconnections (i.e., wires) connect the output of a logic gate to the input of another logic gate. All inputs and outputs are in Boolean form (i.e., take on the values one or zero). Each logic gate in the circuit takes one or two inputs and computes a single output according to the Boolean function associated with its type. In particular, six basic types of logic gates are listed in Table 2.1.

Table 2.1: Six Basic Types of Logic Gates

Type	Common Notation	Associated Boolean Function ($x \in \{0, 1\}$ and $y \in \{0, 1\}$ are inputs, $z \in \{0, 1\}$ is output)
NOT	\neg	$z \Leftrightarrow \neg x$
OR	\vee	$z \Leftrightarrow (x \vee y)$
NOR (NOT-OR)	$\neg \vee$	$z \Leftrightarrow \neg(x \vee y)$
AND	\wedge	$z \Leftrightarrow (x \wedge y)$
NAND (NOT-AND)	$\neg \wedge$	$z \Leftrightarrow \neg(x \wedge y)$
XOR (Exclusive-OR)	\oplus	$z \Leftrightarrow (x \oplus y)$ where $(x \oplus y) = (x \vee y) \wedge (\neg x \vee \neg y)$

To illustrate the operation of a combinational circuit, a simple circuit with three logic gates is depicted in Figure 2.1.

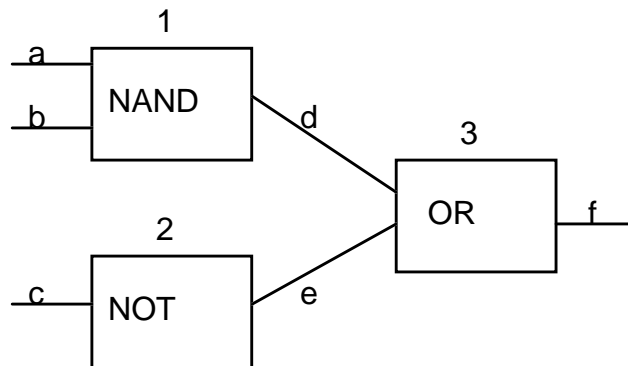


Figure 2.1: A Combinational Circuit

In Figure 2.1, logic gates are labeled by the numbers 1, 2, 3 and the wires are labeled by the letters a, b, c, d, e, f. Wires a, b and c connect the inputs from an external source (e.g., another circuit) to the circuit, while wire f connects the output of the circuit to an external source. A circuit input can be defined as a wire without connection to an output of a logic gate (e.g., wires a, b and c in Figure 2.1) while a circuit output can be defined as a wire without connection to an

input of a logic gate (e.g., the wire f in Figure 2.1). See Cormen et al. (1997) for more detailed information on combinational circuits.

The CIRCUI-T-SATISFIABILITY problem is a decision problem directly related to the combinational circuits. Using the definitions stated above, CIRCUI-T-SATISFIABILITY can be formally described.

Given a combinational circuit with a single circuit output, is there a Boolean assignment for the circuit input(s) that satisfy the circuit (i.e., the circuit output is one).

CIRCUI-T-SATISFIABILITY has numerous practical applications in computer hardware design and VLSI chip design. For example, if a circuit is unsatisfiable (i.e., no assignment for the circuit inputs yield the circuit output to be one), then the circuit output is always zero. In such cases, the entire circuit can be replaced by a single component that always outputs zero, which translates into a reduction in both the number of logic gates and in the time required for the operation of the logic gates. Unfortunately, CIRCUI-T-SATISFIABILITY is *NP-complete* (Cormen et al. 1997). Moreover, approximate (i.e., near optimal) solutions are not useful since an exact “yes” or “no” answer must be obtained to decide whether such a reduction is possible.

CIRCUI-T-SATISFIABILITY problems can be formulated as SATIFIABILITY problems. Note that each logic gate has a well-defined Boolean function associated with its type (as listed in Table 2.1). Moreover, the entire circuit can be defined by a Boolean function. To see this, let a combinatorial circuit, \mathfrak{N} , with n logic gates be given. Let B_1, B_2, \dots, B_n denote the Boolean functions associated with the logic gates. Then the Boolean function, β , for the circuit \mathfrak{N} is

$$\beta = \zeta \wedge B_1 \wedge B_2 \wedge \dots \wedge B_n$$

where ζ is the Boolean variable for the circuit output. Moreover, \mathfrak{N} is satisfiable if and only if β is satisfiable (Cormen et al. 1997, page 942). The following example illustrates how this result can be used to formulate an instance of CIRCUI-T-SATISFIABILITY as an instance of SATISFIABILITY.

Consider the combinational circuit depicted in Figure 2.1. Let the labels on the wires denote the Boolean variables $\{a, b, c, d, e, f\}$. Let B_1, B_2 , and B_3 denote the Boolean functions associated with the logic gates. Note that the Boolean variable f is the circuit output. Then the Boolean function, β , for the circuit is

$$\beta = f \wedge (d \Leftrightarrow \neg(a \wedge b)) \wedge (e \Leftrightarrow \neg c) \wedge (f \Leftrightarrow (d \vee e)).$$

The first logic gate is a NAND gate and its associated Boolean function, B_1 , is

$$d \Leftrightarrow \neg(a \wedge b).$$

Using Boolean algebra, this function can be rewritten as

$$\begin{aligned} d \Leftrightarrow \neg(a \wedge b) &= (d \Rightarrow \neg(a \wedge b)) \wedge (\neg(a \wedge b) \Rightarrow d) \\ &= (\neg d \vee \neg a \vee \neg b) \wedge ((a \wedge b) \vee d) \\ &= (\neg d \vee \neg a \vee \neg b) \wedge (a \vee d) \wedge (b \vee d). \end{aligned}$$

Boolean functions for the second and third logic gates can be obtained in a similar way. In particular,

$$e \Leftrightarrow \neg c = (e \Rightarrow \neg c) \wedge (\neg c \Rightarrow e) = (\neg e \vee \neg c) \wedge (c \vee e)$$

and

$$\begin{aligned} f \Leftrightarrow (d \vee e) &= (f \Rightarrow (d \vee e)) \wedge ((d \vee e) \Rightarrow f) \\ &= (\neg f \vee d \vee e) \wedge (f \vee \neg d) \wedge (f \vee \neg e). \end{aligned}$$

Then the Boolean function for the circuit is $\beta = f \wedge (\neg d \vee \neg a \vee \neg b) \wedge (a \vee d) \wedge (b \vee d) \wedge (\neg e \vee \neg c) \wedge (c \vee e) \wedge (\neg f \vee d \vee e) \wedge (f \vee \neg d) \wedge (f \vee \neg e)$. Note that the Boolean function β is an instance of SATISFIABILITY with nine clauses and six Boolean variables.

SATISFIABILITY can also be used in Automated Test Pattern Generation (ATPG) systems. ATPG systems are used for testing and detecting faulty circuits in computers. Larrabee (1992) presents a SATISFIABILITY model to this problem. In this approach, a wire is assumed to be faulty. Based on this assumption, a set of clauses representing this faulty circuit is extracted and added to the clauses for the non-faulty circuit. Output wires of faulty and non-faulty circuits are then connected with an XOR gate. Clauses representing this XOR gate are also added to the clauses for the faulty and non-faulty circuits. If the resulting instance of SATISFIABILITY is satisfied, then this fault is detectable; otherwise it is undetectable. Computational results have shown that this approach is more efficient than the traditional methods (e.g., Boolean difference method). The primary reason for this improvement is that approximately 80% of the clauses generated by this approach contain only two literals (i.e., an instance of 2-SATISFIABILITY). Note that there are only two clauses with more than two literals in the example depicted in Figure 2.1.

2.2.2. Machine Shop Scheduling

Scheduling problems generally consider how to schedule several tasks subject to a given set of constraints and resources. One important type of scheduling problem is the machine shop scheduling problem. This problem consists of a number of operations to be scheduled subject to a collection of constraints, where each operation requires a specified processing time. Smith and Cheng (1993) categorize the constraints as sequencing restrictions (i.e., an operation must finish before another operation starts), resource capacity constraints (i.e., two operations require the same source, hence cannot be scheduled concurrently), ready times (i.e., the earliest time at which an operation can start) and deadlines (i.e., the latest time at which an operation must be completed). The machine shop scheduling problem then asks if all operations can be completed on time, without violating any of the constraints.

Several definitions are needed to describe how to model machine shop scheduling problems as SATISFIABILITY problems. Consider a machine shop scheduling problem that requires q operations to be scheduled. Let p_i be the processing time required for operation $i = 1, 2, \dots, q$. Let d_i and r_i be the deadline and ready time of operation $i = 1, 2, \dots, q$, respectively. Moreover, let s_i denote the start time of operation $i = 1, 2, \dots, q$. Define the Boolean variable

$$X_{i,t} = \begin{cases} 1 & \text{if } s_i \geq t \\ 0 & \text{if } s_i < t \end{cases}, i = 1, 2, \dots, q \text{ and } t = 0, 1, \dots, d,$$

where t denotes discrete time points over the time line and d denotes the overall deadline (i.e., the time at which all operations must be completed).

By definition, $X_{i,t} = 1$ corresponds to “operation i starts at time t or later,” while $X_{i,t} = 0$ corresponds to “operation i starts before time t .” Note that a solution (i.e., values for the Boolean variables) is a Boolean matrix with q rows and $d + 1$ columns.

Using these definitions, the machine shop scheduling problem can be modeled as a SATISFIABILITY problem. To see this, note that the first set of clauses can be obtained from the Boolean function

$$X_{i,t} \Rightarrow X_{i,t-1} = (\neg X_{i,t} \vee X_{i,t-1}) \quad \text{for all } i = 1, 2, \dots, q \text{ and } t = 1, 2, \dots, d. \quad (2.1)$$

This Boolean function captures “if operation i starts at time t or later, then it starts at time $t - 1$ or later.” Note that index i is quantified over the values $1, 2, \dots, q$, while index t is quantified over the values $1, 2, \dots, d$. This means that a total of $d * q$ clauses are obtained from (2.1). For example, if $q = 2$ and $d = 3$, then the resulting set of six clauses is

$$\{(\neg X_{1,1} \vee X_{1,0}), (\neg X_{1,2} \vee X_{1,1}), (\neg X_{1,3} \vee X_{1,2}), (\neg X_{2,1} \vee X_{2,0}), (\neg X_{2,2} \vee X_{2,1}), (\neg X_{2,3} \vee X_{2,2})\}.$$

The set of clauses obtained from (2.1) ensures that the solutions (i.e., values for the Boolean variables) are consistent with the definition of $X_{i,t}$. To see this, consider the Boolean assignment $(\dots, X_{i,t-1} = 0, X_{i,t} = 1, \dots)$. By definition, $X_{i,t} = 1$ corresponds to “operation i starts at time t or later” while $X_{i,t-1} = 0$ corresponds to “operation i starts before time $t - 1$,” which are complementary events. Therefore, this assignment is not consistent with the definition of $X_{i,t}$ (note that $X_{i,t-1} = 0$ implies $s_i < (t - 1)$ while $X_{i,t} = 1$ implies $s_i \geq t$). The clause $(\neg X_{i,t} \vee X_{i,t-1})$ is not satisfied by this assignment, hence this assignment cannot satisfy all the clauses obtained from (2.1). Moreover, for the three other possible value assignments for the Boolean variables $X_{i,t-1}$ and $X_{i,t}$, the clause $(\neg X_{i,t} \vee X_{i,t-1})$ is satisfied and the assignments are consistent with the definition of $X_{i,t}$. Note that for a solution (i.e., a Boolean matrix) to satisfy all the clauses obtained from (2.1), a zero must never come before a one, in any row of the solution. This condition (i.e., a consistency condition) ensures that the solutions are consistent with the definition of $X_{i,t}$.

To illustrate this condition, consider a machine shop scheduling problem with $d = 4$. For operation i , Table 2.2 lists the six possible assignments for the Boolean variables $\{X_{i,0}, X_{i,1}, \dots, X_{i,4}\}$ that satisfy the clauses $\{(\neg X_{i,0} \vee X_{i,1}), (\neg X_{i,1} \vee X_{i,2}), \dots, (\neg X_{i,4} \vee X_{i,3})\}$. Note that each of these assignments correspond to a single row i of a solution (i.e., a partial solution) with $d = 4$.

Table 2.2:: Consistency Condition for the Operation i (where $d = 4$)

Assignment	$X_{i,0}$	$X_{i,1}$	$X_{i,2}$	$X_{i,3}$	$X_{i,4}$
1	1	1	1	1	1
2	1	1	1	1	0
3	1	1	1	0	0
4	1	1	0	0	0
5	1	0	0	0	0
6	0	0	0	0	0

In Table 2.2, each row corresponds to an assignment for the Boolean variables $\{X_{i,0}, X_{i,1}, \dots, X_{i,4}\}$. Moreover, each of these six assignments satisfy all the clauses involving operation i . Note that the set of clauses obtained from (2.1) are similar for each operation $i = 1, 2, \dots, q$. Therefore, if a solution (i.e., a Boolean matrix) consists of a combination of rows in Table 2.2, then this solution satisfies all the clauses obtained from (2.1). Note that if a Boolean variable takes on the value one, then all the preceding Boolean variables must also take on the value one. Similarly, if a

Boolean variable takes on the value zero, then all the succeeding Boolean variables must also take on the value zero. This pattern holds for all values of d .

The second set of clauses is obtained from the ready times of the operations using the Boolean function

$$X_{i,r_i} \quad \text{for all } i = 1, 2, \dots, q. \quad (2.2)$$

This Boolean function corresponds to “operation i starts at time r_i (i.e., ready time of operation i) or later,” which is equivalent to the definition of the ready time of an operation. If $X_{i,r_i} = 0$, then this clause is not satisfied, since it means that operation i is started prior to its ready time, which is a violation. Note that index i is quantified over all the values $1, 2, \dots, q$. Therefore, a total of q clauses are obtained from (2.2).

The third set of clauses is obtained from the deadlines. In particular

$$\neg X_{i,(d_i-p_i+1)} \quad \text{for all } i = 1, 2, \dots, q. \quad (2.3)$$

This Boolean function corresponds to “operation i starts before time $(d_i - p_i + 1)$,” which is equivalent to meeting the operation deadline. If $X_{i,(d_i-p_i+1)}$ takes on the value one, then this clause is not satisfied, hence operation i starts at time $(d_i - p_i + 1)$ or later, which is equivalent to missing the deadline. Also, note that index i is quantified over all the values $1, 2, \dots, q$. Therefore, a total of q clauses are obtained from (2.3).

The fourth set of clauses is obtained from the sequencing restrictions, such as an operation must wait until another operation is completed. Let i and j denote two such operations, hence operation j can only begin once operation i is completed. The Boolean function for this constraint is

$$X_{i,t} \Rightarrow X_{j,(t+p_i)} = (\neg X_{i,t} \vee X_{j,(t+p_i)}) \quad \text{for all } t = 0, 1, \dots, d - p_i. \quad (2.4)$$

This Boolean function corresponds to “if operation i starts at time t or later, then operation j starts at time $t + p_i$ or later.” Note that if $X_{i,t}$ takes on the value one, then this clause cannot be satisfied if $X_{j,(t+p_i)}$ takes on the value zero. This means that “operation i starts at time t or later” while “operation j starts before time $t + p_i$,” which violates this sequencing restriction since operation j would then have started before operation i is finished (note that in this case, this clause is not satisfied). If $X_{i,t}$ takes on the value zero and $X_{j,(t+p_i)}$ takes on the value one, then “operation i starts before time t ” while “operation j starts at time $t + p_i$ or later,” hence operation j starts after i finishes. This is consistent with the sequencing restriction (note that in this case, this clause is

satisfied). If both Boolean variables are assigned the same values (zero or one), then the clause is satisfied. Note that index t is quantified over all the values $0, 1, \dots, d-p_i$. This means that for this sequencing restriction, there are $d - p_i + 1$ clauses that must all be satisfied. These clauses are designed to detect whether a solution violates this sequencing restriction.

The fifth set of clauses is obtained from the resource capacity constraints. The resource capacity constraint requires two operations not to be scheduled simultaneously. Let i and j denote two such operations. The Boolean function for this constraint is

$$(X_{i,t} \Rightarrow X_{j,(t+p_i)}) \vee (X_{j,t} \Rightarrow X_{i,(t+p_j)}) = (\neg X_{i,t} \vee \neg X_{j,t} \vee X_{j,(t+p_i)} \vee X_{i,(t+p_j)}) \quad (2.5)$$

for all $t = 0, 1, \dots, d$.

Note that this clause has 4 literals. Moreover, this constraint is written as two sequencing constraints connected with a \vee , which corresponds to “operation i starts after operation j is finished or operation j starts after operation i is finished;” this is equivalent to the resource capacity constraint. Note that the only way to violate this constraint is to violate both the sequencing constraints, which can occur only when $X_{i,t}$ and $X_{j,t}$ both take on the value one and $X_{j,(t+p_i)}$ and $X_{i,(t+p_j)}$ both take on the value zero. This corresponds to “start operation i at time t or later” while “start operation j before time $t + p_i$,” which violates the first sequencing restriction since operation j starts before i is finished. In a similar way, the second sequencing restriction is also violated. Therefore, the resource capacity constraint is violated, and the clause is unsatisfied.

The clauses obtained from (2.1)–(2.5) uniquely model a machine shop scheduling problem. Note that when a clause is unsatisfied, then there is a violation of constraints in the problem. Moreover, when a solution violates any of the constraints, then a clause is unsatisfied. Therefore, there is a schedule for the machine shop scheduling problem (without violating any scheduling constraints) if and only if there exist a solution that satisfy all the clauses in the associated SATISFIABILITY problem.

Assume that there exists a solution for this instance of SATISFIABILITY and that this solution (i.e., a Boolean matrix) is given. Note that each row in the Boolean matrix corresponds to a distinct operation, while each column corresponds to a discrete time point over the time line. A start time for operation i can be obtained by taking corresponding row i in the Boolean matrix and finding the latest time at which this solution takes on the value one. Note that this time point corresponds to the latest time at which operation i can start (by the definition of $X_{i,t}$).

Numerous other applications of SATISFIABILITY can be found in the literature. Channel assignment in cellular mobile networks (Rushforth and Wang 1997), channel routing (Devadas 1989), asynchronous circuit design (Gu 1997), optical character recognition (Gu 1997), automated test pattern generation (ATPG) systems (Larrabee 1992) and planning (Kautz and Selman 1992) represent some such applications, and have been the driving force behind much of the research done to address the problem.

2.3. Algorithms

This section briefly discusses algorithms for SATISFIABILITY. These algorithms can be described as either *complete* (i.e., they are able to find a “yes” or “no” answer to the problem) or *incomplete* (i.e., they sometimes find a “yes” answer, but they cannot find a “no” answer to the problem). Complete algorithms typically use *splitting* or *resolution* techniques (described in Section 2.3.1). Incomplete algorithms typically use integer programming or local search techniques (described in Section 2.3.2 and Section 2.3.3, respectively).

2.3.1. Splitting and Resolution

Several algorithms for SATISFIABILITY are based on the recursive elimination of Boolean variables (one variable at a time). *Splitting* and *resolution* are the two techniques used in this elimination process.

Davis et al. (1962) introduced splitting techniques, where a Boolean variable is selected from the given problem and the original problem is replaced by a sub-problem for each of two possible value assignments to this variable. Sub-problems have all the clauses from the original problem except the clauses that are satisfied by this assignment and all the literals from the original problem except the literals that are zero by the assignment. The same process is repeated for the other assignment. This technique recursively deletes the satisfied clauses and zero valued literals from the problem. Note that neither sub-problem includes the selected Boolean variable and the original problem is satisfiable if and only if either sub-problem is satisfiable. This process is repeated until a sub-problem that includes no clauses (i.e., terminate with the problem deemed satisfiable) or an empty clause is generated (i.e., the problem is not satisfiable). Note that at each step, a Boolean variable is selected and eliminated from the problem. An effective rule for selecting a Boolean variable may greatly increase the performance of splitting techniques. One of the most commonly used rules is the pure literal rule, which gives a priority to the Boolean

variables that appear only negated (or unnegated) in the problem instance. Several other rules have been defined for splitting technique (e.g., see Purdom 1983, Buro and Buning 1993, Bitner and Reingold 1975).

Davis and Putnam (1960) introduced resolution techniques, where a Boolean variable is selected and the original problem is replaced by another problem (i.e., a resolvent) that does not include the selected Boolean variable. To see how a resolvent can be obtained, consider a set of m clauses involving n Boolean variables. According to a predefined rule, select a Boolean variable X_i among these n Boolean variables. Find all the clauses that include the literal X_i and delete this literal from all of these clauses. Let $A = \{A_1, A_2, \dots, A_k\}$ be the set of k ($\leq m$) clauses resulted from this process. Similarly, find all the clauses that include the literal \bar{X}_i and delete this literal from all of these clauses. Let $B = \{B_1, B_2, \dots, B_r\}$ be the set of r ($\leq m$) clauses resulted from this process. Moreover, let $R = \{R_1, R_2, \dots, R_{(m-k-r)}\}$ be the set of $(m - k - r)$ clauses that include neither of these two literals. Using these definitions, the *resolvent* of the original problem is obtained from the Boolean function

$$\beta = \left(\bigwedge_{i=1}^k \bigwedge_{j=1}^r (A_i \vee B_j) \right) \bigwedge_{l=1}^{m-r-k} R_l$$

Note that the resolvent does not contain the Boolean variable X_i , since none of the clauses in the sets A , B and R include X_i . Moreover, the original problem is not satisfiable if and only if the resolvent is not satisfiable (see Davis and Putnam 1960 for a proof of this result). Note that the resolvent has a total of $(k * r) + m - k - r$ clauses. Therefore, in most cases, the resolvent will have more clauses than the original problem. However, if $(k * r)$ is small compared to $(r + k)$, then the resolvent will have fewer clauses than the original problem.

As noted earlier, in resolution, a Boolean variable is selected and the original problem is replaced by another problem (i.e., a resolvent) that does not include this Boolean variable. This process is repeated until either an empty clause is generated (i.e., the problem is not satisfiable) or a resolvent with no clauses is obtained (i.e., the problem is satisfiable). Similar to the splitting technique, an effective rule for selecting a Boolean variable is also important for the resolution technique. The pure literal rule that is used in the splitting technique can also be used in the resolution technique. Note that if a Boolean variable X_i appears only as a negated (or unnegated) literal in the problem instance, then selecting this Boolean variable results in a resolvent involving only $(m - k - r)$ clauses, since either k or r is zero in this case. Several algorithms have

been developed based on the resolution technique to address SATISFIABILITY (see Galil 1977, Tseitin 1968, Haken 1985 for examples of such algorithms).

Splitting and resolution technique are the earliest and most commonly used algorithms for SATISFIABILITY. Note that such algorithms treat SATISFIABILITY as a decision problem, hence terminate with either a “yes” or a “no” answer. Moreover, such algorithms may require exponential time (in the size of the problem instance) to solve the problem. In particular, Mitchell et al. (1992) and Cook and Mitchell (1997) note that in the worst case, splitting and resolution algorithms execute in exponential time to solve 3-SATISFIABILITY for $3 < (m/n) < 6$. On the other hand, these algorithms are usually the only alternatives available when optimal solutions are needed.

2.3.2. Integer Programming

Integer programming is another approach that has been used to address the SATISFIABILITY problem. An integer programming approach requires the clauses and Boolean variables to be transformed into linear inequalities. Joy et al. (1997) provides a simple integer programming formulation of SATISFIABILITY involving n Boolean variables and m clauses.

$$\begin{aligned} \min f &= \sum_{j=1}^m W_j \\ \text{subject to} \quad & \left(\sum_{i=1}^n Q_{i,j} \right) + W_j \geq 1 \quad j = 1, 2, \dots, m \\ & X_i \in \{0, 1\}, W_j \in \{0, 1\}, i = 1, 2, \dots, n, j = 1, 2, \dots, m. \end{aligned}$$

with

$$Q_{i,j} = \begin{cases} X_i & \text{if literal } X_i \text{ is in clause } C_j \\ 1 - X_i & \text{if literal } \bar{X}_i \text{ is in clause } C_j \\ 0 & \text{if neither } X_i \text{ nor } \bar{X}_i \text{ is in } C_j \end{cases}.$$

For example, the instance of SATISFIABILITY

$$(X_1 \vee \bar{X}_2) \wedge (X_1 \vee X_2 \vee \bar{X}_3) \wedge (X_1 \vee \bar{X}_3)$$

can be formulated as

$$\begin{aligned}
& \min f = W_1 + W_2 + W_3 \\
& \text{s.t.} \quad X_1 + (1-X_2) + W_1 \geq 1 \\
& \quad \quad X_1 + X_2 + (1-X_3) + W_2 \geq 1 \\
& \quad \quad X_1 + (1-X_3) + W_3 \geq 1 \\
& \quad \quad X_i \in \{0, 1\}, W_j \in \{0, 1\}, i = 1, 2, \dots, n, j = 1, 2, \dots, m.
\end{aligned}$$

Note that for a solution $\omega \in \Omega$ (i.e., values for the Boolean variables $\{X_1, X_2, \dots, X_n\}$), W_j , $j = 1, 2, \dots, m$, takes on the value one (zero) if the clause C_j is satisfied (not satisfied). Therefore, the objective function, $f: \Omega \rightarrow [0, m]$, denotes the number of unsatisfied clauses for the solution $\omega \in \Omega$.

The resulting problem can then be solved using integer programming techniques such as branch and bound (Blair et al. 1986), branch and cut (Joy et al. 1997), or cutting planes (Hooker 1988). Note that this integer programming approach treats SATISFIABILITY as a discrete optimization problem (i.e., MAX SATISFIABILITY) and searches for a solution that minimizes the number of unsatisfied clauses (or equivalently, maximizes the number of satisfied clauses.)

2.3.3. Local Search Algorithms

Local search algorithms use local information to intelligently explore the solution space with the hope of finding optimal/near-optimal solutions. Local search algorithms offer an effective means to address a wide variety of hard discrete optimization problems. Several definitions are needed to describe local search algorithms for MAX SATISFIABILITY. Define n to be the number of Boolean variables and m to be the number of clauses for a given instance of MAX SATISFIABILITY. By the definition of SATISFIABILITY, the solution space Ω is the set of all possible (2^n) solutions, where every solution $\omega \in \Omega$ is represented as an n -tuple Boolean vector. Define an *objective function* $f: \Omega \rightarrow [0, m]$, such that $f(\omega)$ is the number of satisfied clauses associated with solution $\omega \in \Omega$. An important component of local search algorithms is the *neighborhood function* $\eta: \Omega \rightarrow 2^\Omega$, where $\eta(\omega) \subseteq \Omega$ for all $\omega \in \Omega$. At each iteration of a local search algorithm, a solution is generated from the set of neighbors by a *neighborhood probability generating function* $g: \Omega \times \Omega \rightarrow [0, 1]$, where $g(\omega, \omega') = 0$ for all $\omega' \notin \eta(\omega)$ and $\sum_{\omega' \in \eta(\omega)} g(\omega, \omega') = 1$ for all $\omega \in \Omega$. For example, neighbors are said to be generated uniformly at each iteration of a local search algorithm execution if for all $\omega \in \Omega$, with $\omega' \in \eta(\omega)$,

$$g(\omega, \omega') = P\{\omega' \text{ is selected as the neighbor of } \omega \text{ at an iteration of a local search algorithm}\}$$

$$= 1 / |\eta(\omega)|.$$

Various neighborhood functions can be defined for SATISFIABILITY and MAX SATISFIABILITY. One of the most commonly used neighborhood functions is to flip the value of a single Boolean variable X_i from zero (one) to one (zero) in the current solution ω_k . The Boolean variable X_i , whose value is flipped, may be selected randomly among all n Boolean variables. A generalization of this neighborhood function is to allow more than one Boolean variable to be flipped.

Tabu search (Mazure et al. 1995, Jaumard et al. 1996), simulated annealing (Spears 1996, Hansen and Jaumard 1990) and several variations of deterministic local search (Selman et al. 1992, Resende and Feo 1996, Selman et al. 1996, Gu 1993) are the most widely studied local search algorithms for SATISFIABILITY and MAX SATISFIABILITY. By design, local search algorithms are incomplete. On the other hand, they often perform much better than complete algorithms (such as resolution) in finding a satisfying solution (if one exists) to SATISFIABILITY. An extensive discussion of local search algorithms for SATISFIABILITY can be found in Gu et al. (1997). Tabu search and random restart local search algorithms are also discussed in more detail in Chapter 4.

Chapter 3:

A Probabilistic Study of Randomly Generated 3-SATISFIABILITY Instances

This chapter studies and analyzes properties of randomly generated instances of 3-SATISFIABILITY. The literature on the probabilistic analysis of 3-SATISFIABILITY focuses mainly on obtaining the probability that a specific algorithm finds a “yes” or “no” answer in polynomial time (in the size of the problem instance) to randomly generated instances of 3-SATISFIABILITY (Gu et al. 1997). Algorithms that use splitting or resolution techniques (as described in Chapter 2) are most commonly used in such analyses. Moreover, these analyses provide a means to evaluate the probabilistic performance of these algorithms on instances of 3-SATISFIABILITY, hence can be used to identify those instances of 3-SATISFIABILITY that can be solved in polynomial time. In particular, Mitchell et al. (1992) and Cook and Mitchell (1997) observe that some splitting or resolution algorithms almost always find a “yes” answer (in polynomial time) to randomly generated instances of 3-SATISFIABILITY with $(m/n) \ll 3$. Similarly, some splitting and resolution algorithms almost always find a “no” answer, in polynomial time, to randomly generated instances of 3-SATISFIABILITY with $(m/n) \gg 6$. This means that randomly generated instances of 3-SATISFIABILITY with $(m/n) \ll 3$ are satisfiable (i.e., all m clauses can be satisfied) with probability approaching one as $(m/n) \rightarrow 0$, while randomly generated instances of 3-SATISFIABILITY with $(m/n) \gg 6$ are unsatisfiable (i.e., all m clauses cannot be satisfied) with probability approaching one as $(m/n) \rightarrow +\infty$.

Such studies treat SATISFIABILITY as a decision problem. Moreover, they focus on the probabilistic performance of a specific algorithm, rather than specific characteristics of the SATISFIABILITY problem. This research takes a different approach by studying randomly generated instances of 3-SATISFIABILITY without reference to any one particular algorithm.

3.1. Definitions

Several definitions are needed to define the probability space that will be used in this analysis. Define Ψ to be the set of all possible clauses that can be defined from a set of n Boolean variables. By the definition of 3-SATISFIABILITY, the set Ψ can be constructed as follows:

- 1) List all possible combinations of three Boolean variables $\{X_a, X_b, X_c\}$ taken from the set of n Boolean variables $\{X_1, X_2, \dots, X_n\}$ without replacement (i.e., $a, b, c = 1, 2, \dots, n, a \neq b \neq c$). There exist $\binom{n}{3}$ such combinations.
- 2) For each combination, list all possible clauses involving these three Boolean variables $\{X_a, X_b, X_c\}$. There exist eight such clauses. Therefore, there are $8 \binom{n}{3}$ possible clauses that can be defined from n Boolean variables (i.e., $|\Psi| = 8 \binom{n}{3}$).

Label the clauses in Ψ as τ_r , $r = 1, 2, \dots, |\Psi|$ (i.e., $\Psi = \{\tau_1, \tau_2, \dots, \tau_{|\Psi|}\}$). Consider a random experiment of selecting a clause from the set Ψ . Let A_r be the event that the clause τ_r is selected from Ψ . Define the probability measure P_Ψ on a σ -field \mathfrak{S} in Ψ to be uniform (i.e., $P_\Psi\{A_r\} = 1 / |\Psi|$ for all $r = 1, 2, \dots, |\Psi|$). By this definition, each clause in the set Ψ is equally likely to be selected.

Moreover, define Ω to be the set of all possible solutions (i.e., $|\Omega| = 2^n$) to an instance of 3-SATISFIABILITY involving n Boolean variables. Label the elements of Ω as ω_t , $t = 1, 2, \dots, 2^n$. Consider a random experiment of selecting a single solution from the set Ω . Let B_t be the event that the solution ω_t is selected from Ω . Define the probability measure P_Ω on a σ -field \mathfrak{N} in Ω to be uniform (i.e., $P_\Omega\{B_t\} = 1 / 2^n$ for all $t = 1, 2, \dots, 2^n$). By this definition, each solution in the set Ω is equally likely to be selected.

3.2. Analysis for a Given Solution of 3-SATISFIABILITY

In this section, two random variables, $f_m(\omega)$ and $h_m(\omega)$, are defined for randomly generated instances of 3-SATISFIABILITY. An instance of 3-SATISFIABILITY containing m clauses can be randomly generated by selecting a clause from the set Ψ and repeating this procedure m times. The selection process is carried out either with or without replacement, depending on whether identical clauses are allowed in the problem (i.e., a clause can or can not be selected more than

once). This is the distinguishing factor between the random variables $f_m(\omega)$ and $h_m(\omega)$. As will be seen later in this section, for a randomly generated instance of 3-SATISFIABILITY containing m clauses selected with (or without) replacement, the random variables $f_m(\omega)$ and $h_m(\omega)$ denote the number of satisfied clauses for a given solution $\omega \in \Omega$. Note that these random variables are defined for a given solution $\omega \in \Omega$, hence are conditional random variables on ω .

Theorem 3.1 shows that $f_m(\omega)$ is a binomial random variable with parameters m and $p = 7/8$, while Theorem 3.2 shows that $h_m(\omega)$ is a hypergeometric random variable with parameters $|\Psi|$, m and $7|\Psi|/8$. Theorem 3.3 provides a sufficient condition for $\frac{P\{h_m(\omega) = k\}}{P\{f_m(\omega) = k\}} \rightarrow 1$, $k = 0, 1, \dots, m$, as $m \rightarrow +\infty$ and $n \rightarrow +\infty$. Lastly, Theorem 3.4 proves a Central Limit Theorem for both random variables.

Lemma 3.1 defines a random variable $C_1(\omega)$ and shows that it is a Bernoulli random variable with parameter $p = 7/8$. Note that this random variable is the random variables $h_m(\omega)$ and $f_m(\omega)$ with $m = 1$.

Lemma 3.1: Consider a randomly selected clause $C_1 \in \Psi$ and a given solution $\omega \in \Omega$. Define the random variable

$$C_1(\omega) = \begin{cases} 1 & \text{if } C_1 \text{ is satisfied for solution } \omega \\ 0 & \text{otherwise} \end{cases}.$$

Then $C_1(\omega)$ is a Bernoulli random variable with $p = 7/8$ (i.e., $P\{C_1(\omega)=1\} = 7/8$).

Proof : For any given solution $\omega \in \Omega$, there are $7|\Psi|/8$ clauses in Ψ that are satisfied. To see this, for a given solution $\omega \in \Omega$, consider a combination of three Boolean variables $\{X_a, X_b, X_c\}$ taken from n Boolean variables $\{X_1, X_2, \dots, X_n\}$ without replacement (i.e., $a, b, c = 1, 2, \dots, n$, $a \neq b \neq c$). There are eight possible clauses involving these three Boolean variables $\{X_a, X_b, X_c\}$:

$$\begin{aligned} & \{(X_a \vee X_b \vee X_c), (X_a \vee X_b \vee \bar{X}_c), (X_a \vee \bar{X}_b \vee X_c), (X_a \vee \bar{X}_b \vee \bar{X}_c), \\ & (\bar{X}_a \vee X_b \vee X_c), (\bar{X}_a \vee X_b \vee \bar{X}_c), (\bar{X}_a \vee \bar{X}_b \vee X_c), (\bar{X}_a \vee \bar{X}_b \vee \bar{X}_c)\}. \end{aligned}$$

For solution $\omega \in \Omega$, each Boolean variable is assigned a value of one or zero. For the clauses listed above, only the values of the Boolean variables $\{X_a, X_b, X_c\}$ are important (i.e., assigned values of the Boolean variables other than $\{X_a, X_b, X_c\}$ do not have any impact on the satisfiability of these clauses). Therefore, there are eight different value assignments for the Boolean variables $\{X_a, X_b, X_c\}$:

$$\{(X_a = 1, X_b = 1, X_c = 1), (X_a = 1, X_b = 1, X_c = 0), (X_a = 1, X_b = 0, X_c = 1),$$

$$(X_a = 1, X_b = 0, X_c = 0), (X_a = 0, X_b = 1, X_c = 1), (X_a = 0, X_b = 1, X_c = 0) \\ (X_a = 0, X_b = 0, X_c = 1), (X_a = 0, X_b = 0, X_c = 0) \}.$$

It can be seen that for any of these value assignments, seven of the clauses defined from the Boolean variables $\{X_a, X_b, X_c\}$ are satisfied, while one is not satisfied (e.g., if $(X_a = 1, X_b = 1, X_c = 0)$, then the clause $(\bar{X}_a \vee \bar{X}_b \vee X_c)$ is not satisfied). Since the Boolean variables $\{X_a, X_b, X_c\}$ are arbitrary, then this is true for all $\binom{n}{3}$ Boolean variable combinations.

Therefore, there are $7 \binom{n}{3} = 7 |\Psi| / 8$ clauses in Ψ that are satisfied for solution $\omega \in \Omega$. Since the clause C_1 is randomly selected from Ψ (i.e., $C_1 \in \Psi$) and each clause in Ψ is equally likely to be selected (i.e., $P_\Psi\{A_r\} = 1 / |\Psi|$ for all $r = 1, 2, \dots, |\Psi|\}$), then

$$P\{C_1(\omega) = 1\} = \frac{1}{|\Psi|} \left(\frac{7|\Psi|}{8} \right) = 7/8 \quad \text{and} \quad P\{C_1(\omega) = 0\} = 1 - 7/8 = 1/8. \quad \square$$

Theorem 3.1 formally defines the random variable $f_m(\omega)$ and shows that it is a binomial random variable with parameters m and $p=7/8$.

Theorem 3.1: Consider m randomly selected clauses $C_1, C_2, \dots, C_m \in \Psi$ with replacement (i.e., a clause can be selected more than once) and a given solution $\omega \in \Omega$. Define $f_m(\omega) = \sum_{j=1}^m C_j(\omega)$

for $m \geq 1$, where $C_j(\omega) = \begin{cases} 1 & \text{if } C_j \text{ is satisfied for solution } \omega \\ 0 & \text{otherwise} \end{cases}$. Then $f_m(\omega)$ is a binomial random

variable with parameters m and $p = 7/8$.

Proof: The proof is by induction on m . Let $m = 1$. Then from Lemma 3.1,

$$P\{C_1(\omega) = 1\} = 7/8 = \binom{1}{1} (7/8)^1 (1/8)^0 = P\{f_1(\omega) = 1\}$$

and

$$P\{C_1(\omega) = 0\} = 1/8 = \binom{1}{0} (7/8)^0 (1/8)^1 = P\{f_1(\omega) = 0\}.$$

Therefore, $f_1(\omega)$ is a binomial random variable with parameters $m = 1$ and $p = 7/8$.

Suppose that $f_m(\omega)$ is a binomial random variable with parameters m and $p = 7/8$. Then

$$P\{f_m(\omega) = k\} = \binom{m}{k} (7/8)^k (1/8)^{m-k}, \quad k = 0, 1, \dots, m.$$

To show that $f_{m+1}(\omega)$ is a binomial variable with parameters $m + 1$ and $p = 7 / 8$, by conditioning on $f_m(\omega)$,

$$\begin{aligned} P\{f_{m+1}(\omega) = k\} &= \sum_{j=0}^m P\{f_{m+1}(\omega) = k \mid f_m(\omega) = j\}P\{f_m(\omega) = j\} \\ &= P\{f_{m+1}(\omega) = k \mid f_m(\omega) = k - 1\}P\{f_m(\omega) = k - 1\} \\ &\quad + P\{f_{m+1}(\omega) = k \mid f_m(\omega) = k\}P\{f_m(\omega) = k\} \end{aligned}$$

Since $f_{m+1}(\omega) = \sum_{j=1}^{m+1} C_j(\omega) = \left(\sum_{j=1}^m C_j(\omega) \right) + C_{m+1}(\omega) = f_m(\omega) + C_{m+1}(\omega)$, then

$$\begin{aligned} P\{f_{m+1}(\omega) = k\} &= P\{C_{m+1}(\omega) = 1 \mid f_m(\omega) = k - 1\}P\{f_m(\omega) = k - 1\} \\ &\quad + P\{C_{m+1}(\omega) = 0 \mid f_m(\omega) = k\}P\{f_m(\omega) = k\}. \end{aligned} \tag{3.1}$$

Moreover, since the clauses are randomly selected from Ψ with replacement, then each C_j , $j = 1, 2, \dots, m + 1$, is randomly selected from the same set, Ψ , hence from Lemma 3.1, for each C_j , $j = 1, 2, \dots, m + 1$, given $\omega \in \Omega$, the probability of selecting a clause from the set Ψ such that $C_j(\omega) = 1$ or $C_j(\omega) = 0$ is $7 / 8$ or $1 / 8$, respectively, (i.e. $P\{C_j(\omega) = 1\} = 7 / 8$ and $P\{C_j(\omega) = 0\} = 1 / 8$ for all $j = 1, 2, \dots, m + 1$), independent of the previously selected clauses. Therefore,

$$P\{C_{m+1}(\omega) = 1 \mid f_m(\omega) = k - 1\} = P\{C_{m+1}(\omega) = 1\} = \mathbf{7/8}$$

and

$$P\{C_{m+1}(\omega) = 0 \mid f_m(\omega) = k\} = P\{C_{m+1}(\omega) = 0\} = \mathbf{1/8}.$$

Substituting these probabilities into (3.1) leads to

$$\begin{aligned} P\{f_{m+1}(\omega) = k\} &= \frac{7}{8} P\{f_m(\omega) = k - 1\} + \frac{1}{8} P\{f_m(\omega) = k\} \\ &= \frac{7}{8} \binom{m}{k-1} \left(\frac{7}{8}\right)^{k-1} \left(\frac{1}{8}\right)^{m-k+1} + \frac{1}{8} \binom{m}{k} \left(\frac{7}{8}\right)^k \left(\frac{1}{8}\right)^{m-k} \\ &= \left[\binom{m}{k-1} + \binom{m}{k} \right] \left(\frac{7}{8}\right)^k \left(\frac{1}{8}\right)^{m-k+1} \\ &= \binom{m+1}{k} \left(\frac{7}{8}\right)^k \left(\frac{1}{8}\right)^{m+1-k}, \quad k = 0, 1, 2, \dots, m + 1. \end{aligned}$$

Therefore, $f_{m+1}(\omega)$ is a binomial random variable with parameters $m + 1$ and $p = 7 / 8$. \square

Theorem 3.2 formally defines the random variable $h_m(\omega)$ and shows that it is a hypergeometric random variable.

Theorem 3.2: Consider m randomly selected clauses $C_1, C_2, \dots, C_m \in \Psi$ without replacement (i.e., a clause can not be selected more than once) and a given solution $\omega \in \Omega$. Define $y = \binom{n}{3} =$

$|\Psi|/8$ and $h_m(\omega) = \sum_{j=1}^m C_j(\omega)$, $1 \leq m \leq 8y$, where $C_j(\omega)$ is defined as in Theorem 3.1. Then

$h_m(\omega)$ is a hypergeometric random variable with parameters $8y$, m and $7y$.

Proof: The proof is by induction on m .

Let $m = 1$. Then from Lemma 3.1

$$P\{C_1(\omega) = 1\} = 7/8 = \frac{7y}{8y} = \frac{\binom{7y}{1} \binom{y}{0}}{\binom{8y}{1}} = P\{h_1(\omega) = 1\}$$

and

$$P\{C_1(\omega) = 0\} = 1/8 = \frac{y}{8y} = \frac{\binom{7y}{0} \binom{y}{1}}{\binom{8y}{1}} = P\{h_1(\omega) = 0\}.$$

Therefore, $h_1(\omega)$ is a hypergeometric random variable with parameters $8y$, $m = 1$ and $7y$.

Suppose that $h_m(\omega)$ is a hypergeometric random variable with parameters $8y$, m and $7y$. Then

$$P\{h_m(\omega) = k\} = \frac{\binom{7y}{k} \binom{y}{m-k}}{\binom{8y}{m}}, \quad k = 0, 1, \dots, m \text{ and } 1 \leq m \leq 8y.$$

To show that $h_{m+1}(\omega)$ is a hypergeometric random variable with parameters $8y$, $m + 1$ and $7y$, by conditioning on $h_m(\omega)$,

$$\begin{aligned} P\{h_{m+1}(\omega) = k\} &= \sum_{j=0}^m P\{h_{m+1}(\omega) = k \mid h_m(\omega) = j\} P\{h_m(\omega) = j\} \\ &= P\{h_{m+1}(\omega) = k \mid h_m(\omega) = k-1\} P\{h_m(\omega) = k-1\} \\ &\quad + P\{h_{m+1}(\omega) = k \mid h_m(\omega) = k\} P\{h_m(\omega) = k\} \end{aligned}$$

Since $h_{m+1}(\omega) = \sum_{j=1}^{m+1} C_j(\omega) = \left(\sum_{j=1}^m C_j(\omega) \right) + C_{m+1}(\omega) = h_m(\omega) + C_{m+1}(\omega)$, then

$$\begin{aligned} P\{h_{m+1}(\omega) = k\} &= P\{C_{m+1}(\omega) = 1 \mid h_m(\omega) = k-1\} P\{h_m(\omega) = k-1\} \\ &\quad + P\{C_{m+1}(\omega) = 0 \mid h_m(\omega) = k\} P\{h_m(\omega) = k\}. \end{aligned} \tag{3.2}$$

Moreover, since the clauses are randomly selected from Ψ without replacement, then the clause C_j cannot be selected from any of the previously selected clauses $C_1, C_2, \dots, C_{j-1} \in \Psi$. Therefore, let Ψ_j be the set of all possible clauses from which the clause C_j can be selected (i.e., $\Psi_1 = \Psi$, $\Psi_j = \Psi_{j-1} \setminus \{C_{j-1}\}$ for all $j \geq 2$ where $|\Psi_j| = |\Psi| - j + 1 = 8y - j + 1$). By the induction assumption, $h_m(\omega) = k$ is given and the clause C_{m+1} is to be selected from Ψ_{m+1} . By the definition of Ψ_{m+1} , there are $|\Psi| - m = 8y - m$ possible clauses in Ψ_{m+1} . From Lemma 3.1, given $\omega \in \Omega$, there are $7y$ clauses in Ψ_1 that are satisfied. Note that $h_m(\omega) = k$ means that k of these clauses have already been selected. Therefore, there are $7y - k$ clauses left in Ψ_{m+1} that are satisfied for solution ω (i.e., $P\{C_{m+1}(\omega) = 1 \mid h_m(\omega) = k\} = (7y - k) / (8y - m)$). Similarly, there are $y + k - m$ clauses left in Ψ_{m+1} that are not satisfied for solution ω (i.e., $P\{C_{m+1}(\omega) = 0 \mid h_m(\omega) = k\} = (y + k - m) / (8y - m)$). Therefore,

$$P\{C_{m+1}(\omega) = 1 \mid h_m(\omega) = k - 1\} = \frac{7y - k + 1}{8y - m}$$

and

$$P\{C_{m+1}(\omega) = 0 \mid h_m(\omega) = k\} = \frac{y + k - m}{8y - m}.$$

Substituting these probabilities into (3.2) leads to

$$\begin{aligned} P\{h_{m+1}(\omega) = k\} &= \frac{7y - k + 1}{8y - m} P\{h_m(\omega) = k - 1\} + \frac{y + k - m}{8y - m} P\{h_m(\omega) = k\} \\ &= \frac{(7y - k + 1) \binom{7y}{k-1} \binom{y}{m-k+1}}{(8y - m) \binom{8y}{m}} + \frac{(y + k - m) \binom{7y}{k} \binom{y}{m-k}}{(8y - m) \binom{8y}{m}} \\ &= \frac{(7y - k + 1)(7y)!y!}{(k-1)!(7y-k+1)!(m-k+1)!(y+k-m-1)!} + \frac{(y+k-m)(7y)!y!}{k!(7y-k)!(m-k)!(y+k-m)!} \\ &\quad \frac{(8y-m)(8y)!}{m!(8y-m)!} \\ &= \frac{(7y)!y!}{(k-1)!(7y-k)!(m-k+1)!(y+k-m-1)!} + \frac{(7y)!y!}{k!(7y-k)!(m-k)!(y+k-m-1)!} \\ &\quad \frac{(8y)!}{m!(8y-m-1)!} \end{aligned}$$

$$\begin{aligned}
&= \frac{\frac{k(7y)!y!}{k!(7y-k)!(m-k+1)!(y+k-m-1)!} + \frac{(m-k+1)(7y)!y!}{k!(7y-k)!(m-k+1)!(y+k-m-1)!}}{\frac{(m+1)(8y)!}{(m+1)!(8y-m-1)!}} \\
&= \frac{\frac{(7y)!y!}{k!(7y-k)!(m-k+1)!(y+k-m-1)!} \left(\frac{k+m-k+1}{m+1} \right)}{\frac{(8y)!}{(m+1)!(8y-m-1)!}} \\
&= \frac{\binom{7y}{k} \binom{y}{m+1-k}}{\binom{8y}{m+1}}, \quad k=0,1,2,\dots,m+1 \text{ and } m+1 \leq 8y.
\end{aligned}$$

Therefore, $h_{m+1}(\omega)$ is a hypergeometric random variable with parameters $8y$, $m+1$ and $7y$. \square

Corollary 3.1, which follows from Theorem 3.1 and Theorem 3.2, provides a lower bound and an upper bound for the ratio $\frac{P\{h_m(\omega) = k\}}{P\{f_m(\omega) = k\}}$. These bounds are used in Theorem 3.3 to show that

$$\frac{P\{h_m(\omega) = k\}}{P\{f_m(\omega) = k\}} \rightarrow 1 \text{ for } k=0,1,\dots,m \text{ as } m \rightarrow +\infty, n \rightarrow +\infty \text{ and } (m/y) \rightarrow 0.$$

Corollary 3.1: For the random variables $f_m(\omega)$ and $h_m(\omega)$, let $\Theta(k,m,y) = \frac{P\{h_m(\omega) = k\}}{P\{f_m(\omega) = k\}}$. Then

$$0 < \left(1 - \frac{m-1}{8y}\right)^m \left(1 - \frac{k-1}{7y}\right)^k \left(1 - \frac{m-k-1}{y}\right)^{m-k} \leq \Theta(k,m,y) \leq 1 \text{ for all } 1 \leq m \leq 8y \text{ and all } 0 \leq k \leq \min\{m,7y\}.$$

Proof : Consider any $k=0,1,2,\dots,\min\{m,7y\}$ with $1 \leq m \leq 8y$. Since, $P\{f_m(\omega) = k\} > 0$ and $P\{h_m(\omega) = k\} > 0$, then $\Theta(k,m,y) > 0$. The expression for $P\{h_m(\omega) = k\}$ can be rewritten as

$$\begin{aligned}
P\{h_m(\omega) = k\} &= \frac{\binom{7y}{k} \binom{y}{m-k}}{\binom{8y}{m}} = \frac{\frac{(7y)!y!}{k!(7y-k)!(m-k)!(y-m+k)!}}{\frac{(8y)!}{m!(8y-m)!}} = \binom{m}{k} \frac{\prod_{j=0}^{k-1} 7y(1-\frac{j}{7y}) \prod_{j=0}^{m-k-1} y(1-\frac{j}{y})}{\prod_{j=0}^{m-1} 8y(1-\frac{j}{8y})} \\
&= \binom{m}{k} \frac{(7y)^k y^{m-k} \prod_{j=0}^{k-1} (1-\frac{j}{7y}) \prod_{j=0}^{m-k-1} (1-\frac{j}{y})}{(8y)^m \prod_{j=0}^{m-1} (1-\frac{j}{8y})} = \binom{m}{k} \frac{7^k \prod_{j=0}^{k-1} (1-\frac{j}{7y}) \prod_{j=0}^{m-k-1} (1-\frac{j}{y})}{8^m \prod_{j=0}^{m-1} (1-\frac{j}{8y})}.
\end{aligned}$$

Since $P\{f_m(\omega) = k\} = \binom{m}{k} \frac{7^k}{8^m}$, then

$$\Theta(k, m, y) = \frac{\mathbb{P}\{h_m(\omega) = k\}}{\mathbb{P}\{f_m(\omega) = k\}} = \frac{\prod_{j=0}^{k-1} \left(1 - \frac{j}{7y}\right) \prod_{j=0}^{m-k-1} \left(1 - \frac{j}{y}\right)}{\prod_{j=0}^{m-1} \left(1 - \frac{j}{8y}\right)}. \quad (3.3)$$

Note that

$$\left(1 - \frac{k-1}{7y}\right)^k \leq \prod_{j=0}^{k-1} \left(1 - \frac{j}{7y}\right) \leq 1, \text{ since } 0 \leq j \leq k-1,$$

$$\left(1 - \frac{m-k-1}{y}\right)^{m-k} \leq \prod_{j=0}^{m-k-1} \left(1 - \frac{j}{y}\right) \leq 1, \text{ since } 0 \leq j \leq m-k-1, \text{ and}$$

$$\left(1 - \frac{m-1}{8y}\right)^m \leq \prod_{j=0}^{m-1} \left(1 - \frac{j}{8y}\right) \leq 1, \text{ since } 0 \leq j \leq m-1.$$

Therefore, (3.3) can be bounded by

$$\left(1 - \frac{k-1}{7y}\right)^k \left(1 - \frac{m-k-1}{y}\right)^{m-k} \leq \Theta(k, m, y) \leq \frac{1}{\left(1 - \frac{m-1}{8y}\right)^m},$$

which leads to

$$\left(1 - \frac{m-1}{8y}\right)^m \left(1 - \frac{k-1}{7y}\right)^k \left(1 - \frac{m-k-1}{y}\right)^{m-k} \leq \Theta(k, m, y) \leq 1. \quad \square$$

Theorem 3.3 shows that $\frac{\mathbb{P}\{h_m(\omega) = k\}}{\mathbb{P}\{f_m(\omega) = k\}} \rightarrow 1$ for $k = 0, 1, \dots, m$ as $m \rightarrow +\infty$, $n \rightarrow +\infty$ and $(m/y) \rightarrow 0$.

Theorem 3.3: Consider the random variables $f_m(\omega)$ and $h_m(\omega)$. Then $\frac{\mathbb{P}\{h_m(\omega) = k\}}{\mathbb{P}\{f_m(\omega) = k\}} \rightarrow 1$ for $k = 0, 1, \dots, m$ as $m \rightarrow +\infty$, $n \rightarrow +\infty$ and $(m/y) \rightarrow 0$.

Proof: Let $m \rightarrow +\infty$, $n \rightarrow +\infty$ and $(m/y) \rightarrow 0$. Then $y \rightarrow +\infty$ since $y = \binom{n}{3}$ and $n \rightarrow +\infty$. To

show the result, pick any $k = 0, 1, \dots, m$. Then $\frac{m-1}{8y} \rightarrow 0$, $\frac{k-1}{7y} \rightarrow 0$ and $\frac{m-k-1}{y} \rightarrow 0$ as

$m \rightarrow +\infty$, $n \rightarrow +\infty$ and $(m/y) \rightarrow 0$. Therefore, $\left(1 - \frac{m-1}{8y}\right)^m \left(1 - \frac{k-1}{7y}\right)^k \left(1 - \frac{m-k-1}{y}\right)^{m-k} \rightarrow 1$

as $m \rightarrow +\infty$, $n \rightarrow +\infty$ and $(m/y) \rightarrow 0$. It then follows from Corollary 3.1 that

$$\frac{P\{h_m(\omega) = k\}}{P\{f_m(\omega) = k\}} \rightarrow 1 \text{ for } k = 0, 1, \dots, m \text{ as } m \rightarrow +\infty, n \rightarrow +\infty \text{ and } (m/y) \rightarrow 0. \quad \square$$

Theorem 3.4 proves a Central Limit Theorem for the random variables $f_m(\omega)$ and $h_m(\omega)$ by using the results obtained from Theorem 3.1 and Theorem 3.2.

Theorem 3.4: Consider the random variables $f_m(\omega)$ and $h_m(\omega)$. Then

- (i) $\frac{f_m(\omega) - (7m/8)}{\sqrt{(7m/64)}} \rightarrow_D N(0,1)$ as $m \rightarrow +\infty$ and
- (ii) $\frac{h_m(\omega) - (7m/8)}{\sqrt{(7m/64)}} \rightarrow_D N(0,1)$ as $m \rightarrow +\infty$, $n \rightarrow +\infty$ and $(m/n^3) \rightarrow 0$

where $N(0,1)$ denotes the standard normal distribution and “ \rightarrow_D ” denotes convergence in distribution.

Proof .: From Theorem 3.1, $f_m(\omega)$ is a binomial random variable with parameters m and $p = 7/8$. Then, as $m \rightarrow +\infty$, $n \rightarrow +\infty$ and $(m/n^3) \rightarrow 0$ (i) follows from the DeMoivre-Laplace Theorem (see Hoel et al. 1971, page 184).

Let B_m be a binomial random variable with parameters m and p . Then

$$\frac{B_m - mp}{\sqrt{mp(1-p)}} \rightarrow_D N(0,1) \text{ as } m \rightarrow +\infty.$$

Moreover, (ii) follows from (i) and the result (see Wadsworth and Bryan 1960, page 62),

Let H_m be a hypergeometric random variable with parameters $8y$, m and $7y$, and B_m be a binomial distribution with parameters m and $p = 7/8$. Then distribution of H_m converges to distribution of B_m (i.e., convergence in distribution) as $m \rightarrow +\infty$, $n \rightarrow +\infty$ and $(m/y) \rightarrow 0$.

Also note that since $y = O(n^3)$ for n large, then $(m/y) \rightarrow 0$ and $(m/n^3) \rightarrow 0$ are equivalent as $m \rightarrow +\infty$ and $n \rightarrow +\infty$. □

The results in this section provide information on randomly generated instances of 3-SATISFIABILITY, given a particular solution. Theorem 3.1 shows that if identical clauses can be generated (i.e., a clause may appear more than once) in a randomly generated instance of 3-SATISFIABILITY (containing m clauses), then the number of satisfied clauses for a given solution follows a binomial distribution with parameters m and $p = 7/8$. Theorem 3.2 shows that

if identical clauses cannot be generated (i.e., a clause cannot appear more than once) in a randomly generated instance of 3-SATISFIABILITY (containing m clauses), then the number of satisfied clauses for a given solution follows a hypergeometric distribution with parameters $8\binom{n}{3}$, m and $7\binom{n}{3}$. Corollary 3.1 and Theorem 3.3 show that if the number of clauses, m , is asymptotically sufficiently small compared to number of all possible clauses (i.e., $m/n^3 \rightarrow 0$), then $\frac{P\{h_m(\omega) = k\}}{P\{f_m(\omega) = k\}}$ converges to one (for $k = 0, 1, \dots, m$) as $m \rightarrow +\infty$ and $n \rightarrow +\infty$. Lastly,

Theorem 3.4 proves a Central Limit Theorem for the random variables $h_m(\omega)$ and $f_m(\omega)$.

The results presented in this section hold for a fixed solution $\omega \in \Omega$. To obtain expressions for the number of solutions that satisfies $k = 0, 1, \dots, m$ clauses in a randomly generated instance of 3-SATISFIABILITY, all solutions in the solution space Ω need to be considered. This analysis is presented in Section 3.3.

3.3. Analysis for the Solution Space, Ω

This section probabilistically studies the solution space Ω (i.e., the set of all possible solutions) for randomly generated instances of 3-SATISFIABILITY. In particular, a matrix structure is defined and studied for randomly generated instances of 3-SATISFIABILITY. Several definitions are needed for this analysis. Define a matrix, A , with m rows and eight columns, where each row corresponds to a given clause involving three Boolean variables $\{X_a, X_b, X_c\}$ and each column along this row corresponds to an assignment for these three Boolean variables. For example, suppose that a given clause at row j involves the Boolean variables $\{X_{10}, X_{14}, X_{16}\}$, then the first column along this row corresponds to the assignment $\{X_{10} = 1, X_{14} = 1, X_{16} = 1\}$, the second column corresponds to the assignment $\{X_{10} = 1, X_{14} = 1, X_{16} = 0\}$, and so on. Note that for each row (i.e., a given clause) in matrix A , the assignments (i.e., the columns) consider only the three Boolean variables that are included in this clause. A generic form of matrix A is given in Table 3.1.

Table 3.1: Generic Form of Matrix A

	1,1,1	1,1,0	1,0,1	1,0,0	0,1,1	0,1,0	0,0,1	0,0,0
Clause 1	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}	A_{18}
Clause 2	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	A_{27}	A_{28}
...
Clause j	A_{j1}	A_{j2}	A_{j3}	A_{j4}	A_{j5}	A_{j6}	A_{j7}	A_{j8}
...
Clause m	A_{m1}	A_{m2}	A_{m3}	A_{m4}	A_{m5}	A_{m6}	A_{m7}	A_{m8}

For matrix A, define a *path*, π , to be a sequence of m cells (i.e., one cell for each row). For example, the sequence of cells $(A_{13}, A_{25}, \dots, A_{j8}, \dots, A_{m3})$ is a path on matrix A. By this definition, a path, π , is a sequence of moves from the first row to the m^{th} row and visit exactly one cell at each row. Note that a path can select any of the eight cells at each row. Therefore, there are 8^m distinct paths on matrix A. Define $\Xi = \{1, 2, \dots, 8\}^m$ to be the set of all distinct paths that can be defined on matrix A (i.e., $|\Xi| = 8^m$).

Given an instance of 3-SATISFIABILITY, each row on matrix A corresponds to a single clause in this instance. Moreover, for a clause at row j, the cells $(A_{j1}, A_{j2}, \dots, A_{j8})$ take on the value one (i.e., $A_{jv} = 1$ if clause j is satisfied for the assignment in column v) or zero (i.e., $A_{jv} = 0$ if clause j is not satisfied for the assignment in column v). Note that the cell values provide information on the instance of 3-SATISFIABILITY. In particular, the value of each cell is an indicator of whether a clause is satisfied (or unsatisfied) for the Boolean variable assignment in the corresponding column. Furthermore, for a path, $\pi \in \Xi$, the sum of the cell values along the path is the number of satisfied clauses for this path. Let $d_m(\pi)$ denote the number of satisfied clauses

(i.e., sum of the cell values) for a path $\pi \in \Xi$ defined on matrix A (i.e., $d_m(\pi) = \sum_{j=1}^m \sum_{v=1}^8 A_{jv} I_{\{v \in \pi\}}$,

$0 \leq d_m(\pi) \leq m$).

Example 3.1 illustrates the design and construction of matrix A.

Example 3.1: Consider an instance of 3-SATISFIABILITY containing five Boolean variables and six clauses (listed in Table 3.2). Note that the first row of matrix A corresponds to the clause $(X_1 \vee \bar{X}_2 \vee \bar{X}_4)$. The first column along this row corresponds to the Boolean variable assignment $\{X_1 = 1, X_2 = 1, X_4 = 1\}$, the second column corresponds to the Boolean variable assignment $\{X_1 = 1, X_2 = 1, X_4 = 0\}$ (and so on). It can be seen that the clause $(X_1 \vee \bar{X}_2 \vee \bar{X}_4)$ is not satisfied for the assignment in column 5 (i.e., $\{X_1 = 0, X_2 = 1, X_4 = 1\}$), while it is satisfied

for the assignments in the seven other columns. Therefore, the cells $A_{11}, A_{12}, \dots, A_{18}$ take on the values 1, 1, 1, 1, 0, 1, 1, 1, respectively. The second row in matrix A corresponds to the clause $(X_1 \vee \bar{X}_3 \vee X_4)$. Note that for this row, only the Boolean variables $\{X_1, X_3, X_4\}$ are considered. Therefore, first column corresponds to the Boolean variable assignment $\{X_1 = 1, X_3 = 1, X_4 = 1\}$, hence the cell A_{21} takes on the value one. Values for the other cells are evaluated in the same way. Table 3.2 provides all the cell values for this instance of 3-SATISFIABILITY.

Table 3.2: Matrix A for an Instance of 3-SATISFIABILITY

	1,1,1	1,1,0	1,0,1	1,0,0	0,1,1	0,1,0	0,0,1	0,0,0
$(X_1 \vee \bar{X}_2 \vee \bar{X}_4)$	1	1	1	1	0	1	1	1
$(X_1 \vee \bar{X}_3 \vee X_4)$	1	1	1	1	1	0	1	1
$(\bar{X}_1 \vee X_4 \vee X_5)$	1	1	1	0	1	1	1	1
$(X_3 \vee \bar{X}_4 \vee X_5)$	1	1	1	1	1	0	1	1
$(X_1 \vee X_3 \vee X_5)$	1	1	1	1	1	1	1	0
$(X_2 \vee X_4 \vee \bar{X}_5)$	1	1	1	1	1	1	0	1

An important property of matrix A is that at each row, exactly seven cells take on the value one while one cell takes on the value zero. Furthermore, this holds for any instance of 3-SATISFIABILITY. To see this, note that a clause at row j involves only three Boolean variables and the columns at row j correspond to all eight possible assignments for these three Boolean variables. From definition of 3-SATISFIABILITY, a clause involving three Boolean variables is satisfied for seven Boolean variable assignments, while it is not satisfied for one Boolean variable assignment.

This property of matrix A leads to the result given in Lemma 3.2.

Lemma 3.2: Consider an instance of 3-SATISFIABILITY containing m clauses defined by n Boolean variables. For this problem, construct the matrix A and find all 8^m paths in this matrix (i.e., find the set Ξ). Define $N_k(m)$ to be the number of paths in Ξ such that the sum of the cell values is k (i.e., the number of paths in Ξ such that $d_m(\pi) = k$). Then,

$$N_k(m) = \binom{m}{k} 7^k, \quad k = 0, 1, \dots, m.$$

Proof : By definition, a path, $\pi \in \Xi$, visits one cell at each row on matrix A. Therefore, a path $\pi \in \Xi$ visits exactly m cells. For a fixed $k = 0, 1, \dots, m$, $d_m(\pi) = k$ means that a path visits exactly k cells with the value one and $m - k$ cells with the value zero. Using the binomial property, it can

be seen that there are $\binom{m}{k} 7^k$ distinct ways in which a path can visit k cells with the value one and $m - k$ cells with the value zero. \square

Another property of matrix A is that each solution $\omega \in \Omega$ defines a unique path on the matrix. To see this, pick a solution $\omega \in \Omega$. In this solution, all n Boolean variables are assigned a value of one or zero. By definition, each row of matrix A corresponds to a clause involving three Boolean variables. Therefore, the assigned values for these three Boolean variables correspond to a particular column for some path, $\pi \in \Xi$. To illustrate this, consider Example 1 and Table 3.2. The solution $\omega = \{X_1 = 1, X_2 = 0, X_3 = 1, X_4 = 0, X_5 = 1\}$ defines the path $(A_{14}, A_{22}, A_{33}, A_{43}, A_{51}, A_{67})$ on matrix A . Note that the first clause in Table 3.2 contains the Boolean variables $\{X_1, X_2, X_4\}$. For this solution ω , these three Boolean variables take on the values 1,0,0, respectively. Since the fourth column of matrix A contains the same Boolean variable assignment, then A_{14} must be the first cell in the path. The other cells along this path can be found in a similar way (this path is marked on Table 3.2 using bold fonts).

By the definition of 3-SATISFIABILITY, there are 2^n possible solutions in the solution space Ω . For each solution $\omega \in \Omega$, there exist a unique path on matrix A . Therefore, each solution $\omega \in \Omega$ is also a path on matrix A (i.e., $\Omega \subseteq \Xi$). On the other hand, every path $\pi \in \Xi$ on matrix A may not define a solution $\omega \in \Omega$. To see this, consider the path $(A_{11}, A_{23}, A_{35}, A_{47}, A_{54}, A_{66})$ in Table 3.2. This path requires the Boolean variable X_1 to be assigned a value one in cell A_{11} , while it requires the Boolean variable X_1 to be assigned a value zero in cell A_{35} . Since a Boolean variable can take on either one or zero, but not both values, then this path does not define a solution. By the definition of Ξ , there are 8^m distinct paths that can be defined from matrix A , with 2^n of these paths corresponding to solutions in Ω , and the remaining $(8^m - 2^n)$ paths not corresponding to any solutions. Define a partition of Ξ , Ξ_π and Ξ_ω , to be the set of all paths that do not correspond to solutions and set of all paths that do correspond to solutions, respectively (i.e., $\Xi = \Xi_\pi \cup \Xi_\omega$).

From Lemma 3.2, the number of paths that satisfy k clauses (i.e., $N_k(m)$) is same for all m clause instances of 3-SATISFIABILITY. Note that for all $k = 0, 1, \dots, m$, $N_k(m)$ does not depend on the number of Boolean variables. Define $s_k(m)$ to be the number of solutions that satisfy k clauses for an m clause instance of 3-SATISFIABILITY. By definition, each solution satisfying k clauses is also a path satisfying k clauses in matrix A . Therefore, $s_k(m) \leq N_k(m)$ for all $k = 0,$

1, ..., m. For an m clauses instance of 3-SATISFIABILITY, $\mathbf{s}(\mathbf{m}) = (s_0(\mathbf{m}), s_1(\mathbf{m}), \dots, s_m(\mathbf{m}))$ is the frequency distribution of all solutions (i.e., the solution space Ω) over the number of satisfied clauses k. For a randomly generated m clause instance of 3-SATISFIABILITY, $\mathbf{S}(\mathbf{m}) = (S_0(\mathbf{m}), S_1(\mathbf{m}), \dots, S_m(\mathbf{m}))$ is a random vector of size m + 1. This vector is formally defined in Corollary 3.2, for the case when randomly generated clauses are generated with replacement.

Corollary 3.2: Consider randomly generating m clauses with replacement $C_1, C_2, \dots, C_m \in \Psi$ (i.e., a clause can be selected more than once) with solution space $\Omega = \{\omega_1, \omega_2, \dots, \omega_t, \dots, \omega_{2^n}\}$. For each solution, $\omega_t \in \Omega$, and a value $k = 0, 1, \dots, m$, define the indicator random variable,

$$I_{tk} = \begin{cases} 1 & \text{if } f_m(\omega_t) = k \\ 0 & \text{otherwise} \end{cases} \quad t = 1, 2, \dots, |\Omega|, \quad k = 0, 1, \dots, m.$$

Then for each $k = 0, 1, \dots, m$, the number of solutions that satisfy k clauses, $S_k(\mathbf{m})$, is the random variable

$$S_k(\mathbf{m}) = \sum_{t=1}^{2^n} I_{tk}, \quad \text{where } S_k(\mathbf{m}) = 0, 1, \dots, \min\{2^n, N_k(\mathbf{m})\}. \quad (3.4)$$

Moreover, $\mathbf{S}(\mathbf{m}) = (S_0(\mathbf{m}), S_1(\mathbf{m}), \dots, S_m(\mathbf{m}))$, is a random vector of size m + 1 such that

$$S_0(\mathbf{m}) + S_1(\mathbf{m}) + S_2(\mathbf{m}) + \dots + S_m(\mathbf{m}) = 2^n \quad (3.5)$$

and

$$E[S_k(\mathbf{m})] = E\left[\sum_{t=1}^{2^n} I_{tk}\right] = 2^n \binom{m}{k} \frac{7^k}{8^m}. \quad (3.6)$$

Proof : From Theorem 3.1, for each solution, $\omega_t \in \Omega$, $f_m(\omega_t)$ is a binomial random variable with parameters m and $p = 7/8$. Note that m randomly selected clauses $C_1, C_2, \dots, C_m \in \Psi$ define the matrix A. Then (3.4) follows from the definition of $S_k(\mathbf{m})$ and (3.5) follows from the definition of 3-SATISFIABILITY (i.e., $|\Omega| = 2^n$). To show (3.6), note that

$$E[S_k(\mathbf{m})] = E\left[\sum_{t=1}^{2^n} I_{tk}\right] = \sum_{t=1}^{2^n} E[I_{tk}], \quad \text{where } E[I_{tk}] = P\{f_m(\omega_t) = k\} = \binom{m}{k} \frac{7^k}{8^m}. \quad \text{Therefore,}$$

$$E[S_k(\mathbf{m})] = \sum_{t=1}^{2^n} E[I_{tk}] = \sum_{t=1}^{2^n} \binom{m}{k} \frac{7^k}{8^m} = 2^n \binom{m}{k} \frac{7^k}{8^m}. \quad \square$$

Corollary 3.3 shows a similar result for the case when clauses are randomly generated without replacement.

Corollary 3.3: Consider randomly generating m clauses without replacement $C_1, C_2, \dots, C_m \in \Psi$ (i.e., a clause cannot be selected more than once) with solution space $\Omega = \{\omega_1, \omega_2, \dots, \omega_t, \dots, \omega_2^n\}$. For each solution, $\omega_t \in \Omega$, and a value $k = 0, 1, \dots, m$, define the indicator random variable,

$$I_{tk} = \begin{cases} 1 & \text{if } h_m(\omega_t) = k \\ 0 & \text{otherwise} \end{cases}, \quad t = 1, 2, \dots, |\Omega|, k = 0, 1, \dots, m.$$

Then for the random variable $S_k(m)$ and the random vector $\mathbf{S}(m) = (S_0(m), S_1(m), \dots, S_m(m))$ defined in Corollary 3.2,

$$E[S_k(m)] = E\left[\sum_{t=1}^{2^n} I_{tk}\right] = 2^n \frac{\binom{7y}{k} \binom{y}{m-k}}{\binom{8y}{m}}, \quad \text{where } y = \binom{n}{3}.$$

Proof: Substituting $f_m(\omega_t)$ with $h_m(\omega_t)$ into Corollary 3.2 leads to the result. \square

Corollary 3.2 and Corollary 3.3 provide mathematical expressions for the expected number of solutions that satisfy k clauses (i.e., $E[S_k(m)]$). For a randomly generated instance of 3-SATISFIABILITY, the random vector $\mathbf{S}(m) = (S_0(m), S_1(m), \dots, S_m(m))$ denotes the distribution of the solution space over the number of satisfied clauses, k , where the expected value vector for $\mathbf{S}(m)$ is $(E[S_0(m)], E[S_1(m)], \dots, E[S_m(m)])$. From Lemma 3.2, the frequency distribution of all paths over k (i.e., $(N_0(m), N_1(m), \dots, N_m(m))$) is same for all m clause instances of 3-SATISFIABILITY. Figure 3.1 depicts values for $(E[S_0(m)], E[S_1(m)], \dots, E[S_m(m)])$ and the values of $(N_0(m), N_1(m), \dots, N_m(m))$ for a randomly generated instance of 3-SATISFIABILITY with $m = 15$ and $n = 44$.

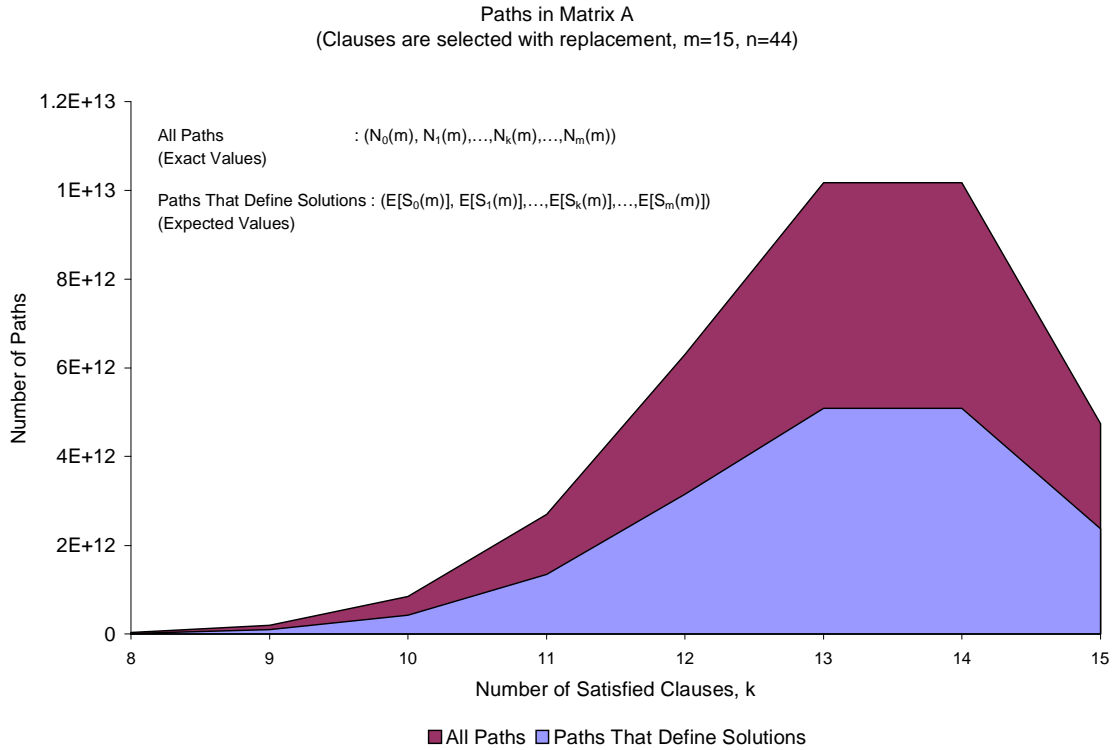


Figure 3.1: Paths on the Matrix A

Figure 3.1 provides useful information on randomly generated instances of 3-SATISFIABILITY. For example, for a given m and n , the values of the vector $(E[S_0(m)], E[S_1(m)], \dots, E[S_m(m)])$ can be computed and an upper bound (see Section 5.4) for the maximum number of satisfied clauses can be obtained. To see this, consider an instance of 3-SATISFIABILITY with $m = 100$ and $n = 10$. The expected number of solutions that satisfy all one hundred clauses in this problem instance (i.e., $E[(S_{100}(100))]$) is approximately 0.002. Therefore, it is highly unlikely that this problem is satisfiable. A simple calculation shows that if $k = 96$, then $E[S_{96}(100)] = 2.65$, which means that the expected number of solutions that satisfy 96 clauses is less than three. For this instance, $E[(S_{97}(100))] \approx 0.766$, which is less than one. From these calculations, the maximum number of clauses that are satisfied for at least one solution can be estimated (i.e., 96 for this instance). Note that this number provides an upper bound (see the Section 5.4) for the maximum number of satisfied clauses. Furthermore, this number can be used to define a stopping criterion for a local search algorithm (i.e., the algorithm should be terminated when a solution that satisfy 96 clauses is found or a pre-specified number of iterations have been executed). This analysis can also be used to decide if an algorithm has been prematurely terminated. To see this, suppose that an algorithm is terminated and reports that the best solution found results in 90 clauses being

satisfied. It can then be concluded that the algorithm may have been terminated prematurely, since on average, there are $2^{10} \sum_{j=91}^{100} \binom{100}{j} \frac{7^j}{8^{100}} = 188$ solutions (i.e., approximately 18% of the $1024 = 2^{10}$ solutions) that are better than the solution found, as measured by the number of satisfied clauses.

Note that Corollary 3.2 and Corollary 3.3 only provide expressions for the expected values of the random vector $\mathbf{S}(\mathbf{m}) = (S_0(\mathbf{m}), S_1(\mathbf{m}), \dots, S_m(\mathbf{m}))$. Work is in progress to obtain the joint probability distribution for $\mathbf{S}(\mathbf{m})$; properties of matrix A may be useful for this purpose. For example, note that for each solution $\omega \in \Omega$, there is an *inverse solution* $\omega' \in \Omega$ defined as a solution that is obtained by switching the values (i.e., from one (zero) to zero (one)) of all n Boolean variables in a given solution. Note that in matrix A , if a solution visits a cell with value zero at row j , then the inverse solution has to visit a cell with value one at row j . This means that if there exists a solution $\omega \in \Omega$ such that $d_m(\omega) \leq k$, then there exists an inverse solution $\omega' \in \Omega$ such that $d_m(\omega') \geq m-k$. To see this, note that $d_m(\omega) \leq k$ means that the solution $\omega \in \Omega$ visits k or fewer cells with value one, hence it must visit $m-k$ or more cells with value zero. Therefore, the inverse path visits $m-k$ or more cells with value one. Such symmetry may be helpful for obtaining the joint probability distribution of random vector $\mathbf{S}(\mathbf{m})$. On the other hand, note that $S_0(\mathbf{m}) + S_1(\mathbf{m}) + S_2(\mathbf{m}) + \dots + S_m(\mathbf{m}) = 2^n$, hence the random variables contained in the random vector $\mathbf{S}(\mathbf{m})$ are not independent. Moreover, these random variables are not identically distributed (e.g., $E[S_{m-1}(\mathbf{m})] \neq E[S_m(\mathbf{m})]$), hence making the probabilistic analysis of $\mathbf{S}(\mathbf{m})$ difficult. Obtaining the joint probability distribution of the random vector $\mathbf{S}(\mathbf{m})$ is a problem to be considered for future research.

3.4. The Threshold Conjecture

The two parameters m (i.e., the number of clauses) and n (i.e., the number of Boolean variables) are the key factors in determining whether a randomly generated instance of 3-SATISFIABILITY can be satisfied. It is conjectured that for n fixed, randomly generated instances of 3-SATISFIABILITY are less (more) likely to be satisfied for m sufficiently large (small). This phenomenon, termed *the threshold conjecture*, is formally described (Cook and Mitchell 1997).

Let c_l be the clause-to-variable ratio (i.e., m/n) for an instance of l -SATISFIABILITY, where each clause has exactly l literals. It is conjectured that for every $l > 2$, there is a constant c_l^ such that for each fixed value of $c_l < c_l^*$, the probability that a randomly generated instance of l -SATISFIABILITY (with n Boolean variables and m clauses) is satisfiable approaches one as $n \rightarrow \infty$. Moreover when $c_l > c_l^*$, the probability that a randomly generated instance of l -SATISFIABILITY is unsatisfiable approaches one as $n \rightarrow \infty$.*

For $l=2$ (i.e., the 2-SATISFIABILITY problem), it has been shown that $c_2^* = 1$ (see Goerdt 1992). However, no proof exists for $l > 2$ and the threshold conjecture remains an open problem. For $l=3$ (i.e., 3-SATISFIABILITY problem) current best known bounds for the constant c_3^* are $3.003 < c_3^* < 4.596$. The lower bound is reported in Frieze and Suen (1996). This result comes with an algorithm that, with probability approaching one, finds a satisfying solution for a randomly generated instance of 3-SATISFIABILITY (with $m/n < 3.003$), in polynomial time (in the size of the problem instance). Janson et al. (2000) reported the upper bound. Empirical studies (e.g., see Mitchell et al. 1992 for the first published results) suggest that this constant is approximately 4.24 (i.e., $c_3^* \approx 4.24$).

Note that the threshold conjecture refers to a single random variable $S_m(m)$ on the vector $\mathbf{S}(m)$ defined in Section 3.3. In particular, $P\{S_m(m) = 0\}$ is the probability that a randomly generated instance of 3-SATISFIABILITY is unsatisfiable (i.e., there is no solution that satisfies all m clauses). If the marginal probability distribution of the random variable $S_m(m)$ can be obtained (e.g., the joint probability distribution of random vector $\mathbf{S}(m)$ can be used for this purpose), then the result follows easily. On the other hand, dependency between the solutions make it quite challenging to obtain the joint probability distribution of random vector $\mathbf{S}(m)$.

Note that all the probabilistic analysis in this research focuses on randomly generated instances of 3-SATISFIABILITY (i.e., $l=3$). On the other hand, this analysis can be extended to randomly generated instances of l -SATISFIABILITY for any $l = 2, 3, \dots$. In particular, using a similar analysis as presented in Section 3.2, for any value of $l = 2, 3, \dots$, the random variable $f_m(\omega)$ is a binomial random variable with parameters m and $p = (2^l - 1) / 2^l$, and the random variable $h_m(\omega)$ is a hypergeometric random variable with parameters $2^l \binom{n}{l}$, m and $(2^l - 1) \binom{n}{l}$.

Given a solution $\omega \in \Omega$ and a randomly generated instance of l -SATISFIABILITY, Θ_l , with m clauses and n Boolean variables, let $c_l = m/n$ be the clause-to-variable ratio for Θ_l and define the random variable $Y_l(\omega)$ to be the number of clauses in Θ_l that are satisfied for solution $\omega \in \Omega$. Define $\Gamma = 2^n P\{Y_l(\omega) = m\}$ to be the expected number of solutions that satisfy all m clauses in Θ_l . By the extension of Theorem 3.4 (i.e., the Central Limit Theorem) to l -SATISFIABILITY,

$$Z = \frac{Y_l(\omega) - \frac{(2^l - 1)m}{2^l}}{\sqrt{\frac{(2^l - 1)m}{2^{2l}}}} \rightarrow_D N(0,1).$$

Note that

$$\begin{aligned} P\{Y_l(\omega) = m\} &\approx P\left\{m - \frac{1}{2} \leq Y_l(\omega) \leq m + \frac{1}{2}\right\} = P\left\{\frac{m - 2^{l-1}}{\sqrt{m(2^l - 1)}} \leq Z \leq \frac{m + 2^{l-1}}{\sqrt{m(2^l - 1)}}\right\} \\ &\approx \frac{2^l}{\sqrt{2\pi m(2^l - 1)}} e^{-\frac{m}{2(2^l - 1)}} = e^{-\frac{m}{2(2^l - 1)} + L_n(2) - \frac{1}{2} L_n[2\pi m(2^l - 1)]}. \end{aligned}$$

Substituting m with nc_l leads to $P\{Y_l(\omega) = m\} \approx e^{-\frac{nc_l}{2(2^l - 1)} + L_n(2) - \frac{1}{2} L_n[2\pi nc_l(2^l - 1)]}$, hence

$$\Gamma = 2^n P\{Y_l(\omega) = m\} \approx e^{\frac{nL_n(2) - \frac{nc_l}{2(2^l - 1)} + L_n(2) - \frac{1}{2} L_n[2\pi nc_l(2^l - 1)]}{1}}. \quad \text{Since } \Gamma \text{ is dominated by the term}$$

$e^{\frac{nL_n(2) - \frac{nc_l}{2(2^l - 1)}}{1}}$ (as $n \rightarrow +\infty$), then $\Gamma \rightarrow +\infty$ as $n \rightarrow +\infty$ if $c_l < 2(2^l - 1)L_n(2)$, while $\Gamma \rightarrow 0$ as $n \rightarrow +\infty$ if $c_l > 2(2^l - 1)L_n(2)$. Table 3.3 contains values for $\alpha_l = 2(2^l - 1)L_n(2)$.

Note that $\Gamma \rightarrow 0$ means that the expected number of solutions that satisfy all m clauses in Θ_l approaches zero as $n \rightarrow +\infty$, hence the number of solutions that satisfy all m clauses in Θ_l approaches zero as $n \rightarrow +\infty$. Using Markov's inequality (i.e., $P\{X \geq a\} \leq E[X]/a$), $P\{S_m(m) \geq 1\} \leq E[S_m(m)]$ implies that $P\{\Theta_l \text{ is satisfiable}\} \rightarrow 0$ as $n \rightarrow +\infty$ (note that $S_m(m) \geq 1$ means that Θ_l is satisfiable, by definition). On the other hand, $\Gamma \rightarrow +\infty$ does not imply that $P\{\Theta_l \text{ is satisfiable}\} \rightarrow 1$ as $n \rightarrow +\infty$. Therefore, the values in Table 3.3 can only be considered as upper bounds for the threshold values, c_l^* , $l = 2, 3, \dots, 9, 19$, (i.e., $c_l^* < 2(2^l - 1)L_n(2)$, $l = 2, 3, \dots, 9, 19$). For $l = 3$, the current upper bound, $c_3^* < 4.596$, is clearly better than the value given in Table 3.3 (i.e., 9.704).

Consider an alternate method to estimate the threshold values. Since $Y_l(\omega)$ is a binomial random variable with parameters m and $p = (2^l - 1) / 2^l$, then $P\{Y_l(\omega) = m\} = \binom{2^l - 1}{2^l}^m = \left(\frac{2^l - 1}{2^l}\right)^{nc_l}$ is the probability that a fixed solution $\omega \in \Omega$ satisfies all m clauses in a randomly generated instance of l -SATISFIABILITY. Let Φ be a subset of solutions from the solution space Ω (i.e., $\Phi \subseteq \Omega$) with $|\Phi| = \left[\left(\frac{2^l - 1}{2^{l-1}}\right)^n\right]$. Note that for fixed values of l and c_l , $|\Phi| \rightarrow +\infty$ and $P\{Y_l(\omega) = m\} \rightarrow 0$ as $n \rightarrow +\infty$. On the other hand, if $c_l > \beta_l = Ln(2^{l-1} / (2^l - 1)) / Ln((2^l - 1) / 2^l)$, then $P\{Y_l(\omega) = m\} \rightarrow 0$ faster than $|\Phi| \rightarrow +\infty$ as $n \rightarrow +\infty$, hence $P\{\Theta_l \text{ is satisfiable}\} \rightarrow 0$ as $n \rightarrow +\infty$. Note that letting $|\Phi| = |\Omega| = 2^n$ and $P\{Y_l(\omega) = m\} = \left(\frac{2^l - 1}{2^l}\right)^{m+n} = \left(\frac{2^l - 1}{2^l}\right)^{n(c_l+1)}$ yield the same result (i.e., if $c_l > \beta_l$, then $P\{\Theta_l \text{ is satisfiable}\} \rightarrow 0$ as $n \rightarrow +\infty$). Table 3.3 contains several values for β_l .

Dubois et al. (1996) lists estimated values for c_l^* , $l = 3, 4, \dots, 10, 20$, based on an empirical study. Table 3.3 provides Dubois's estimates of the threshold values, c_l^* , along with the β_l and α_l values described earlier in this section.

Table 3.3: β_l and α_l Values Compared to Dubois's Estimates for c_l^* Values

l	c_l^*	β_l	α_l
2	-	-	4.16
3	4.24	4.19	9.70
4	9.88	9.74	20.79
5	21.05	20.83	42.97
6	43.31	43.01	87.34
7	87.70	87.38	176.06
8	176.41	176.10	353.50
9	353.88	353.54	708.40
10	708.78	708.44	1418.18
20	726816.49	726816.20	1453633.61

Note that Dubois's estimates for c_l^* values are very close to the $\alpha_{(l-1)}$ values listed in Table 3.3 (i.e., $c_l^* \approx \alpha_{(l-1)} = 2(2^{l-1} - 1)Ln(2)$). This means that a loose upper bound (obtained using the normal distribution approximation) for the threshold value c_{l-1}^* appears to provide a tight lower bound for the threshold value c_l^* , $l = 2, 3, \dots$ (i.e., $c_{(l-1)}^* < \alpha_{(l-1)} \approx c_l^* < \alpha_l \approx c_{(l+1)}^* < \dots$).

Moreover, note that Dubois's estimates for c_l^* values are very close to the β_l values listed in Table 3.3 (i.e., $c_l^* \approx \beta_l = Ln(2^{l-1} / 2^l - 1) / Ln(2^l - 1 / 2^l)$). These interesting and curious observations do not appear to have any mathematical basis. They are presented here as an open conjecture for future research, since an explanation for why β_l and/or $\alpha_{(l-1)}$ provide tight lower bounds for c_l^* will be a huge step forward in resolving the threshold conjecture.

3.5. Concluding Remarks

This chapter analyzed randomly generated instances of 3-SATISFIABILITY. In Section 3.2, two random variables, $f_m(\omega)$ and $h_m(\omega)$, were defined and analyzed for randomly generated instances of 3-SATISFIABILITY, given a solution $\omega \in \Omega$. The analysis showed that $f_m(\omega)$ is a binomial random variable with parameters m and $p = 7/8$, while $h_m(\omega)$ is a hypergeometric random variable with parameters $8 \binom{n}{3}$, m and $7 \binom{n}{3}$. A Central Limit Theorem for these random variables was also presented.

In Section 3.3, the solution space, Ω , was studied for randomly generated instances of 3-SATISFIABILITY. A matrix structure was introduced and used for this purpose. Lastly, an expression was obtained for the expected number of solutions satisfying k clauses (i.e., $E[S_k(m)]$). This analysis is used in Chapter 5 for deriving stopping rules for local search algorithms.

Section 3.4 described and analyzed the threshold conjecture using results from Section 3.2 and Section 3.3. The analysis yielded two curious observations indicating a possible interrelationship between the threshold values c_3^* , c_4^* , ..., c_l^* for 3-SATISFIABILITY, 4-SATISFIABILITY, ..., l -SATISFIABILITY problems, respectively.

Chapter 4 :

Tabu Search and Random Restart Local Search Algorithms for MAX 3-SATISFIABILITY

The analysis in Chapter 3 provides useful information for randomly generated instances of 3-SATISFIABILITY. On the other hand, these results assume that the problem instance is randomly generated, and hence, cannot be applied for a given problem instance without taking this into account. In such cases, an algorithmic approach must be used. This chapter introduces two local search algorithms (tabu search and random restart local search) to address MAX 3-SATISFIABILITY. Section 4.1 describes these two algorithms for a generic discrete optimization (maximization) problem. Section 4.2 describes how these two algorithms can be applied to MAX 3-SATISFIABILITY.

4.1. Tabu Search and Random Restart Local Search

Framework

To describe the general framework for tabu search and random restart local search for a discrete optimization problem, define Ω to be a finite set of feasible solutions (i.e., the solution space). Define an objective function $f: \Omega \rightarrow R^n$ that maps elements of the solution space into an n -tuple of reals. Note that the range of this function is often just R , the set of real scalars. Using these definitions, a discrete optimization problem, can be represented as

$$\underset{\omega \in \Omega}{\text{Maximize}} f(\omega).$$

4.1.1. Tabu Search Framework

Tabu search is a metaheuristic for addressing a wide variety of intractable discrete optimization problems, ranging from mixed integer programming to space planning (Glover and Laguna 1997). To describe tabu search, several definitions are needed. Define a *neighboring function*, $\eta: \Omega \rightarrow 2^\Omega$, where $\eta(\omega) \subseteq \Omega$ for all $\omega \in \Omega$. Define the *tabu list*, $T \subset \Omega$, to be a set of elements

identified as forbidden (tabu) solutions. T is initially defined to be empty, and subsequently updated at each iteration by prespecified rules (tabu conditions) that use historical information from the search. Define BEST to be an evaluator function that finds the best solution among a set of neighboring solutions $\eta(\omega) \setminus T$. Let ω^* denote the best solution found to date and define a stopping condition which terminates the algorithm (e.g., a desired number of iterations has elapsed or all neighboring solutions are tabu). Using these definitions and notations, a generic tabu search algorithm can be outlined in pseudo-code form (Glover and Laguna 1997).

1. Select an initial solution $\omega \in \Omega$
 $\omega^* \leftarrow \omega$
 Set the iteration counter $k = 0$ and $T = \emptyset$
2. $k \leftarrow k+1$
 Select $\omega_k = \text{BEST}(\eta(\omega) \setminus T)$
 $\omega \leftarrow \omega_k$
 If $f(\omega) > f(\omega^*)$, then $\omega^* \leftarrow \omega$
3. If the stopping condition is met, then STOP and report $f(\omega^*)$.
 Otherwise, update T and return to Step 2.

Two important aspects of tabu search should be emphasized

- (1) the tabu list, T ,
- (2) the evaluator function, BEST.

The tabu list is a memory-based structure that guides the search away from undesirable solutions (e.g., local optima or previously visited solutions) by defining a set of tabu solutions. Tabu search traverses the solution space by selecting the best possible solution at each step, determined by the function BEST. It should be noted that the best possible solution might not always yield an improvement in the objective function value. In such cases, tabu search allows the search to select an inferior solution (with the minimum deterioration in the objective function value). The function BEST also gives tabu search an ability to use other methods (e.g., linear programming) to make the search more flexible.

There are numerous tabu search strategies that have been developed, including long-term memory, and aspiration criteria (see Glover and Laguna 1997 for a list of other strategies). In the tabu search framework, T is usually used as a short-term memory tool that forbids a solution to be visited over the next L iterations. After L iterations have occurred, the same solution can be repeated since its tabu status has been removed. It is not desirable for an algorithm to revisit the same solution too often. To avoid such a situation, it may be necessary to incorporate long-term memory into the algorithm. Long-term memory utilizes the frequency information of previously

visited solutions and guides the search into unexplored regions of the solution space. In some cases, the simultaneous application of short-term and long-term memories may result in the neighboring solutions becoming too restricted. To circumvent this problem, aspiration criteria may be defined for overriding the tabu status of a solution (e.g., a tabu solution produces a better solution than the best solution found to date, ω^*). Both the type of memory and the tabu strategies used are important factors that affect the performance of tabu search. These factors need to be individually designed for each specific problem.

4.1.2. Random Restart Local Search Framework

Deterministic local search is a simple search strategy for addressing discrete optimization problems. Deterministic local search traverses the solution space by selecting the best neighboring solution at each step. In contrast to tabu search, deterministic local search does not allow an inferior solution to be selected, hence stops if an improving solution (among the neighboring solutions of the current solution) cannot be found. Therefore, deterministic local search does not have the ability to escape from local optima. There are numerous variations of deterministic local search that can be found in the literature, including random restart local search (Jacobson and Yücesan 2001), random-walk GSAT (Selman et al. 1992), and tunneling heuristics (Gu 1993) that offers strategies to escape from local optima. Random restart local search is a modification of deterministic local search. Random restart local search randomly selects an initial solution to restart the search whenever the algorithm becomes trapped in a local optimum. Using the same notation as described for tabu search, random restart local search can be outlined in pseudo-code form.

1. Set the random restart counter $r = 0$
2. $r \leftarrow r+1$
 Select an initial solution $\omega \in \Omega$
 Set the iteration counter $k = 0$
3. $k \leftarrow k+1$
 Select $\omega_k = \text{BEST}(\eta(\omega))$
 if $f(\omega_k) > f(\omega^*)$, then $\omega^* \leftarrow \omega_k$
 if $f(\omega_k) > f(\omega)$, then $\omega \leftarrow \omega_k$ and return to Step 3
4. If $r < R$, then return to Step 2
 Otherwise STOP and report $f(\omega^*)$

4.2. Algorithm Implementations for MAX

3-SATISFIABILITY

This section describes how tabu search and random restart local search algorithms can be applied to MAX 3-SATISFIABILITY. Note that the tabu search and random restart algorithms are designed to address a wide variety of discrete optimization problems. Therefore, implementations of these algorithms for MAX 3-SATISFIABILITY require an objective function, a neighborhood function, and a tabu list (for tabu search) to be specified.

4.2.1. Definitions

Several definitions are needed to describe the tabu search and random restart local search algorithms for MAX 3-SATISFIABILITY. Let n be the number of Boolean variables and m be the number of clauses for a given instance of MAX 3-SATISFIABILITY. Recall that Ω is the set of all possible (2^n) solutions (i.e., the solution space), where every solution $\omega \in \Omega$ is represented as a Boolean vector of size n . Define an objective function, $f: \Omega \rightarrow [0, m]$, such that

$$f(\omega) = \sum_{j=1}^m C_j(\omega)$$

denotes the number of *satisfied* clauses associated with solution $\omega \in \Omega$. Let $\omega_k \in \Omega$ be the current solution at iteration k . A neighboring solution, $\omega_k(i) \in \eta(\omega_k)$, is then obtained by switching the value of the Boolean variable X_i , in the current solution ω_k (e.g., neighboring solution $\omega_k(50)$ is obtained by changing $x(50)$ from 0 to 1 (or 1 to 0) in ω_k). This switching procedure defines the neighborhood function, hence there are exactly n distinct neighboring solutions $\{\omega_k(1), \omega_k(2), \dots, \omega_k(n)\}$ associated with the current solution ω_k . For tabu search, define the set $\eta(\omega_k) \setminus T$, to be the candidate neighboring solutions (i.e., the set of all neighboring solutions that are not tabu). Define $\omega_{k,best}$ to be the solution with the biggest (i.e., the best) objective function value among all candidate neighboring solutions obtained from ω_k (i.e., $\omega_{k,best} = \operatorname{argmax}\{f(\omega_k(i)), \omega_k(i) \notin T, i = 1, 2, \dots, n\}$).

4.2.2. Tabu List

Three types of tabu search strategies (short-term tabu list, long-term tabu list, and aspiration criteria) are described and will be applied to MAX 3-SATISFIABILITY. It should be noted that the main difference between tabu search and random restart local search algorithms lies in the

definition of a candidate neighboring solution. In random restart local search, the word “candidate” is redundant, since every neighboring solution is a candidate neighboring solution. On the other hand, tabu search restricts the candidate neighboring solution set by defining a set of tabu (forbidden) solutions (i.e., the tabu list).

A neighboring solution $\omega_k(i)$ is defined as tabu if the Boolean variable X_i , which is to be switched in ω_k to obtain $\omega_k(i)$, is either on the short-term tabu list, Υ , or the long-term tabu list, Γ . To avoid excessive memory requirements, both lists are defined as a set of Boolean variables rather than a set of solutions. It should be noted that both the short-term and long-term tabu lists are dynamic and therefore proper updating procedures need to be defined. In particular, at the end of iteration k , Boolean variable $X_{i_{\text{best}}}$, which is switched to obtain $\omega_{k, \text{best}}$, immediately enters the short-term tabu list and stays tabu for the next L iterations (i.e., $X_{i_{\text{best}}}$ leaves the list at the end of iteration $k + L$). Note that any Boolean variable X_i may re-enter the short-term tabu list later in the algorithm’s execution. If a Boolean variable X_i enters the short-term tabu list more than F times, it is then included in the long-term tabu list. Once a Boolean variable is included in the long-term tabu list, it stays tabu for the rest of the algorithm’s execution.

Two n -tuple vectors, u and v , store all information required to verify the tabu status of the Boolean variables. When a Boolean variable X_i enters to the short-term tabu list, $u(i)$ records the current iteration counter (i.e., $u(i) = k$) and $v(i)$ records the number of times that the Boolean variable X_i has become tabu (i.e., $v(i) = v(i) + 1$). The same updating process is repeated at all iterations $k = 1, 2, \dots, K$. These two vectors makes it possible to check the tabu status of any Boolean variable X_i with just two comparisons (i.e., $X_i \in \Upsilon$ if $k \leq u(i) + L$ and $X_i \in \Gamma$ if $v(i) \geq F$). As noted earlier, a neighboring solution $\omega_k(i)$ is said to be tabu if the Boolean variable X_i is tabu (i.e., $X_i \in \Upsilon \cup \Gamma$). However if a tabu neighboring solution $\omega_k(i)$ produces a better solution than the best solution found to date (i.e., $f(\omega_k(i)) > f(\omega^*)$), then its tabu status is overridden. This describes an aspiration criterion for the algorithm.

4.2.3. Tabu Search Implementation

The tabu search algorithm starts with a randomly generated initial solution, ω_1 , and empty tabu lists, Υ and Γ . A set of n neighboring solutions $\{\omega_1(1), \omega_1(2), \dots, \omega_1(n)\}$ are obtained, where for each such solution, $\omega_1(i)$, $f(\omega_1(i))$ is evaluated, its short-term and long-term tabu status is checked (i.e., $X_i \in \Upsilon \cup \Gamma$ is checked), and the solutions ω^* and $\omega_{1, \text{best}}$ are updated. Note that during the

first iteration, none of the neighboring solutions are tabu, since both tabu lists are initialized to be empty. Once the objective function value and tabu status of all n neighboring solutions are obtained, the solution $\omega_{1,\text{best}}$ becomes the current solution for the next iteration, $k = 2$ (even if it is worse than the current solution ω_1) and the Boolean variable $X_{i,\text{best}}$ enters the short-term tabu list. This process is repeated until either: 1) K iterations have been executed, 2) all neighboring solutions have become tabu, or 3) a solution satisfying all m clauses is found. The algorithm is then terminated and the solution with the best objective function value found to date, $f(\omega^*)$, is reported. A tabu search algorithm for MAX 3-SATISFIABILITY is outlined in pseudo-code form, where short-term tabu list, long-term tabu list, and aspiration criterion are applied as described in Section 4.2.2.

```

Generate a random initial solution  $\omega_1$ 
Evaluate  $f(\omega_1)$ 
 $\omega^* \leftarrow \omega_1, f(\omega^*) \leftarrow f(\omega_1)$ 
Initialize the tabu lists,  $\Upsilon = \emptyset, \Gamma = \emptyset$ 
Set iteration counter  $k = 1$ 
Repeat
  Set Move Indicator MOVE = NO
  Set Boolean variable index  $i = 1$ 
  Repeat
    Obtain a neighboring solution  $\omega_k(i) \in \eta(\omega_k)$ 
    Evaluate  $f(\omega_k(i))$ 
    if  $f(\omega_k(i)) = m$ , then STOP and report  $\omega_k(i)$  (all  $m$  clauses are satisfied)
    if  $f(\omega_k(i)) > f(\omega^*)$ , then MOVE = YES,  $\omega^* \leftarrow \omega_k(i), f(\omega^*) \leftarrow f(\omega_k(i)),$ 
       $\omega_{k,\text{best}} \leftarrow \omega_k(i), f(\omega_{k,\text{best}}) \leftarrow f(\omega_k(i))$ 
    if  $X_i \notin (\Upsilon \cup \Gamma)$ 
      if MOVE = NO, then MOVE = YES,  $\omega_{k,\text{best}} \leftarrow \omega_k(i), f(\omega_{k,\text{best}}) \leftarrow f(\omega_k(i))$ 
      if  $f(\omega_k(i)) > f(\omega_{k,\text{best}})$ , then  $\omega_{k,\text{best}} \leftarrow \omega_k(i), f(\omega_{k,\text{best}}) \leftarrow f(\omega_k(i))$ 
     $i \leftarrow i + 1$ 
  Until  $i = n$ 
  If MOVE = NO, then STOP and report  $\omega^*, f(\omega^*)$  (all neighboring solutions are tabu)
   $\omega_{k+1} \leftarrow \omega_{k,\text{best}}, f(\omega_{k+1}) \leftarrow f(\omega_{k,\text{best}})$ 
  Update the tabu lists  $\Upsilon, \Gamma$  as described in Section 4.2.2.
   $k \leftarrow k + 1$ 
Until  $k = K$ 
Report  $\omega^*, f(\omega^*)$ 

```

4.2.4. Random Restart Local Search Implementation

Using the same notation defined previously, random restart local search starts with a random initial solution, ω_1 . The best solution, $\omega_{1,\text{best}}$, is then selected among a set of n neighboring

solutions $\{\omega_1(1), \omega_1(2), \dots, \omega_1(n)\}$. If $\omega_{1,best}$ yields an improvement over ω_1 (i.e., $f(\omega_{1,best}) > f(\omega_1)$), then $\omega_{1,best}$ becomes the current solution (i.e., $\omega_2 \leftarrow \omega_{1,best}$). This process is repeated until the algorithm reaches a solution ω_k , such that among the set of n neighboring solutions $\{\omega_k(1), \omega_k(2), \dots, \omega_k(n)\}$, none of the solutions yield an improvement over ω_k (i.e., $f(\omega_k(i)) \leq f(\omega_k)$ for all $i = 1, 2, \dots, n$). In such cases, random restart local search is then restarted with a new (randomly generated) initial solution. Random restart local search for MAX 3-SATISFIABILITY is outlined in pseudo-code form.

Set random restarts counter $r = 1$

Repeat

Generate a random initial solution $\omega_1 \in \Omega$

Evaluate $f(\omega_1)$

if $r = 1$, then $\omega^* \leftarrow \omega_1$, $f(\omega^*) \leftarrow f(\omega_1)$

Set Local Optimum Indicator LOCAL = NO

Set iteration counter $k = 1$

Repeat

Set Boolean variable index $i = 1$

Repeat

Obtain a neighboring solution $\omega_k(i) \in \eta(\omega_k)$

Evaluate $f(\omega_k(i))$

if $i = 1$, then $\omega_{k,best} \leftarrow \omega_k(i)$, $f(\omega_{k,best}) \leftarrow f(\omega_k(i))$

If $f(\omega_k(i)) = m$, then STOP and report $\omega_k(i)$ (all m clauses are satisfied)

If $f(\omega_k(i)) > f(\omega^*)$, then $\omega^* \leftarrow \omega_k(i)$, $f(\omega^*) \leftarrow f(\omega_k(i))$

If $f(\omega_k(i)) > f(\omega_{k,best})$, then $\omega_{k,best} \leftarrow \omega_k(i)$, $f(\omega_{k,best}) \leftarrow f(\omega_k(i))$

$i \leftarrow i + 1$

Until $i = n$

If $f(\omega_{k,best}) > f(\omega_k)$, then $\omega_{k+1} \leftarrow \omega_{k,best}$

If $f(\omega_{k,best}) \leq f(\omega_k)$, then LOCAL = YES

$k = k + 1$

Until LOCAL = YES

$r \leftarrow r + 1$

Until $r = R$

Report ω^* , $f(\omega^*)$

Chapter 5 :

Computational Results

This chapter reports computational experiments on randomly generated instances of 3-SATISFIABILITY. These experiments are designed to illustrate the theoretical results obtained in Section 3.3, to test and compare the performance of tabu search and random restart local search algorithms (as described in Section 4.2) on MAX 3-SATISFIABILITY, and to show how the results of probabilistic analysis in Section 3.3 can be used to obtain a stopping criterion for local search algorithms.

Section 5.1 reports results that use complete enumeration over the entire solution space (i.e., each problem is solved using all possible 2^n solutions in the solution space). Section 5.2 and Section 5.3 report results using tabu search and local search algorithms, respectively.

Randomly generated instances of 3-SATISFIABILITY are obtained by selecting m clauses (with or without replacement) from Ψ , where each clause is equally likely to be selected (see Section 3.1). A Dell Precision Model 610 computer with two 500 MHz Pentium III processors running Windows NT 4.0 was used for all the computational experiments.

5.1. Complete Enumeration Results

This section contains the results using complete enumeration over all possible solutions. Note that the probabilistic analysis in Section 3.3 needed to look at all the solutions in the solution space Ω . Moreover, to estimate the values for the expected number of solutions that satisfy k clauses (i.e., $E[S_k(m)]$, $k = 0, 1, \dots, m$) in a randomly generated instance of 3-SATISFIABILITY (with m clauses and n Boolean variables), all 2^n solutions must be considered. Due to the exponential time needed for complete enumeration, only a small number of Boolean variables (e.g., $n = 10, 15, 20$) are used when generating random instances of 3-SATISFIABILITY.

The first set of experiments illustrates the result in Corollary 3.2. In particular, for fixed values of n and m , a set of one thousand instances of 3-SATISFIABILITY was randomly generated (with replacement) and each instance was solved using complete enumeration. For each instance, a frequency distribution of all 2^n solutions (i.e., the solution space) over the number of satisfied

clauses k (i.e., the vector $s(m) = (s_0(m), \dots, s_k(m), \dots, s_m(m))$) was obtained. This data was then used to obtain estimates for $s(m)$, $\bar{s}(m) = (\bar{s}_0(m), \dots, \bar{s}_k(m), \dots, \bar{s}_m(m))$. A total of 20 experiments were executed using different values for m and n . The CPU time for these experiments depended on the values of m and n . For example, for $n = 10$, each experiment took less than one minute. For $n = 15$, the experiments took between 40 and 70 minutes (depending on the value of m). For $n = 20$, experiments took between 10 and 20 hours. Tables A.1.1 - A.1.20 list the estimated values of the minimum, the maximum, the average and the standard deviation of the number of solutions that satisfy $k = 0, 1, \dots, m$ clauses along with the theoretical values (for the average number of solutions) obtained using Corollary 3.2.

The results in Tables A.1.1 - A.1.20 are consistent with the theoretical results obtained from Corollary 3.2. To gain some insights into the marginal distribution of the random variable $S_m(m)$ (i.e., number of solutions that satisfy all m clauses), data obtained from complete enumeration was fit to a probability distribution using the Arena Input Analyzer. The resulting (best fit) probability distributions have small p values (i.e., $p < 0.005$) indicating a poor fit. Moreover, the form for these probability distributions changed as m and n changed. For example, for $n = 10$ and $m = 30$, $S_m(m)$ follows a Weibull distribution while for $n = 10$ and $m = 35$, $S_m(m)$ follows a Beta distribution.

The second set of experiments was designed to estimate the distribution of the globally optimal objective function values, for randomly generated instances of 3-SATISFIABILITY (with the same size). In particular, for fixed values of n and m , a set of five hundred instances of 3-SATISFIABILITY was randomly generated (with replacement) and each instance was solved using complete enumeration. For each instance, the maximum number of satisfied clauses, $MAX(n,m)$, was found (i.e., a solution to MAX 3-SATISFIABILITY was obtained). A total of 26 experiments were executed using several different values for n and m . The CPU time for the experiments depended on the values of n and m . For example, for $n = 10$, each experiment took less than two minutes. For $n = 20$, each experiment took between 10 and 57 hours (depending on m). Table A.2.1 and Table A.2.2 list the frequency distribution (for each value of n and m) of the five hundred instances over the maximum number of satisfied clauses, $MAX(n,m)$.

Table A.2.1 and Table A.2.2 suggest that for a fixed value of n and m , the $MAX(n,m)$ values are concentrated over a small range of values, for randomly generated instances of 3-SATISFIABILITY. To better explain this situation, consider the case $n = 20$ and $m = 120$ in Table A.2.2. For 95% of the five hundred randomly generated instances, $MAX(n,m)$ is between

117 and 119. Moreover, in most cases, the probability distribution of the maximum number of satisfied clauses $\text{MAX}(n,m)$ appears to be symmetric. These experiments provide useful information on the distribution of globally optimal objective function values for randomly generated instances of 3-SATISFIABILITY (with the same size). For example, consider the case $n = 20$ and $m = 600$ in Table A.2.2. Note that for 95% of the instances, the globally optimal objective function values (i.e., $\text{MAX}(n,m)$) are between 280 and 273. This means that it is very unlikely (likely) to find a solution that satisfies more than 280 (273) clauses in a randomly generated instance of 3-SATISFIABILITY (with $n = 20$ and $m = 300$). Such information may be useful for defining a stopping criterion for local search algorithms. For example, an algorithm may be terminated when it finds a solution satisfying 280 clauses in a randomly generated 3-SATISFIABILITY instance (with $n = 20$ and $m = 300$). This approach is further discussed and analyzed in Section 5.4.

The third set of experiments was designed to gain insight into the threshold conjecture. In particular, for fixed values of n and m , a set of one thousand instances of 3-SATISFIABILITY was randomly generated (without replacement) and solved using complete enumeration. For each instance, a “yes/no” answer to 3-SATISFIABILITY was obtained. If a “yes” answer was obtained before all solutions in the solution space were evaluated, then the enumeration was halted. Such a stopping rule reduced the CPU time for these experiments (especially for small values of $c = m/n$). A total of thirty-two experiments were conducted using several different values for n and m . For $n = 10$, each experiment took less than one minute, while for $n = 20$, the experiments took between 50 minutes and 23 hours (depending on the value of m). Table A.3 lists the percentage of satisfiable instances (i.e., the probability that a randomly generated instance of 3-SATISFIABILITY is satisfiable).

From Table A.3, it can be seen that the randomly generated instances of 3-SATISFIABILITY are satisfiable with high (low) probability when $c = m/n$ is small (large). In particular, when $c = m/n = 3$, almost all of the instances are satisfiable. Similarly, when $c = m/n = 6$, almost all of the instances are unsatisfiable. This result suggest that the probability that a randomly generated instance of 3-SATISFIABILITY is satisfiable shifts from one to zero as $c = m/n$ increases. Note that the threshold conjecture refers to this phenomenon and suggest that there exists a constant c^* such that when $m/n = c^*$, the probability that a randomly generated instance of 3-SATISFIABILITY is satisfiable almost instantly shifts from one to zero if $c = m/n$ is increased.

The fourth set of experiments was designed to gain insight into the distribution of the solution space (i.e., all possible 2^n solutions) over the number of satisfied clauses, $k = 0, 1, \dots, m$. In particular, for fixed values of n and m , a set of ten instances of 3-SATISFIABILITY was randomly generated (with replacement) and each instance was solved using complete enumeration over all 2^n solutions. For each instance, a frequency distribution of all 2^n solutions (i.e., the solution space) over the number of satisfied clauses, $k = 0, 1, \dots, m$ (i.e., the vector $s(m) = (s_0(m), \dots, s_k(m), \dots, s_m(m))$) was obtained. This vector was then tested for normality using Geary's test, which is known to be more powerful than the chi-squared goodness-of-fit test when testing normality (Walpole and Myers 1993, page 347). To describe Geary's test for normality, consider a random sample y_1, y_2, \dots, y_T taken from a normal distribution, $N(\mu, \sigma)$ and let

$$U = \frac{(V-1)\sqrt{T}}{0.2661}, \text{ where } V = \frac{\sqrt{\pi/2} \sum_{i=1}^T |y_i - \bar{y}| / T}{\sqrt{\sum_{i=1}^T (y_i - \bar{y})^2 / T}}. \text{ Then the goodness-of-fit test p-value for this}$$

sample is $p = 2 * (1 - P\{Z \geq U\})$, where Z is a standard normal random variable. Note that each randomly generated instance defines a random sample of size $T = 2^n$, where $y_t, t = 1, 2, \dots, T$, is the number of satisfied clauses for solution $\omega_t \in \Omega$ in each instance. Note also that the sample size T can be reduced by randomly selecting a subset, $\Omega' \subseteq \Omega$, from the solution space and observing the frequency distribution of Ω' over the number of satisfied clauses $k = 0, 1, \dots, m$. Law and Kelton (1991, page 382) suggests that goodness-of-fit tests based on large sample sizes almost always yield small p-values (hence indicating a poor fit). A total of twenty-one experiments were conducted using several different values for n and m . These experiments took between 10 seconds and 4 minutes (depending on the values of n and m). For each value of n and m , Tables A.4.1 – A.4.3 list the sample averages (i.e., the average number of satisfied clauses), the sample standard deviations (along with the U statistic) and the p-values for all ten instances (for both $T = 2^n = |\Omega|$ and $T = |\Omega'|$). In addition, for selected values of n and m , the frequency distribution of the solution space over the number of satisfied clauses is depicted in Figures A.2.1 - A.2.6 for all ten instances.

Tables A.4.1 - A.4.3 suggest that when $n = 10$ and $T = 2^n$ (i.e., the entire solution space is taken as the sample), most of the instances have large p-values, hence indicating a good fit to a normal distribution. On the other hand, when $n = 15, n = 20$ and $T = 2^n$, the p-values for almost all the instances are very close to zero (hence indicating a poor fit to a normal distribution). Also note that the sample size, T , is very big when $n = 15$ and $n = 20$ (e.g., $T = 2^{20} = 1048576$), hence these

results may be due to the fact that the sample is too large (as noted in Law and Kelton 1991). Moreover, when the sample size, T , is reduced (e.g., $T = 100$ for $n = 10$, $T = 1000$ for $n = 15$ and $T = 10000$ for $n = 20$), then the p-values are larger (hence indicating a good fit to a normal distribution). Lastly, Figures A.2.1 – A.2.6 suggest that the frequency distribution of the solution space over the number of satisfied clauses is normally distributed.

5.2. Tabu Search Results

This section reports results with the tabu search algorithm described in Section 4.2.3. These experiments are designed to test the performance of tabu search algorithm and to illustrate a stopping criterion for local search algorithms based on the results from the experiments using tabu search.

Note that the tabu search algorithm requires several parameters (e.g., tabu list parameters L , F and aspiration criteria that are explained in Section 4.2.2) to be set before the algorithm is executed. The best parameter values were obtained by first executing several different parameter values over a small set of randomly generated instances. During this preliminary analysis, it was observed that the length of short-term tabu list (i.e., the value for L) is the most critical factor affecting the performance of the tabu search algorithm. The use of long-term tabu list and/or aspiration criteria did not affect the algorithm's performance, particularly when the value for L was near its optimum value. Moreover, when the value for L was small or large compared to its optimum value, the use of long-term tabu list and/or aspiration criteria resulted in only a slight improvement in performance.

The first set of experiments was designed to test the performance of tabu search on the MAX 3-SATISFIABILITY problem. In particular, for fixed values of n and m , a set of five hundred instances of 3-SATISFIABILITY was randomly generated (with replacement) with tabu search (using $L = 4$, and $K = 200$ iterations) applied to each instance. A frequency distribution for the maximum number of satisfied clauses, $\text{MAX}(n,m)$, was obtained. The same set of five hundred instances of 3-SATISFIABILITY was then solved using complete enumeration. A total of ten such experiments were executed using $n = 20$ and several different values for m . Each experiment took between 1 and 10 minutes (depending on m) for tabu search, while each experiment took between 10 and 57 hours for complete enumeration. The resulting frequency distributions (for both tabu search and complete enumeration) are reported in Table B.1. To better explain the results reported in Table B.1, consider the case $n = 20$ and $m = 60$. Tabu search

reports that $0.944 * 500 = 472$ instances are satisfiable, with the remaining 18 instances unsatisfiable, while complete enumeration shows that 498 instances are satisfiable, with just 2 instances unsatisfiable.

The results in Table B.1 suggest that the $MAX(n,m)$ values obtained using tabu search algorithm are very close to their actual values for the most of the instances. For example, for case $n = 20$ and $m = 60$, the actual value of $MAX(n,m)$ (i.e., the global optimum) is either 60 or 59 for all five hundred instances. Moreover, tabu search was able to get within one clause of the optimal solution for all five hundred instances. Tabu search also correctly found the globally optimal solutions for 95% of the instances. Similarly, for the case with $n = 20$ and $m = 110$, the optimal solution errors were always within 5 clauses. Note that these errors were usually much smaller (if not zero) for the most of the instances. In addition, tabu search was able to find the optimal solution for those instances with the highest globally optimal values. For example, when $n = 20$ and $m = 350$, there was a single instance with $MAX(n,m) = 337$ and tabu search was able to solve this instance.

The second set of experiments was designed to compare the effectiveness of tabu search and random restart local search algorithms. The results of these experiments were also used for deriving a stopping criterion for these algorithms. In particular, for fixed values of n and m , two different sets of instances of 3-SATISFIABILITY (with each set containing one hundred instances) were randomly generated (with replacement). Tabu search (with random restarts) was then applied to each instance. Note that tabu search, as described in Chapter 4, does not use random restarts. However, by their very nature, the use of random restarts may improve the performance of tabu search, since it provides an independent sample of solutions, rather than a single solution. In particular, it was observed that the best solution is typically found early in the algorithm's execution, hence making the rest of the iterations unnecessary. Moreover, restarting the algorithm from a randomly generated initial solution is an effective way of escaping from local optima. A total of forty-seven experiments were executed using several different values of n and m . Each experiment took between 10 minutes and 15 hours (increasing linearly with n and m). For each experiment, twenty restarts were executed, where each restart was run for $K = 500$ iterations. Tables B.2.1 - B.2.6 list (for each value of n and m) the minimum, the maximum, the average and the standard deviation of the maximum number of satisfied clauses, $MAX(n,m)$, taken over the one hundred instances. In addition, the average number of iterations that were executed until tabu search became trapped in a local optimum was also obtained and listed in

Tables B.2.1 - B.2.6. This provides information on how quickly tabu search algorithm found (and became trapped in) a local optimum.

Tables B.2.1 - B.2.6 suggest that less than 500 iterations can be executed without affecting the performance of tabu search. Note that when $n = 50$, tabu search found a local optima after 70 iterations (on average), with no further improvement made during the rest of the iterations. For $n = 100$ and $n = 150$, it took longer for tabu search to find a local optimum. Moreover the average number of iterations to find a local optimum is more affected by n than by m . Also note that for each value of n and m , the two sets of one hundred randomly generated instances of 3-SATISFIABILITY yielded comparable results.

The third set of experiments was designed to observe how some important statistics such as the minimum, the maximum, the average and the standard deviation of the maximum number of satisfied clauses, $MAX(n,m)$, change as m increases. In particular, for fixed values of n and m , tabu search was applied to a set of one hundred randomly generated (without replacement) instances of 3-SATISFIABILITY. Due to the excessive time requirements, $K = 200$ iterations were executed without restarting. A total of one hundred and ten experiments were executed using $n = 20$ and several different values of m (ranging from 5 to 9100). Note that for $n = 20$, there are $8 \binom{20}{3} = 9120$ clauses in Ψ . The experiments took a total of 26 hours to run. Table B.3 contains (for each value of m) the minimum, the maximum, the average and the standard deviation of the maximum number of satisfied clauses, $MAX(n,m)$, taken over the one hundred instances.

Table B.3 provides information on how $MAX(n,m)$ is distributed for randomly generated instances of 3-SATISFIABILITY. Note that for small values of m , the range of $MAX(n,m)$ (i.e., the difference between the maximum and minimum values) is also small. As m increases, the range of $MAX(n,m)$ first slowly increases, then starts to fluctuate after some value of m , and finally starts to decrease. The same observation holds true for the standard deviation.

5.3. Random Restart Local Search Results

This section reports computational results with random restart local search. These experiments focused on comparing the effectiveness of tabu search with random restart local search for the MAX 3-SATISFIABILITY problem near the threshold values. In particular, for fixed values of n

and m , a set of one hundred instances of 3-SATISFIABILITY was randomly generated. Random restart local search was applied to each such instance, first with five hundred restarts and then with one thousand restarts. A total of eleven experiments were executed using several different values of n and m . Each experiment took between 2 hours and 16 hours (depending on n , m and the number of restarts). Tables C.1.1 - C.1.4 contain (for each value of n and m) the minimum, the maximum, and the average and the standard deviation of the maximum number of satisfied clauses, $MAX(n,m)$, taken over the one hundred instances.

Tables C.1.1 - C.1.4 and Tables B.2.1 - B.2.6 suggest that tabu search and random restart local search yield similar results in terms of the minimum, the maximum, the average and the standard deviation of the maximum number of satisfied clauses. Note that random restart local search yielded slightly better results when $n = 50$, while tabu search yielded slightly better results when $n = 100$. Assuming that the optimal values of the tabu list parameters (i.e., L , F and the aspiration criteria) are used, tabu search required considerably less computing time than the random restart local search (e.g., when $n = 50$ and $m = 150$, tabu search took 10 minutes while random restart local search took 2 hours). Therefore, given a limited computing budget, tabu search is significantly more efficient than random restart local search. However, the computational results also suggest that if sufficient computing time is available, a simple algorithm like random restart local search performs as well as a more sophisticated algorithm like tabu search. This is consistent with the theoretical results reported in Jacobson and Yücesan (2001).

Note that from Tables C.1.1 - C.1.4, increasing the number of restarts yields only a modest improvement in the results. This means that the marginal value of increasing the number of restarts decreases after a sufficiently large number of restarts have been executed.

5.4. Stopping Criterion for Local Search Algorithms

This section describes a stopping criterion for local search algorithms using the expected value vector $(E[S_0(m)], E[S_1(m)], \dots, E[S_k(m)], \dots, E[S_m(m)])$ defined in Section 3.3. To describe this stopping criterion, several definitions are needed. Consider m clauses $C_1, C_2, \dots, C_m \in \Psi$ randomly generated (with or without replacement) using a set of n Boolean variables. Let $\omega \in \Omega$ denote a

solution for these m clauses. Define $\theta_m(\omega) = \sum_{j=1}^m C_j(\omega)$ for $m \geq 1$, where

$$C_j(\omega) = \begin{cases} 1 & \text{if } C_j \text{ is satisfied for solution } \omega \\ 0 & \text{otherwise} \end{cases}.$$

Note that if identical clauses are permitted (i.e., clauses are generated with replacement, hence can be selected more than once), then by Theorem 3.1, $\theta_m(\omega)$ is a binomial random variable with parameters m and $p = 7/8$. If identical clauses are not permitted (i.e., clauses are generated without replacement, hence cannot be selected more than once), then by Theorem 3.2, $\theta_m(\omega)$ is a

hypergeometric random variable with parameters $8y$, m and $7y$, where $y = 8 \binom{n}{3}$. Label the

solutions in Ω as ω_t , $t = 1, 2, \dots, 2^n = |\Omega|$. For each solution, $\omega_t \in \Omega$, and a value $k = 0, 1, \dots, m$, define the indicator random variable,

$$I_{t,k} = \begin{cases} 1 & \text{if } \theta_m(\omega_t) \geq k \\ 0 & \text{otherwise} \end{cases} \quad t = 1, 2, \dots, |\Omega|, \quad k = 0, 1, \dots, m.$$

Then for each $k = 0, 1, \dots, m$, the number of solutions that satisfy k or more clauses is the random

variable $S_k^+(m) = \sum_{t=1}^{2^n} I_{t,k}$, where $S_k^+(m) = 0, 1, \dots, |\Omega|$. Moreover,

$$E[S_k^+(m)] = E\left[\sum_{t=1}^{2^n} I_{t,k}\right] = 2^n P\{\theta_m(\omega_t) \geq k\}. \quad (5.1)$$

By Theorem 3.4,

$$Z = \frac{\theta_m(\omega) - (7m/8)}{\sqrt{(7m/64)}} \rightarrow_D N(0, 1) \text{ as } n \rightarrow +\infty, m \rightarrow +\infty \text{ and } m/n^3 \rightarrow 0.$$

Therefore,

$$E[S_k^+(m)] = 2^n P\{\theta_m(\omega_t) \geq k\} = 2^n P\{Z \geq z\}, \text{ where } z = \frac{k - (7m/8)}{\sqrt{(7m/64)}}.$$

For a fixed value of n , let z^* denote the real value such that $P\{Z \geq z^*\} = 2^{-n}$. Moreover:

if $z > z^*$, then $E[S_k^+(m)] \rightarrow 0$ as $z \rightarrow +\infty$.

Using Markov's inequality, $P\{S_k^+(m) \geq 1\} \leq E[S_k^+(m)]$, hence if $z > z^*$, then $P\{S_k^+(m) \geq 1\} \rightarrow 0$ as $z \rightarrow +\infty$. Note that the random variable $S_k^+(m)$ is defined as the number of solutions that satisfy k or more clauses. Recall that $\text{MAX}(n, m)$ denotes the maximum number of satisfied clauses in a randomly generated instance of 3-SATISFIABILITY. Therefore, if $z > z^*$, then $P\{S_k^+(m) \geq 1\} = P\{\text{MAX}(n, m) \geq k\} \rightarrow 0$ as $z \rightarrow +\infty$, hence

$$k^* = \frac{(z^* \sqrt{7m}) + 7m}{8} \quad (5.2)$$

provides an upper bound (estimate) for $\text{MAX}(n,m)$, the maximum number of satisfied clauses.

Note that obtaining the value for z^* may be difficult when n is large, since there is no closed form expression for the probability distribution function of a standard normal random variable. Mathematica 4.0 includes a built-in function for this distribution. However, this function is unable to compute the value of z^* for $n > 50$. To circumvent this problem, the following approximation can be used:

$$P\{Z \geq z^*\} \approx \frac{1}{\sqrt{2\pi}} e^{-\frac{z^{*2}}{2}}, \text{ hence } \frac{1}{2^n} = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^{*2}}{2}} \text{ and } z^* = \sqrt{2nLn(2) - Ln(2\pi)}.$$

Table 5.1 contains values for z^* .

Table 5.1: Values for z^*

n	z^*
10	3.10
20	4.76
50	7.96
100	11.70
150	14.36
200	16.60

For fixed values of n and m , Table 5.1 and (5.2) provide an estimate for the upper bound on the maximum number of satisfied clauses, $\text{MAX}(n,m)$, for randomly generated instances of 3-SATISFIABILITY. For example, for $n = 20$ and $m = 400$, Table 5.1 and (5.2) yield $z^* = 4.763$ and $k^* = \frac{(4.763\sqrt{2800}) + 2800}{8} \approx 381$, respectively. This means that for a randomly generated instance of 3-SATISFIABILITY with $n = 20$ and $m = 400$, it is unlikely to find a solution $\omega \in \Omega$ that satisfy more than 381 clauses (i.e., $P\{\text{MAX}(n,m) \geq 382\} \approx 0$). Note that from Table A.2.2, only 1.8% of five hundred instances (with $n = 20$ and $m = 400$) resulted in $\text{MAX}(n,m) = 382$ (i.e., $P\{\text{MAX}(n,m) \geq 382\} = 0.018$). Moreover, the results in Table A.2.1 and Table A.2.2 were obtained using complete enumeration (i.e., an exact solution was obtained for each instance), hence provide upper bounds for local search algorithms executed on the same instances. Therefore, for a given value of n and m , the value for k^* can be computed and used as a stopping criterion for local search algorithms. More specifically, local search algorithm can be terminated when a solution, $\omega \in \Omega$, that satisfies k^* (or more) clauses is found during the algorithm's

execution. Note that k^* is typically a real value, hence must be rounded to the next highest integer.

Using k^* as a stopping criterion for a local search algorithm provides a reasonable guideline to address the hypothesis that the algorithm has been executed sufficiently long, hence can be terminated. For example, consider a local search algorithm being executed on a randomly generated instance of 3-SATISFIABILITY with $n = 20$ and $m = 400$. If the algorithm finds a solution $\omega \in \Omega$ that satisfies $k^* = 381$ or more clauses during the execution, then it may be reasonable to terminate the algorithm execution, particular of no better solution has been found for several iterations. Moreover, Table A.2.2 suggests that when $n = 20$ and $m = 400$, $k^* = 381$ can be used as a stopping criterion with 98% confidence, since $P\{\text{MAX}(n,m) > k^*\} \approx 0.02$ in this case. Table 5.2 contains the values for k^* (i.e., the stopping values) for several different values of n and m . Computed values for $P\{\text{MAX}(n,m) > k^*\}$ (based on the complete enumeration results from Table A.2.1 and Table A.2.2) are given in Table 5.2.

Table 5.2: Values for k^*

n = 10			n = 20		
m	k^*	$P\{\text{MAX}(n,m) > k^*\}$	m	k^*	$P\{\text{MAX}(n,m) > k^*\}$
30	32	0	60	65	0
35	37	0	70	75	0
40	42	0	80	85	0
42	44	0	85	89	0
45	47	0	90	94	0
50	51	0	100	104	0
55	56	0	110	113	0
60	61	0	120	123	0
175	167	0.020	350	336	0.002
200	190	0.022	400	381	0.018
225	213	0.012	450	428	0.004
250	235	0.024	500	473	0.004
300	281	0.008	600	564	0.012

Note that from Table 5.2, for small values of $c = m/n$ (e.g., $c \leq 6$), the values for k^* are greater than m . This suggests that the local search algorithm should be executed long enough to find a solution $\omega \in \Omega$ that satisfies all m (or close to m) clauses. Table 5.2 also suggests that using k^* values as a stopping criterion prevents the algorithm from being terminated prematurely (note that $P\{\text{MAX}(n,m) > k^*\}$, hence the probability that the algorithm terminates early is small in most cases).

For a fixed value of n , the stopping criterion treats z^* as a constant to obtain values for k^* . On the other hand, tabu search experiments (i.e., the second set of experiments described in Section 5.2) suggest that the z^* values change as m changes (for a fixed value of n), hence should be treated as a function of m . Figures D.1 – D.3 illustrate this observation (for several different values of n and m) by comparing the z values obtained from Table B.2.1, Table B.2.3 and Table B.2.5 (i.e., tabu search experiments) to the z^* values listed in Table 5.1. Figures D.1 – D.3 show that the z values obtained from tabu search experiments are much smaller than the z^* values listed in Table 5.1, when $c = m/n$ is small (e.g, $c \leq 20$ for $n = 100$). As $c = m/n$ increases (hence m increases, since n is fixed), the z values obtained from the tabu search experiments move closer to the z^* values. In general, the tabu search experiments suggest that for a fixed value of n , the following k^* values should be used for defining a stopping criterion for the tabu search algorithm:

$$k^* = \frac{(z^+ \sqrt{7m}) + 7m}{8}, \text{ where } z^+ = z^* - \kappa(m) \text{ and } \kappa(m) \text{ is a positive valued function such that } \kappa(m) \rightarrow 0 \text{ as } m \rightarrow M \in \mathbb{Z}^+.$$

Note that the function $\kappa(m)$ appears to change as n changes, hence making it difficult to estimate the z^+ values for tabu search algorithm. On the other hand, the tabu search experiments suggest that using the z^* values listed in Table 5.1 to obtain a stopping criterion usually results in the k^* values being slightly higher than necessary. This may prevent the algorithm from terminating prematurely. However it may also result in the algorithm executing longer than necessary. Several strategies can be used to obtain a better stopping criterion for local search algorithms. One of such strategy is to iteratively observe the likelihood of finding a solution that satisfies more clauses than the best solution found to date. For example, consider a local search algorithm being executed on a given instance of 3-SATISFIABILITY defined with n Boolean variables and m clauses. After a small number of iterations have been executed, the algorithm can be temporarily stopped and the best solution found to date could be obtained. Assuming that the best solution results in k clauses being satisfied, then the expected number of solutions that satisfy more than k clauses (i.e., $E[S_{k+1}^+]$) can be computed using equation (5.1). If $E[S_{k+1}^+]$ is sufficiently small (big) compared to a pre-specified value, then the algorithm can be terminated (restarted).

An alternative strategy is to execute the local search algorithm for a pre-specified number of iterations without using any pre-defined stopping criterion. After a pre-specified number of

iterations have been executed, the algorithm can be temporarily stopped and a stopping criterion (i.e., a k value to terminate the algorithm) can be defined by comparing the best solution found to date and the k^* value given by (5.2). The algorithm can then be restarted with this additional stopping criterion. Note that these two strategies iteratively adjust the stopping criterion using information from the local search algorithm, hence may be useful for defining an effective stopping criterion for such algorithms.

Chapter 6 :

Conclusion and Future Research

This research probabilistically studied and analyzed randomly generated instances of 3-SATISFIABILITY to obtain information on the structure of the solution space Ω . Several structural properties of randomly generated instances of 3-SATISFIABILITY were explored, including the probability that a given solution $\omega \in \Omega$ satisfies a pre-specified number of clauses in a randomly generated instance of 3-SATISFIABILITY defined with n Boolean variables and m clauses. In Section 3.2, two random variables $f_m(\omega)$ and $h_m(\omega)$ were defined and their properties analyzed. The random variable $f_m(\omega)$, defined for randomly generated instances of 3-SATISFIABILITY with replacement (i.e., identical clauses are allowed), was shown to be a binomial random variable with parameters m and $p = 7/8$. The random variable $h_m(\omega)$, defined for randomly generated instances of 3-SATISFIABILITY without replacement (i.e., identical clauses are not allowed), was shown to be a hypergeometric random variable with parameters $8\binom{n}{3}$, m and $7\binom{n}{3}$. In addition, a Central Limit Theorem was proven for both these random variables.

This research also analyzed all 2^n solutions in the solution space Ω for randomly generated instances of 3-SATISFIABILITY. In particular, a matrix structure and a random vector, $\mathbf{S}(\mathbf{m}) = (S_0(m), S_1(m), \dots, S_k(m), \dots, S_m(m))$, were defined in Section 3.3 to analyze the solution space Ω . The random vector $\mathbf{S}(\mathbf{m})$ denotes the distribution of the solution space Ω over the number of satisfied clauses, $k = 0, 1, \dots, m$, hence provides valuable information on the structure of the solution space Ω , for randomly generated instances of 3-SATISFIABILITY. Mathematical expressions for the expected value of $\mathbf{S}(\mathbf{m})$ (i.e., $(E[S_0(m)], E[S_1(m)], \dots, E[S_m(m)])$) were obtained using the matrix structure defined in Section 3.3 and the results from Section 3.2. In addition, a phenomenon, termed the threshold conjecture, was described and analyzed using the results from Section 3.2 and Section 3.3. This analysis yielded interesting and curious observations that need further attention by researchers working on this problem.

In addition to the probabilistic studies, this research also described how tabu search and random restart local search algorithms could be implemented on MAX 3-SATISFIABILITY. Moreover,

several computational experiments were conducted to illustrate the results obtained in Chapter 3, to computationally observe the distribution of the solution space Ω over the number of satisfied clauses, and to compare the effectiveness of tabu search and random restart local search algorithms. Lastly, several strategies were suggested for defining a stopping criterion for local search algorithms, using the computational experiments and the expected value vector $(E[S_0(m)], E[S_1(m)], \dots, E[S_k(m)], \dots, E[S_m(m)])$.

This research suggests several new directions for future studies. In particular, obtaining the joint probability distribution of the random vector $\mathbf{S}(\mathbf{m})$ defined in Section 3.3 is an important research question that requires further research. If such a distribution could be obtained, both 3-SATISFIABILITY and MAX 3-SATISFIABILITY can be addressed more efficiently. For example, using the joint probability distribution of the random vector $\mathbf{S}(\mathbf{m})$, the threshold conjecture could then be addressed analytically. Moreover, the probability that a randomly generated instances of 3-SATISFIABILITY will result in the maximum number of k clauses being satisfied can be obtained for a given value of n and m (e.g., $P\{(S_0(m) \geq 0, \dots, S_{m-2}(m) \geq 0, S_{m-1}(m) \geq 1, S_m(m) = 0\} = P\{\text{MAX}(n, m) = m-1\}$). On the other hand, the dependency between the random variables $S_k(m)$, $k = 0, 1, \dots, m$, contained in $\mathbf{S}(\mathbf{m})$ makes such a probabilistic analysis very challenging. Further research into properties of the random vector $\mathbf{S}(\mathbf{m})$ using the matrix structure defined in Section 3.3 may shed new lights into such questions. Moreover, additional experiments could be executed to better explore the distribution of the solution space over the number of satisfied clauses.

Lastly, the analysis of the threshold conjecture described in Section 3.4 also suggests several new directions for future research. In particular, the curious observations made during this analysis show that there may be relationships between randomly generated instances of 2-SATISFIABILITY, 3-SATISFIABILITY, ..., and l -SATISFIABILITY. Insights into the mathematical basis for these observations, focusing on structural similarities between randomly generated instances of 2-SATISFIABILITY and 3-SATISFIABILITY, may lead to a solution to the threshold conjecture.

Appendix A

Table A.1.1: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 10, m = 30					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 13	0	0	0	0	0.00*
14	0	1	0.04	0.00*	0.00*
15	0	1	0.04	0.00*	0.00*
16	0	2	0.10	0.01	0.00*
17	0	5	0.31	0.05	0.02
18	0	8	0.82	0.22	0.12
19	0	11	1.68	0.74	0.52
20	0	20	3.18	2.38	1.98
21	0	33	5.64	7.02	6.61
22	0	57	8.57	19.57	18.93
23	4	80	10.79	46.42	46.09
24	50	128	11.33	93.37	94.10
25	98	226	20.97	155.84	158.08
26	124	304	28.45	212.03	212.80
27	146	297	24.38	219.46	220.69
28	119	235	16.73	166.40	165.51
29	19	162	20.38	81.26	79.90
30	1	76	11.24	19.24	18.64

* denotes a value below .005

For each n and m, 1000 instances are randomly generated (with replacement)

k = # of Clauses That are Satisfied

Minimum = Minimum # of Solutions that Satisfy k Clauses

Maximum = Maximum # of Solutions that Satisfy k Clauses

St.Dev = Sample Standard Deviation of # of Solutions that Satisfy k Clauses

Average = Average # of Solutions that Satisfy k Clauses

Expected = Expected # of Solutions that Satisfy k Clauses (Corollary 3.2)

Table A.1.2: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 10, m = 35					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 19	0	0	0	0	0.00*
20	0	2	0.11	0.01	0.01
21	0	4	0.22	0.03	0.03
22	0	6	0.58	0.15	0.15
23	0	8	1.26	0.58	0.58
24	0	17	2.77	2.12	2.02
25	0	33	5.10	6.46	6.21
26	0	49	8.09	17.21	16.73
27	13	71	9.59	39.56	39.04
28	47	119	10.44	77.88	78.08
29	85	188	16.03	131.03	131.93
30	105	266	24.38	184.36	184.71
31	113	290	25.77	207.80	208.54
32	112	233	16.69	181.60	182.47
33	71	177	16.44	116.58	116.12
34	12	112	15.52	49.08	47.81
35	0	101	7.61	9.56	9.56

Table A.1.3: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 10, m = 40					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 21	0	0	0	0	0.00*
22	0	1	0.05	0.00*	0.00*
23	0	2	0.11	0.01	0.00*
24	0	2	0.19	0.03	0.01
25	0	6	0.50	0.10	0.04
26	0	8	0.95	0.30	0.17
27	0	14	1.81	0.88	0.61
28	0	20	3.10	2.41	1.98
29	0	26	5.16	6.12	5.73
30	0	36	7.34	14.90	14.72
31	5	70	9.86	33.48	33.24
32	36	106	10.26	65.02	65.43
33	68	161	14.03	109.59	111.04
34	95	241	23.37	159.55	160.02
35	99	268	26.24	191.17	192.03
36	101	246	20.68	186.54	186.69
37	97	183	15.06	140.75	141.28
38	33	156	17.39	79.09	78.08
39	1	99	12.42	28.88	28.03
40	0	30	5.00	5.21	4.90

* denotes a value below .005

Table A.1.4: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 10, m = 45					
k	Minimum	Maximum	Std.Dev	Average	Expected
0,...,27	0	0	0	0	0.00*
28	0	3	0.14	0.01	0.01
29	0	3	0.28	0.05	0.05
30	0	8	0.61	0.18	0.18
31	0	9	1.30	0.63	0.62
32	0	14	2.64	2.03	1.90
33	0	28	4.42	5.39	5.23
34	0	43	6.63	13.40	12.91
35	6	57	8.11	29.12	28.41
36	23	81	9.25	55.52	55.25
37	60	132	11.16	92.41	94.07
38	82	209	17.28	137.94	138.62
39	89	259	22.82	174.06	174.17
40	110	256	22.06	182.25	182.88
41	107	204	15.47	156.72	156.11
42	59	159	13.92	103.89	104.08
43	14	103	13.88	51.22	50.83
44	0	71	9.09	16.72	16.17
45	0	22	2.84	2.46	2.52

* denotes a value below .005

Table A.1.5: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n =10, m = 50					
k	Minimum	Maximum	Std.Dev	Average	Expected
0,....,30	0	0	0	0	0.00*
31	0	1	0.03	0.00*	0.00*
32	0	4	0.19	0.02	0.01
33	0	4	0.35	0.07	0.05
34	0	4	0.61	0.20	0.19
35	0	7	1.17	0.61	0.61
36	0	12	2.15	1.70	1.78
37	0	20	3.90	4.58	4.73
38	0	33	6.00	11.35	11.32
39	4	51	8.25	24.72	24.38
40	19	77	9.24	47.36	46.92
41	52	122	10.86	80.58	80.11
42	81	161	14.34	119.68	120.17
43	106	225	20.20	155.67	156.50
44	114	256	22.00	174.13	174.28
45	106	225	17.83	162.93	162.67
46	77	170	12.78	124.06	123.77
47	31	118	13.43	73.21	73.73
48	3	73	11.32	32.08	32.26
49	0	36	6.36	9.44	9.22
50	0	15	2.43	1.63	1.29

* denotes a value below .005

Table A.1.6: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 10, m = 55					
k	Minimum	Maximum	Std.Dev	Average	Expected
0,...,33	0	0	0	0	0.00*
34	0	3	0.12	0.01	0.00*
35	0	3	0.16	0.01	0.00*
36	0	3	0.21	0.03	0.02
37	0	5	0.54	0.13	0.06
38	0	9	1.00	0.35	0.19
39	0	10	1.69	0.88	0.59
40	0	17	2.81	2.06	1.66
41	0	22	4.12	4.65	4.25
42	0	33	6.17	10.33	9.91
43	3	51	7.65	20.98	20.98
44	14	69	8.89	40.07	40.05
45	39	108	10.02	68.15	68.53
46	57	143	14.15	103.43	104.28
47	83	215	19.65	138.76	139.78
48	92	238	21.25	162.32	163.08
49	98	232	19.36	162.18	163.08
50	90	189	15.00	136.60	136.99
51	60	142	13.60	94.58	94.01
52	8	92	12.74	51.05	50.62
53	0	64	9.60	20.90	20.06
54	0	31	4.89	5.76	5.20
55	0	8	1.39	0.77	0.66

* denotes a value below .005

Table A.1.7: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 10, m = 60					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 37	0	0	0	0	0.00*
38	0	2	0.07	0.00*	0.00*
39	0	3	0.22	0.02	0.00*
40	0	4	0.34	0.05	0.02
41	0	4	0.45	0.10	0.06
42	0	10	1.11	0.34	0.19
43	0	11	1.61	0.79	0.57
44	0	17	2.72	1.92	1.53
45	0	23	3.99	4.28	3.80
46	0	28	5.46	9.24	8.68
47	2	41	7.17	18.27	18.10
48	12	56	8.11	34.25	34.32
49	33	92	9.47	58.25	58.83
50	50	127	12.16	90.13	90.60
51	56	171	17.57	122.92	124.36
52	85	224	20.48	149.76	150.66
53	96	228	19.76	159.34	159.19
54	99	187	16.34	143.59	144.45
55	62	172	13.38	109.99	110.31
56	33	120	13.57	69.29	68.94
57	6	78	11.11	34.76	33.87
58	0	49	7.47	12.99	12.26
59	0	25	3.51	3.29	2.91
60	0	6	0.93	0.41	0.34

* denotes a value below .005

Table A.1.8: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 15, m = 45					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 25	0	0	0	0	0.00*
26	0	1	0.06	0.00*	0.02
27	0	5	0.43	0.05	0.08
28	0	12	1.46	0.39	0.38
29	0	44	5.39	1.73	1.57
30	0	84	12.25	6.48	5.85
31	0	168	27.12	21.82	19.81
32	0	308	56.06	65.45	60.66
33	3	501	101.58	176.86	167.27
34	91	973	155.24	430.05	413.26
35	385	1450	192.13	929.23	909.18
36	1262	2235	173.13	1785.68	1767.84
37	2458	3571	165.72	3004.54	3010.11
38	3352	5301	368.01	4401.65	4435.95
39	4272	7119	528.74	5509.69	5573.38
40	4639	7273	474.42	5807.87	5852.05
41	4248	5682	249.52	4986.99	4995.65
42	2410	4149	270.25	3350.43	3330.43
43	874	2660	339.26	1665.86	1626.49
44	135	1348	218.94	538.72	517.52
45	0	430	64.77	84.52	80.50

* denotes a value below .005

Table A.1.9: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 15, m = 53					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 31	0	0	0	0	0.00*
32	0	2	0.12	0.01	0.02
33	0	10	0.47	0.06	0.07
34	0	25	1.39	0.27	0.29
35	0	57	3.56	1.04	1.10
36	0	90	7.84	3.48	3.84
37	0	143	17.21	11.47	12.36
38	0	214	34.51	34.29	36.42
39	0	347	63.72	94.95	98.05
40	16	596	106.10	237.86	240.22
41	120	973	147.99	535.33	533.16
42	521	1539	162.70	1074.73	1066.32
43	1457	2424	139.56	1923.84	1909.46
44	2316	3532	178.08	3055.04	3037.78
45	3352	5368	331.97	4255.22	4252.89
46	3959	6672	442.86	5158.59	5177.43
47	4266	6745	403.52	5374.62	5397.75
48	3753	5369	249.34	4703.78	4723.03
49	2714	4234	216.87	3375.34	3373.59
50	1090	2729	275.49	1897.56	1889.21
51	246	1548	218.42	789.39	777.91
52	20	682	106.41	213.25	209.44
53	0	208	26.61	27.89	27.66

* denotes a value below .005

Table A.1.10: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 15, m = 60					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 34	0	0	0	0	0.00*
35	0	1	0.03	0.00*	0.00*
36	0	1	0.04	0.00*	0.00*
37	0	3	0.13	0.01	0.01
38	0	4	0.22	0.02	0.04
39	0	11	0.86	0.12	0.16
40	0	20	2.05	0.42	0.57
41	0	47	4.50	1.42	1.95
42	0	84	10.37	4.98	6.17
43	0	134	20.24	14.89	18.08
44	0	267	40.40	44.04	48.90
45	1	442	71.38	112.76	121.71
46	25	739	109.65	261.88	277.82
47	191	1010	145.87	559.12	579.28
48	587	1503	155.13	1080.75	1098.21
49	1393	2212	129.12	1873.79	1882.65
50	2260	3597	162.65	2911.21	2899.28
51	2934	4946	301.49	4014.58	3979.40
52	3489	6277	417.52	4877.27	4821.20
53	3722	6331	408.10	5152.20	5094.09
54	3893	5559	262.28	4649.72	4622.42
55	2970	4162	170.60	3512.95	3529.85
56	1490	2939	244.67	2171.82	2206.15
57	451	1842	234.75	1050.05	1083.73
58	77	996	146.81	375.99	392.38
59	1	392	56.63	87.84	93.11
60	0	123	12.52	10.18	10.86

* denotes a value below .005

Table A.1.11: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 15, m = 68					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 42	0	0	0	0	0.00*
43	0	2	0.10	0.01	0.01
44	0	2	0.13	0.01	0.03
45	0	5	0.31	0.04	0.11
46	0	14	1.01	0.18	0.38
47	0	28	2.54	0.75	1.25
48	0	40	6.00	2.77	3.84
49	0	80	12.67	8.50	10.97
50	0	147	25.42	24.51	29.19
51	0	262	46.13	64.08	72.12
52	9	477	77.80	152.75	165.05
53	74	789	112.08	329.88	348.79
54	267	1071	137.97	655.92	678.20
55	734	1566	139.29	1193.02	1208.43
56	1513	2332	120.44	1956.74	1963.69
57	2457	3408	164.55	2914.02	2893.86
58	2963	4734	284.14	3885.43	3841.85
59	3405	5838	372.76	4613.26	4558.13
60	3642	6205	367.65	4845.93	4786.03
61	3635	5183	256.79	4428.48	4393.74
62	2912	3955	159.88	3463.26	3472.47
63	1612	2949	202.88	2276.01	2314.98
64	613	1968	217.62	1227.20	1266.00
65	163	1094	160.53	521.93	545.36
66	12	552	85.37	165.57	173.52
67	0	178	28.88	34.23	36.26
68	0	41	5.71	3.53	3.73

* denotes a value below .005

Table A.1.12: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 15, m = 75					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 47	0	0	0	0	0.00*
48	0	2	0.09	0.01	0.00*
49	0	4	0.15	0.01	0.02
50	0	6	0.45	0.05	0.06
51	0	18	0.95	0.18	0.20
52	0	24	2.17	0.64	0.64
53	0	32	4.56	2.07	1.94
54	0	66	9.34	6.21	5.52
55	0	121	19.08	16.89	14.75
56	0	226	35.81	42.12	36.88
57	3	337	60.64	97.22	86.06
58	13	475	93.58	206.17	186.96
59	100	766	127.84	405.39	377.10
60	315	1199	150.06	739.70	703.92
61	798	1721	140.86	1242.25	1211.66
62	1588	2252	110.06	1923.89	1915.20
63	2284	3333	159.22	2742.37	2766.40
64	2864	4431	282.59	3573.69	3630.90
65	3278	5584	375.88	4217.03	4301.22
66	3229	5583	382.71	4481.44	4561.90
67	3145	5144	292.80	4244.09	4289.55
68	2866	4119	176.62	3524.36	3532.57
69	1944	3146	177.42	2539.92	2508.64
70	976	2233	225.46	1555.30	1505.18
71	275	1495	205.56	781.58	741.99
72	50	918	137.85	311.18	288.55
73	2	364	68.26	93.17	83.01
74	0	162	23.28	18.97	15.70
75	0	31	4.18	2.11	1.47

* denotes a value below .005

Table A.1.13: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 15, m = 83					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 54	0	0	0	0	0.00*
55	0	1	0.04	0.00*	0.01
56	0	3	0.21	0.03	0.04
57	0	5	0.47	0.08	0.13
58	0	16	1.44	0.35	0.41
59	0	28	3.27	1.14	1.21
60	0	58	6.57	3.30	3.38
61	0	102	12.98	9.03	8.91
62	0	192	24.64	23.25	22.13
63	0	271	42.66	53.75	51.63
64	0	400	68.30	117.70	112.94
65	24	603	100.89	239.61	231.09
66	83	883	134.01	454.29	441.18
67	305	1214	153.07	797.63	783.58
68	779	1686	148.57	1302.34	1290.61
69	1554	2349	121.43	1965.87	1963.97
70	2213	3159	156.88	2737.07	2749.56
71	2533	4468	278.87	3493.89	3524.08
72	3000	5468	389.20	4079.56	4111.43
73	3328	5504	402.70	4313.06	4336.72
74	3181	4968	308.35	4089.81	4102.30
75	2925	4107	172.72	3444.61	3445.93
76	1988	3057	173.81	2550.53	2539.11
77	981	2291	224.77	1627.01	1615.80
78	363	1525	215.02	883.75	870.04
79	102	930	155.50	396.01	385.46
80	8	539	86.05	140.34	134.91
81	0	212	33.21	36.92	34.98
82	0	56	8.72	6.60	5.97
83	0	15	1.29	0.47	0.50

* denotes a value below .005

Table A.1.14: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 15, m = 90					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 60	0	0	0	0	0.00*
61	0	3	0.12	0.01	0.02
62	0	2	0.32	0.06	0.07
63	0	8	0.86	0.24	0.21
64	0	21	1.92	0.65	0.61
65	0	31	4.87	2.09	1.71
66	0	48	8.88	4.95	4.54
67	0	79	16.12	12.68	11.39
68	0	127	28.33	29.28	26.96
69	1	247	47.84	64.80	60.17
70	16	374	74.34	134.90	126.36
71	49	553	103.89	260.50	249.17
72	168	834	129.89	474.66	460.27
73	402	1185	142.32	803.99	794.43
74	878	1645	136.85	1286.70	1277.53
75	1574	2282	123.45	1911.10	1907.78
76	2236	3011	147.68	2619.06	2635.75
77	2667	4021	259.88	3324.59	3354.59
78	3024	4855	363.30	3875.61	3913.69
79	3194	5130	374.39	4134.24	4161.39
80	3132	4731	306.24	3983.74	4005.34
81	2990	3940	188.80	3459.20	3461.41
82	2379	3090	148.66	2673.25	2659.37
83	1307	2267	198.01	1808.38	1794.28
84	528	1561	208.36	1060.65	1046.66
85	201	1052	172.25	530.41	517.17
86	37	661	109.73	219.83	210.48
87	2	340	54.42	71.96	67.74
88	0	115	19.05	17.67	16.17
89	0	24	4.34	2.68	2.54
90	0	8	0.61	0.14	0.20

* denotes a value below .005

Table A.1.15: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 20, m = 60					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 31	0	0	0	0	0.00*
32	0	4	0.14	0.01	0.00*
33	0	8	0.41	0.03	0.00*
34	0	22	1.08	0.08	0.00*
35	0	52	3.19	0.28	0.01
36	0	108	7.27	0.70	0.07
37	0	195	14.08	1.53	0.30
38	0	373	27.89	3.73	1.26
39	0	667	55.10	9.82	4.97
40	0	1284	109.46	27.45	18.26
41	0	2161	207.52	78.23	62.36
42	0	3639	394.92	223.64	197.46
43	0	5856	741.88	619.18	578.60
44	6	9276	1331.31	1625.39	1564.84
45	162	14016	2225.75	3981.91	3894.72
46	1426	21275	3330.79	9035.66	8890.11
47	6850	31010	4256.16	18785.62	18536.83
48	19745	46165	4314.32	35514.33	35142.75
49	46879	70569	3294.00	60687.94	60244.71
50	72918	103347	4413.29	93044.96	92776.85
51	93534	152155	8812.48	127097.20	127340.80
52	113862	197231	12115.50	153347.70	154278.30
53	124002	201269	11583.21	161650.30	163011.00
54	124124	164465	7061.49	146724.50	147917.40
55	99164	128163	4351.37	112651.10	112955.10
56	47483	98479	7059.29	71197.59	70596.93
57	17133	62983	7184.28	35564.69	34679.19
58	3840	33810	4586.48	13132.62	12556.26
59	374	13447	1809.66	3185.57	2979.45
60	4	3253	369.48	384.20	347.60

* denotes a value below .005

Table A.1.16: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 20, m = 70					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 42	0	0	0	0	0.00*
43	0	2	0.07	0.00*	0.03
44	0	6	0.30	0.04	0.11
45	0	16	1.29	0.24	0.44
46	0	54	4.71	1.22	1.67
47	0	131	13.36	4.96	5.99
48	0	300	36.28	18.56	20.08
49	0	606	93.92	62.67	63.10
50	0	1331	224.71	189.22	185.53
51	0	2478	492.45	525.35	509.29
52	11	4710	974.76	1337.11	1302.61
53	205	8115	1724.51	3142.10	3096.77
54	1236	14834	2712.92	6850.26	6824.37
55	4656	24494	3694.91	13838.21	13896.89
56	14032	37113	4149.14	25862.51	26056.68
57	33738	53183	3530.89	44423.23	44799.20
58	60965	77683	2694.85	69819.91	70288.40
59	84593	115415	5385.29	99706.83	100071.60
60	103314	154203	9292.42	128352.00	128425.20
61	115997	179035	11282.54	147825.70	147373.20
62	122714	175349	9834.43	150645.10	149750.20
63	116351	150598	5709.05	134087.80	133111.30
64	90995	116575	4035.82	102457.00	101913.30
65	48796	83300	6159.47	65754.07	65851.70
66	21000	53454	6227.42	34432.80	34921.35
67	5715	27696	4290.12	14096.12	14594.00
68	997	12502	2039.69	4227.24	4506.97
69	47	4573	628.15	834.47	914.46
70	0	1010	98.98	81.35	91.45

* denotes a value below .005

Table A.1.17: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 20, m= 80					
k	Minimum	Maximum	Std.Dev	Average	Expected
0,....,49	0	0	0	0	0.00*
50	0	1	0.04	0.00*	0.01
51	0	5	0.23	0.02	0.04
52	0	15	1.03	0.14	0.15
53	0	45	3.14	0.65	0.56
54	0	118	8.95	2.45	1.97
55	0	246	23.27	8.40	6.52
56	0	524	54.26	25.55	20.37
57	0	977	118.45	72.55	60.05
58	0	1951	249.12	196.53	166.68
59	0	3410	483.27	497.29	435.06
60	2	5374	882.73	1184.66	1065.89
61	102	8319	1499.25	2639.82	2446.30
62	592	12587	2333.53	5534.01	5247.71
63	2601	19628	3233.00	10844.78	10495.42
64	8025	30624	3885.17	19853.73	19514.92
65	20378	44534	3804.69	33816.11	33625.71
66	43471	61591	2818.69	53378.60	53495.44
67	68465	85527	2908.59	77672.44	78247.06
68	86570	118868	6141.53	103721.90	104713.00
69	102841	155260	9479.47	126423.20	127476.70
70	112124	173374	10806.17	139481.40	140224.30
71	113826	165222	9061.14	138106.20	138249.40
72	102487	137255	5068.14	121372.90	120968.20
73	83937	102987	3484.18	93491.93	92797.51
74	42985	77802	5704.22	62047.03	61447.00
75	16357	52333	6183.35	34739.06	34410.32
76	4785	31324	4676.37	15902.61	15846.86
77	1104	16956	2574.39	5737.11	5762.49
78	94	7481	1010.85	1532.16	1551.44
79	3	2224	269.59	270.32	274.94
80	0	514	39.72	22.39	24.06

* denotes a value below .005

Table A.1.18: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 20, m = 90					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 58	0	0	0	0	0.00*
59	0	2	0.15	0.02	0.05
60	0	9	0.63	0.10	0.19
61	0	25	2.05	0.45	0.65
62	0	58	5.73	1.65	2.13
63	0	122	15.05	5.64	6.63
64	0	310	36.67	17.43	19.57
65	0	640	84.63	50.89	54.81
66	0	1130	179.86	138.69	145.32
67	0	2099	356.49	352.07	364.39
68	3	3701	668.30	844.51	862.74
69	57	6007	1157.27	1895.08	1925.53
70	331	9829	1858.35	4000.06	4043.62
71	1540	15426	2688.26	7891.06	7973.33
72	5073	23182	3442.35	14583.33	14728.51
73	13919	34053	3735.48	25166.79	25421.81
74	31001	48461	3224.51	40544.69	40881.03
75	52215	68032	2299.55	60685.58	61049.00
76	70539	94395	3851.60	84079.82	84344.01
77	90323	127901	7060.18	107365.00	107346.90
78	102354	152601	9504.64	125637.60	125238.10
79	108165	158181	9723.88	133911.10	133164.50
80	106421	148910	7496.46	129190.10	128170.90
81	96511	128992	4200.26	111647.90	110764.90
82	73722	99860	3623.94	85539.32	85099.90
83	42304	71956	5220.44	57277.83	57416.80
84	19725	47997	5467.81	32993.60	33493.13
85	6880	30193	4211.30	15986.80	16549.55
86	1959	17095	2476.45	6318.13	6735.28
87	315	8056	1095.71	1946.69	2167.68
88	11	3124	357.60	437.21	517.29
89	0	979	81.81	62.64	81.37
90	0	164	10.73	4.27	6.33

* denotes a value below .005

Table A.1.19: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 20, m = 100					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 64	0	0	0	0	0.00*
65	0	2	0.07	0.00*	0.00*
66	0	2	0.13	0.01	0.02
67	0	6	0.33	0.03	0.06
68	0	14	0.85	0.12	0.22
69	0	30	2.85	0.58	0.70
70	0	70	6.64	1.96	2.17
71	0	158	16.92	6.71	6.42
72	0	331	38.22	19.44	18.09
73	0	582	79.40	53.17	48.58
74	0	1096	157.48	135.35	124.08
75	0	1823	299.46	328.06	301.09
76	8	3111	537.48	747.18	693.31
77	88	5087	906.27	1615.37	1512.67
78	467	8195	1419.27	3299.07	3122.30
79	1931	12450	2022.70	6357.35	6086.51
80	5270	18880	2586.33	11562.58	11183.96
81	12272	27107	2859.53	19822.26	19330.31
82	23524	38656	2607.29	31880.25	31352.81
83	40348	53402	1956.26	48013.02	47595.84
84	60131	74845	2484.55	67589.87	67427.44
85	76378	99592	4432.34	88532.87	88845.57
86	90604	123795	6393.20	107597.80	108474.20
87	100179	142847	7379.24	120814.90	122189.40
88	104626	148284	6892.23	124838.30	126354.90
89	103214	135982	5134.43	118025.30	119256.30
90	93688	113699	3138.48	101472.60	102030.40
91	71512	87649	3039.87	78699.72	78484.93
92	42358	64682	3963.92	54524.02	53745.12
93	20415	45555	4069.23	33279.15	32362.65
94	8022	29010	3303.70	17583.44	16869.89
95	2634	16388	2152.86	7867.41	7458.27
96	630	7699	1120.68	2878.70	2719.16
97	66	3424	461.19	827.83	784.91
98	0	1145	140.60	175.89	168.20
99	0	214	30.56	24.18	23.79
100	0	34	4.05	1.56	1.66

* denotes a value below .005

Table A.1.20: Number of Solutions that Satisfy $k = 0, 1, \dots, m$ Clauses

n = 20, m = 120					
k	Minimum	Maximum	Std.Dev	Average	Expected
0, ..., 82	0	0	0	0	0.00*
83	0	1	0.07	0.01	0.08
84	0	7	0.40	0.06	0.24
85	0	10	1.32	0.29	0.70
86	0	35	4.20	1.34	1.99
87	0	78	11.09	4.36	5.45
88	0	172	24.25	13.06	14.30
89	0	309	51.15	34.90	36.00
90	0	668	105.01	87.31	86.78
91	0	1085	203.25	210.04	200.26
92	1	1947	370.48	469.90	441.88
93	30	3210	630.67	997.71	931.28
94	206	5270	1003.74	1994.90	1872.48
95	815	8124	1481.85	3803.94	3587.27
96	2637	12050	1972.41	6868.26	6539.29
97	6130	17760	2351.43	11772.79	11325.79
98	11890	25402	2487.73	19136.87	18606.65
99	21797	34682	2212.68	29506.87	28943.68
100	36992	47706	1740.21	43044.76	42547.21
101	54192	65505	2110.51	59243.31	58976.33
102	68182	85904	3627.58	76728.73	76900.50
103	80249	107432	5261.76	93345.74	94072.46
104	90570	124721	6234.25	106350.10	107640.60
105	97102	132587	6285.39	113218.00	114816.60
106	97489	126520	5241.96	112189.60	113733.50
107	92515	114083	3617.31	103096.50	104167.10
108	79227	93881	2556.13	87402.17	87770.43
109	59730	74005	2921.13	67980.13	67639.60
110	38169	56586	3456.47	48167.64	47347.72
111	21929	40213	3430.38	30822.24	29858.92
112	11008	26232	2838.43	17577.35	16795.64
113	4612	15471	2001.27	8808.92	8323.50
114	1405	8266	1188.73	3799.47	3577.65
115	350	3955	597.97	1377.23	1306.62
116	39	1677	256.62	410.80	394.24
117	0	639	85.76	93.96	94.35
118	0	148	20.39	15.01	16.79
119	0	30	3.54	1.68	1.98
120	0	5	0.27	0.04	0.12

* denotes a value below .005

Table A.2.1: Maximum Number of Satisfied Clauses

n = 10								
m	k	P{MAX(n,m) = k}	m	k	P{MAX(n,m) = k}	m	k	P{MAX(n,m) = k}
30	30	0.998	200	192	0.004	300	284	0.002
	29	0.002		191	0.018		283	0.004
35	35	0.954	190	190	0.048	282	282	0.002
	34	0.046		189	0.088		281	0.020
40	40	0.850	188	188	0.164	280	280	0.044
	39	0.150		187	0.232		279	0.082
42	42	0.794	186	186	0.256	278	278	0.122
	41	0.206		185	0.144		277	0.160
45	45	0.622	184	184	0.044	276	276	0.214
	44	0.370		183	0.002		275	0.208
	43	0.008		225	216		0.002	274
50	50	0.482	215	215	0.002	273	273	0.048
	49	0.474		214	0.008		272	0.006
	48	0.044		213	0.018			
55	55	0.310	212	212	0.064			
	54	0.564		211	0.110			
	53	0.126		210	0.212			
60	60	0.144	209	209	0.262			
	59	0.550		208	0.188			
	58	0.294		207	0.096			
	57	0.012		206	0.036			
175	169	0.004	205	205	0.002			
	168	0.016		250	239	0.002		
	167	0.050		238	0.004			
	166	0.118		237	0.008			
	165	0.240		236	0.010			
	164	0.276		235	0.038			
	163	0.208		234	0.096			
	162	0.074		233	0.180			
	161	0.014		232	0.196			
				231	0.208			
				230	0.168			
				229	0.070			
				228	0.018			
227			0.002					

For each n and m, 500 instances are randomly generated (with replacement)

MAX(n,m) = Maximum number of satisfied clauses (given n and m)

P{MAX(n,m) = k} = Frequency Distribution of 500 instances over MAX(n,m) values

Table A.2.2: Maximum Number of Satisfied Clauses

n = 20								
m	k	P{MAX(n,m) = k}	m	k	P{MAX(n,m) = k}	m	k	P{MAX(n,m) = k}
60	60	0.996	400	382	0.018	600	571	0.002
	59	0.004		381	0.016		570	0
70	70	0.966		380	0.056		569	0
	69	0.034		379	0.078		568	0
80	80	0.782		378	0.124		567	0.002
	79	0.218		377	0.164		566	0
85	85	0.644		376	0.242		565	0.008
	84	0.348		375	0.166		564	0.004
	83	0.008		374	0.100		563	0.018
90	90	0.488		373	0.030		562	0.040
	89	0.484		372	0.006		561	0.042
	88	0.028	450	430	0.002		560	0.078
100	100	0.266		429	0.002		559	0.118
	99	0.584		428	0.012		558	0.106
	98	0.146		427	0.018		557	0.140
	97	0.004		426	0.028		556	0.146
110	110	0.110		425	0.080		555	0.128
	109	0.460		424	0.108		554	0.102
	108	0.398		423	0.156		553	0.042
	107	0.032		422	0.174		552	0.018
120	120	0.044		421	0.216		551	0.006
	119	0.266		420	0.124			
	118	0.524		419	0.070			
	117	0.152		418	0.010			
	116	0.014	500	476	0.002			
350	337	0.002		475	0			
	336	0.014		474	0.002			
	335	0.058		473	0.010			
	334	0.056		472	0.016			
	333	0.094		471	0.054			
	332	0.168		470	0.070			
	331	0.226		469	0.104			
	330	0.158		468	0.146			
	329	0.124		467	0.170			
	328	0.086		466	0.178			
	327	0.014		465	0.122			
				464	0.090			
				463	0.018			
				462	0.012			
				461	0.004			
				460	0.002			

Table A.3: The Threshold Conjecture

n = 10		n = 20	
c = m / n	P{Satisfiable}	c = m / n	P{Satisfiable}
3.0	0.996	3.0	1
3.2	0.986	3.2	0.991
3.4	0.968	3.4	0.986
3.6	0.946	3.6	0.967
3.8	0.904	3.8	0.918
4.0	0.848	4.0	0.825
4.2	0.785	4.2	0.748
4.4	0.680	4.4	0.618
4.6	0.624	4.6	0.487
4.8	0.509	4.8	0.397
5.0	0.472	5.0	0.244
5.2	0.408	5.2	0.192
5.4	0.315	5.4	0.136
5.6	0.244	5.6	0.082
5.8	0.191	5.8	0.076
6.0	0.149	6.0	0.043

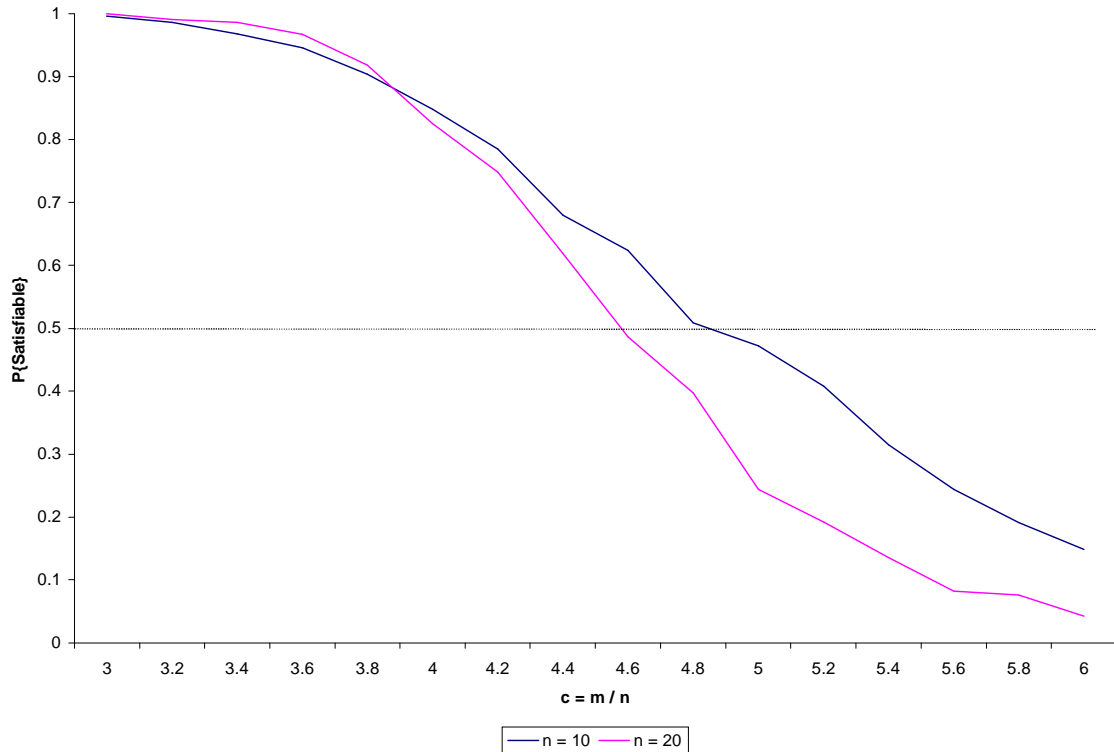


Figure A.1: The Threshold Conjecture

For each n and c = m / n, 1000 instances are randomly generated (without replacement)
 P{Satisfiable} = Proportion of the instances that are satisfiable

Table A.4.1: Geary's Test Results for Normality

		n = 10							
m	Inst.#	T = 2 ⁿ				T = 100			
		Avg.	Std. Dev.	U	p	Avg.	Std. Dev.	U	p
30	1	26.25	1.64	2.93	0.00*	25.58	1.87	-0.47	0.64
	2	26.25	1.76	0.54	0.59	26.16	1.72	-1.65	0.10
	3	26.25	1.65	0.74	0.46	25.72	1.64	0.92	0.36
	4	26.25	1.87	1.49	0.14	26.03	1.79	-1.77	0.08
	5	26.25	1.85	2.63	0.01	26.50	1.84	1.53	0.12
	6	26.25	1.69	0.89	0.37	26.14	1.65	-0.25	0.80
	7	26.25	1.78	2.07	0.04	26.37	1.55	-0.04	0.97
	8	26.25	1.84	1.48	0.14	25.53	2.08	1.83	0.07
	9	26.25	1.98	2.21	0.03	25.88	2.22	1.01	0.31
	10	26.25	1.62	0.89	0.38	26.46	1.38	1.23	0.22
35	1	30.63	1.87	2.36	0.02	30.49	1.98	0.18	0.85
	2	30.63	2.10	-1.72	0.09	30.14	1.85	-1.36	0.17
	3	30.63	1.47	3.24	0.00*	30.46	1.44	1.55	0.12
	4	30.63	2.13	1.57	0.12	30.67	2.29	-1.10	0.27
	5	30.63	2.52	0.61	0.54	30.75	2.11	-1.68	0.09
	6	30.63	1.98	2.80	0.01	30.80	1.78	1.35	0.18
	7	30.63	1.93	1.44	0.15	31.21	1.75	-0.04	0.97
	8	30.63	1.74	4.66	0.00*	30.61	1.84	2.43	0.01
	9	30.63	1.72	0.12	0.90	30.51	1.96	-1.18	0.24
	10	30.63	1.89	4.44	0.00*	30.73	1.72	2.96	0.00*
40	1	35.00	1.60	-1.28	0.20	34.97	1.62	1.53	0.13
	2	35.00	1.99	-1.87	0.06	35.30	2.02	-0.74	0.46
	3	35.00	2.06	-8.31	0.00*	34.80	2.43	-0.86	0.39
	4	35.00	1.71	-2.32	0.02	35.02	1.74	-1.57	0.12
	5	35.00	2.33	-3.11	0.00*	33.83	2.52	1.08	0.28
	6	35.00	2.04	-0.84	0.40	35.54	2.23	1.38	0.17
	7	35.00	2.02	-0.64	0.52	34.85	1.95	-0.96	0.34
	8	35.00	1.99	-1.43	0.15	35.23	1.78	1.30	0.19
	9	35.00	1.85	-0.54	0.59	34.95	1.94	0.01	0.99
	10	35.00	1.74	-1.23	0.22	35.30	1.53	0.78	0.43

* denotes a value below .005

For each n and m, 10 instances are randomly generated (with replacement)

Inst.# = Instance ID

T = Sample size

Avg. = Sample average (i.e., average number of satisfied clauses)

Std. Dev = Sample standard deviation (i.e., standard deviation of the number of satisfied clauses)

U = Geary's test statistic for the goodness-of-fit to normality

p = p-value for the goodness-of-fit to normality

Table A.4.1 (Continued...)

		n = 10							
m	Inst.#	T = 2 ⁿ				T = 100			
		Avg.	Std. Dev.	U	p	Avg.	Std. Dev.	U	p
45	1	39.38	2.29	1.17	0.24	39.52	2.33	-1.86	0.06
	2	39.38	2.10	4.60	0.00*	39.22	2.04	1.97	0.05
	3	39.38	2.06	0.12	0.90	39.30	1.84	-1.04	0.30
	4	39.38	2.10	1.97	0.05	39.49	1.81	0.97	0.33
	5	39.38	2.38	2.70	0.01	39.85	2.18	-0.70	0.49
	6	39.38	2.31	2.16	0.03	39.78	2.42	-0.18	0.85
	7	39.38	2.21	-1.89	0.06	39.56	1.78	-0.29	0.77
	8	39.38	2.10	3.30	0.00*	39.13	2.02	-0.21	0.83
	9	39.38	2.49	2.10	0.04	39.63	2.46	1.44	0.15
	10	39.38	2.06	0.86	0.39	39.54	2.13	0.66	0.51
50	1	43.75	2.31	3.41	0.00*	43.19	2.27	2.78	0.01
	2	43.75	2.32	2.75	0.01	43.67	2.33	1.47	0.14
	3	43.75	2.58	0.80	0.42	43.89	2.82	-1.98	0.05
	4	43.75	1.99	1.69	0.09	43.71	2.03	0.54	0.59
	5	43.75	2.31	-0.46	0.65	44.29	2.56	-0.09	0.93
	6	43.75	2.63	0.13	0.90	43.23	2.94	0.98	0.33
	7	43.75	2.22	2.38	0.02	44.69	2.31	1.25	0.21
	8	43.75	2.30	-0.68	0.50	43.95	2.29	-0.67	0.50
	9	43.75	2.64	1.88	0.06	43.64	2.77	1.84	0.07
	10	43.75	2.33	3.06	0.00*	43.41	2.51	1.11	0.27
55	1	48.13	2.31	-0.36	0.72	48.86	2.06	0.69	0.49
	2	48.13	2.34	4.01	0.00*	48.01	2.12	1.16	0.25
	3	48.13	2.70	1.14	0.25	48.94	2.58	1.90	0.06
	4	48.13	2.34	1.83	0.07	47.88	2.01	0.45	0.65
	5	48.13	2.47	0.86	0.39	48.52	2.41	-0.09	0.93
	6	48.13	2.80	2.51	0.01	48.85	2.68	-0.59	0.56
	7	48.13	2.33	-2.49	0.01	48.15	2.65	-0.75	0.45
	8	48.13	2.58	2.29	0.02	47.25	2.71	0.63	0.53
	9	48.13	2.82	0.12	0.90	48.52	2.86	1.22	0.22
	10	48.13	2.49	2.89	0.00*	48.81	2.55	-0.28	0.78
60	1	52.50	3.31	-0.35	0.72	52.44	3.24	0.49	0.63
	2	52.50	2.58	0.17	0.86	52.96	2.39	-0.28	0.78
	3	52.50	3.03	2.66	0.01	51.93	3.35	0.70	0.49
	4	52.50	2.29	0.19	0.85	53.06	2.17	0.15	0.88
	5	52.50	2.40	4.41	0.00*	52.32	2.57	0.15	0.88
	6	52.50	2.35	-0.72	0.47	53.02	2.22	-0.07	0.94
	7	52.50	2.85	2.56	0.01	52.22	3.08	1.30	0.20
	8	52.50	2.65	0.24	0.81	52.67	2.75	-0.22	0.83
	9	52.50	2.97	6.42	0.00*	52.19	2.89	2.72	0.01
	10	52.50	2.17	3.20	0.00*	52.51	2.12	-0.66	0.51

* denotes a value below .005

Table A.4.2: Geary's Test Results for Normality

		n = 15							
m	Inst.#	T = 2 ⁿ				T = 1000			
		Avg.	Std. Dev.	U	p	Avg.	Std. Dev.	U	p
45	1	39.38	2.25	10.27	0.00*	39.41	2.18	2.51	0.01
	2	39.38	2.30	4.03	0.00*	39.37	2.34	1.07	0.29
	3	39.38	2.14	15.34	0.00*	39.39	2.26	1.45	0.15
	4	39.38	2.24	8.42	0.00*	39.44	2.19	1.40	0.16
	5	39.38	2.22	-1.00	0.32	39.26	2.37	-0.48	0.63
	6	39.38	2.34	-4.62	0.00*	39.24	2.21	-1.25	0.21
	7	39.38	2.12	11.52	0.00*	39.20	2.09	1.18	0.24
	8	39.38	2.15	10.70	0.00*	39.39	2.04	1.61	0.11
	9	39.38	2.50	9.38	0.00*	39.25	2.57	0.85	0.39
	10	39.38	2.02	6.72	0.00*	39.29	2.09	2.26	0.02
53	1	46.38	2.11	6.49	0.00*	46.38	2.15	1.16	0.25
	2	46.38	2.32	6.56	0.00*	46.32	2.23	3.41	0.00*
	3	46.38	2.30	2.08	0.04	46.44	2.27	1.14	0.26
	4	46.38	2.43	6.87	0.00*	46.23	2.43	2.02	0.04
	5	46.38	2.46	6.70	0.00*	46.28	2.55	1.56	0.12
	6	46.38	2.03	12.75	0.00*	46.33	2.09	2.78	0.01
	7	46.38	2.35	9.66	0.00*	46.21	2.42	1.42	0.16
	8	46.38	2.58	3.16	0.00*	46.14	2.69	-1.64	0.10
	9	46.38	2.63	0.81	0.42	46.47	2.57	1.20	0.23
	10	46.38	2.26	4.12	0.00*	46.37	2.18	1.36	0.17
60	1	52.50	2.79	6.20	0.00*	52.64	2.63	0.68	0.50
	2	52.50	3.02	4.85	0.00*	52.48	3.04	0.81	0.42
	3	52.50	2.38	2.30	0.02	52.38	2.41	0.91	0.36
	4	52.50	3.29	-2.74	0.01	52.38	3.41	1.66	0.10
	5	52.50	2.42	7.73	0.00*	52.44	2.39	1.73	0.08
	6	52.50	2.88	9.75	0.00*	52.42	2.85	2.08	0.04
	7	52.50	2.69	-5.38	0.00*	52.30	2.64	0.87	0.39
	8	52.50	2.77	10.53	0.00*	52.39	2.74	1.12	0.26
	9	52.50	2.43	6.51	0.00*	52.67	2.41	0.31	0.76
	10	52.50	2.61	7.20	0.00*	52.42	2.63	0.99	0.32
68	1	59.50	3.42	8.13	0.00*	59.31	3.48	0.05	0.96
	2	59.50	2.33	7.57	0.00*	59.31	2.30	0.67	0.50
	3	59.50	2.70	-0.05	0.96	59.48	2.64	2.12	0.03
	4	59.50	2.55	12.02	0.00*	59.15	2.60	1.61	0.11
	5	59.50	2.57	4.74	0.00*	59.53	2.55	1.73	0.08
	6	59.50	2.73	2.56	0.01	59.33	2.85	-0.18	0.86
	7	59.50	2.96	7.14	0.00*	59.31	2.97	1.43	0.15
	8	59.50	2.97	12.95	0.00*	59.46	2.92	2.40	0.02
	9	59.50	2.93	1.25	0.21	59.45	2.84	0.26	0.79
	10	59.50	2.73	-3.23	0.00*	59.42	2.90	0.27	0.79

* denotes a value below .005

Table A.4.2 (Continued...)

		n = 15							
m	Instd.#	T = 2 ⁿ				T = 1000			
		Avg.	Std. Dev.	U	p	Avg.	Std. Dev.	U	p
75	1	65.63	2.74	4.70	0.00*	65.44	2.81	2.04	0.04
	2	65.63	2.55	3.50	0.00*	65.62	2.60	-0.29	0.77
	3	65.63	2.91	10.34	0.00*	65.52	3.00	1.72	0.09
	4	65.63	2.79	-8.88	0.00*	65.77	2.60	-2.34	0.02
	5	65.63	2.81	2.95	0.00*	65.47	2.76	0.16	0.88
	6	65.63	2.53	6.58	0.00*	65.75	2.61	1.38	0.17
	7	65.63	2.63	8.47	0.00*	65.53	2.57	1.45	0.15
	8	65.63	2.78	8.42	0.00*	65.48	2.80	0.06	0.95
	9	65.63	3.16	10.67	0.00*	65.38	3.19	2.51	0.01
	10	65.63	3.07	14.23	0.00*	65.66	3.07	4.00	0.00*
83	1	72.63	3.14	0.35	0.72	72.68	3.07	0.13	0.89
	2	72.63	2.95	7.73	0.00*	72.53	2.87	1.94	0.05
	3	72.63	3.62	10.72	0.00*	72.37	3.54	2.15	0.03
	4	72.63	2.68	3.45	0.00*	72.82	2.59	0.45	0.65
	5	72.63	3.01	7.10	0.00*	72.38	2.86	0.32	0.75
	6	72.63	3.21	4.73	0.00*	72.47	3.29	0.97	0.33
	7	72.63	3.25	8.84	0.00*	72.57	3.33	2.21	0.03
	8	72.63	2.96	8.30	0.00*	72.57	2.92	1.49	0.14
	9	72.63	2.57	7.45	0.00*	72.50	2.55	0.97	0.33
	10	72.63	2.69	-8.80	0.00*	72.65	2.69	-1.94	0.05
90	1	78.75	3.04	5.06	0.00*	78.85	3.03	1.56	0.12
	2	78.75	3.40	5.74	0.00*	78.54	3.44	1.14	0.25
	3	78.75	3.04	3.88	0.00*	78.88	3.13	0.75	0.45
	4	78.75	3.14	3.31	0.00*	78.58	3.17	2.16	0.03
	5	78.75	3.05	10.55	0.00*	78.81	3.07	2.24	0.03
	6	78.75	3.22	-2.33	0.02	78.75	3.30	0.06	0.95
	7	78.75	3.01	7.99	0.00*	78.59	3.01	0.68	0.49
	8	78.75	3.33	-5.03	0.00*	78.70	3.25	-0.29	0.77
	9	78.75	3.01	8.22	0.00*	78.64	3.05	0.97	0.33
	10	78.75	2.75	3.87	0.00*	78.59	2.86	-0.19	0.85

* denotes a value below .005

Table A.4.3: Geary's Test Results for Normality

		n = 20							
m	Inst.#	T = 2 ⁿ				T = 10000			
		Avg.	Std. Dev.	U	p	Avg.	Std. Dev.	U	p
60	1	52.50	2.54	27.20	0.00*	52.49	2.52	2.71	0.01
	2	52.50	2.77	18.59	0.00*	52.65	2.69	2.39	0.02
	3	52.50	2.40	47.37	0.00*	52.62	2.43	3.23	0.00*
	4	52.50	2.40	38.40	0.00*	52.59	2.33	1.96	0.05
	5	52.50	2.61	51.53	0.00*	52.82	2.55	0.09	0.93
	6	52.50	2.78	7.00	0.00*	52.48	2.80	0.47	0.64
	7	52.50	2.76	57.29	0.00*	52.31	2.86	5.80	0.00*
	8	52.50	2.48	54.08	0.00*	52.17	2.55	3.35	0.00*
	9	52.50	2.22	34.85	0.00*	52.42	2.17	2.81	0.01
	10	52.50	2.47	51.60	0.00*	52.55	2.40	5.51	0.00*
70	1	61.25	3.01	37.60	0.00*	61.36	2.90	3.50	0.00*
	2	61.25	2.89	32.70	0.00*	61.19	2.91	2.83	0.00*
	3	61.25	2.78	35.39	0.00*	60.94	2.75	-0.97	0.33
	4	61.25	3.36	-5.87	0.00*	61.22	3.39	0.19	0.85
	5	61.25	2.76	-9.58	0.00*	61.18	2.82	-0.91	0.36
	6	61.25	2.90	-15.38	0.00*	61.55	3.00	-1.37	0.17
	7	61.25	2.82	31.98	0.00*	61.45	2.80	5.45	0.00*
	8	61.25	3.14	33.50	0.00*	61.28	3.09	3.43	0.00*
	9	61.25	2.73	30.52	0.00*	61.51	2.71	3.71	0.00*
	10	61.25	2.72	13.47	0.00*	61.12	2.67	-2.51	0.01
80	1	70.00	2.67	-48.00	0.00*	70.06	2.68	-1.02	0.31
	2	70.00	2.74	-41.20	0.00*	70.12	2.69	-2.40	0.02
	3	70.00	2.94	-11.06	0.00*	70.00	2.94	-0.08	0.93
	4	70.00	3.05	-45.54	0.00*	70.01	3.08	-3.31	0.00*
	5	70.00	2.81	-18.63	0.00*	69.98	2.86	-0.54	0.59
	6	70.00	2.81	-33.52	0.00*	69.80	2.80	0.19	0.85
	7	70.00	3.25	-14.17	0.00*	70.34	3.17	1.75	0.08
	8	70.00	3.01	-5.10	0.00*	69.84	3.01	2.90	0.00*
	9	70.00	3.01	-26.92	0.00*	69.95	3.05	-0.74	0.46
	10	70.00	3.05	15.80	0.00*	69.87	3.09	3.46	0.00*
90	1	78.75	2.87	11.73	0.00*	78.68	2.89	1.64	0.10
	2	78.75	2.94	28.71	0.00*	78.85	2.95	0.72	0.47
	3	78.75	3.06	10.82	0.00*	78.70	3.02	1.70	0.09
	4	78.75	3.35	0.23	0.82	78.51	3.42	4.20	0.00*
	5	78.75	3.54	28.86	0.00*	78.92	3.54	-0.61	0.54
	6	78.75	2.94	47.27	0.00*	78.53	2.98	5.95	0.00*
	7	78.75	3.13	39.07	0.00*	78.70	3.09	2.35	0.02
	8	78.75	3.15	-5.92	0.00*	79.02	3.09	-3.15	0.00*
	9	78.75	2.89	22.19	0.00*	78.79	2.90	0.42	0.67
	10	78.75	3.06	15.29	0.00*	78.52	3.06	2.23	0.03

* denotes a value below .005

Table A.4.3 (Continued...)

		n = 20							
m	Inst.#	T = 2 ⁿ				T = 10000			
		Avg.	Std. Dev.	U	p	Avg.	Std. Dev.	U	p
100	1	87.50	3.45	66.75	0.00*	87.59	3.44	6.82	0.00*
	2	87.50	3.67	22.86	0.00*	87.09	3.83	0.72	0.47
	3	87.50	3.38	24.03	0.00*	87.90	3.32	-1.32	0.19
	4	87.50	3.57	40.46	0.00*	87.78	3.54	1.87	0.06
	5	87.50	3.43	-8.24	0.00*	87.53	3.52	-0.08	0.94
	6	87.50	3.26	-11.42	0.00*	87.48	3.16	-0.54	0.59
	7	87.50	3.85	30.10	0.00*	87.46	3.79	1.92	0.06
	8	87.50	3.50	11.41	0.00*	87.83	3.50	-1.48	0.14
	9	87.50	3.74	23.57	0.00*	86.85	3.81	0.79	0.43
	10	87.50	3.43	10.18	0.00*	87.74	3.37	-0.74	0.46
110	1	96.25	3.77	39.19	0.00*	96.56	3.85	5.02	0.00*
	2	96.25	3.50	0.96	0.33	96.14	3.45	1.30	0.20
	3	96.25	3.47	25.44	0.00*	96.22	3.43	1.44	0.15
	4	96.25	3.50	27.32	0.00*	96.25	3.58	2.76	0.01
	5	96.25	3.88	-0.50	0.62	96.01	4.08	-5.37	0.00*
	6	96.25	3.67	-10.34	0.00*	96.37	3.65	0.04	0.97
	7	96.25	3.89	24.63	0.00*	96.37	3.86	2.34	0.02
	8	96.25	3.68	35.89	0.00*	96.66	3.70	4.78	0.00*
	9	96.25	3.43	45.00	0.00*	96.25	3.46	4.40	0.00*
	10	96.25	3.16	-12.16	0.00*	96.23	3.22	-0.55	0.58
120	1	105.00	3.61	-4.86	0.00*	105.04	3.66	1.77	0.08
	2	105.00	4.02	-0.78	0.44	105.14	4.01	2.80	0.01
	3	105.00	3.67	-4.62	0.00*	105.14	3.67	2.01	0.04
	4	105.00	3.56	5.17	0.00*	105.25	3.53	2.01	0.04
	5	105.00	3.75	-6.81	0.00*	104.44	3.78	2.59	0.01
	6	105.00	3.94	-8.47	0.00*	105.57	3.81	3.36	0.00*
	7	105.00	3.91	-7.83	0.00*	104.91	3.91	0.38	0.70
	8	105.00	3.76	-16.00	0.00*	104.82	3.86	2.42	0.02
	9	105.00	3.26	-47.80	0.00*	105.15	3.28	-3.88	0.00*
	10	105.00	3.46	-42.66	0.00*	104.72	3.51	-2.47	0.01

* denotes a value below .005

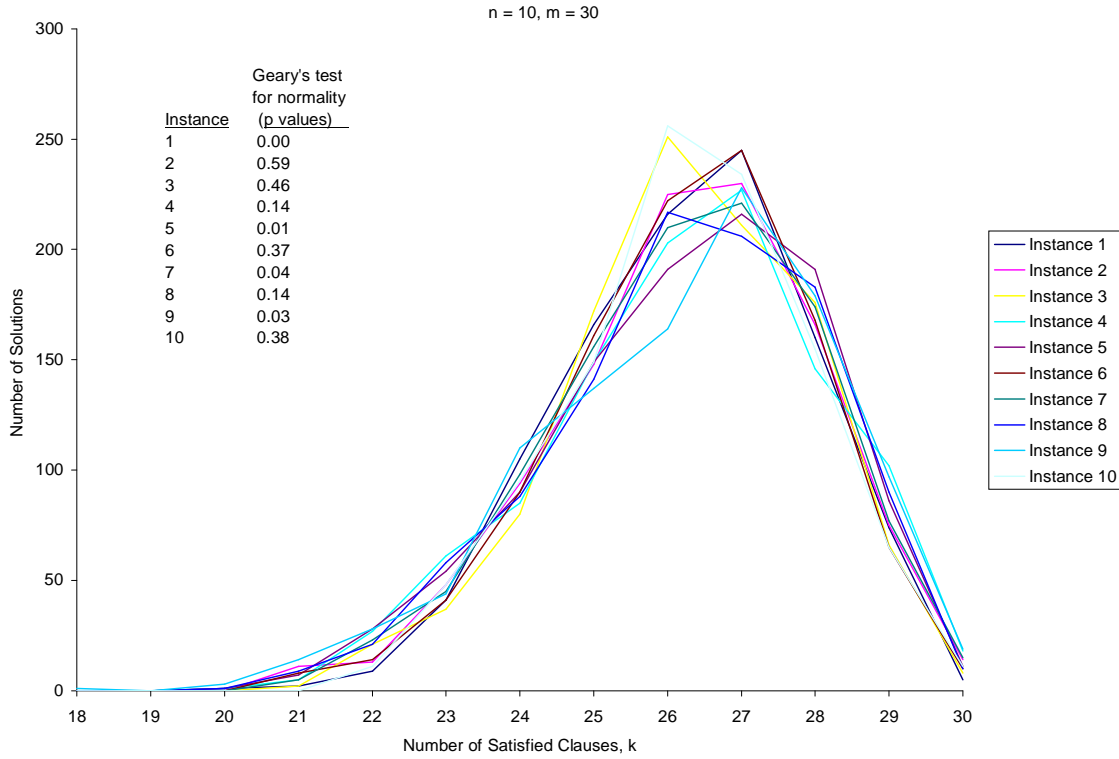


Figure A.2.1: Distribution of the Solution Space over the Number of Satisfied Clauses

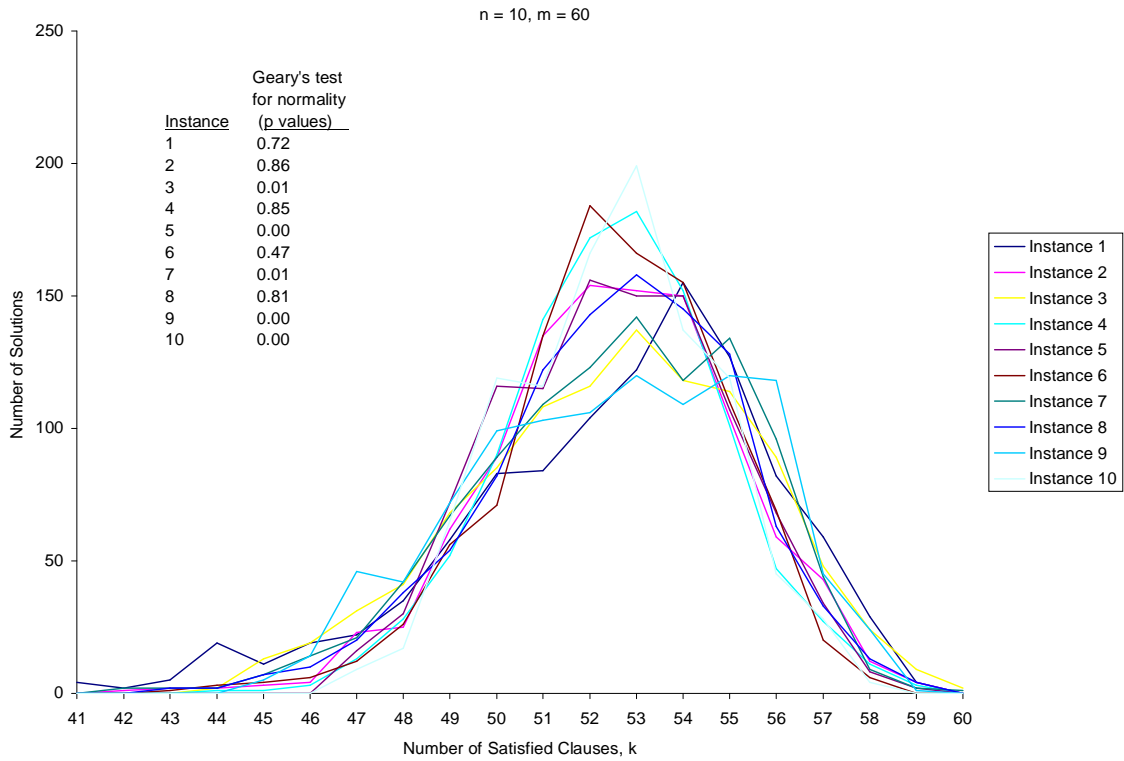


Figure A.2.2: Distribution of the Solution Space over the Number of Satisfied Clauses

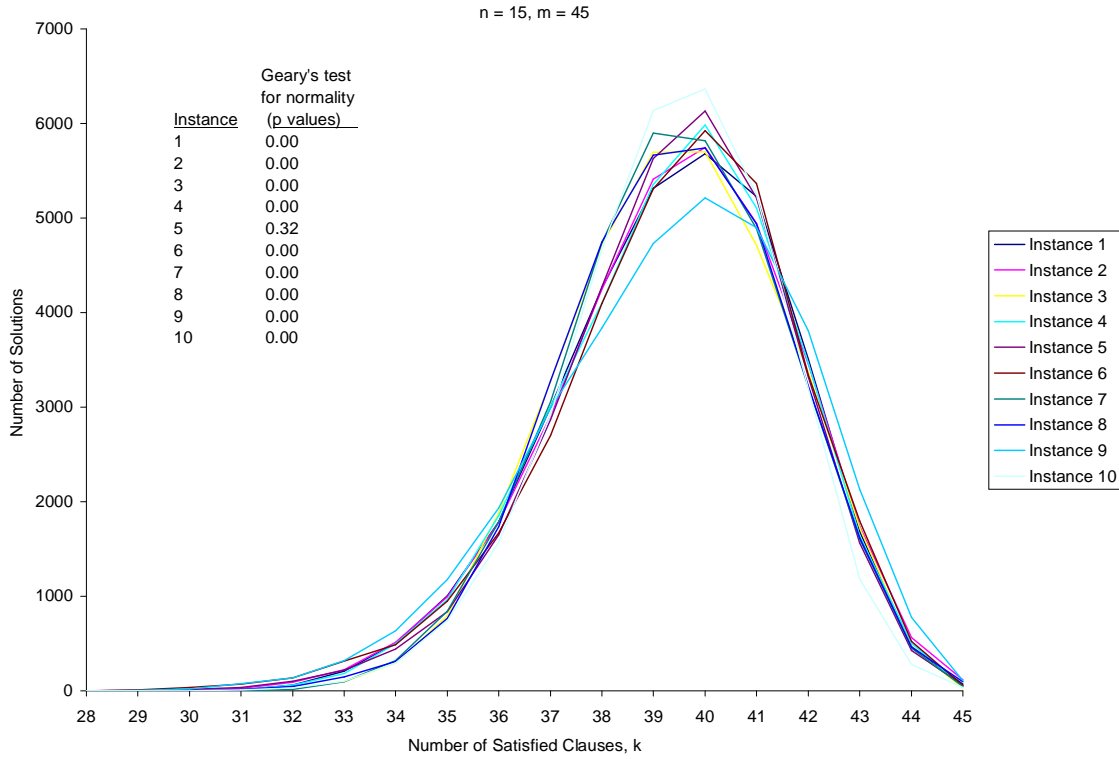


Figure A.2.3: Distribution of the Solution Space over the Number of Satisfied Clauses

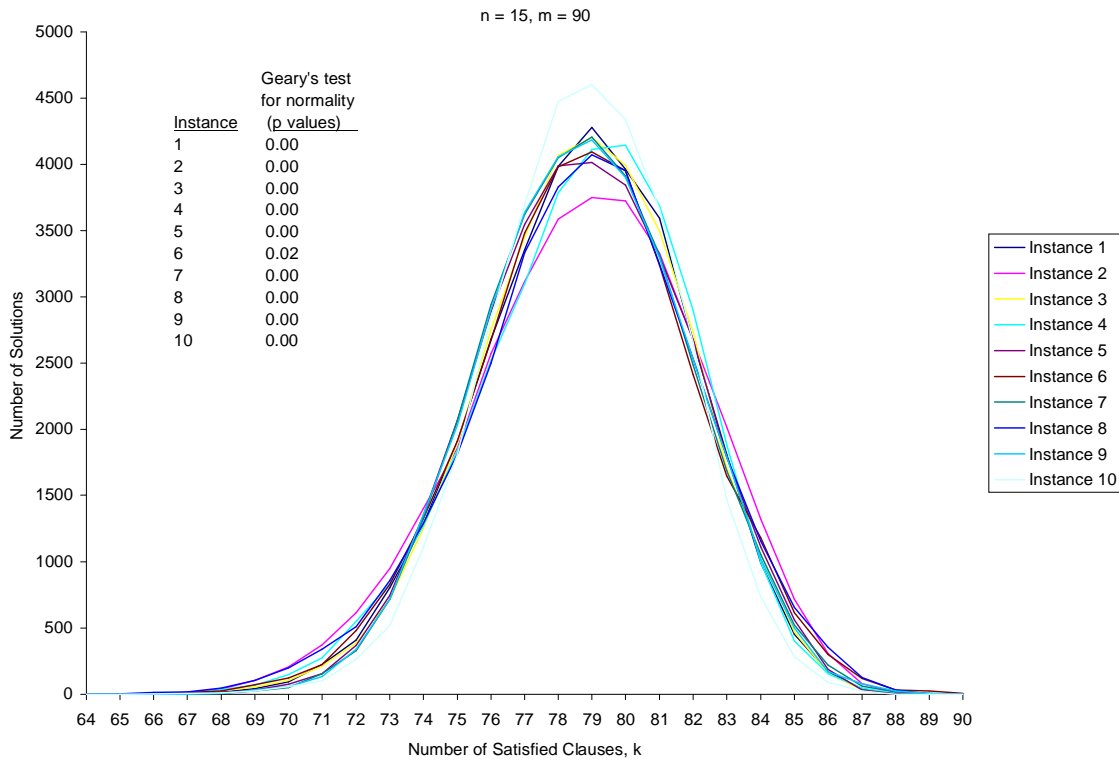


Figure A.2.4: Distribution of the Solution Space over the Number of Satisfied Clauses

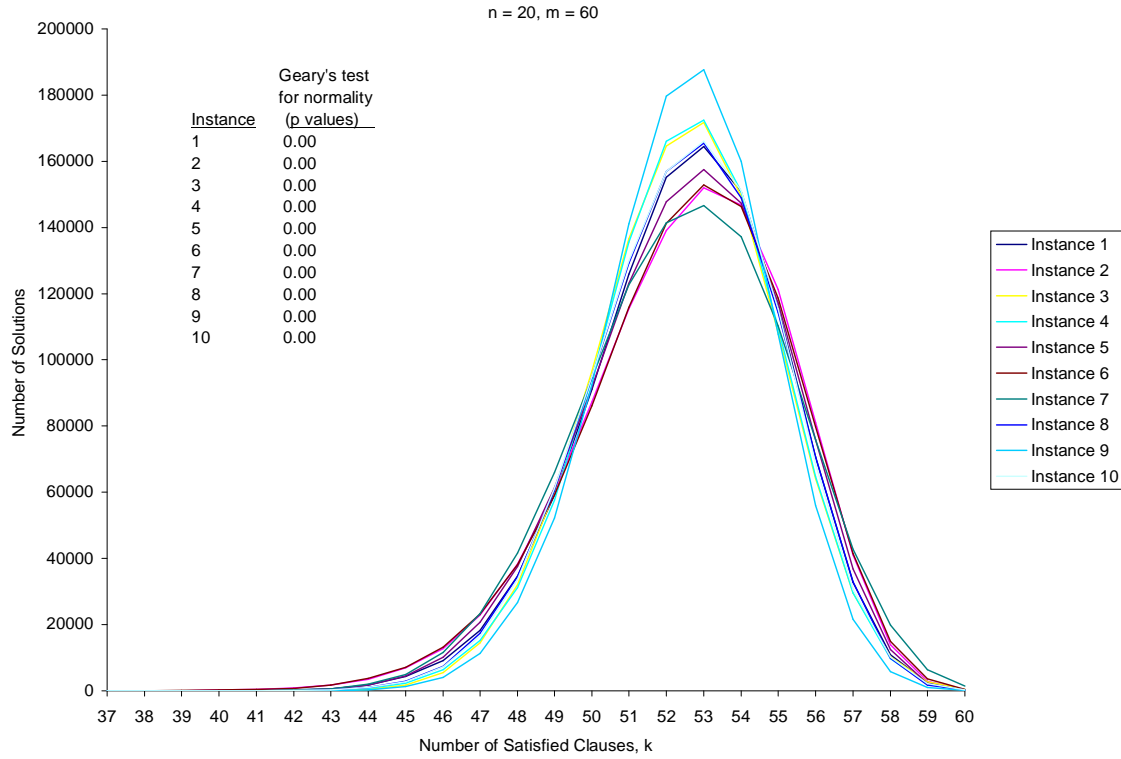


Figure A.2.5: Distribution of the Solution Space over the Number of Satisfied Clauses

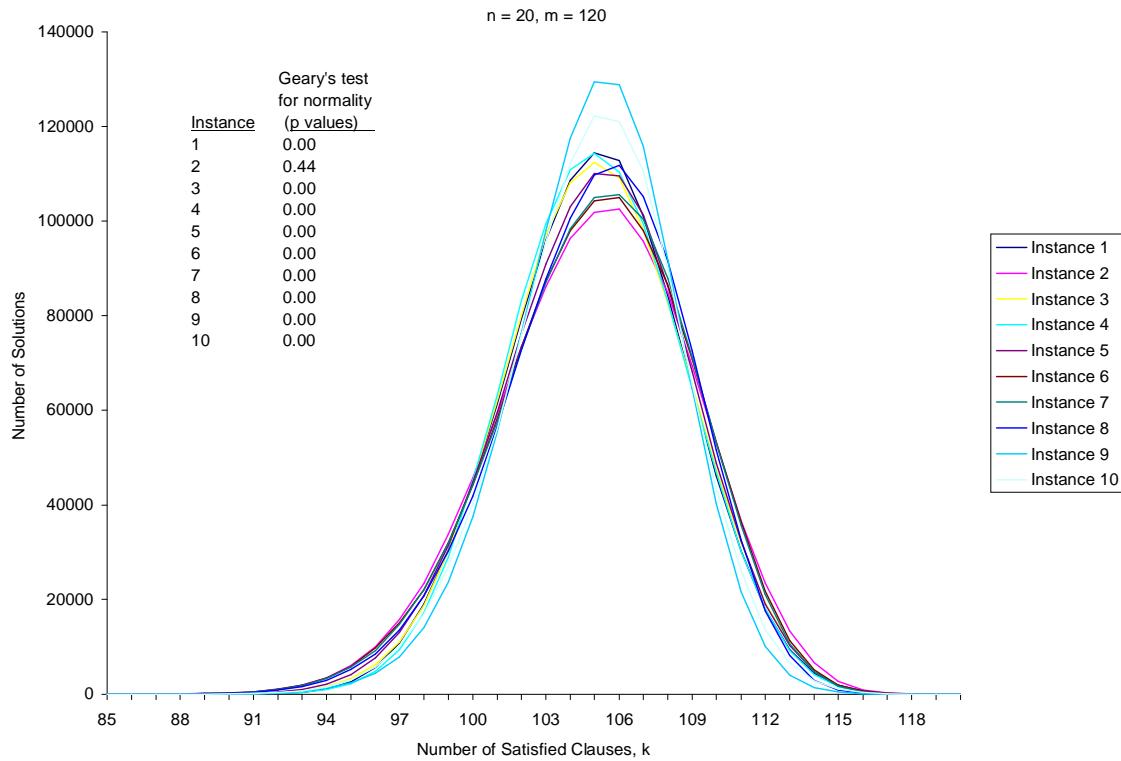


Figure A.2.6: Distribution of the Solution Space over the Number of Satisfied Clauses

Appendix B

Table B.1: Tabu Search versus Complete Enumeration

n = 20							
m	k	P{MAX(n,m) = k}		m	k	P{MAX(n,m) = k}	
		Complete Enumeration	Tabu Search			Complete Enumeration	Tabu Search
60	60	0.996	0.944	350	337	0.002	0.002
	59	0.004	0.056		336	0.014	0.014
70	70	0.966	0.866		335	0.058	0.054
	69	0.034	0.130		334	0.056	0.054
	68	0	0.002		333	0.094	0.088
	67	0	0.002		332	0.168	0.150
80	80	0.782	0.658		331	0.226	0.206
	79	0.218	0.312		330	0.158	0.150
	78	0	0.030		329	0.124	0.122
85	85	0.644	0.524		328	0.086	0.096
	84	0.348	0.400	327	0.014	0.052	
	83	0.008	0.066	326	0	0.008	
	82	0	0.010	325	0	0.002	
90	90	0.488	0.380	324	0	0.002	
	89	0.484	0.508	400	382	0.018	0.014
	88	0.028	0.100		381	0.016	0.018
	87	0	0.012		380	0.056	0.050
100	100	0.266	0.234		379	0.078	0.068
	99	0.584	0.506		378	0.124	0.118
	98	0.146	0.224		377	0.164	0.156
	97	0.004	0.034	376	0.242	0.210	
	96	0	0.002	375	0.166	0.166	
110	110	0.110	0.088	374	0.100	0.114	
	109	0.460	0.378	373	0.030	0.062	
	108	0.398	0.426	372	0.006	0.016	
	107	0.032	0.098	371	0	0.006	
	106	0	0.008	370	0	0.002	
	105	0	0.002				
120	120	0.044	0.024				
	119	0.266	0.238				
	118	0.524	0.460				
	117	0.152	0.234				
	116	0.014	0.034				
	115	0	0.006				
	114	0	0.002				

For each n and m, 500 instances are randomly generated (with replacement)
 MAX(n,m) = Maximum Number of Satisfied Clauses (given n and m)
 P{MAX(n,m) = k} = Frequency Distribution of 500 Instances over MAX(n,m)
 Obtained using "Complete Enumeration" and "Tabu Search"

Table B.1 (Continued....)

n = 20			
P{MAX(n,m) = k}			
m	k	Complete	Tabu
		Enumeration	Search
450	430	0.002	0.002
	429	0.002	0.002
	428	0.012	0.010
	427	0.018	0.018
	426	0.028	0.028
	425	0.080	0.070
	424	0.108	0.108
	423	0.156	0.152
	422	0.174	0.166
	421	0.216	0.172
	420	0.124	0.126
	419	0.070	0.076
	418	0.010	0.042
	417	0	0.020
	416	0	0.004
	415	0	0.002
	414	0	0.002

Table B.2.1: Tabu Search Results I

n = 50, L = 5, Problem Set # 1					
m	Minimum	Maximum	Average	Std. Dev.	Avg(Iter.#)
150	146	150	149.41	0.87	25.92
200	195	200	198.94	1.05	59.15
250	243	250	247.40	1.26	71.30
300	291	298	295.01	1.41	64.47
400	384	393	388.96	1.74	62.67
500	476	488	482.30	2.56	66.04
600	564	581	573.44	2.64	64.10
700	661	672	666.45	2.18	72.46
800	745	765	757.09	3.28	70.00
900	844	863	850.84	3.83	63.57
1000	932	947	941.83	3.06	63.86
1500	1381	1403	1395.49	4.15	66.72
2000	1834	1854	1844.27	4.82	76.30
2500	2288	2305	2296.41	3.73	63.43
3000	2732	2756	2743.00	4.91	72.42
3500	3180	3191	3186.46	2.41	101.11
4000	3629	3656	3647.59	5.07	72.76
4500	4081	4122	4108.36	6.55	72.17
5000	4531	4570	4557.10	6.57	66.27

For each n and m, 2 sets of 100 instances are randomly generated (with replacement)

Minimum = Minimum of "Maximum # of Satisfied Clauses", among 100 instances

Maximum = Maximum of "Maximum # of Satisfied Clauses", among 100 instances

Average = Average of "Maximum # of Satisfied Clauses" over 100 instances

Std. Dev = Standard Deviation of "Maximum # of Satisfied Clauses"

Avg(Iter#) = Average number of iterations to find the best solution (within each restart)

L = Length of the short-term tabu list (See the Section 4.2.2)

Table B.2.2: Tabu Search Results I

n = 50, L = 5, Problem Set # 2					
m	Minimum	Maximum	Average	Std. Dev.	Avg(Iter.#)
150	146	150	149.41	0.83	23.88
200	195	200	198.76	0.99	49.56
250	244	250	247.46	1.27	65.13
300	292	298	295.06	1.36	64.81
400	386	394	389.04	1.56	61.71
500	477	485	482.09	1.68	67.02
600	570	580	574.91	2.26	69.02
700	661	672	667.31	2.20	69.63
800	750	764	759.21	2.48	66.63
900	842	862	850.66	3.92	58.34
1000	933	950	939.30	3.29	69.08
1500	1385	1403	1395.38	3.35	62.67
2000	1840	1858	1848.47	4.33	74.34
2500	2280	2314	2303.08	5.97	70.57
3000	2727	2760	2742.63	5.34	79.30
3500	3162	3190	3180.90	3.57	75.18
4000	3628	3672	3658.48	6.67	67.59
4500	4087	4124	4115.13	6.39	63.33
5000	4527	4586	4572.34	8.95	63.30

Table B.2.3: Tabu Search Results I

n = 100, L = 10, Problem Set # 1					
m	Minimum	Maximum	Average	Std. Dev.	Avg(Iter.#)
300	298	300	299.77	0.45	147.38
400	396	400	398.35	0.95	135.36
500	491	499	494.98	1.61	132.05
600	586	594	590.50	1.93	127.92
1000	954	970	963.99	2.91	130.18
1500	1417	1433	1423.85	3.10	131.48
2000	1870	1892	1882.56	4.36	128.95
2500	2327	2367	2352.27	7.22	140.56
3000	2798	2822	2810.84	4.76	142.38
3500	3234	3251	3244.37	3.68	135.68
4000	3696	3716	3707.18	4.38	136.48
4500	4128	4166	4144.55	7.15	130.25
5000	4574	4615	4593.38	9.05	137.47

Table B.2.4: Tabu Search Results I

n = 100, L = 10, Problem Set # 2					
m	Minimum	Maximum	Average	Std. Dev.	Avg(Iter.#)
300	298	300	299.67	0.54	144.13
400	394	400	398.25	1.27	139.89
500	491	499	495.05	1.71	134.02
600	586	596	590.65	1.92	126.56
1000	960	970	965.34	2.50	121.69
1500	1418	1435	1426.68	3.39	133.30
2000	1869	1897	1883.10	5.07	127.98
2500	2329	2352	2338.84	5.14	139.56
3000	2787	2810	2798.90	4.43	146.67
3500	3241	3267	3253.79	5.36	137.17
4000	3694	3722	3712.27	5.72	138.38
4500	4165	4192	4179.02	5.40	131.58
5000	4589	4651	4638.05	8.98	135.51

Table B.2.5: Tabu Search Results I

n = 150, L = 15, Problem Set # 1					
m	Minimum	Maximum	Average	Std. Dev.	Avg(Iter.#)
500	491	500	497.86	1.74	157.09
600	593	600	597.33	1.39	247.24
700	691	699	695.10	1.71	285.42
800	787	796	791.61	1.99	312.93
900	882	891	887.01	1.96	300.36
1000	975	988	981.48	2.50	312.40
1500	1443	1459	1448.99	3.13	316.38
2000	1898	1925	1911.51	4.08	304.96
2500	2355	2382	2370.50	4.98	299.68
3000	2823	2841	2832.49	3.96	325.71
3500	3278	3297	3290.40	3.99	310.98
4000	3743	3771	3759.76	4.67	289.71
4500	4209	4233	4221.14	5.15	310.22
5000	4668	4711	4699.91	6.12	317.52

Table B.2.6: Tabu Search Results I

n = 150, L = 15, Problem Set # 2					
m	Minimum	Maximum	Average	Std. Dev.	Avg(Iter.#)
500	494	500	498.09	1.50	167.40
600	592	600	597.53	1.45	250.72
700	691	699	695.51	1.70	295.82
800	786	796	791.60	1.92	292.04
900	882	892	887.46	2.25	289.93
1000	975	987	981.93	2.35	325.52
1500	1440	1457	1448.97	3.31	307.44
2000	1899	1924	1911.83	4.32	297.97
2500	2360	2380	2370.07	4.56	302.90
3000	2824	2855	2843.46	5.92	288.39
3500	3277	3292	3284.71	2.90	308.83
4000	3737	3765	3757.37	3.96	328.23
4500	4208	4234	4221.26	5.30	302.53
5000	4633	4721	4705.88	11.10	302.76

Table B.3: Tabu Search Results II

n = 20, L = 4					
m	Minimum	Maximum	Average	Std. Dev.	Avg(Iter.#)
5	5	5	5.00	0	0.52
10	10	10	10.00	0	1.07
15	15	15	15.00	0	1.68
20	19	20	19.98	0.14	1.90
25	24	25	24.99	0.10	2.17
30	29	30	29.99	0.10	3.36
35	34	35	34.98	0.14	3.65
40	39	40	39.99	0.10	4.51
45	44	45	44.96	0.20	5.16
50	49	50	49.97	0.17	7.05
55	54	55	54.96	0.20	9.91
60	59	60	59.97	0.17	10.78
65	64	65	64.98	0.14	12.96
70	68	70	69.93	0.29	17.36
75	73	75	74.78	0.48	17.56
80	78	80	79.77	0.49	18.78
85	82	85	84.46	0.66	16.07
90	88	90	89.27	0.68	19.58
95	92	95	94.18	0.69	21.37
100	96	100	98.92	0.80	16.28
200	189	196	192.57	1.26	17.95
300	280	289	284.85	1.83	21.75
400	371	381	375.95	2.34	23.74
500	461	473	467.04	2.50	20.92
600	549	563	556.54	2.95	21.92
700	639	654	646.81	3.23	20.54
800	731	747	737.10	3.38	18.52
900	818	834	826.29	3.52	20.83
1000	908	929	915.88	3.75	20.24
1100	996	1017	1005.63	4.01	23.23
1200	1084	1104	1094.58	4.11	21.16
1300	1169	1193	1183.57	4.07	18.37
1400	1262	1283	1272.16	4.07	24.38
1500	1353	1374	1362.66	4.74	24.09
1600	1440	1463	1450.69	4.69	20.50
1700	1531	1552	1539.80	4.17	20.13
1800	1617	1643	1627.90	5.01	18.20
1900	1706	1727	1716.00	4.16	20.13
2000	1789	1823	1805.81	6.24	21.71

Table B.3 (Continued...)

m	Minimum	Maximum	Average	Std. Dev.	Avg(Iter.#)
2100	1883	1905	1893.24	4.55	22.71
2200	1969	2000	1982.54	6.03	20.22
2300	2056	2085	2070.97	5.63	20.34
2400	2147	2172	2159.22	5.29	24.88
2500	2234	2267	2248.16	5.59	19.83
2600	2321	2353	2336.45	6.21	20.14
2700	2410	2439	2422.81	6.06	20.88
2800	2501	2528	2512.56	5.18	22.98
2900	2588	2618	2601.00	6.06	21.47
3000	2676	2703	2689.01	5.45	20.93
3100	2763	2797	2775.37	5.72	20.49
3200	2850	2879	2864.52	5.86	21.36
3300	2940	2967	2951.95	5.52	21.80
3400	3025	3057	3040.32	6.53	21.20
3500	3110	3145	3129.34	6.67	21.18
3600	3200	3236	3215.78	6.73	22.23
3700	3286	3322	3303.75	6.36	22.36
3800	3378	3418	3390.88	6.77	21.07
3900	3464	3499	3478.72	6.38	21.68
4000	3551	3584	3566.67	6.12	20.99
4100	3642	3683	3655.76	7.41	22.35
4200	3728	3756	3742.68	6.26	20.50
4300	3817	3855	3830.06	6.96	23.18
4400	3904	3941	3917.18	7.49	20.28
4500	3990	4020	4004.09	5.65	22.91
4600	4079	4113	4091.97	6.21	20.79
4700	4166	4195	4180.62	5.80	18.94
4800	4241	4294	4266.82	7.31	23.06
4900	4342	4373	4355.09	5.68	21.51
5000	4427	4466	4443.01	6.73	20.65
5100	4514	4544	4529.19	6.51	20.00
5200	4600	4640	4617.80	7.66	23.56
5300	4690	4724	4704.77	6.96	20.32
5400	4782	4816	4792.26	6.58	21.67
5500	4862	4900	4879.85	7.38	19.27
5600	4952	4981	4965.97	5.82	18.38
5700	5039	5072	5053.50	6.54	20.93
5800	5121	5161	5140.78	7.19	20.54
5900	5217	5244	5228.74	6.01	21.14
6000	5303	5328	5314.74	5.52	22.51

Table B.3 (Continued...)

m	Minimum	Maximum	Average	Std. Dev.	Avg(Iter.#)
6100	5390	5416	5402.32	5.89	21.50
6200	5475	5506	5489.14	6.26	18.24
6300	5561	5592	5575.02	6.01	20.80
6400	5652	5679	5662.11	6.05	19.58
6500	5737	5764	5749.01	5.49	19.01
6600	5825	5857	5836.30	6.48	21.52
6700	5907	5943	5923.46	6.34	21.63
6800	5992	6025	6009.16	5.81	19.10
6900	6084	6110	6096.89	6.38	22.32
7000	6169	6204	6183.31	5.27	20.06
7100	6259	6286	6270.64	5.68	18.74
7200	6344	6367	6355.81	5.34	19.44
7300	6426	6463	6443.39	6.08	19.93
7400	6517	6547	6529.48	5.82	21.74
7500	6604	6629	6615.49	4.85	20.25
7600	6689	6716	6702.09	5.57	18.23
7700	6774	6803	6788.12	4.87	22.07
7800	6861	6888	6873.29	5.21	19.45
7900	6950	6972	6959.99	4.72	17.61
8000	7037	7061	7046.71	4.54	19.90
8100	7118	7144	7131.11	4.60	22.09
8200	7208	7231	7217.27	4.51	18.79
8300	7293	7321	7302.56	4.95	16.83
8400	7381	7397	7388.29	3.77	17.57
8500	7460	7484	7473.03	4.56	18.15
8600	7549	7572	7557.66	3.73	17.28
8700	7636	7650	7641.91	2.96	19.38
8800	7722	7733	7726.15	2.52	17.17
8900	7803	7817	7810.18	2.69	16.62
9000	7887	7897	7891.72	1.94	18.45
9100	7965	7972	7969.27	1.36	10.44

For each n and m, a set of 100 instances are randomly generated (without replacement)

Minimum = Minimum of "Maximum # of Satisfied Clauses", among 100 instances

Maximum = Maximum of "Maximum # of Satisfied Clauses", among 100 instances

Average = Average of "Maximum # of Satisfied Clauses" over 100 instances

Std. Dev. = Standard Deviation of "Maximum # of Satisfied Clauses"

Avg(Iter#) = Average number of iterations to find the best solution (within each restart)

L = Length of the short-term tabu list (See the Section 4.2.2)

Appendix C

Table C.1.1: Random Restart Local Search Results

n = 50, 500 Restarts				
m	Minimum	Maximum	Average	Std.Dev.
150	150	150	150.00	0
200	198	200	199.56	0.54
215	212	215	214.16	0.71
225	223	225	223.84	0.65
300	294	298	295.72	1.07

Table C.1.2: Random Restart Local Search Results

n = 50, 1000 Restarts				
m	Minimum	Maximum	Average	Std.Dev.
150	150	150	150.00	0
200	199	200	199.72	0.45
215	213	215	214.30	0.68
225	223	225	223.88	0.69
300	294	298	295.84	1.04

Table C.1.3: Random Restart Local Search Results

n = 100, 500 Restarts				
m	Minimum	Maximum	Average	St.Dev.
300	299	300	299.68	0.47
400	395	400	397.34	1.12
425	420	423	421.50	0.91
450	443	448	445.34	1.21
600	585	593	589.50	1.69

For each n and m, 50 instances are randomly generated (with replacement)
 Minimum = Minimum of "Maximum # of Satisfied Clauses", among 50 instances
 Maximum = Maximum of "Maximum # of Satisfied Clauses", among 50 instances
 Average = Average of "Maximum # of Satisfied Clauses" over 50 instances
 Std.Dev. = Standard Deviation of "Maximum # of Satisfied Clauses"

Table C.1.4: Random Restart Local Search Results

n = 100, 1000 Restarts				
m	Minimum	Maximum	Average	Std.Dev.
300	299	300	299.88	0.33
400	397	399	397.76	0.74
425	419	424	421.98	1.10
450	444	448	445.74	0.92
600	586	594	589.94	1.70

Appendix D

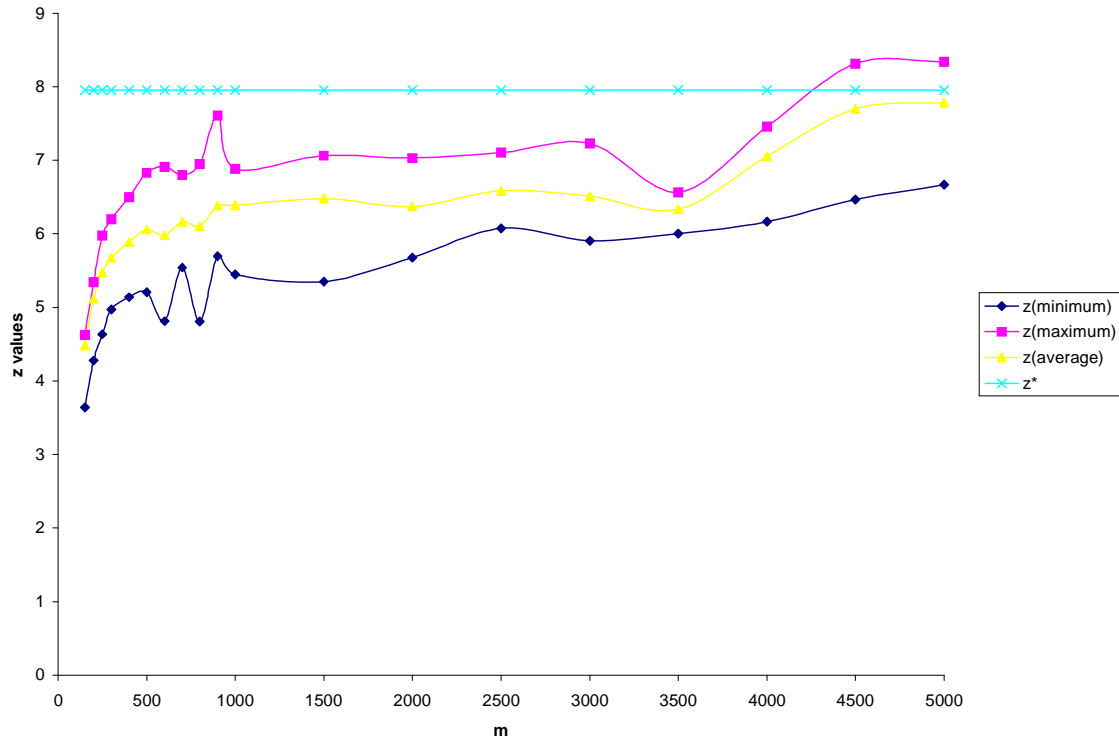


Figure D.1: z Values for Tabu Search (n = 50)

z(minimum), z(maximum) and z(average) values are obtained from

Table B.2.1 (for Figure D.1), Table B.2.3 (for Figure D.2) and Table B.2.5 (for Figure D.3)

using the formula
$$z(x) = \frac{x - (7m/8)}{\sqrt{(7m/64)}}$$

z* value (fixed) is obtained from Table 5.1

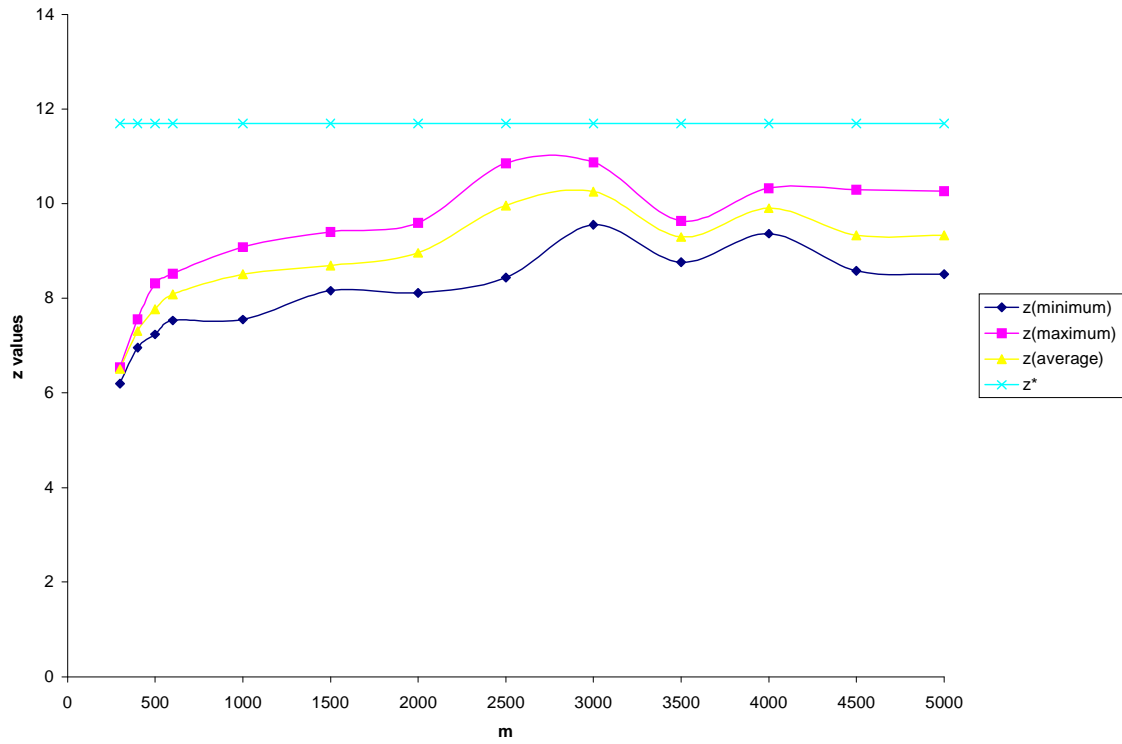


Figure D.2: z Values for Tabu Search (n = 100)

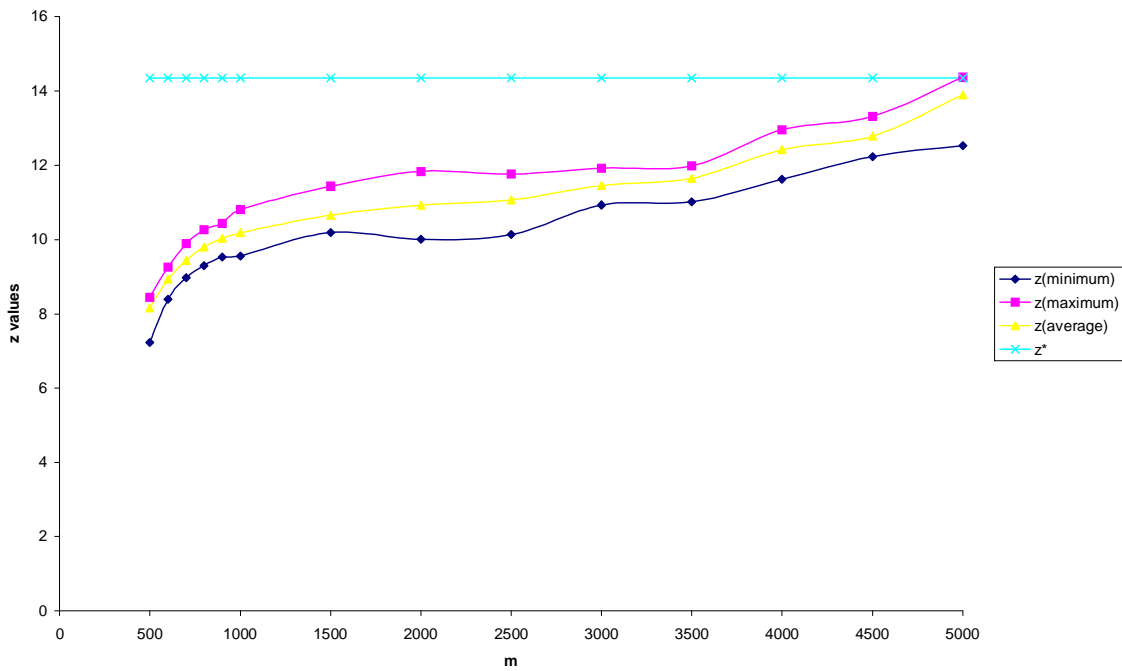


Figure D.3: z Values for Tabu Search (n = 150)

Bibliography

Aspvall, B., M. F. Plass, and R. E. Tarjan, (1979). "A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas." *Information Processing Letters* 8(3), 121-132.

Bitner, J. R., and E. M. Reingold, (1975). "Backtrack Programming Techniques." *Communications of the ACM* 18(11), 651-656.

Buro, M., and H. K. Buning, (1993). "Report on a SAT Competition." *Bulletin of the European Association for Theoretical Computer Science* 49, 143-151.

Blair, C. E., R. G. Jeroslow, and J. K. Lowe, (1986). "Some Results and Experiments in Programming Techniques for Propositional Logic." *Computers and Operations Research* 5, 633-645.

Ceria, S., P. Nobili, and A. Sassano, (1998). "A Lagrangian-Based Heuristic for Large-Scale Set Covering Problems." *Mathematical Programming* 81(2), 215-228.

Charon, I., and O. Hudry, (1993). "The Noising Method: A New Method for Combinatorial Optimization." *Operations Research Letters* 14, 133-137.

Cook, S. A., (1971). "The Complexity of Theorem-Proving Procedures." *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York, 151-158.

Cook, S. A., and D. G. Mitchell, (1997). "Finding Hard Instances of the Satisfiability Problem : A Survey." *DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 35: Satisfiability Problem: Theory and Applications: Proceedings of a DIMACS Workshop*, March 11-13, 1996, (D. Du, J. Gu and P. M. Pardalos, Editors), 1-17.

Cormen, T. H., C. Leiserson, and R. Rivest, (1997). *Introduction to Algorithms*, McGraw-Hill, New York.

Coullard, C. R., A. B. Gamble, and P. C. Jones, (1998). "Matching Problems in Selective Assembly Operations." *Annals of Operations Research* 76, 95-107.

Creignou, N., (1993). "The Class of Problems that are Linearly Equivalent to Satisfiability or a Uniform Method for Proving NP-Completeness." *Lecture Notes in Computer Science* 702, 115-133.

Davis, M., G. Logemann, and D. Loveland, (1962). "A Machine Program for Theorem-Proving." *Communications of the ACM* 5, 394-397.

Davis, M., and H. Putnam, (1960). "A Computing Procedure for Quantification Theory." *Journal of the ACM* 7, 201-215.

Devadas, S., (1989). "Optimal Layout via Boolean Satisfiability." *1989 IEEE International Conference on Computer-Aided Design*, 294-297.

Dowling, W. F., and J. H. Gallier, (1984). "Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae." *Journal of Logic Programming* 1, 267-284.

Dubois, O., P. Andre, Y. Boufkhad, and J. Carlier, (1996). "SAT versus UNSAT." *DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 26: Cliques, Coloring and Satisfiability: Second DIMACS Challenge*, 415-434.

Dueck, G., and T. Scheuer, (1990). "Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing." *Journal of Computational Physics* 90, 161-175.

Eglese, R. W., (1990). "Simulated Annealing: A Tool for Operational Research." *European Journal of Operational Research* 46, 271-281.

Fleischer, M. A., (1995). "Simulated Annealing: Past, Present, and Future." *Proceedings of the 1995 Winter Simulation Conference*, 155-161.

Frieze, A., and S. Suen, (1996). "Analysis of Two Simple Heuristics on a Random Instance of k-SAT." *Journal of Algorithms* 20(2), 312-355.

Galil, Z., (1977). "On the Complexity of Regular Resolution and the Davis-Putnam Procedure." *Theoretical Computer Science* 4, 23-46.

Garey, M. R., and D. S. Johnson, (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman and Company, San Francisco, California.

Glover, F., and M. Laguna, (1997). *Tabu Search*, Kluwer Academic Publishing, Norwell, Massachusetts.

Goerdts, A., (1992). "A Threshold for Unsatisfiability." *Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science*, Prague, August 1992.

Gu, J., (1993). "Local Search for Satisfiability (SAT) Problem." *IEEE Transactions on Systems, Man, and Cybernetics* 23(4), 1108-1129.

Gu, J., (1997). "Multispace Search for Satisfiability and NP-Hard Problems." *DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 35: Satisfiability Problem: Theory and Applications: Proceedings of a DIMACS Workshop*, March 11-13, 1996, (D. Du, J. Gu and P. M. Pardalos, Editors), 407-517.

Gu, J., P. W. Purdom, J. Franco, and B. W. Wah, (1997). "Algorithms for the Satisfiability Problem: A Survey." *DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 35: Satisfiability Problem: Theory and Applications: Proceedings of a DIMACS Workshop*, March 11-13, 1996, (D. Du, J. Gu and P. M. Pardalos, Editors), 19-130.

Haken, A., (1985). "The Intractability of Resolution." *Theoretical Computer Science* 39, 297-308.

Hansen, P., and B. Jaumard, (1990). "Algorithms for the Maximum Satisfiability Problem." *Computing* 44, 279-303.

Hochbaum, D. S., and A. Pathria, (1997). "Generalized p-Center Problems: Complexity Results and Approximation Algorithms." *European Journal of Operational Research* 100(3), 594-607.

Hoel, P. G., S. C. Port, and C. J. Stone, (1971). *Introduction to Probability Theory*, Houghton Mifflin, Boston.

Hooker, J. N., (1988). "Generalized Resolution and Cutting Planes." *Annals of Operations Research* 12, 217-239.

Jacobson, S. H., and E. Yücesan, (2001). "Assessing the Performance of Generalized Hill Climbing Algorithms." Technical Report.

Janson, S., Y. C. Stamatiou, and M. Vamvakari, (2000). "Bounding the Unsatisfiability Threshold of Random 3-SAT." *Random Structures and Algorithms* 17, 103-116.

Jaumard, B., M. Stan, and J. Desrosiers, (1996). "Tabu Search and a Quadratic Relaxation for the Satisfiability Problem." *DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 26: Cliques, Coloring and Satisfiability: Second DIMACS Challenge*, 457-477.

Joy, S., J. Mitchell, and B. Borchers, (1997). "A Branch and Cut Algorithm for MAX-SAT and Weighted MAX-SAT." *DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 35: Satisfiability Problem: Theory and Applications: Proceedings of a DIMACS Workshop*, March 11-13, 1996, (D. Du, J. Gu and P. M. Pardalos, Editors), 519-536.

Kautz, H., and B. Selman, (1992). "Planning as Satisfiability." in *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 92)*, August 1992.

Kumar, A., S. H. Jacobson, and E. C. Sewell, (2000), "Computational Analysis of a Flexible Assembly System Design Problem." *European Journal of Operational Research*, 123(3), 453-472.

Larrabee, T., (1992). "Test Pattern Generation Using Boolean Satisfiability." *IEEE Transactions on Computer Aided Design* 11(1), 4-15.

Law, A. M., and W. David Kelton, (1991). *Simulation Modeling & Analysis*, McGraw-Hill, New York.

Liepins, G. E., and M. R. Hilliard, (1989). "Genetic Algorithms: Foundations and Applications." *Annals of Operations Research* 21, 31-58.

Mazure, B., L. Sais, and E. Gregorie, (1995). "Tabu Search for SAT." *Proceedings of CP'95 Workshop on Solving Really Hard Problems*, 127-130.

Muhlenbein, H., (1997). "Genetic Algorithms." *Local Search in Combinatorial Optimization* (E. Aarts and J. K. Lenstra, Editors), 137-172.

Mitchell, D., B. Selman and H. Levesque, (1992). "Hard and Easy Distributions of SAT Problems." *Proceedings of AAAI'92*, 459-465.

Purdom, P. W., (1983). "Search Rearrangement Backtracking and Polynomial Average Time." *Artificial Intelligence* 21, 117-133.

Resende, M. G. C., and T. A. Feo, (1996). "A GRASP for Satisfiability." *DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 26: Cliques, Coloring and Satisfiability: Second DIMACS Challenge*, 499-520.

Rushforth, C. K., and W. Wang, (1997). "Local Search for Channel Assignment in Cellular Mobile Networks." *DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 35: Satisfiability Problem: Theory and Applications: Proceedings of a DIMACS Workshop*, March 11-13, 1996, (D. Du, J. Gu and P. M. Pardalos, Editors), 689-709.

Selman, B., H. Kautz, and B. Cohen, (1996). "Local Search Strategies for Satisfiability Testing." *DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 26: Cliques, Coloring and Satisfiability: Second DIMACS Challenge*, 521-531.

Selman, B., H. Levesque, and D. Mitchell, (1992). "A New Method for Solving Hard Satisfiability Problems." *Proceedings of AAAI'92*, 440-446.

Smith, S., and C. C. Cheng, (1993). "Slack-Based Heuristics for Constraint Satisfaction Scheduling." *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 440-446.

Spears, M. W., (1996). "Simulated Annealing for Hard Satisfiability Problems." *DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 26: Cliques, Coloring and Satisfiability: Second DIMACS Challenge*, 533-557.

Tseitin, G. S., (1968). "On the Complexity of Derivations in Propositional Calculus." *Structures in Constructive Mathematics and Mathematical Logic*, Part II, 115-125.

Wadsworth, G. P., and J. G. Bryan, (1960). *Introduction to Probability and Random Variables*, McGraw-Hill, New York.

Walpole, R. E., and H. Myers, (1993). *Probability and Statistics for Engineers and Scientists*, MacMillan, New York.

Vita :

Tevfik Aytemiz

Mersin Universitesi

İktisadi ve İdari Bilimler Fakültesi

Ciftlikkoyu Kampusu, 33342, MERSİN

TURKEY

Education

2001 Ph.D. in Industrial and Systems Engineering, Virginia Tech, Virginia.

1997 M.S. in Operations Research, University of New Haven, Connecticut.

1991 B.S. in Business Administration, Ankara University, Ankara, Turkey.

Employment

1994-Present Research Assistant, Mersin Universitesi, Mersin, Turkey.

1993-1994 Managerial Accounting Specialist, MAGE Inc., Ankara, Turkey.