RICE UNIVERSITY

**Analysis on the Assignment Landscape of 3-SAT problems**

by

**Xiaoxu Wang**

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

**MASTER OF SCIENCE**

Approved, Thesis Committee:

---

Moshe Vardi , Chair
Professor of Computer Science

---

Devika Subramanian
Professor of Computer Science

---

Robert Cartwright
Professor of Computer Science

---

Daniel Osherson
Professor of Psychology

Houston, Texas

June, 2004

# ABSTRACT

## Analysis on the Assignment Landscape of 3-SAT problems

by

Xiaoxu Wang

The complexity of random 3-SAT problems is one of the fundamental problems in Computer Science theory. 3-SAT problems shift from satisfiable to unsatisfiable at the crossover point, and show high complexity around it. Using the CUDD package and the image computation method, we empirically characterize the structure of all solutions to random 3-SAT instances. This thesis confirms that solutions to a 3-SAT instance are clustered as predicted by the spin model in statistical physics[]. Our experimental results indicate that the solution set breaks into several clusters when the density falls into a specific range. As the density grows, the number of clusters increases and peaks at density 3.2. Beyond density 3.2, the number of clusters in the solution set drops and non-solution basins in the assignment landscape emerge. These non-solution basins cause high complexity in search-based SAT solvers at densities right below the crossover point of SAT problems.

# Acknowledgments

First and foremost, I wish to express sincere appreciation to my advisor Dr. Vardi for his insightful instructions on the work in this thesis. I would like to thank Dr. Subramanian many for important suggestions on this project. The thoughts they offered have enriched my thesis. It was their guidance and encouragement make the work proceed. I would like the thank Dr. Cartwright and Dr. Osherson for being members of my thesis committee. In addition, special thanks are due to Demetrios D. Demopoulos and Guoqiang Pan who were helpful during the early programming phase. I also thank Deian Tabakov and James Hsia for their careful proof reading. I would like to thank my parents and my husband for their support.

This project was supported in part by NSF grants CCR-9700061 and IIS-9908435, and by a grant from the Intel Corporation.

# Contents

# List of Figures

# Chapter 1
# Introduction

Many computational tasks of practical interest, such as the "Hamilton cycle" and the "Travelling salesman problem", are NP-complete problems [34]. As introduced in [22] and [14], NP stands for "non-deterministic polynomial time", which means that the set of decision problems is solvable in polynomial time on a non-deterministic Turing machine. Informally, the correctness of a solution can be checked in polynomial time, but it usually takes no less than exponential time to find a solution. If all other NP-problems can be reduced into this problem, it is called an NP-hard problem. An NP-complete problem is both NP-hard and in NP. The complexity of NP-complete problem is critical in the fundamental theory of computer science. If an NP-complete problem can be solved in polynomial time, then all of the NP problems can be solved in polynomial time. 3-SAT problem is an archetypal NP-complete problem which has received a great deal of attention in the last few decades.

An instance of 3-SAT is a boolean formula in conjunctive normal form(CNF) [43], which consists of a conjunction of clauses, each one a disjunction of three literals. A literal is either a proposition or the negation of a proposition. If a clause is a disjunction of K literals, the problem is called a K-SAT problem. When K is not specified, the problem is generally referred as a SAT problem. The goal is to find a truth assignment that satisfies all clauses. The number of variables is called the order

of the instance. The ratio of the number of clauses to the order is called density. Intuitively, a higher density suggests more constraints and lower probability of satisfiability. Surprisingly, the probability of satisfiability does not decrease gradually as the density grows. Previous experiments [6, 15] have shown that it suddenly drops from one to zero at density 4.26. This point is called the crossover point.

A number of different algorithms [30] have been developed to solve SAT problems. Some of them transform the 3-SAT problem to another problem and solve it with the existing methods in that field. For example, a 3-CNF formula can be transformed to an instance of integer programming and then solved using an approach, such as linear programming relaxations [25], branch-and-bound [13], cutting-plane and branch-and-cut [26]. These methods are fast for certain classes of problems. However, these methods often fail to solve hard instances of satisfiability. This thesis treat 3-SAT as a search/inference problem.

Traditional methods treat 3-SAT as a discrete, constrained decision problem and apply discrete search and inference procedures to determine a solution. This category of algorithms includes backtracking algorithms [40], consistency algorithms, Binary Decision Diagrams [2], and resolution [18], and so on. Most of them are complete, which means they can definitely determine whether an input has a solution or not. In addition, they can find all solutions of an instance. One of their drawbacks is that they consume too many resources when run on an instance with a high order. Instances with higher order cannot be solved in reasonable time and space. For

example, the Davis-Putnam-Logemann-Loveland (DPLL) [18, 40] procedure solves 3-SAT problems by unit propagation and splitting. In the worst case, the search tree generated by DPLL will be proportional to $2^{order}$. The size of search tree quickly grows out of control.

To trade off completeness for scalability, another approach transforms 3-SAT problems to discrete, unconstrained minimization problems and then solves them by local search [24]. Each set of truth assignments of all the variables can be viewed as a configuration in these methods. The number of unsatisfied clauses is usually taken as the energy of the configuration. The goal is to find the configuration with minimum energy. For a satisfiable formula, the minimal energy configuration has an energy value of zero. A local search algorithm starts from a random configuration, moves to one of its neighbors on each step, and stops when there is no suitable neighbor to move to. Many heuristic methods are often combined and applied together in local search. A greedy algorithm always selects the best neighbor to move to. The goal of local search is to find a global energy minimum, which usually is the optimum solution of the problem. Unfortunately, greedy algorithms can be easily trapped at local minima. To avoid being trapped at local minima, random walk, simulated annealing, tabu list and tunnelling heuristic are introduced [7, 8, 9]. This group of methods is incomplete. If a formula has no solution, the algorithms keep searching till they reach the number of maximum moves, which is set in advance. However, they can find solutions for those satisfiable instances with high order and high density within

a reasonable resource limit.

Since local search methods are not complete, most research on the complexity of 3-SAT problems are based on the DPLL algorithm. Both the complexity of finding a solution if the instance is satisfiable and the complexity of deciding there is no solution if unsatisfiable are examined as functions of density. Intuitively, at low densities, variables are under-constrained. There are many solutions and they are easy to be found. At high densities, variables are over-constrained. Unsuccessful branches can be found early during tree search procedures. Selman's experiments [17] show that hard instances of 3-SAT problems that demand far more effort than others emerge around the crossover point. Lots of attention has been cast on those high complexity cases near the crossover point for years [6, 32, 53]. Experiments [1, 12] reveal that the complexity of 3-SAT is highly solver-dependant. When solved by DPLL solvers such as GRASP or by integer programming solvers such as CPLEX, the complexity of 3-SAT instance peaks at the crossover point. The complexity of 3-SAT solved by Binary Decision Diagram system such as CUDD shows a phase transition at density 2.

Monasson (2001) [47, 51] and Mezard (2002) [42] use the spin model in statistical mechanics on 3-SAT problems. The basic idea of statistical mechanics is to describe a system of particles in a probabilistic way in order to deduce macroscopic features as emergent statistical properties. The system will be in state $S$ with probability $p(S)$, which depends on the temperature $T$ and the energy of the state $E(S)$. An Ising spin

system consists of N spins $S = (S_1, S_2, ..., S_N)$, $S_i = \{-1, 1\}$. Usually the energy is defined as a function of the states, depending on the real problem to which the spin model is applied. The entropy is the natural logarithm of the number of states having a specific energy. It is more convenient to be defined as a function of the temperature $T$. The particles in liquid can be taken as a spin system. When a liquid is cooled extremely slowly, there might be a sudden drop of the entropy, like solidification. This reminds of the crossover point of 3-SAT problem, where the number of solutions suddenly drops at some specific density.

An instance of 3-SAT problems is equivalent to the spin system if a Boolean variable $v_i = \{false, true\}$ is mapped onto a spin $S_i = \{-1, 1\}$, each assignment to the variables represents a state of the spin system, and the energy of a state is defined as the number of unsatisfied clauses. Therefore, the ground state energy in the spin system corresponds to the lowest number of unsatisfied clauses in 3-SAT problems. In the spin model, the SAT-UNSAT phase transition shows as an increase of the ground state energy from zero to a positive value. The ground state entropy is the natural logarithm of the number of solutions to a MAX-SAT problem, which looks for truth assignments that maximizes the number of satisfied clauses of a SAT instance. Analytical calculations on both energy and entropy [47] in the spin model confirm the crossover point obtained from experiments.

Replica method is a mathematical approximation first introduced by Mezard [44]. The basic idea is to involve multiple replicas of the original system and investigate

one of them to remove the difficulty of getting an average of a logarithm in the energy. It can be applied to those 3-SAT instances with low density, say, below 3.94. That point is another important phase transition, called the replica symmetry breaking (RSB) [45] point. Above that, systems at different time are not symmetric anymore, and we are not able to take data obtained from one of them as the average value.

Below this density, all the N-dimensional spins are replica symmetry. Any pair of them is separated by the same Hamming distance $d$. Solutions are gathered in a single cluster of diameter $dN$. Above this density of RSB, the space of solutions breaks into an exponential number (in N) of different clusters. The Hamming distance between solutions belonging to different clusters remains nearly constant up to the crossover point. Meanwhile, the intra-cluster Hamming distance decreases rapidly, which means that the solutions in the same cluster become more and more similar. The complexity of 3-SAT instances solved by GRASP and COLEX MIP solver also exhibit a phase transition around density 3.8, where the median running time shifts from being polynomial to being exponential in order [12]. It shows that after RSB, the thermal distribution for a single spin is simply a sum of Gaussians of magnitude specified by exponentially distributed energies, each centered on some random point in the N dimensional space. This naturally produces a picture in which the distribution function has many minima of different sizes, distributed in a hierarchical way.

The clustering behavior of 3-SAT problems is also showed by Enrenrich's experiments (2003) [19]. He compared the performance of genetic algorithm, simulated

annealing and tabu algorithm in solving 3-SAT problems. High dimensional solutions obtained from different methods are plotted on a plane by multiple dimensional scaling (MDS) [39]. The clusters of solutions can be clearly observed from the figures. However, do these algorithms give us a real view of the whole solution set? The heuristic methods used in Enrenrich's experiments are not complete. We cannot tell whether or not they favor some of the solutions . The clustering phenomenon can be explored more thoroughly by complete algorithms. Our methods are based on binary decision diagrams (BDDs) [2, 11, 38], which provide efficient symbolic representation of both Boolean formulas and their solution sets, gives the exact description of the whole solution set.

Consider all assignments to N variables in a 3-SAT instance. Each assignment takes the number of unsatisfied clauses as its energy. All assignments constitute a landscape with their energy as their altitude. The complexity of the local search methods is highly related to the ruggedness of landscape,based on the N-k model introduced in Kauffman's book [35, 36]. The local search easily finds the goal on a smooth landscape and will spend lots of time searching in a rugged landscape. On a rugged surface, it is more likely to be trapped in some local minima, which leads to a large amount of useless wandering.

This thesis explores the relationship between the complexity of a random 3-SAT problem. We usually plot the complexity with two parameters, the order and the density of a 3-SAT instance. The thesis focuses on the assignment landscape, the

structure of the solution set and their influence on the complexity of a 3-SAT problem. The findings coming from the statistical model are introduced at first, followed by some search methods used in SAT solvers. After that, the symbolic methods used in our experiments to explore the assignment landscape and the solution set are described. Right after the methods, the result of the experiments are presented with analysis. The intuitive conclusions and future work are discussed at the end of the thesis.

# Chapter 2
# Phase Transitions in SAT

From a statistical mechanics perspective, a phase transition is the emergence of non-trivial macroscopic behavior in a system composed of a large number of elements that follow simple microscopic laws. A system may have totally different macroscopic properties in different phases although it is in the same microscopic structure, say, molecules and magnetic spins. In our case, a SAT instance is composed of clauses. It may present totally different properties from a physical, mechanical or electrical point of view. The SAT problems have different computational complexity in different phases. What's more, the abrupt change of phase follows from very a slight modification of a control parameter. For instance, a slightly increase of the density in SAT problems will cause large difference of the computational complexity. Statistical mechanics focuses on the fundamental questions like: How can main properties of a system change abruptly and how are these properties related to the microstructure of the system?

## 2.1 Ising model

The early effort to investigate the microscopic structure by Newton's fundamental law is not successful for that it yields complicated equations that are hard to handle. As described in K.Huang's book(1967) [28], the basic idea of statistical mechanics is to describe a system of particles in a probabilistic way and deduce macroscopic

properties as emergent statistical properties.

The ideas of ergodicity and thermodynamical equilibrium are first introduced by Boltzmann in the 19th century. Suppose there is a system of N particles. For convenience, assume each particle has two positions, one positive and one negative, denoted by 1 and -1 respectively. A specification of the positions of all N particles is called a configuration of the system, denoted by $C$. When the system is in equilibrium a configuration $C$ has a probability $p(C)$ to be realized. The probability depends on temperature $T$, and equals

$$p(C) = \frac{1}{Z} exp(-\frac{1}{T} E(C)) \tag{2.1}$$

In the above expression, energy $E(C)$ is a real-valued function defined on the configuration C.

The partition function Z ensures the correct normalization of the probability distribution p,

$$Z = \sum_{C} exp(-\frac{1}{T} E(C)) \tag{2.2}$$

Consider the two limiting cases:

- Infinite temperature $T \to \infty$: The probability of spin in configuration C $p(C) \to \frac{1}{Z}$ is independent of configuration $C$. $Z$ is a parameter obtained from a whole system. It can be considered as a constant for a single configuration. because $Z$ is symmetric over all the configurations. The system is in a fully disordered phase, like gas. A spin could be in any configuration with same probability.

- Zero temperature $T = 0$: From the analytic computation of formula (2.1), The probabilities will concentrate on those configurations with the minimum energy, in other words, most of spins are fixed at the ground states. Those stable spins are solutions of the $MAX\_SAT$ problems, which ask for the assignments with lowest number of unsatisfied clauses. These strong constraints make the system achieve a perfect "ordered" state. As mentioned in the introduction, the energy and the entropy of the state are closely related to some features of SAT problems.

Consider the basic definition of average energy at a specific temperature, called the inner energy of a system at temperature $T$

$$\langle E \rangle_T = \sum_C p(C)E(C) \tag{2.3}$$

The following identity can be derived by combining the above definition with the equation of probability. Denote $\beta = \frac{1}{T}$.

$$\langle E \rangle_T = -\frac{d}{d\beta} lnZ \tag{2.4}$$

The above equation can be extended to higher moments of the energy. The vaiance of E can be computed from the following equation.

$$\langle E^2 \rangle_T - \langle E \rangle_T^2 = \frac{d}{d\beta} lnZ \tag{2.5}$$

Such equations suggest that the partition function is the generating function of energies. Therefore, once the partition function is known, the inner energy of a system can be derived from the above equations directly.

The "disorder" degree can be measured by the entropy $\widehat{S}(E) = lnN(E)$, where $N(E)$ is the number of configurations having energies $E$. Intuitively, if a spin with a

certain energy can be located at many probable positions, the system is disorder and tends to have high entropy. $\widehat{S}$ is showed as the average value over the temperature $\langle S\rangle_T$ more often. $\langle S\rangle_T$ can be obtained from thermodynamics theory,

$$\langle S\rangle_T = -\frac{1}{T}(F(T) - \langle E\rangle_T) \tag{2.6}$$

where

$$F(T) = -TlnZ(T) \tag{2.7}$$

is called the free energy of the system. This formula applies to very slow changes in systems with constant temperatures. So far, as long as the partition function is known, both the energy and the entropy of the ground state can be obtained by leting the temprature get close to zero.

The inner energy of system is composed of two parts. One part is the free energy, which can do work. The other part cannot do work and is called binding energy. The binding energy is defined as the product of entropy and temperature. According the second law of thermodynamics, the world acts spontaneously to maximize the entropy. The entropy of a system increases when the system is heated, or when its volume is increased while keeping the temperature constant. The entropy also changes during phase transitions, say, evaporation and dissolution.

## 2.2 Ising spin model for SAT

A K-SAT instance with N variables and M clauses can be viewed as a system composed of N particles. The logical value true is mapped to the positive spin position and the logical value false is mapped to the negative spin position. An assignment to

N variables can be viewed as a configuration of N particles. The random clauses are encoded into an $M * N$ matrix $C_{li}$ as this:

$$C_{li} = \begin{cases} 1 & \text{if the clause } C_l \text{ includes } v_i; \\ -1 & \text{if the clause } C_l \text{ includes } \neg v_i; \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

For a single clause $l$, $\sum_{i=1}^{N} C_{li} v_i$ reaches its lower bound $-K$ when all the literals in the clause are evaluated to be false. Using the Kronecker function

$$\delta(x) = \begin{cases} 1 & \text{if } x = 0; \\ 0 & \text{Otherwise.} \end{cases} \tag{2.9}$$

the value of $\delta(\sum_{i=1}^{N} C_{li} v_i + K)$ equal one iff the clause is evaluated to false, otherwise it quals zero. The number of unsatisfied clauses, denoted as the energy of a configuration, can be obtained from

$$E(C, V) = \sum_{l=1}^{M} \delta(\sum_{i=1}^{N} C_{li} v_i + K) \tag{2.10}$$

The ground state energy is the lowest number of unsatisfied clauses that could be reached by possible truth assignments (R. Monasson, 1997) [46]. Spins with ground state energy are solutions of the MAX-SAT problems. Indicated in A. M. Frienze [5], with a fixed number of clauses and a fixed number of variables, the ground state energy is tightly concentrated on its mean value. Therefore, the average ground state energy can be taken as the groud state energy of this pair of M and N. Those unfrozen spins do not contribute to the energy when $T \to 0$, but to the entropy (R. Monasson, 2000) [20]. With the partition function

$$Z[C] = \sum_{V} exp(-E[C, V]/T), \tag{2.11}$$

the ground energy is can be derived as fowlling

$$E_{GS} = -T\overline{lnZ[C]} + O(T^2) \tag{2.12}$$

$E_{GS}$ equals to zero in the sat phase and is strictly positive in the unsat phase. This is confirmed by analytical calculations.

## 2.3 Replica methods

It is very difficult to carry out such an average over the logarithm of partition function. Assuming that the system is symmetry versus time, averaging over the logarithm can be circumvented by computing the n$th$ moment of Z for integer-valued n and perform an analytic continuation to real n to exploit the identity $\overline{Z[C]^n} = 1 + n\overline{logZ[C]} + O(n^2)$. Using the replica method (G.Porrisi, 1987) [41], the n$th$ moment of Z can be obtained by summing up over the truth assignments $V$ and averaging over distributions of fixed number of clauses.

$$\overline{Z[C]^n} = \sum_{V^1,V^2,\cdots,V^n} \overline{exp(-\sum_{a=1}^{n} E[C,V^a])} \tag{2.13}$$

Since the clauses are arbitrarily selected, the energy in the formula can be represented by M times the average energy in one clause. Notice that each individual term of (2.13) factorizes over different clauses due to random probability distribution.

$$z[V^a] = \overline{exp(-\frac{1}{T}\sum_{a=1}^{n} E[C,V^a])} = \overline{(exp[-\frac{1}{T}\sum_{a=1}^{n}\delta(\sum_{a=1}^{N} C_i V_i^a + K)])}^M. \tag{2.14}$$

Based on the fact that SAT instances are arbitrarily generated, the above equation can be transformed further. The energy and entropy of ground state can be derived from complicated replica-symmetric saddle-point equations.

The effective field $h$ is defined as the average magnetization of those frozen spins when $T \to 0$. $H(m)$ is the distribution of $h$ over the interval $m \in [-1, 1]$. In the absence of clauses, $H(m) = \delta(m)$. Oppositively, the average magnetizations of over-contrained variables are mostly concentrated around 1 and -1. $H(m)$ can be drived from (2.14) and contributes to the study of SAT-UNSAT transition.

## 2.4 Phase transitions

The leap of the ground state energy from 0 to a strictly positive value is showed through the analytic computation of the spin model. Besides this crossover point, some other features of SAT problems can be derived from the Ising spin model introduced before. When there is no constraint, any assignment is a solution, and the entropy $S_{GS}(\alpha = 0) = ln2$. The Taylor expansion of $S_{GS}$ in the vicinity of $\alpha = 0$ is computed in Monasson's paper (1997) [46]. Results show that $S_{GS}(\alpha_{crossover} = 4.2) = 0.1$ for 3-SAT. Below this threshold, solutions are exponentially numerous, which is comfirmed by Y. Boufkhad's work (1999) [10].

Recent research (R. Monasson, 1999) [50] showes that Replica Symmetery theory breaks down at a density $\alpha_{RSB}$ that is below the crossover point. The solution sets of SAT problems are in completely different structures before and after this phase transition.

- Replica symmetry assumes that each pair of solutions has Hamming distance $dN$. All the solutions can be considered as gathered in a single cluster of diameter $dN$. There only exist one cluster of solutions characterized by a single

probability distribution of magnetizations. The $d$ in the Hamming distance is a decreasing function of densities, starting at $d(\alpha = 0) = 0.5$.

- At $\alpha_{RSB} \simeq 4.0$, the solutions set breaks into an exponential (in N) number of different clusters. The Hamming distance between the solutions belonging to different clusters keeps around $0.3N$, while the distance inside one cluster decreases quickly as density grows. This indicates the solutions inside a cluster move closer to each other. The sizes of clusters shrink as the density grows, while the clusters don't move closer to each other.

When $T \to 0$, some spins with the ground state energy are completely frozen at stable positions. These spins are in the backbone of a solution set. In other words, a backbone is composed of those bits at where all solutions share the same truth values. The results also show that the SAT-UNSAT transition is accompanied by a abrupt change of backbone components of 3-SAT problems. This is comfirmed by the data obtained from exhausitive enumaration of all the assignments of low order 3-SAT instances [51].

# Chapter 3
# SAT Solvers

As introduced before, there are many SAT solvers approaching SAT problems in many different ways. In this chapter we focus on those solvers based on search, both tree search and local search, for the complexities of solvers in this group are directly affected by the structure of the assignment landscape.

## 3.1 Tree search algorithms

The earliest complete algorithm for SAT problems is Davis-Putnam algorithm (M. Davis and H. Putnam1960) [18], which uses resolution to determine if a SAT instance is satisfiable.

Davis-Logemann-Loveland (DLL) procedure (1962) [40] is the most widely used refinement of the DP algorithm. It replaced the resolution rule in DP with the splitting rule. The formula F can also be viewed as a set of clauses. Arbitrarily selecting a variable $x$ in the clause set, $F \cup x \cup \neg x$ is satisfiable iff $F$ is satisfiable. Therefore, $F$ is satisfiable iff either $F \cup x$ or $F \cup \neg x$ is satisfiable. In this way, the clause set can be split into two parts that the satisfiability of each part can be checked respectively and recursively. In another word, DLL algorithm is based on tree search, which can be implemented by recursive calls of the function. After a variable is assigned a truth value, all satisfied clauses are removed from the clause set and all unsatisfied literals are removed from the remaining clauses. Tree search is

monotonic in the sense that constraints get tightened when going down the tree, and this is undone in reverse order when backtracking to a parent node. The algorithm returns true if there is no clause left and return false if an empty clause is found. An empty clause set shows that all clauses are removed because they are all satisfied by the partial assignment. An empty clause shows that all literals in this clause are unsatisfied, therefore this clause is unsatisfied. With this unsatisfied clause the formula cannot be satisfied by the partial assignment.

```
Algorithm 3.1 DLL(φ)
BEGIN
    If φ is trivially satisfiable (has no clause)return 1;
    If φ is trivially unsatisfiable (contains empty clauses), return 0;
    Choose one of φ's variables in p and its value c
    If DLL( φ[p → c] ) return 1;
    If DLL( φ[p → 1 − c] ) return 1;
    Return 0;
END
```

Unit clauses are those clauses having only one literal. The clause set can be satisfied only if the all the unit clauses in it are satisfied. When DLL algorithm splits on the unit clause, one of the branches which unsatisfies the unit clause will terminates immediately. Starting from the easiest part, the algorithm reduces the clause set $F$ with regard to all the unit clauses at first. The value of the variable $x$ in unit clause $U$ is set to the true if $U$ is a positive literal, and to false if $U$ is a negative literal. The reduction of a set of clauses $F$ with regard to the literal $U$ includes the

following two actions: remove all clauses having literal $U$ in $F$ and remove all literals $\neg U$ in any other clauses. Suppose variable $x$ is in $U$. Notice that all the satisfied clauses and the unsatisfied literals related to $x$ are removed in reduction. Expressions related to $x$ are eliminated from the formula as much as possible while keeping the satisfiability of the formula unchanged.

Besides a formula with no clauses, there are other kinds of easily satisfiable formulas. To introduce them, we firstly introduce a few definitions at first.

**Def2.1 Fact**: A clause containing only positive literals is called a fact.

**Def2.2 Constraint**: A clause containing only negative literals is called a constraint.

**Def2.3 Horn clause**: A clause containing at most one positive literal is called a Horn clause.

Notice that a formula is satisfiable as long as it is either fact free or constraint free. For instance, suppose that a formula doesn't have any fact. It is satisfied if all variables are assigned to false. In this case, all negative literals are evaluated true. In a non-trivial formula, there is at least one negative literal in each clause by the assumption and this negative literal makes the clause satisfied. The value of a formula is true when all of its clauses are satisfied. Given the fact that a fact free formula is satisfiable, our goal is to eliminate all facts from the original formula. In splitting it is critical to choose a good variable which will push the problem closer to the tractable case of Horn clauses. Splitting on a literal in facts will reduce the size of facts.

A model is defined as a set of variables that are assigned to true in a solution

of the formula $F$. A model is minimal if and only if none of its subsets is a model. All facts can be eliminated and all minimal models can be obtained through unit propagation and splitting.

```
Algorithm 3.2 DPLL(F,σ)
Input:
◇ A propositional CNF formula F;
◇ A partial model σ.
Output:
◇ All minimum models of F;
Function Unit-Propagation(F,σ)
    For each unit clause α in F
        If α is a positive literal a, σ' = σ ∪ a
            Remove all unit clauses α from F (unit subsumption)
        End if
        Remove all literals ¬α from any clauses (unit resolution)
    End for
End
BEGIN
    If there are any empty clauses in F, then return empty set;
    If there are no clauses left in F, then return {σ};
    /* Unit Propagation */
    (F', σ') ←Unit-Propagation(F,σ);
    If F' is fact free, then return σ';
    /* Splitting */
    a ← Select-Branching-Atom(F',σ) ;
    return DPLL(F' ∪ ¬a, σ' ∪ a = false)∪ DPLL(F' ∪ a, σ' ∪ a = true) ;
End
```

As introduced in V. Kumar, 1992, [37], backtracking can be optimized by lots of refinements, for example, backjumping, backmarking and arc consistency. In backtracking, the algorithm backs up one level in the searching tree and tries the other branch. Most of the time conflicts do not occur between the new current variable and its parent. The assignment of the current variable may conflict with the assignment in a node expanded much earlier in the search tree. For instance, assigning variable $x$ to the truth value $a$ makes the formula unsatisfiable. However, the conflict does not arise immediately but until reaching variable $y$, which is interpreted much later than $x$. Any partial interpretation rooted from the node which sets $x$ to $a$ cannot satisfy the formula. Saving some searching history, making out the exact place where conflicts happen and directly jumping back to the earliest assignment causing the conflict can avoid spending a lot of effort on searching those unsatisfiable subtrees. This is called backjumping [23].

Backmarking [48] is to avoid duplicate consistency checking in the searching. For example, if those assignments to some variables make the current assignment satisfiable or unsatisfiable have not been changed, the consistency of current assignment can be known directly without checking. These two refinements are based on the search history and require recording of some information during the searching procedure.

Arc consistency[29] is one of the forward checking methods. Forward checking methods check the consistency between the current variable and the remaining variables not yet instantiated. It rules out those values of the remaining variables which

conflict with the current assignment and guarantee that each assignment is consistent with past assignments. It saves a lot of effort on backtracking. Hybrids of these methods are used with DPLL algorithm.

## 3.2 Complexity of tree search

Tree search algorithms are complete. The worst case complexity is $O(2^{order})$. Many optimizations either trade off time complexity for space complexity, or vise verse. The exponential complexity cannot be lowered by any optimization. The computational effort is showed by Selman's experiments [52] by DPLL calls. Some very hard instances emerge near around the crossover point of satisfiability. Instances at lower densities or higher densities are easier to solve. Why are hard distances distributed around the crossover piont? Intuitively, for an under-constrained instance, solutions are dense and easy to find. To solve an over-constrained instance, all the branches of the search tree are cut short and the unsatisfiability could be decided earlier. The searching tree over the crossover point may have many long failure branches which introduce high complexity.

To demonstrate the above explanation, we implemented the basic DPLL algorithm (Algorithm 3.3). Without any refinement, the complexity will be higher than Selman's experiments, especially at high densities where the searching involves lots of backtracking. We record the depth of each leaf in the search tree, at the end of either a successful branch and a failure branch. Here we define the longest failure branch as the longest length of backtracking in the search procedure. At each splitting point, we

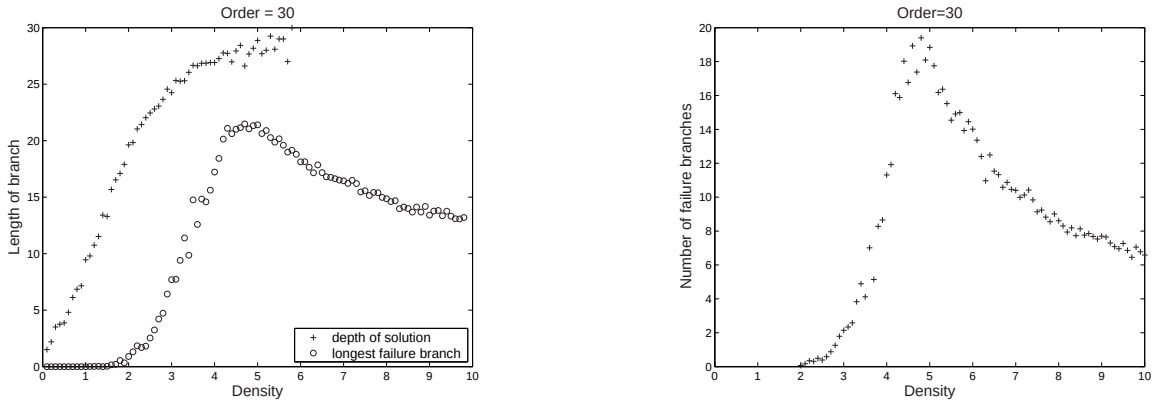
check the difference between its depth and the maximum depth visited before. The maximum difference obtained are taken as the length of the longest failure branch.

```
Algorithm 3.3 Length of longest failure branch(φ)
BEGIN
    While searching
    If reaching a unsuccessful leaf α
        If depth(α) > maxDepth
            maxDepth = depth(α);
    If reaching a splitting point β
        If maxDepth - depth(β) > maxDiff
            maxDiff = maxDepth - depth(β);
    End while;
    Return maxDiff;
END
```

The depth of a solution and the length of the longest failure branch are plot in the Figure 3.1. The number of failure branches is showed in Figure 3.2.

As the figure 3.1 shows, the depth of a solution is always the depth of the search tree in the satisfiable case. the depth of the search tree is increasing linearly below the density 2.0 and slows down after density 2.0. Above density 3.8, it needs to search almost full depth N to get a solution.

Below density 2.0, there is no failure branch. The search tree has a single branch to a solution. After the first few splits, unit propagation leads all the way down to a solution. The complexity of finding a solution is proportional to the depth of the searching tree, which is less than or equal to N.




**(a)** **(b)**

**Figure 3.1** (a) Length of successful path and failure branch in DPLL (b) Number of failure branches

Above density 2.0, the number of failure branches increases quickly. After reaching its peak around the crossover point, it drops again. Meanwhile, the length of longest failure branch increases linearly from 0 to about 2/3 of the order before reaching the crossover point, then drops down gradually after it.

Traversing many long failure branches easily raises the complexity. The high complexity over the crossover point can be viewed a combination effect of the increasing number of failure branches and their increasing lengths. What could be the reason causing these? That's the problem we are going to investigate further.

## 3.3 Local search algorithms

In contrast with tree search algorithms, local search methods can find a solution of a 3-SAT problem with larger number of variables, if there exists one. We call an assignment to all the variables in an CNF formula as a state in local search algorithm.

As mentioned in introduction, in local search methods the state space can be viewed as a landscape. The number of unsatisfied clauses is taken as the energy of a state, which evaluates how well the state fits the constraints. The search goal is a solution, which is a state with zero energy. Local search methods do not keep a global view of the state space. They only keep the information of the current state and check its neighbors. Then they move to one of their neighbors by flipping the truth value of one variable. Which state is chosen to be the next one depends on different algorithms. The generic local search algorithm takes a propositional CNF formula $F$ as input, and outputs a satisfying assignment of $F$ if it can find one.

```
Algorithm 3.4 Generic-Local-Search(F,σ)
BEGIN
    For i ← 1 to MAX − TRIES, do
        σ ← randomly generated truth assignment;
        For j ← 1 to MAX − FLIPS, do
            If σ satisfies F, return σ;
            a ← Pick-Variable-to-Flip(F,σ);
            Flip(F,σ,a);
        End for
    End for
END
```

One of the local search algorithms is called Random Walk because it always select an arbitrary neighbor to move to. It makes Brownian movement in the state space, and thus it is not a very efficient algorithm. GSAT [8] stands for greedy strategy SAT. It always selects the neighbor that yields the largest decrease of energy. GSAT takes

as input a propositional CNF formula $F$ and a truth assignment $\sigma$. GSAT algorithm is as follows

```
Algorithm 3.5 Pick-Var-GSAT(F,σ)
BEGIN
    For each variable x in F, do
        σ′ the truth assignment obtained by flipping x in σ;
        p1 the number of unsatisfied clauses in F given σ;
        p2 the number of unsatisfied clauses in F given σ;
        Δp(x) ← p2 − p1;
    End for
    Return minarg_x(Δp(x) : Δp(x) ≤ 0) if x exists
    Return null if such x does not exist;
End
```

GSAT easily got stuck at the a local minimum, when a state has no neighbor with lower energy. One way to escape from a local minima is called sidewalk. It will move to a neighbor with the same energy. Though it cannot make improvement in the local minima, at least it will not lose any fitness already gained. Experiments show that selecting a variable from those unsatisfied clauses will save much effort in searching.

Sidewalk cannot completely eliminate the effect of the local minima. To lead the search pointer out of local minima, random walk is introduced. A refined algorithm called WalkSAT [7, 9] takes some chances for random walk, other chances for greedy strategy. Simulated Annealing is another heuristic searching methods working quite well with SAT. It simulates the process of metal cooling down. The flexibility of metals decreases when temperature lowers down. The chance for random walk in Simulated

Annealing exponentially decreases when the temperature cools down. Keeping half chance of random walking, WalkSAT has higher capability to escape from local minima and has better performance in solving SAT problem. The following algorithm is WalkSAT which takes probability $p$ for greedy strategy.

```
Algorithm3.6 Pick-Var-WalkSAT(F,σ)
BEGIN
     C ← Randomly selected unsatisfied clause;
     For each variable x in C,
          compute Δp(x) as in Algorithm 2.5;
     With probability p,
          return minarg_x(Δp(x) : Δp(x) ≤ 0);
     With probability 1 − p,
          return a randomly chosen variable in C;
END
```

The goal of this function is to find the variable that can decrease the energy of the current state to the maximum extent after its value is flipped. Such a variable must be located in some unsatisfied clauses to make at least one unsatisfied clause satisfied. Selman's experiments show that always selecting an unsatisfied clause, even during the random walk, makes search faster [7]. Here "neighbor" is redefined as a state with distance one from the current state and at least make one of the unsatisfied clause satisfied. The refined algorithm only checks those variables showing up in unsatisfied clauses.

Even after introducing random walk, local minima could cost much time since the algorithm tends to search a small area back and forth. To circumvent behaviors like this, a tabu list is kept used to record all those variables being flipped within the past few steps. The new variable to flip should not be already on the tabu list. If all K variables in the selected unsatisfied clause are on the tabu list, we reselect another unsatisfied clause.

The latest algorithms used in WalkSAT are called NOVELTY and R_NOVELTY [16].

```
Algorithm3.7 Pick-Var-NOVELTY(F,σ)
BEGIN
    Randomly selected unsatisfied clause;
    For each variable x in C,
        compute Δp(x) as in Algorithm 2.5;
    Best ← minarg_{x∈Var(C)}(Δp(x) : Δp(x) ≤ 0);
    SecondBest ← minarg_{x∈(Var(C)−Bbest)}(Δp(x) : Δp(x) ≤ 0);
    If the Best ≠ the variable flipped most recently in the clause
        Return the best variable.
    Otherwise,
        With probability p,
            return Best;
        With probability 1 − p,
            return the SecondBest variable;
END
```

The algorithm NOVELTY make a compromise between the WalkSAT and Tabu algorithm. Maintaining a long tabu list costs time and space. Sometimes it slows

down the search process more than the "back and forth" walk. The algorithm NOVELTY can be viewed as arranging a tabu list of length one.

The algorithm R_NOVELTY takes into consideration how large the improvement is by flipping the variable. If it can make significant improvement, it is worth to step back to the most recently flipped variables. The assignment may be different from last time being flipped at the same variable.

Compared with GSAT, WalkSAT and search with a tabu list, NOVELTY and R_NOVELTY show the highest abilities in solving hard random 3-SAT instances. Call the random parameter $p$ as noise in the searching. Hoos' experiments shows that 0.6 and 0.7 is the most proper value for noise $p$ [27].

## 3.4 Complexity of local search

The total number of flips before a solution is found is taken as the complexity of the GSAT algorithm. We run GSAT from order 10 to order 100 with interval 5. For each order we run GSAT from density 1.0 to density 4.4 with interval 1. For each case we run 100 trials to get the average number of flips. With fixed density, the complexity grows up polynomially in the order. The higher the density, the steeper the climbing slope and the heavier weight of the square term.

At low densities, the complexity increases almost linearly over with order. The linear approximation still fits well at density 2.4. At density 3.6, it shows some curve. As the right figure above shows, the coefficient of square part increases dramatically as the density grows above 4.0. The polynomial approximation at density 4.4 shows

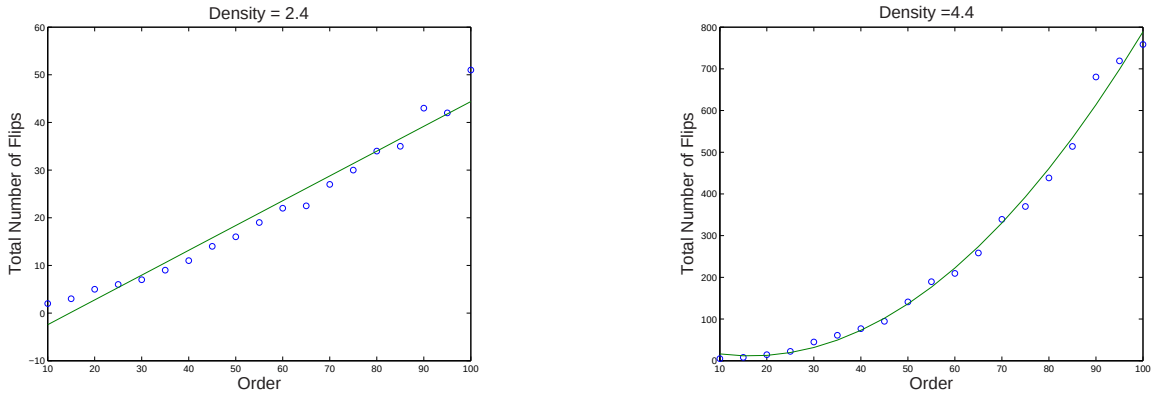



**Figure 3.2** Complexity of GSAT versus order

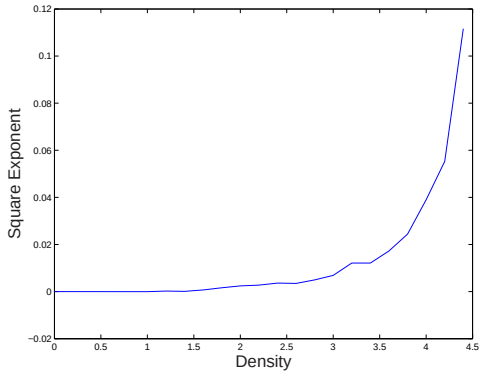



**Figure 3.3** The square coefficient of regression curve of GSAT complexity

a square curve.

The square coefficient of complexity curve goes up gradually above density 3.0, and then rise abruptly after the crossover point. What makes the increasing rate of complexity shift from linear to square before reaching the crossover point? We are going to explore the landscape of the searching space to find the answer.

Similar to the experiments for the GSAT algorithm, we run NOVELTY from order 10 to order 100 with interval 5. For each order we run from density 1.0 to density

4.4 with interval 0.1. For each case we run 1000 trials and get the average number of flips on those satisfied instances.

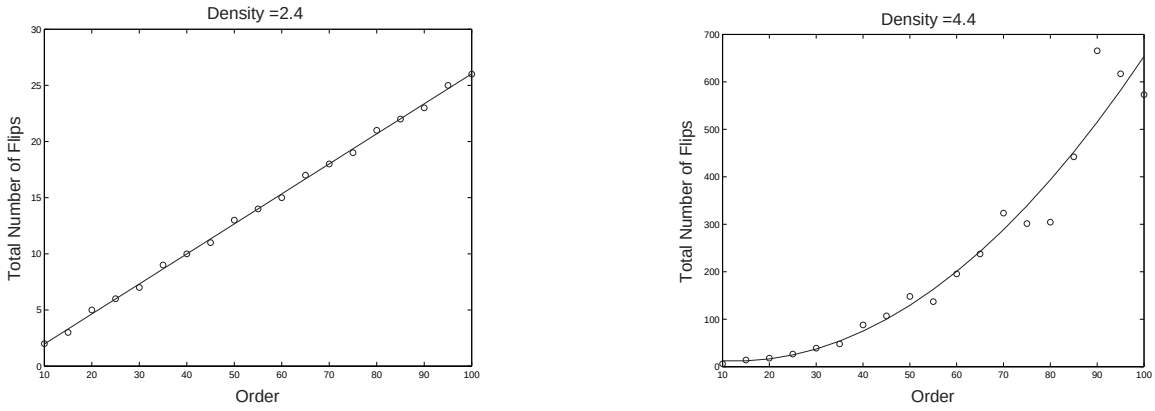



**Figure 3.4** Complexity of WalkSAT versus order at density 2.4

Similar as GSAT algorithm, at densities below 3.5, the complexity goes up linearly with the order. Above density 3.5, the complexity plots change from a line to a square curve before the density reaches 4.4. The square coefficient is derived from the

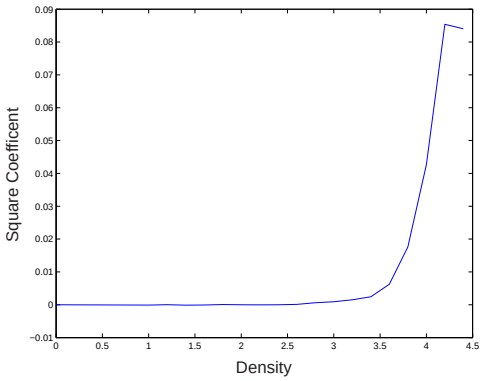



**Figure 3.5** The square coefficient of regression curve of GSAT complexity

complexity curve of WalkSAT algorithm. One difference is that the largest coefficient

of NOVELTY at the highest density below the crossover point is only 2/3 of that for GSAT. It means that NOVELTY has better performance on those hard SAT instances. Another difference is that the square coefficient of NOVELTY emerges at higher density compared to GSAT. Between density 3.0 and 3.6, the complexity plot of GSAT already shows noticeable square curve. However, the square coefficient is still very close to zero within these density ranges. NOVELTY and R_NOVELTY also works much better on these fairly hard SAT instances.

# Chapter 4
# Symbolic Methods

The naive way of describing the solution set of a CNF formula is enumerating all of the solutions, which takes exponentially growing space. The solution set of a formula having 20 variables will be difficult for a Sparc server with 2048M memory to handle. Representing a CNF formula and its solution set with Reduced Binary Decision Diagram (ROBDD) enables us to investigate the CNF formulas with twice of the order than what can be achieved by enumeration.

## 4.1 Binary Decision Diagram

A ROBDD is a compact decision tree which can represent large CNF formulas in feasible space. A Binary Decision Diagram (BDD) [3, 2, 11]is a rooted acyclic graph (DAG). To introduce ROBDD, define an operator "if-then-else" as

$$x \rightarrow y_0, y_1 = (x \wedge y_0) \vee (\neg x \wedge y_0) \tag{4.1}$$

Here $x$ is called a test expression. If $x$ is true, the value of $y_0$ is taken as the value of the expression, otherwise the value of $y_1$ is return. Some more complicated operators can be expressed by overlapping if-then-else operations. For example, $x \Leftrightarrow y$ is $x \rightarrow (y \rightarrow 1, 0), (y \rightarrow 0, 1)$. All boolean operators can be easily expressed by only using the if-then-else operator and constants 0 and 1.

**Def3.1 If-then-else Normal Form (INF)**: A Boolean expression built only with if-then-else operator and constants 0 ad 1. All of the test expressions in INF are

| expressions | INF |
|---|---|
| $\neg x$ | $x \rightarrow 0, 1$ |
| $x \vee y$ | $x \rightarrow 1, y$ |
| $x \wedge y$ | $x \rightarrow y, 0$ |
| $x \rightarrow y$ | $x \rightarrow y, 1$ |
| $x \oplus y$ | $x \Leftrightarrow y$ is $x \rightarrow (y \rightarrow 0, 1), (y \rightarrow 1, 0)$ |

single variables and not complex expressions. We can always make the test expression a single variable which doesn't occur in other part of the expression. Denote $t[a/x]$ as a Boolean expression obtained by replacing all the variables $x$ in expression t with truth value a. The Shannon expansion of $t$ with respect to x is

$$t = x \rightarrow t[1/x], t[0/x]. \tag{4.2}$$

A Shannon expression can help us to translate any expression into an INF. Given an expression t, it can be represented as $t = x \rightarrow t[1/x], t[0/x]$ by arbitrarily picking one of variables in it as a test expression. Recursively use Shannon expressions until there is no uninterpreted variable in $t$. The expression t is either constant 0 or constant 1 if it contains no variable.

By putting all variables in an arbitrary order, a formula can be translated to an INF. Any INF can be represented by a decision tree, also called a rooted acyclic graph (DAG). A decision tree has two different types of vertices, terminal and nonterminal, and the two terminals are one and zero. Each nonterminal vertex has two outgoing edges, a low edge and a high edge. In this thesis, low edges are drawn by dotted lines and high edges are drawn by solid lines. For each Shannon decomposition,

$f = (1 - x) \cdot f_{low}[0/x] + x \cdot f_{high}[1/x]$, $f_{low}$ is put at the low edge and $f_{high}$ is put at the high edge. Recursively we compose $f_{low}$ and $f_{high}$ until reaching the constants 0 and 1.

To build a BDD from a Boolean formula, build a decision tree from the INF at first. Then apply the following two rules as much as possible.

- Deletion rule: Remove those vertices whose two children are the same node.
- Merging rule: Two vertices having the equivalent sub-graphs share the same copy of the sub-graph.

For example, an formula $(x \vee y) \wedge z$ can be written as an INF

$$x \rightarrow (y \rightarrow (z \rightarrow 1, 0), (z \rightarrow 1, 0)), (y \rightarrow (z \rightarrow 1, 0), (z \rightarrow 0, 0)) \tag{4.3}$$

This if-then-else normal form can be represented as the following decision tree.

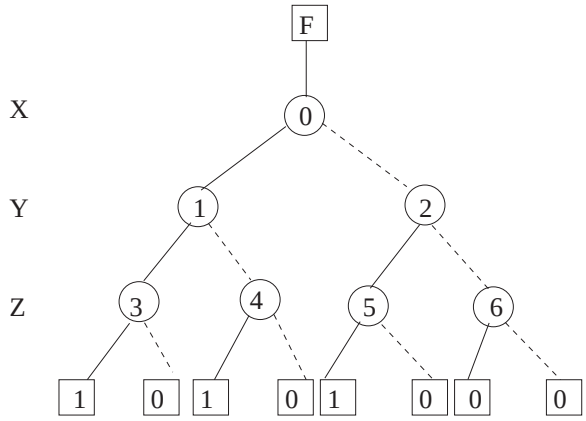



**Figure 4.1** A binary decision tree representing $(x \vee y) \wedge z$

Firstly, the merge rule can be applied to the terminal nodes 1 and 0. Nonterminal nodes 3, 4 and 5 are roots of identical subtrees, and thus they can be merged to one

node. The left figure below shows the graph after the merging phase. The deletion rule can be applied to node 1 and node 6. After deleting these two nodes, we get the BDD showed in the right figure below.

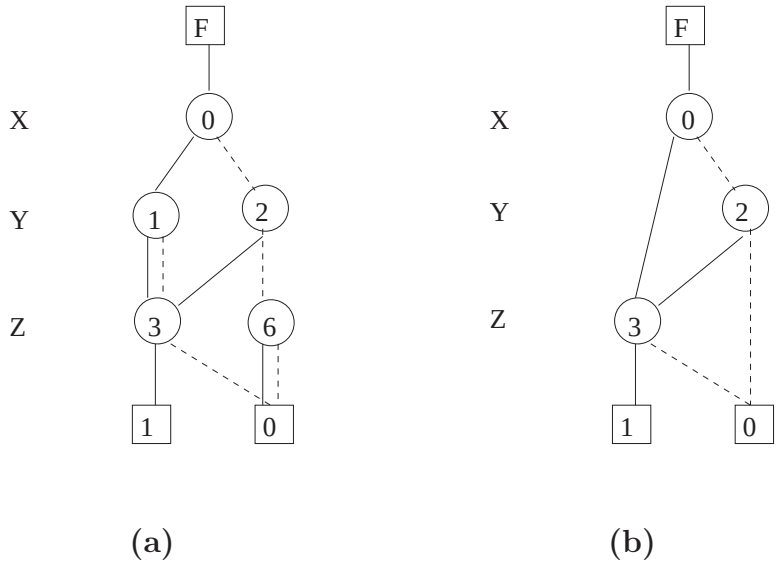



**Figure 4.2** (a) After applying merging rule (b) After applying deletion rule, BDD

After applying these rules, the graph is called a reduced ordered binary decision diagram (ROBDD). In this thesis, BDD stands for ROBDD. With different variable orders, the sizes of BDDs built from the same formula vary a lot. With a fixed variable order, the ROBDDs for the same formula are canonical. We manipulate BDDs with a package called CUDD developed by Somenzi [54, 55] from Colorado University. Operations like 'AND' and 'OR' between BDDs are implemented recursively applying this rule.

Disjunction is handled similarly. To build a BDD for a CNF using CUDD package,

instead of building a decision tree and then reducing it down, we piece together small BDDs representing literals to make a BDD representing a clause and then we piece together clauses to make a BDD representing a CNF formula.

After a BDD is built from a CNF formula, all assignments along the paths to the terminal 1 are solutions. Therefore, a BDD can represent the solution set of a CNF formula. With the highly compact size of BDDs, we can handle SAT instances in much higher order than using enumeration.

## 4.2 Image computation

Local search methods always move to one of the neighbors of the current state. Neighbors are defined as states with Hamming distance. An area in which all points are connected is defined as a cluster. Within one cluster, each pair of states are connected by a path. A method usually used in model checking, called image computation [33, 58], is used here to get clusters in the solution sets of SAT problems. Image computation starts from a single state, and expand its periphery for hamming distance 1 each time until reaching the fix point.

To find all clusters in a set of states $B$, we can start from a random state in that set. First, we put that random state in active set $A$. Each time we add to active set $A$ all its neighbors. Here we define a neighbor of active set $A$ as a state next to at least one state in $A$. By adding new neighbors, the set keeps growing until there is no neighbor of $A$ left outside. If the set does not change after trying to add its neighbors, it has reached a fixed point. All states connected to the original random state have

been collect in set $A$. They construct a cluster. Next we remove this cluster from set $B$ and continue to figure out other clusters left in it until all of them are found.

```
Algorithm4.1 Clusters(B)
/* Find a cluster in set B*/ BEGIN
    While B is not empty
        A' ← ,
        A ← randomstateinB,
        Do
            A' ← A;
            A ← A ∨ Neighbors(A);
        Until A = A';
        B ← B − A;
END
```

In SAT problems, an assignment to variables is viewed as a state. we can find all clusters in a solution set with image computation. Representing a set of solutions with a BDD, the above algorithm can be implemented in the following way. Since BDDs can be directly derived from from Boolean formulas, image computation can be carried out with this algorithm with CUDD package.

```
Algorithm4.2 Clusters(BDD_CNF)
/* Find all clusters in solution set of CNF*/ BEGIN
    BDD_activeSol ← BDD_CNF;
    While(BDD_activeSol ≠ BDD_zero)
        Build BDD_sol from a random solution in BDD_activeSol
        BDD_oneCluster ← Find_one_Cluster(BDD_sol, BDD_CNF);
        // push the information of this cluster into stack;
```

```
        Push(Properties(BDD_new ));
        //remove the cluster from solution set;
        BDD_activeSol ← BDD_activeSol ∧ ¬BDD_new;
    End while
END
```

Suppose the random solution selected is (0,0,1,01,1), construct the initial BDD from the formula $S_0(X) = \neg x_0 \wedge \neg x_1 \wedge x_2 \wedge \neg x_3 \wedge x_4 \wedge x_5$. The S(X) represents the active set, which has only one solution in the beginning. The transition with hamming distance 1 is showed as the following equation. It is to find all the neighbors of state $X$, denoted by $X'$. It can also be represented by a BDD.

$$T(X, X') = \bigvee_{1 \leq i \leq N} [(x_i \oplus x_i') \wedge ( \bigwedge_{1 \leq j \leq N, i \neq j} (x_j = x_j'))] \tag{4.4}$$

The following equation shows how to add those neighbors into the current set. $S_i(X)$ denotes for the current set. After finding all neighbors by making conjunction with the transition relation, we add this to $S_i(X)$ by disjunction and get the expanded set as $S_{i+1(X')}$.

$$S_{i+1}(X') = S_i(X) \vee \exists_X (S_i(X) \wedge T(X, X')) \tag{4.5}$$

To constraint the active set in the solution set, we conjoint it with the formula each time after expansion.

```
Algorithm4.3 Find_one_Cluster(BDD_seed,BDD_CNF)
/* Find a cluster in solution set of CNF that contains a "seed"*/ BEGIN
    BDD_new ← BDD_seed;
```

```
    While(BDD_OldSol ≠ BDD_NewSol),
        BDD_old ← BDD_new;
        BDD_new(X′) ← BDD_old(X′) ∨ BDD_old(X) ∧ BDD_T(X, X′)
        BDD_new ← BDD_new ∧ BDD_CNF;
    End while;
    Return BDD_new;
END
```

However, because the transition BDD has 2N variables, building and operating it may introduce high complexity. Meanwhile, getting a neighbor by flipping one bit in the BDD representing the current state is quite straightforward. It is not difficult to implement by manipulating BDDs directly. Given the current BDD, we switch the low edge and the high edge representing the same variable. This yields neighbors which only differs all such edges in that variable. Disjoining all of its neighbors, we can get the BDD expand the periphery by Hamming distance 1. The algorithm used to get the clusters in an assignment set is as following.

```
Algorithm4.4 Find_one_Cluster(BDD_seed, BDD_CNF)
/* Find a cluster in solution set of CNF that contains a "seed"*/ BEGIN
    BDD_new ← BDD_seed;
    While(BDD_OldSol ≠ BDD_NewSol),
        BDD_old ← BDD_new;
        For i = 1 to N
            BDD_new ← BDD_old ∨ Neighbor(BDD_old, i);
        End for;
        BDD_new ← BDD_new ∧ BDD_CNF;
    End while;
```

```
    Return BDD_new;
END
```

## 4.3 ADD and landscape

The clustering of the solution set gives us an insight into the rugged landscape of the assignment space. In addition to the solution set, which is the bottom of the landscape, we are interested in altitudes of non-solution assignments. Algebraic Decision Diagram (ADD) [49] is a tool to represent the whole landscape. An ADD can be seen as a BDD with not only 0-1 terminals. It can take any numeric value as a terminal and have other rules of BDDs. In ADDs, an assignment is mapped to a path, which leads to the terminal showing the energy of the state. The whole landscape can be described by an ADD in compact size.

Consider the example mentioned before, $(x \vee y) \wedge z$. Solutions like (1, 1, 1) have energy 0. Some assignments like (0, 0, 1) only satisfy one of the two clauses and they have energy 1. The assignment (0, 0, 0) has energy 2 because it doesn't satisfy any of two clauses.

Logic operations of ADDs work in the same way as BDDs. We can build ADDs from small pieces by adopting "$Apply(f, h, op)$" in the CUDD package. There is another function in CUDD package called "$Cudd\_addBddThreshold$", which proves to be very useful in our algorithm. It turns an ADD to a BDD by mapping all of its leaves greater than the threshold to the leaf 1, and all others to the leaf 0.

From the ADD representing a SAT instance, we can easily extract a BDD rep-

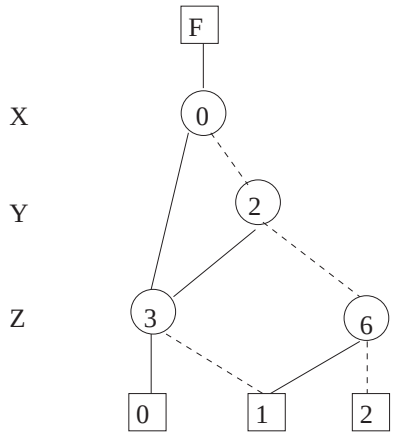



**Figure 4.3** An ADD representing $(x \vee y) \wedge z$

resenting the group of states with a specific energy. The number of states with a specific energy can be obtained by counting the number of "Minterm"s in the BDD. We are interested in the average energy and variation of them of one SAT instance. We are also interested in the ruggedness of the landscape since the more rugged the landscape, the higher complexity will be involved in searching.

## 4.4 Measure ruggedness of the landscape

Define the distance between two states as their hamming distance. The ruggedness of the landscape are usually measured by the landscape autocorrelation function

$$\rho(d) = 1 - \frac{\langle (E(s) - E(t))^2 \rangle_{d(s,t)=d}}{\langle (E(s) - E(t))^2 \rangle} \tag{4.6}$$

with $\langle (E(s) - E(t))^2 \rangle$ the average value of $(E(s) - E(t))^2$ over all pairs $(s,t)$, and $\langle (E(s) - E(t))^2 \rangle_{d(s,t)=d}$ the average value of $(E(s) - E(t))^2$ over all pairs $(s,t)$ with distance $d$ [4]. $\rho(d)$ shows the level of correlation between any two states with distance

$d$. $\rho(1)$ indicates the correlation between neighbors, which play important role in local search. A value close to 1 indicates that the energy of neighbors are very close. A value close to 0 indicates that the energy of adjacent states are almost independent. The more smooth the landscape, the more suitable for local search algorithms.

For an SAT instance with N variables, there are $2^{N-1}\dot{(}2^N - 1)$ pairs of states in total and $2^{N-1}\dot{(}N - 1)$ pairs of adjacent states.

$$\rho(1) = \frac{N}{2^N - 1} \frac{\sum((E(s) \dot{-} E(t))^2)}{\sum_{d(s,t)=1}((E(s) - E(t))^2)} \tag{4.7}$$

$$\rho(2) = \frac{N(N-1)}{2(2^N - 1)} \frac{\sum((E(s) \dot{-} E(t))^2)}{\sum_{d(s,t)=2}((E(s) - E(t))^2)} \tag{4.8}$$

are derived from (4.6).

As mentioned before, the number of assignments with each level of energy, defined as Num(energy), can be easily obtained from the ADD. The nominator is easily calculated from the following equation by sum up values over all pairs of different energy in the ADD.

$$\sum((E(s) - E(t))^2) = \sum_{(E_1,E_2)} Num(E_1)Num(E_2)(E_1 - E_2)^2 \tag{4.9}$$

As for the denominator of (4.6), it is a little bit difficult to identify all pairs of states with a specific distance in the space. We are going to figure out the denominator in a few steps. Let $X$ stand for an assignment, which is mapped to a path in an ADD. Function terminal($X$) gives us the numerical value of the terminal at the end of path $X$. First, we write a function which computes $\sum_X \text{terminal}(X)^2$, the sum of square

energy of all states.

```
Algorithm 4.5 Sum_Square(ADD_cur)
BEGIN
     If ADD_cur is a terminal,
         return 0;
     ADD_left_child = Left_Branch(ADD_cur);
     ADD_right_child = Right_Branch(ADD_cur);
     For each branch
          if ADD_left_child/right_child is a terminal
               return (value of the terminal)^2
     End for
     return 2^(index(ADD_left_child)−index(ADD_cur)−1) × Sum_Square(ADD_left_child)+
            2^(index(ADD_right_child)−index(ADD_cur)−1) × Sum_Square(ADD_right_child)
END
```

Similar as what we did with BDDs, we can switch all the 0-edges and the 1-edges of variable $X_i$ by a function called Switch($ADD_{CNF}$, $i$). In the new ADD return by the function, any assignment $X'$, which only differs from assignment $X$ in the variable $X_i$, leads to the same terminal as $X$ does in the old ADD.

$$\text{terminal}(X') = \text{terminal}(X), X'_i = \neg X_i, X'_j = X_j \text{ when } j \neq i$$

.

If we subtract the new ADD with the old ADD, the result ADD has the energy difference between $X$ and $X'$, which are neighbors. Notice that the difference between

each pair of neighbors appears twice here. Summing up the square of terminals led by all assignments in the result ADD and dividing by 2, we get the nominator of (4.7).

$$\sum_{d(s,t)=2}((E(s)-E(t))^2) = \sum_{i=1}^{N} \text{Sum\_Square}(\text{Switch}(ADD_{CNF}, i) - ADD_{CNF})/2 \quad (4.10)$$

This equation can be computed by the algorithm

```
Algorithm 4.6 Sum_NeighborsA(ADD_CNF)
BEGIN
    sum = 0;
    For i=1 to N
        ADD_neighbor_i = Switch(ADD_CNF, i)
        sum+ = 2^(index(ADD_neighbor_i)−index(ADD_cur)) ×
            Sum_Square(ADD_neighbor_i − ADD_CNF)/2;
    End for
    return sum
END
```

The square sum between neighbors can also be computed by recursion. Consider a subtree of the ADD with index $i$, it represents a set of partial solutions, which only interpret those variables with index no lower than i. Subtracting two children of it, we get difference between neighbors of those partial solutions only differing in variable $X_i$. By recursively calling this function on both its left child and right child, we can get the square sum between neighbors differing in variables with index higher than $i$. The function returns the square sum of difference of between neighbors of partial solutions. When the value is passed upwards, the partial assignments are extended to a full assignments. And the energy difference between neighbors differing in variable $X_i$ is count and only count once here.

Algorithm 4.7 Sum_NeighborsB($ADD_{cur}$)

BEGIN

If $ADD_{cur}$ is a terminal,

return 0;

$ADD_{leftc\_child}$ = Left_Branch($ADD_{cur}$);

$ADD_{right\_child}$ = Right_Branch($ADD_{cur}$);

$new\_diff$ = Sum_Square$(ADD_{left_child} - ADD_{right_child})^2$;

return $2^{index(ADD_{diff})-index(ADD_{cur})-1}\times$ $new\_diff$ +

$2^{index(ADD_{left\_child})-index(ADD_{cur})-1}\times$ Sum_Neighbors($ADD_{left\_child}$)+

$2^{index(ADD_{left\_child})-index(ADD_{cur})-1}\times$ Sum_Neighbors($ADD_{right\_child}$);

END

The square sum of all pairs with distance 2 can also be calculated in the similar way. In a subtree with index $i$, the partial assignments differing in variable $i$ should be different in another bit with index higher than $i$ to make distance 2. For an partial assignment to its left child $X = (X_{i+1}, ..., X_N)$, we are going to find its neighbor $X' = (X'_{i+1}, ..., X'_N)$ in the right child , which is neighbor of $X$. Plus on the difference in variable $i$, $(0, X_{i+1}, ..., X_N)$ and $(1, X'_{i+1}, ..., X'_N)$ has two different bits. $X'$ can be obtained by switching branches of the subtree. Subtracting left child with Switch(right child), we get the result ADD showing energy difference between pairs of partial assignments differing in two variables. Similar as the algorithm computing $\rho(1)$, the difference can be obtained by recursively calling this function on its children.

Algorithm 4.8 Sum_Dist2($ADD_{CNF}$)

BEGIN

If $ADD_{cur}$ is a terminal,

```
        return 0;
    ADD_left_child = Left_Branch(ADD_CNF);
    ADD_right_child = Right_Branch(ADD_CNF);
    new_diff = Σ_{i=1}^{N} Sum_Square(Switch(ADD_left_child, i) − ADD_right_child)/2
    return 2^{index(ADD_diff)−index(ADD_cur)−1}× new_diff +
                2^{index(ADD_left_child)−index(ADD_cur)−1}× Sum_Dist2(ADD_left_child)+
                2^{index(ADD_left_child)−index(ADD_cur)−1}× Sum_Dist2(ADD_right_child);
END
```

The complexity of algorithms calculating the denominators of $\rho(1)$ and $\rho(2)$ are exponential to N because they recursively traverse an ADD. However, experimental results on instances with low orders shed some light on ruggedness of the assignment landscape. The results will be presented in chapter 5.

## 4.5 Count non-solution basins

A local minimum is a state without a neighbor with lower energy. Local search methods easily get stuck at a local minimum. After introducing sidewalk, it can move to a neighbor with the same energy. Given plenty of time, it will move downwards unless it gets stuck in a basin. A basin is a connected area at the same altitude such that no outgoing path leads downwards. Once the GSAT search enters a basin, it is not able to come out of it even with sidewalk. All clusters in solution set are basins with energy zero. Intuitively and confirmed by our experiments, there exist some other basins with energies higher than zero.

The assignments in each basin are all local minima. To find a basin, we start from

a local minimum. First we search for a local minima by examining all neighbors of each state, and then we get a slice of the landscape with the same energy as the local minimum. Next we find by image computation the cluster in the slice which contains this local minimum. To check if there is any outgoing path that leads downwards, we expand the cluster for one more transition. If no state in the big cluster after expanding has lower energy than the original local minimum, we say that this cluster is a basin. After searching from all the local minima, none of the basins will be left out.

Algorithm 4.9 Basin_original($ADD_{CNF}$)
BEGIN
    For each assignment $\alpha$
        check all its neighbors.
        /* If none of them has lower energy, $\alpha$ is a local minimum. */
        If $\alpha$ is a local minimum
            $BDD_{Energy_\alpha} \leftarrow$ All assignments having same energy as $\alpha$
            $BDD_{Cluster} \leftarrow$Find_one_Cluster($BDD_\alpha$ , $BDD_{Energy_\alpha}$ );
            $BDD_{Neighbors}(X') \leftarrow BDD_{Cluster} \wedge T(X, X')$;
            $ADD_{Neighbors}(X) = ((ADD)BDD_{Neighbours}(X) \times ADD_{CNF}(X))$;
            If Min_Leave($ADD_{Neighbors}(X)$) $\geq$ Energy($\alpha$)
                return true;
            else return false;
        End if
    End for
END

Directly converting a BDD to an ADD is simply changing all logic values 0/1

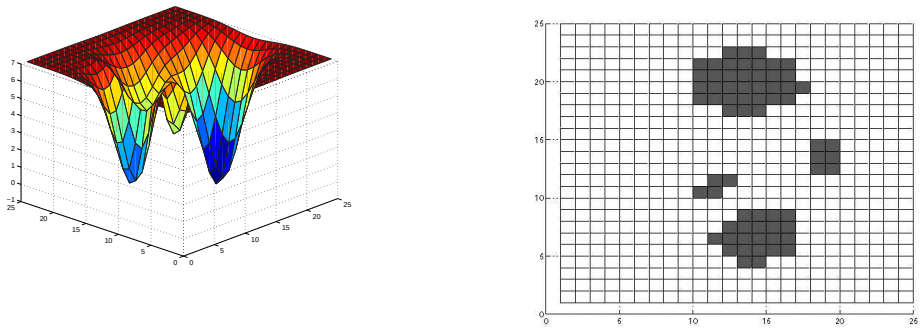
in the BDD to numerical values 0/1 in ADD. And the multiplication of ADDs can be viewed as taking the product of the leaves from the corresponding paths in both BDDs as the new value of the leave. Then we check the minimum values of the leaves in the ADDs representing neighbors. Because all nodes in the cluster have the same energy, a path leading downwards means that one of their neighbors has lower energy. By checking if a cluster has any neighbor with lower energy than what it has, we can tell it is a basin or not.

Given the order 20, the experiments showed that all the basins are close to the bottom of the landscape (Fig5.9). Based on these observations, we can find basins can by an alternative method which can reach higher order.

```
Algorithm4.10 Basin(ADD_CNF)
BEGIN
    For i = 0 to Upbound (Chosen energy level)
        BDD_ActiveSet ← States | Energy(States) ≤ i;
        Clusters ← Clusters(BDD_ActiveSet);
        For each cluster
            If Min_energy(ADD_Cluster)=i
                Count this cluster a basin;
            End if
        End for
    End for
END
```

The method is illustrated by the figures 4.4. Imagine a landscape with a few basins with low altitude. We remove all states with energy higher than a specific

value, for instance, 4. Only those states with energy no higher than energy 4 are left on the landscape. Those basins with their bottoms no higher than energy 4 are still showed in the remaining landscape. If every path connecting two basins involves a state with energy higher than energy 4, these two basins are disconnected in the remaining landscape. Therefore, by projecting the remaining landscape to a plane and then applying image computation to the plane, we can get clusters corresponding to those disconnected basins with their bottoms lower or equal than energy 4.



**(a)** **(b)**

**Figure 4.4** (a) Landscape (b) Clusters corresponding to basins below some latitude

Repeating this process with different energies, we can sperate different basins. All basins will be showed on the projected planes. Some basins appear in more than one projection planes. A basin appear in the projected plane since it is isolated from

others, until its bottom is higher than the threshold and removed from the landscape. How to count them at most once? We count a basin only when the threshold reaches the bottom of the basin. When all those states with energy higher than its bottom are removed, the states in the cluster on the projected plane have the same energy, the energy of its bottom. By checking if the states in a cluster have energy lower than the current threshold, we can tell if it is the bottom of a basin.

We call those basins with energy larger than 0 as non-solution basins. The distance between a state in non-solution basins and its nearest solution is very interesting. Usually we start from one state, put it into a set, expand the set with distance 1 each step as we did in image computation until the lowest energy of the set reaches 0. the lowest energy of the set reaches 0 if and only if the set contains a solution. Then the times of loop before reaching such a solution is the shortest distance to the nearest solution.

```
Algorithm 4.11 Distance_to_the_nearest_solution(State)
BEGIN
    Set ← State;
    distance ← 0;
    Do
        Set← Set ∪ Neighbor(Set);
        distance ← distance + 1;
    until Lowest_energy(Set)=0;
    return distance;
END.
```

## 4.6 Backbone and Distance

Suppose that there is a set of states constrained by formula $F$. The backbone [31, 56] of the states is those variables that are "frozen" to the same value in all states. A long backbone means that the set of states is centralized, and a short one means that the set of states is sparse.

One approach to get the backbone of the solution set of $F$ is to enumerate all states in the set and compare each bit among them. This approach yields high space complexity because all states in the set need to be stored before the comparison. An alternative way is to get backbone directly from the formula $F$. If a variable is in the backbone, the variable is assigned to the same truth value in all states, say $a$. If we constraint the variable to $\neg a$, $(\neg a) \wedge F$ cannot be satisfied any more. Meanwhile, since $F$ is satisfiable, the only fact that could make $(\neg a) \wedge F$ unsatisfiable is that $x = a$ is in the backbone of $F$'s solution set. The following algorithm can produce the backbone of a set of states.

Algorithm 4.12 Backbone($BDD_{Set}$)
BEGIN
    Backbone $\leftarrow \{\}$;
    For i = 1 to order
        If $v_i \wedge BDD_{Set} = BDD_{zero}$
        *** $v_i$ is frozen to 0 in this set ***
            push $v_i = 0$ into backbone;
        End if
        If $\neg v_i \wedge BDD_{Set} = BDD_{zero}$

```
        *** ¬v_i is frozen to 1 in this set ***
            push v_i = 1 into backbone;
        End if
    End for
END
```

# Chapter 5
# Experimental Result and Analysis

To investigate the cause of high complexity around the crossover point of 3-SAT problems, many efforts have been spent on solution sets. Early research on solution sets mostly focus on the number of solutions. Recently the structure of a solution set also raises large interest of people. The spin model, one of the statistical mechanics methods, models a SAT problem very well. It takes an assignments as a state and the number of unsatisfied clauses as the energy of the state. Some conclusions drawn from the analytic computations of the spin model give better insight into the structure of the solution set. Based on the model, the solution set breaks into clusters below the crossover point [47]. In addition, the backbone of the solution set jumps from 0.6 to a value above 1 near the crossover point.

This chapter consists of two main sections. The first section is about our experiments on the structure of the solution set. One of the experiments shows the increase of the backbone of the solution set. Another experiment confirms the clustering behavior of solution sets. In the second section, we expand the field of vision from the bottom of the space, the solution set, to the whole assignment space. First, we describe the general landscape of assignment space. Afterwards, we introduce some experiments indicating that some non-solution basins emerge at the densities below the crossover point and contribute to the high complexity in those solvers based on

searching. To investigate the complexity of local search methods further, we also plot the number of loose local minima.

## 5.1 Structure of the solution set

The number of solutions decrease exponentially as the density grows. As showed in Figure 5.1, the number of solutions decreases exponentially with the density. The SAT instances with low densities have far more solutions than instances with high densities. As the solution set decreases in size, its structure also changes with densities.

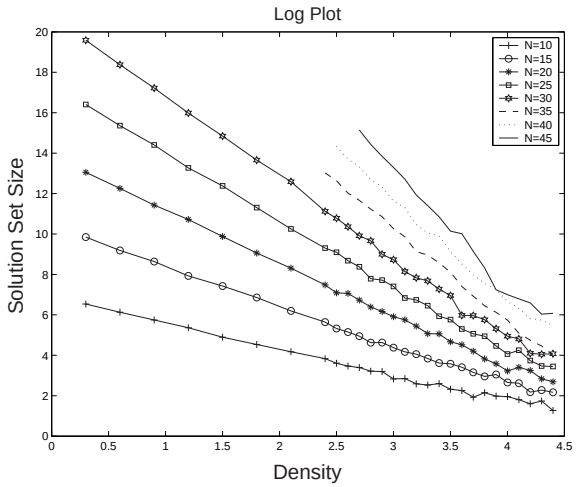



**Figure 5.1** Log-plot of the size of a solution set

### 5.1.1 Backbone of the solution set

Recall that the backbone of a solution set is the set of common bits shared by all solutions. The solutions in a solution set with a long backbone are similar to each other.

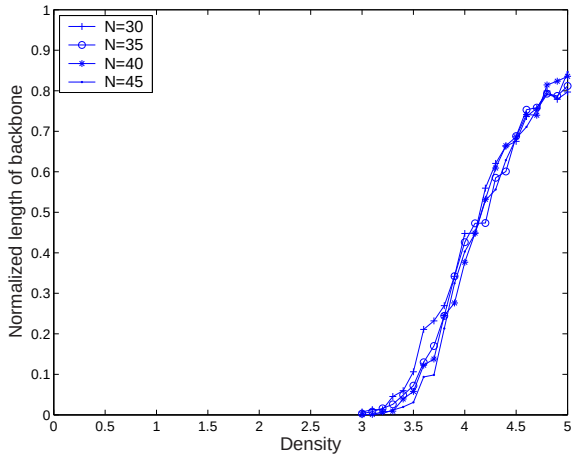



**Figure 5.2** Normalized backbone of a solution set

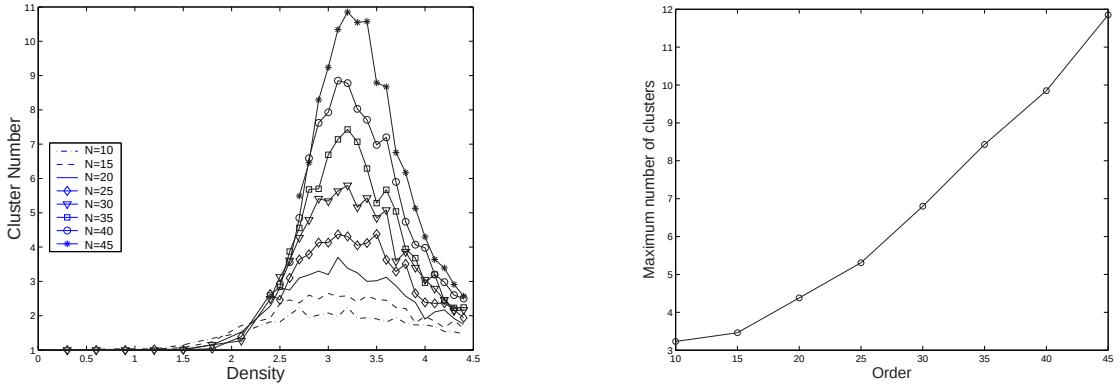
We conduct experiments with algorithm 4.9 on order 30, 35, 40 and 45, densities between 3.0 and 5.0 with interval 0.1. For each pair of order and density, we run 100 instances of random 3-SAT problems. In the absence of clauses, all assignments are solutions. There is no sharing of any common value in any single bit for all of them. The backbone of a solution set in this case is zero. It remains zero until the density grows up to 3.0. An abrupt raise of the length of a backbone happens between density 3.5 and 5.0. The length of a backbone increases from 0 to $80\% \times order$ in these density range. The four curves obtained from different orders almost completely overlap. We can predict that the length of backbone of the solution set of a large order instance will show the same trend when the density grows. Our curves are in the same shape as in Monasson's paper [51]. The only difference is that our backbone grows close to the full length of an assignment. The backbone in Monasson's paper only has a length less than $0.6 \times order$ before the solution set dies out. Our experiments show

that when density is right below the crossover point, the size of solution set could be very small. Consider the backbone with a length 80% $\times$ *order*, these solutions are very close to each other. In addition, they form one single or two small clusters instead of scattering around the space.

### 5.1.2 Clustering behavior of a solution set

A SAT instance and its solution set can both be represented by the same BDD. We run image computation (Algorithm4.2) on the solution set represented by the BDD to get all of the clusters in it. The experiments were running on orders from 10 to 45 with an interval of 5. For each order we ran experiments at different densities. Within the lower density, 0.0, 0.5, 1.0 and 1.5 were checked. Between density 2.0 and 4.4, densities were checked with an interval of 0.1. For each pair of order and density, 500 random 3-SAT problem instances were run. The sizes of all the clusters in each solution set were enumerated.

As predicted by the spin model, the solution set breaks into clusters below the crossover point. We analyzed the average number of clusters in a solution set as a function of the density for fixed order instances. The curves of the eight different orders coming from the experiments demonstrate the clustering behavior of the solution set. At density lower than 2.0, there is only one cluster in each solution set, which means that all of the solutions are connected. The solution set is broken into clusters as the density grows above 2.0. The number of clusters increases abruptly until it reaches its maximum value around density 3.2 when the order is below 45.



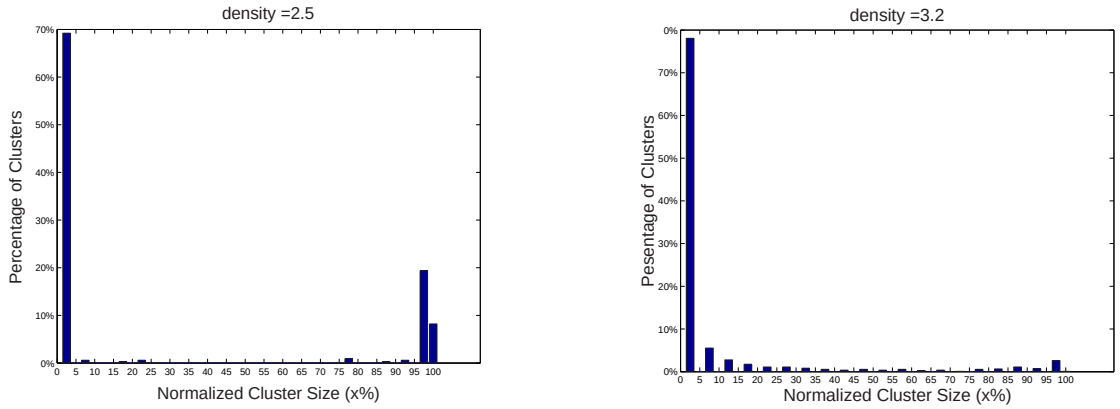

(a) (b)

**Figure 5.3** (a)The number of clusters in solution set, (b)The maximum number of clusters versus order

Above this density, the number of clusters in one solution set declines as the solution set shrinks. There are only one or two clusters in the solution set right before it vanishes. The maximum number of clusters is plot versus the order. The curve is more closer to a line than a exponential curve. That's different from the prediction made by Monasson's paper that the cluster breaks into exponential number of clusters.

The histograms of cluster size with density from 1.7 to 4.8 are presented in Appendix A.1. Here we show some samples. We normalized cluster size with the size of the solution set the cluster comes from, and then classify those clusters with their normalized sizes. Figures are based on our experiments with 500 SAT instances on order 40 with each density.

The solution set does not break evenly into several clusters with similar size. After the solution set breaks down, there still exists a big cluster that contains almost all

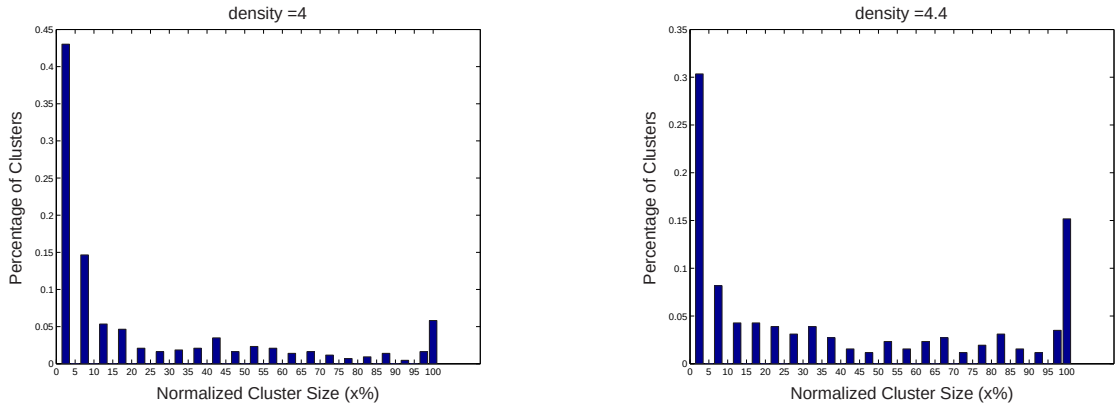



(a) (b)

**Figure 5.4** Probability distribution of clusters with different size (a)density=2.5 (b)density=3.6

of solutions. The other clusters are very small compared to the maximum cluster. Despite the large number of small clusters, they only hold a small percentage of the solutions.

Observed from Figures 5.4 and 5.5, there are extremely small number of medium-sized clusters in the solution set. At density 2.5, a portion of solution sets breaks into clusters and the others still keep whole. Above 95% of solutions are still connected to each other and stay in a big cluster. About 2 or 3 small clusters emerge. Each contains less than 5% solutions. In the figure of the simulations with density 3.6, both the one-cluster solution set and the main cluster in the multi-cluster solution set almost disappear. Most of the clusters are small clusters having less than 5% of the solutions. The solution set is the sparsest at this density. When density is above 4.0, the one-cluster solution sets reappear again and increase when the density keeps




(a) (b)

**Figure 5.5** Probability distribution of clusters with different size (a)density=4.0 (b)density=4.4

growing. The number of small clusters containing less than 5% solutions decreases a lot and the he number of clusters with medium size between 5% and 95% increases.

We remove the maximum cluster in a solution set, normalize the size of rest clusters over the size of the solution set, and plot them as Figure 5.6. With order 45, the average size of a non-maximum cluster is about 5% of the solution set. The average size of non-maximum clusters decreases with the order. We can predict that when $N \to \infty$, the size of a non-maximum cluster could be very small compared to size of the solution set.

The experiments in this part give us the clustering structure of the solution sets. They confirm the result from analytical computations of the spin model. However, clustering behavior emerges at much lower density than density 3.94 as in Monasson's paper [47]. That paper didn't indicate that the number of clusters in solution set will

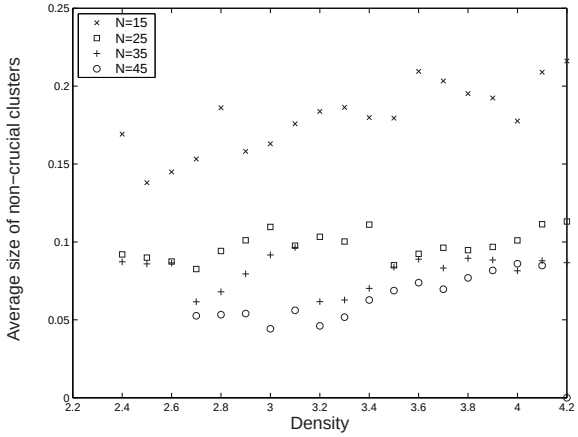



**Figure 5.6** The average normalized size of a non-maximum cluster

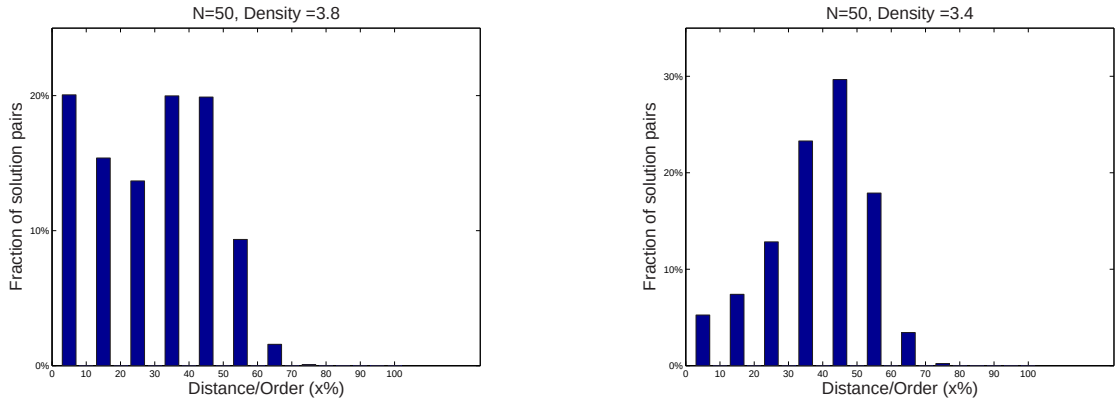
decrease at high densities either. The Spin model assumes that each particle is in a position with some probability. It allows some error range in its result.

### 5.1.3 Distance between solutions

Our definition of clusters are slightly different from the definition in Monasson's paper. We define a cluster as a set of connected points. Clustering behavior in Monasson's paper comes from the analytic computation on distances between solutions. Below RSB, each pair of solutions has distance $\frac{1}{2}order$ with the same probability. After RSB at density 3.94, distances between solutions concentrate at two values. One value shows the distance between solutions in different clusters. The other shows the distance inside one cluster. The difference between these two values increase with the density when the inter-cluster distance keeps stable and the intra-cluster distance decreases quickly.

We conduct experiments with order 50 and different densities. With each pair of

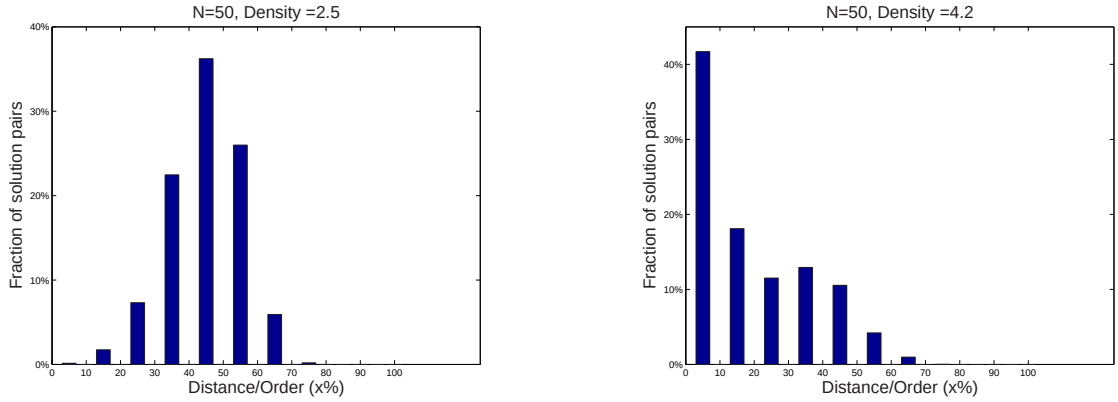
parameters we randomly generated 100 formulas in CNF. For each instance of SAT problems, we randomly selected 100 pairs of solutions and computed the distance between each pair of solutions. Then we normalize the distance with the order of the formula, and plot the histogram of distances. All histograms are presented in the appendix A.2.




**(a)** **(b)**

**Figure 5.7** Histogram of distances between solutions (a)density=2.5 (b)density=3.4

Our experiments confirms the result of Monasson's paper. At low densities like density 1.0, distances are concentrated at $\frac{1}{2}order$. As density grows, the average distance gradually decreases. The distribution of distances slowly move to small values while the primary fraction of distances remains near $\frac{1}{2}order$. This distribution of distances keeps until density 3.6. The plot of density 3.8 shows there are two focuses of the distance distribution. One is around $0.4 \times order$, which is close the unique focus at the low densities. The other is in $[0, 0.1 \times order]$, which emerges as




(a) (b)

**Figure 5.8** Histogram of distances between solutions (a)density=3.8 (b)density=4.2

the distances inside a cluster.

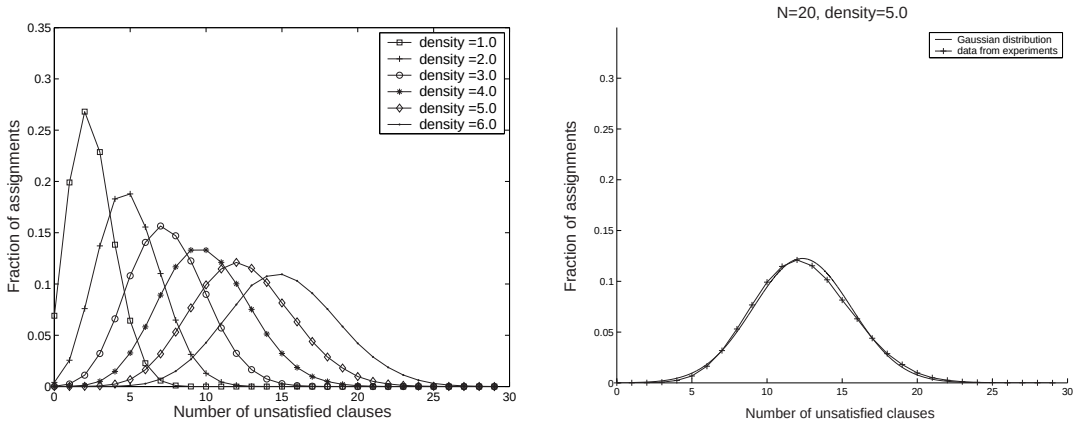
Why does cluster phenomenon observed in this way seems happen at the different density as in section 5.1.2? It is due to the fact that the maximum cluster in the solution set contains more than 95% solutions. Between density 3.0 and 3.7, the size of the maximum cluster still exceeds in size. The distance inside this maximum clusters are close to the average distance inside a solution set. The probability of selecting both solutions inside the maximum cluster is much larger than others. The intra-cluster distance decrease along with the size of maximum cluster. The focus on the small distances becomes noticeable when the maximum cluster no longer dominates the solution set.

## 5.2 The Assignments Landscape

A state is an assignment to all variables in a SAT instance. If we take the number of unsatisfied clauses as the energy of state and represent their energy as altitude, the states constitute a landscape. Before reaching the solution set, local search methods search in non-solution space. The structure of non-solution space is also very interesting.

### 5.2.1 General description

With an ADD representing a 3-CNF, we can easily find out the number of assignments with each level of energy. We average it over 100 instances with each group of parameters and plot distribution of assignments versus their energy. It follows Gaussian distribution perfectly at density 5.0 as showed in Figure 5.6(b). At densities below the crossover point, it only fits part of the Gaussian distribution on the right of the zero. As the density grows, the center of the Gaussian moves to the right gradually. The gaussian distribution at density 3.0 shows a tail on the side close to zero. Above the crossover point, all states moved strictly above the energy 0. That make the Gaussian distribution complete and make the instances unsatisfiable. As long as the mean value of the Gaussian distribution increases linearly with the order, the variance of it decreases with the order. Combining these two facts, the tail of it becomes more and more threadlike. Searching through this critical narrow path requires a significant effort because of the comparably sparse distribution of assignments.



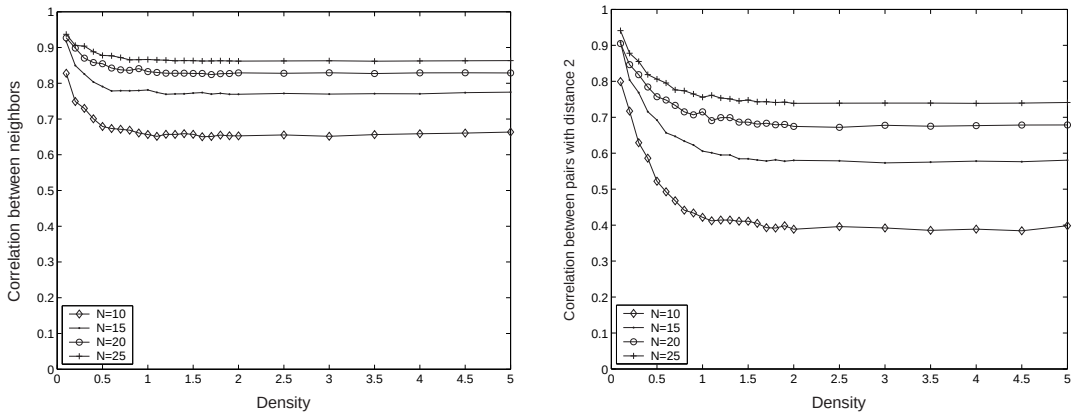

**(a)** **(b)**

**Figure 5.9** (a) The probability distribution of states versus energy;(b)Fits the curve with a Gaussian distribution.

In the absence of clauses, all states are solutions and the space is a plane with energy 0. As the density grows, the main body of states move to higher and higher altitude. The random starting state of a local search is located in the main body of the states. Generally, tfcthe distance from the random starting state to a solution increases with the density. In addition, at high densities, the Gaussian distribution of assignments show a a threadlike tail on the side close to zero, which means the number of those states near the bottom of the landscape is very small. It will take more and more time to search all the way down to the bottom of the landscape, especially at high densities.

### 5.2.2 Ruggedness of landscape

The ruggedness of a landscape is measured in Chapter 4. We plot the correlation between neighbors and the correlation between pairs with distance 2 versus densities. The correlation decreases quickly at low density and stays stable beyond density 2.0. The higher the order, the smooth the landscape. As the rapid increase of ruggedness versus density and the shift of complexity happen at the different densities, they are related with low probability.




**(a)** **(b)**

**Figure 5.10** (a) $\rho(1)$ The correlation between neighbors; (b)$\rho(2)$ The correlation between pairs with distance 2.

N-k model in Kauffman's book [36] is a model derived from biology. A gene have two types, say 0,1. A configuration of genes have N genes. The fitness of such a sequence b is average fitness of its N genes.

$$F(b) = \frac{1}{N}\sum_{i=1}^{N} f_i(s_i) \tag{5.1}$$

The fitness of each gene is a random function $f_i$ of $s_i$, composed of gene $i$ and other $0 \leq k \leq N$ genes, which are called its "neighbors". The neighbors could be randomly chosen or using $k$ bits adjacent to $i$. Correspondingly, the landscape is called random landscape or adjacent neighbor landscape. The landscape of the N-k Model is "tuneably rugged". It gradually evolve from a smooth plane to a rugged landscape as k increases.

The decision problem of an N-k model is defined as: Is the optimum of $f(x)$ equal to N. An N-k decision problem is insoluble if there is no solution for it. The N-k decision problem is NP complete for $k \geq 3$ based on a reduction from 3-SAT to the decision problem of N-k landscape. Meanwhile, a decision problem of the N-k landscape can be reduced to a (k+1)-SAT problem as indicated in Yong Gao's paper [21]. A N-2 landscape problem can be reduced to a 3-SAT problem. A 3-SAT problem can be reduced to N-3 landscape problem.

The exact correlation of two points with distance $d$ in N-k model is computed in Weinberger(1996) [57] For the random landscape, the correlation is

$$R(d) = 1 - \frac{d(k+1)}{N} + \frac{d(d-1)k(k+2)}{2N^2} + O((\frac{dk}{N})^3) \tag{5.2}$$

; for the adjacent neighbor landscape, the correlation is similar

$$R(d) = 1 - \frac{d(k+1)}{N} + \frac{d(d-1)k(k+1)}{2N^2} + O((\frac{dk}{N})^3) \tag{5.3}$$

.

The correlation of 3-SAT problem above density 2.0 obtained from our experiments is between the $R(d)$ with $k = 2$ and $k = 3$. For 3-SAT, the correlation between states

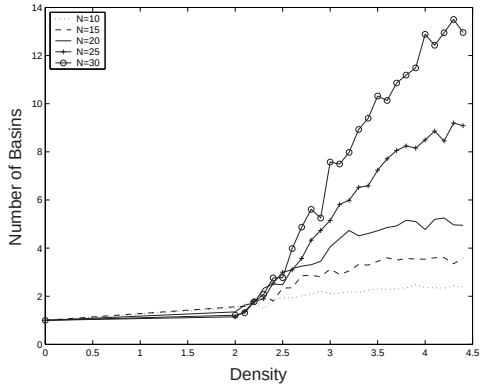
with small distance is very small for those instances with large order. This is one of the reasons why local search methods could solve 3-SAT with large order efficiently. For K-SAT problems with large $k$, the search procedure of local search algorithms significantly slow down because of the ruggedness landscape.

### 5.2.3 Non-solution basins

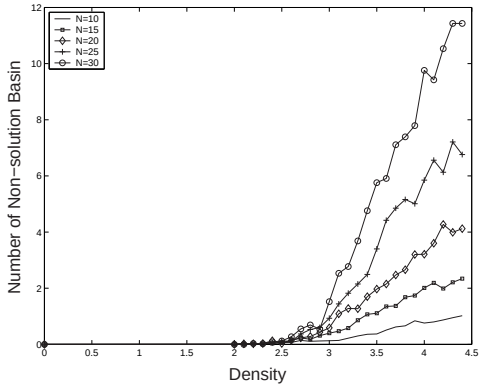
A basin is a flat area in the landscape whose out-going paths are all leading upwards. GSAT cannot make its way out of a basin once getting stuck into it. All the clusters in a solution set are basins, called solution basins in this thesis. There exist some basins with energy higher than 0, which we called non-solution basins.

To get non-solution basins, we have tried two two algorithms (4.3 and 4.4) introduced in Chapter 4, which use image computation algorithm and CUDD package. Similar with the experiments on clusters, the experiments were running on orders from 10 to 30 with an interval 5. With each order we ran experiments at different densities. Between density 2.0 and 4.4, densities are checked with an interval of 0.1. With each pair of order and density, 500 random 3-SAT problem instances were run. The size of an ADD build from a formula is much larger than a BDD of the same formula. We can reach order 30 when manipulating ADDs, lower than order 45 in the experiments of the solution set.

Near density 3.0, the number of non-solution basins is increasing from 0. It happens at the same density where the solution basins just begin to decrease. The number of non-solution basins grows up linearly as the density grows. Similar to the number




(a)




(b)

**Figure 5.11** (a) The number of basins; (b)The number of non-solution basins

of solution basins, it increases with the order. It is predictable that when $N \to \infty$ the number of non-solution basins also goes up without limit.

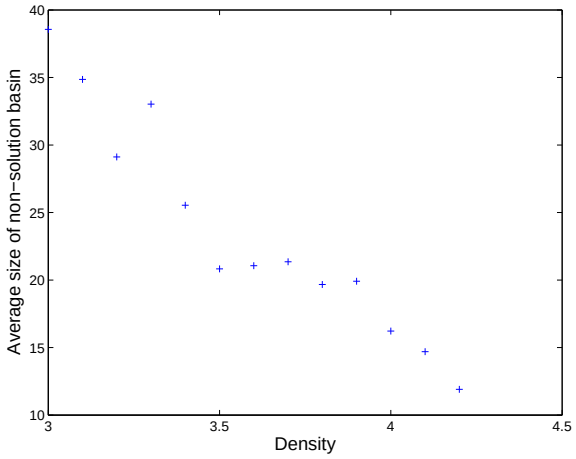
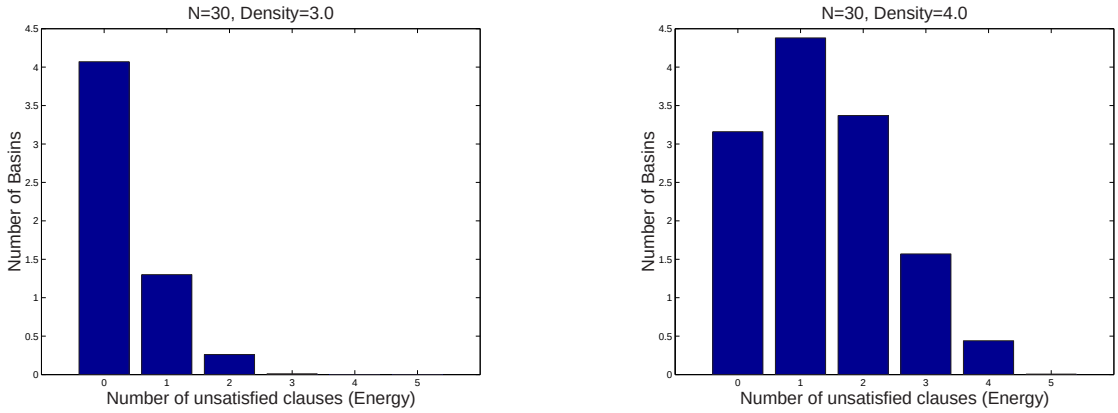



**Figure 5.12** Average size of non-solution basins

As we observed in Figure 5.12, sizes of non-solution basins are small compared to the size of solution set. Their sizes are comparable to the sizes of those small clusters

in the solution set. The sizes of non-solution basins are not even. There exist some large ones and most of them are small ones.




(a) (b)

**Figure 5.13** Basins are located at low energy(the number of unsatisfied clauses

We also plot the number of basins versus energy as Figure 5.13. Pictures with density from 2.0 to 4.4 are showed in appendix A.2. Through experiments, we discovered that basins are mainly located at or near the bottom of the landscape. The states in non-solution basins are quasi-solutions, which stand for those assignments which contains relatively few unsatisfied clauses.

Non-solution basins have an influence on the complexity of SAT solvers based on search. Search-based solvers easily locate quasi-solutions. States in non-solution basins are quasi-solutions far away from their nearest solutions. Searching from these quasi-solutions to a solution will take lots of time because of the long distance. The increase in number of states in non-solution basins will increase the cost of finding a

solution with algorithms based on search. We are going to present some experimental results to back up this theory.

Algorithms based on search locate quasi-solutions easily. The GSAT algorithm easily gets trapped in basins. After that, it cannot leave a non-solution basin without random walk. Even with the random walk, it tends to spend lots of time wandering in the basin because of the inclination to keep the lowest energy. As for the tree searching, quasi-solutions will not trigger backtracking until we go deeply in the search tree.

Our experiments were carried on all states with energy 1. We divided them into two groups, one of the groups contains states in basins and the other one contains the rest. For each state, we get the distance to its nearest solution and average over each group. According to our results in Figure 5.14, quasi-solutions in basins are far away from the nearest solution compared to other states with the same energy.

Distances coming from different groups showed significant difference. The distances between those states not in basins and their nearest solutions are small values between 1.5 and 2.5 and stay almost unchanged with the growing orders. In contrast, the distances between states in basins and their nearest solutions are larger. For example, the average distance is around 7 from instances with order 20. The distance keeps a stable value as the density grows. From the results of low-order instances, we can predict that the distance between states in basins and solutions will keep increasing with orders. When $N \to \infty$, the differences between these two distances could

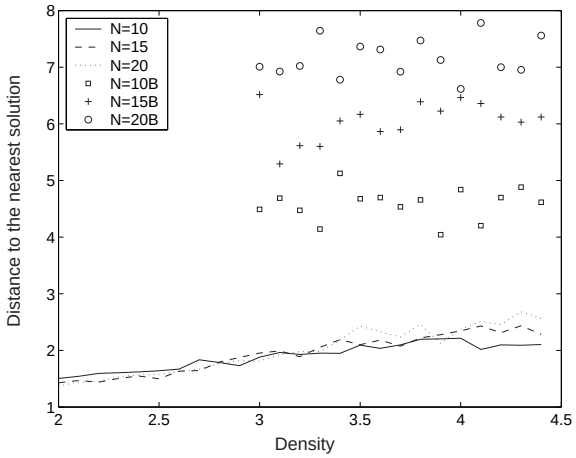
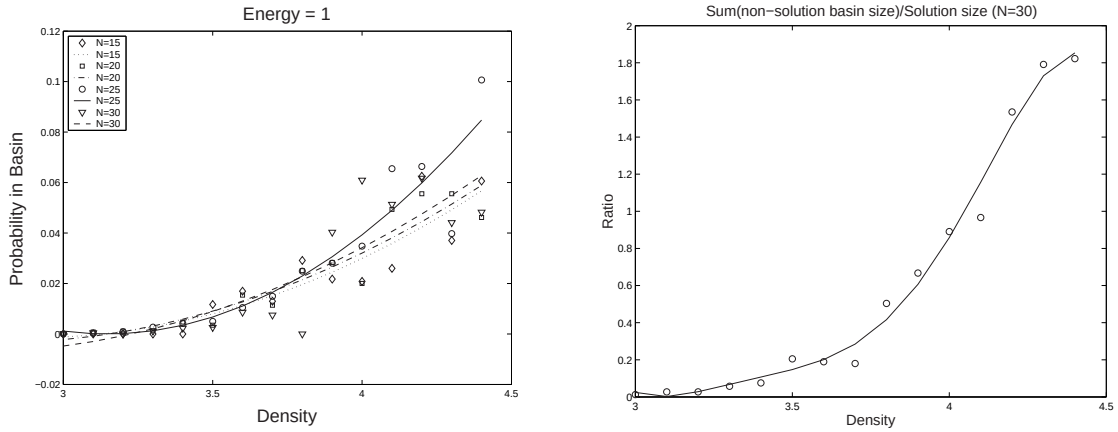



**Figure 5.14** Difference in distances to nearest solutions between states in non-solution basins and other states with the same energy

be very large. The long distance between two states requires large efforts to search from one to the other. With the local search methods, in order to reach another state with longer distance we have to investigate a larger area. With tree search algorithm, the longer distance between two leaves in the search tree usually introduces more backtracking.

In Figure5.15(a), the probability of a state with energy 1 in basins is lower than 10% below density 4.4. With energy 1, a solution has low probability being located in a non-solution basin. However, the sizes of non-solution basins are not trivial compared to the size of the solution set. We also plot the ratio of the number of all states in non-solution basins to the number of states in solution set as Figure5.15(b). The ratio increases from zero near density 3.2, and goes up polynomially. At density 4.0, there is almost the same number of states in non-solution basins as in a solution

set. Between density 3.2 and 4.0, this ratio is not negligible. Above density 4.0, there are more states in non-solution basins than in the solution set.



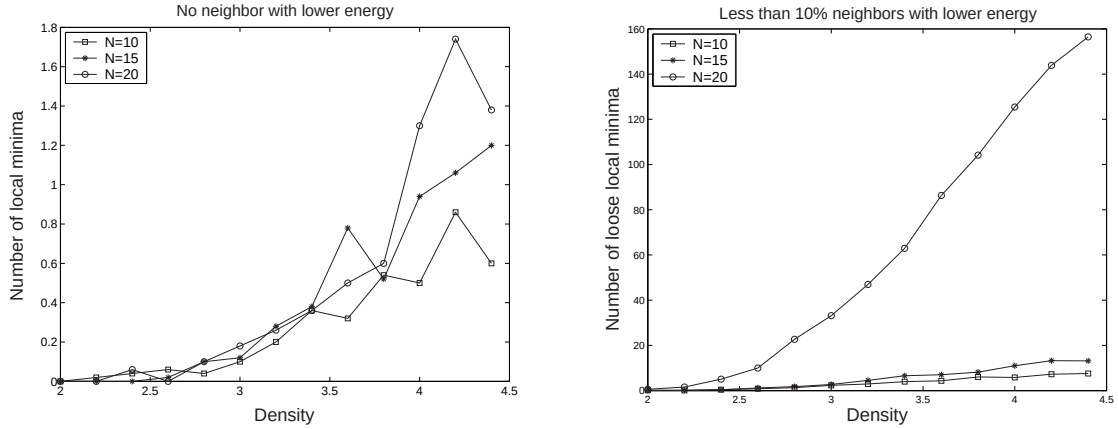

(a) (b)

**Figure 5.15** (a) The ratio of states with energy 1 in basins; (b)$\sum$ (non-solution basin size)/Solution size

### 5.2.4 Loose local minima

A local minimum in a local search space is defined as a state whose all neighbors have higher energy. In another words, it is a basin with size 1. Without sidewalk, the search procedure cannot walk out of a local minimum. As showed in Figure 5.16(a), there are only small number of strict local minima under this definition.

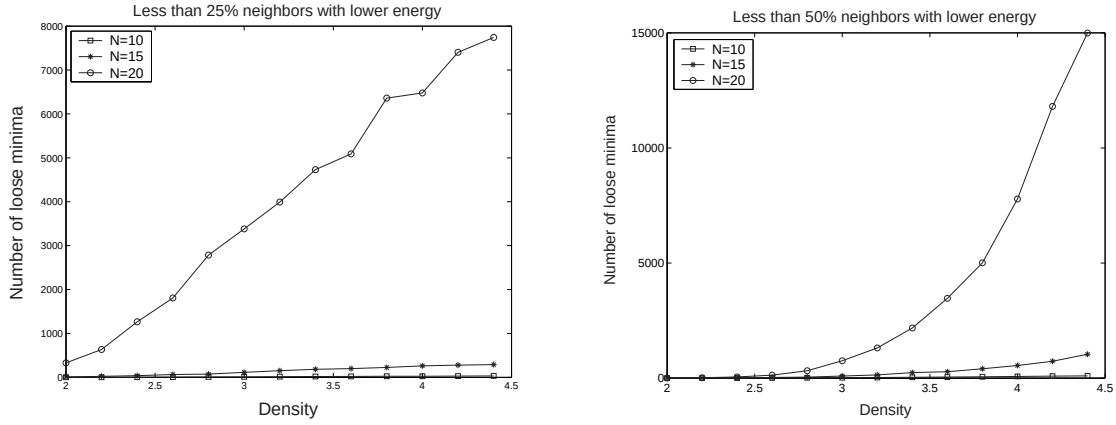
Those refined local search algorithms do not check all neighbors of a state to see if they lead downwards, because this checking process significantly slows down the searching procedure. The compromise is to make a random walk after a few unsuccessful checks. In this way, the algorithm will find another path to move down




**(a)** **(b)**

**Figure 5.16** (a) Number of loose minima; (b)Number of loose local minima(Less than 10% neighbors with lower energy)

in a shorter time. Therefore, the number of local minima does not influence local searching complexity directly. Those states with small percentage of neighbors leading downwards raise the complexity of local search. Because within a few tries, the neighbors with lower energy could be unfound. The local search makes random move in this case and probably move to a neighbor with higher energy. We call those states having less than some fraction of neighbors leading downwards as loose local minima. For each SAT instance, we exhaustively examine all states, and for each state, we exhaustively check all its neighbors. If less than some percent of its neighbors lead downwards, the state is a loose local minimum. We run experiments at densities between 2.0 and 4.4 with interval 0.2, and with three different orders. With each pair of parameters we run 50 instances and take the average value on the number of loose local minima.




**(a)** **(b)**

**Figure 5.17** (a) Number of loose local minima(Less than 25% neighbors with lower energy) (b)Number of loose local minima(Less than 50% neighbors with lower energy)

From Figure 5.16(b) and Figure 5.17, the number of loose local minima increases quick as we loosen the percentage of neighbors with lower energy. The 10% and 25% plot show the number grows up linearly with density. The 50% plot show a fast increase of the number of loose local minima with density. Below density 3.0, the number of loose local minima stays close to zero. Between density 3.0 and 4.5, the number of loose local minima increases quickly. Moreover, the increasing rate also goes up quickly. The large and quickly increasing number of loose local minima in these density range is one of facts making the complexity curve of local search algorithm turn from linear to quadratic.

# Chapter 6
# Conclusion and Future work

Our experiments confirmed the clustering behavior predicted by the analytical results of the spin model. In Monasson's paper, the cluster behavior is deduced from the fact that the distance between solutions splits to two different values beyond density 3.94. Our experiments confirmed Monasson's theory. The distribution of distance concentrates on one value when the density is below 3.6. Beyond density 3.7, The distribution of distance focuses on two different values. From another aspect of view, a cluster can be taken as a connected set. Using symbolic methods, we enumerated clusters under this definition. We observed that all solutions are connected as one cluster at low densities. The solution set breaks into clusters when density is above 2.0. The number of clusters increases as density grows and peaks at density 3.2. It decreases beyond density 3.2. The solutions reunite to fewer clusters before they varnish.

Expanding our field of view from solutions to all assignments, we got some better understanding on complexities of search based SAT solvers. We discovered three facts which could influence the complexity.b

All assignments constitute a landscape by taking the number of unsatisfied clauses as the energy of an assignment. The distribution of assignments versus energy follows partial or whole Gaussian distribution. The center of Gaussian distribution moves to

higher energies as density grows. The tail of Gaussian distribution indicate that there are small number of assignments at low energy area. It could be a reason of the high cost to search from an assignment with average energy to a solution, an assignment with energy 0 by local search algorithms.

The second fact is that some low-energy non-solution basins emerge when density grows beyond 3.2. The assignments in these basins are quasi-solutions, which attract the search pointer. The quasi-solutions in non-solution basins are far away from real solutions. It would take much time to find a path from a quasi-solution in non-solution basin to a solution.

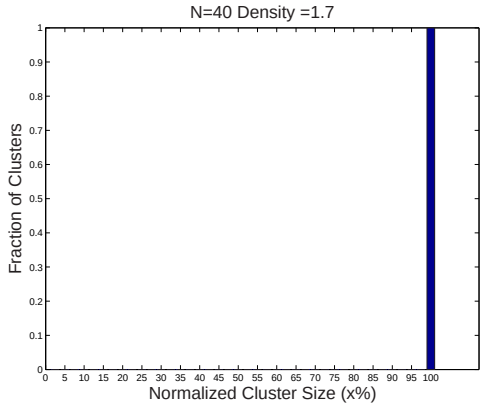
As for refined local search algorithms, the number of loose local minima is another factor to slow down the search procedure. We also found that the number of loose local minima increases quickly with density.

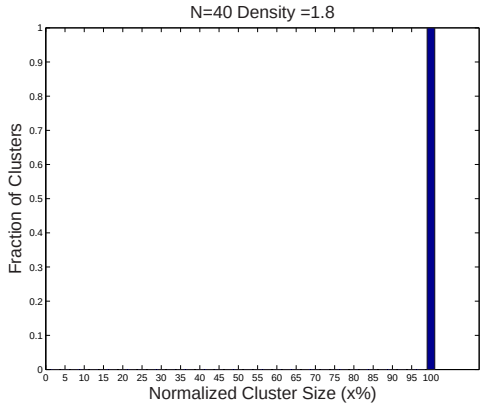
Statistical analysis can be applied to justify our experiment results. How to apply these results to improve the performance of SAT solvers could be explored in the future.

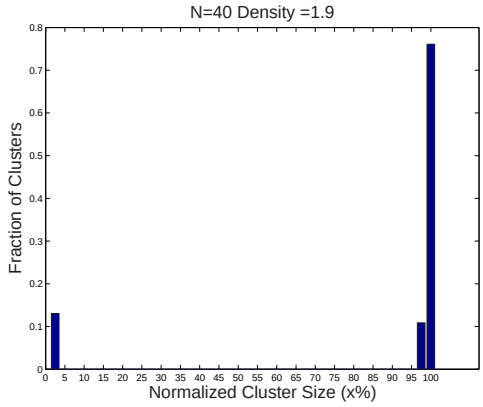
# Appendix A
# An Appendix

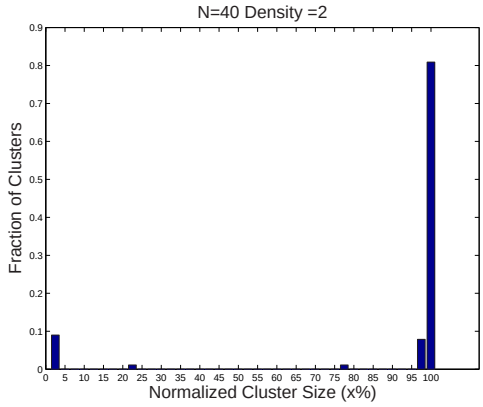
In order to show the change of some parameters versus density, we put all figures here with density interval 0.1. They illustrate the transformation of the structure of solution set.

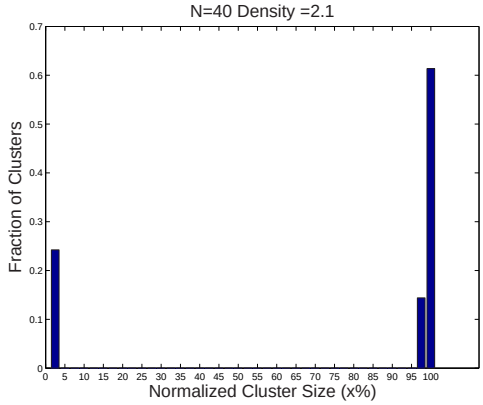
## A.1 The histogram of cluster size

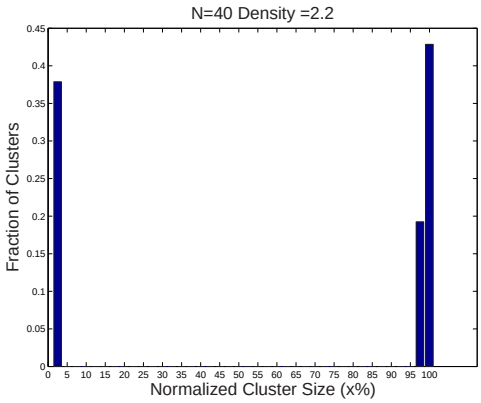


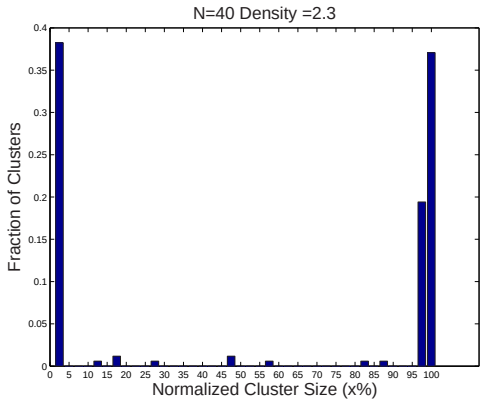



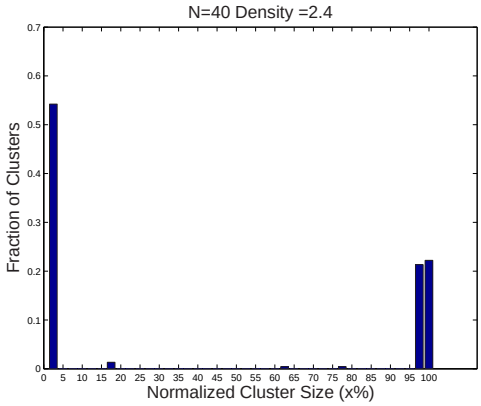



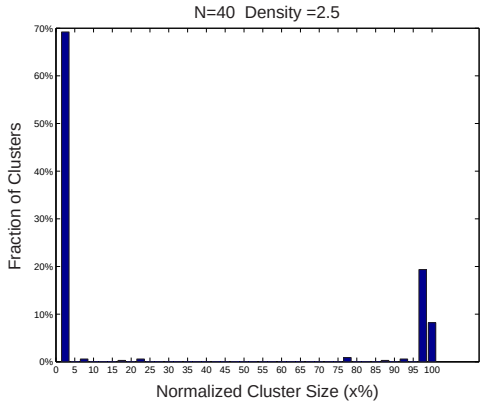



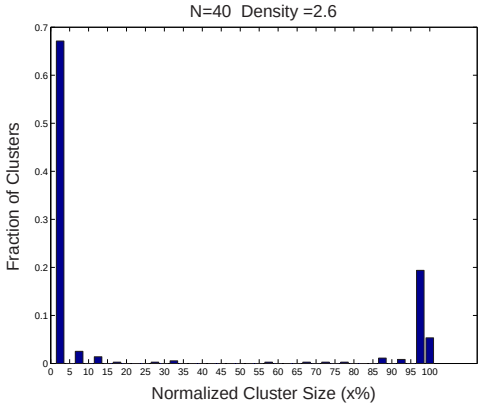

N=40 Density =2.1
Fraction of Clusters
Normalized Cluster Size (x%)

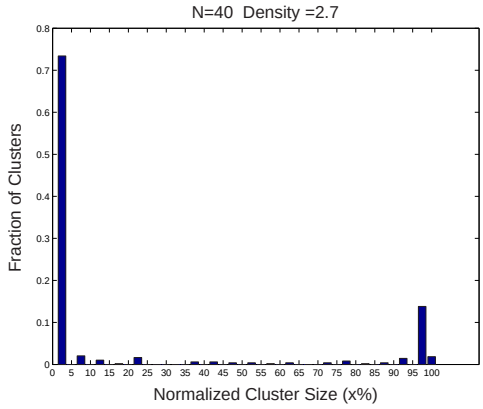

N=40 Density =2.2
Fraction of Clusters
Normalized Cluster Size (x%)

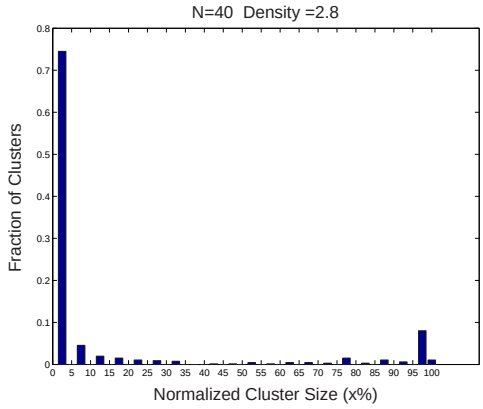

N=40 Density =2.3
Fraction of Clusters
Normalized Cluster Size (x%)

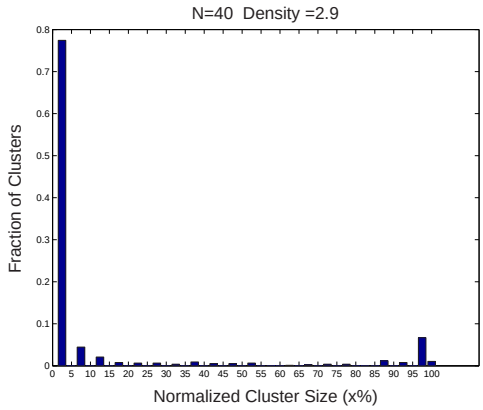

N=40 Density =2.4
Fraction of Clusters
Normalized Cluster Size (x%)

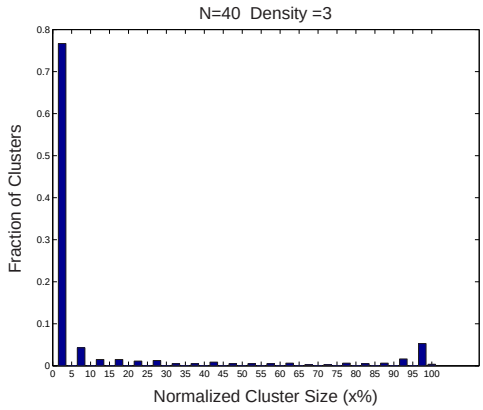

N=40 Density =2.5
Fraction of Clusters
Normalized Cluster Size (x%)

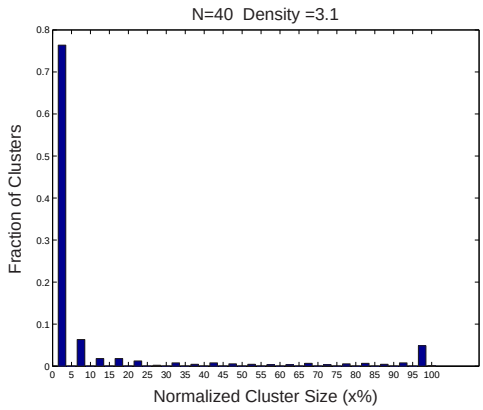

N=40 Density =2.6
Fraction of Clusters
Normalized Cluster Size (x%)

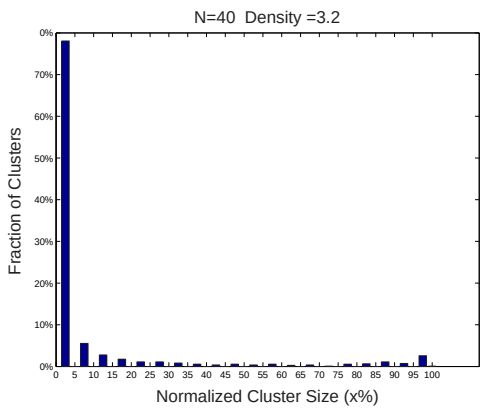

N=40 Density =2.7
Fraction of Clusters
Normalized Cluster Size (x%)

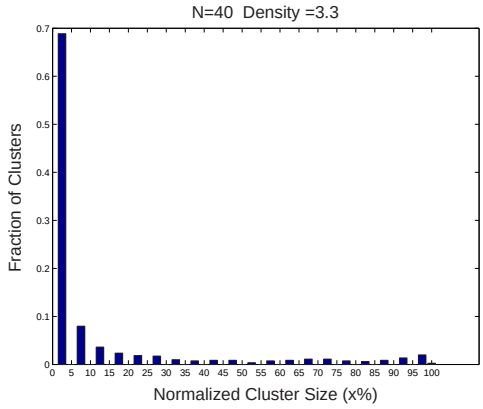

N=40 Density =2.8
Fraction of Clusters
Normalized Cluster Size (x%)

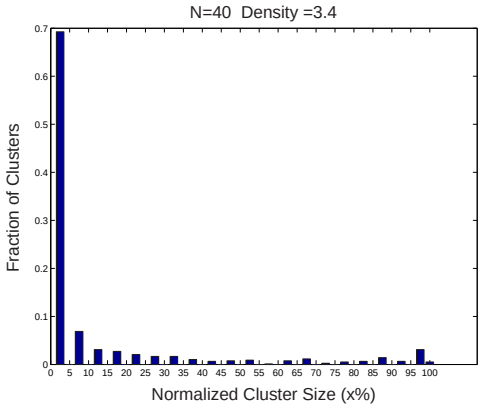

N=40 Density =2.9
Fraction of Clusters
Normalized Cluster Size (x%)

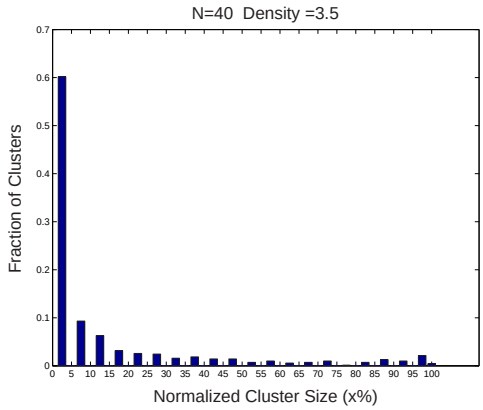

N=40 Density =3
Fraction of Clusters
Normalized Cluster Size (x%)

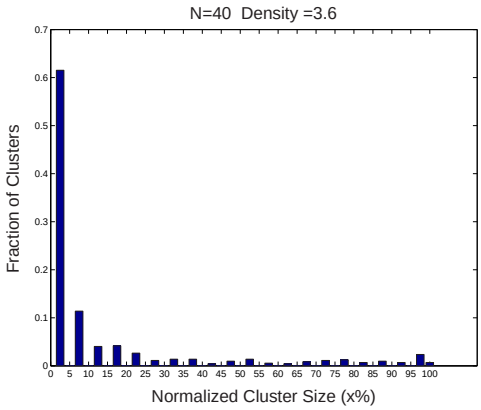

N=40 Density =3.1
Fraction of Clusters
Normalized Cluster Size (x%)

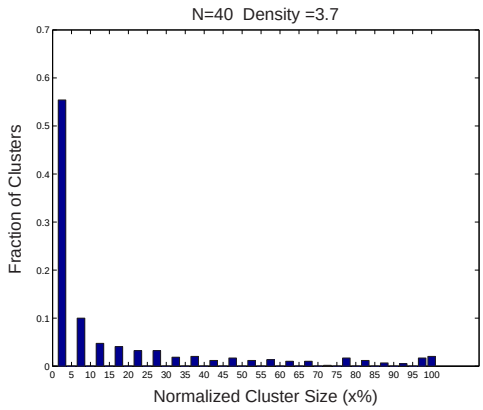

N=40 Density =3.2
Fraction of Clusters
Normalized Cluster Size (x%)

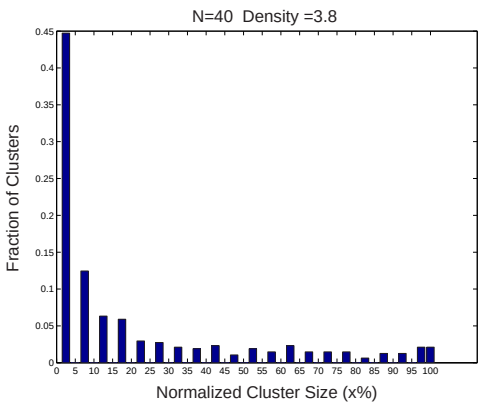

N=40 Density =3.3
Fraction of Clusters
Normalized Cluster Size (x%)

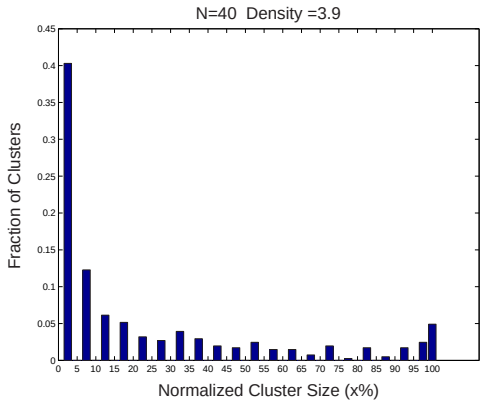

N=40 Density =3.4
Fraction of Clusters
Normalized Cluster Size (x%)

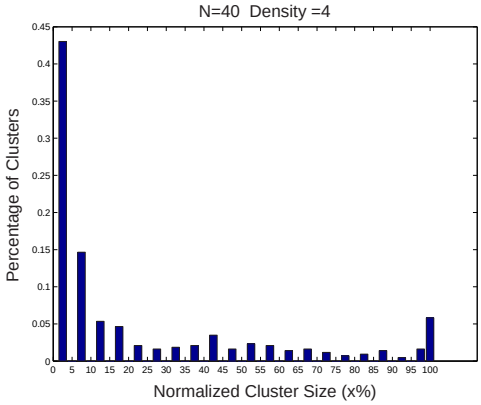

N=40 Density =3.5
Fraction of Clusters
Normalized Cluster Size (x%)

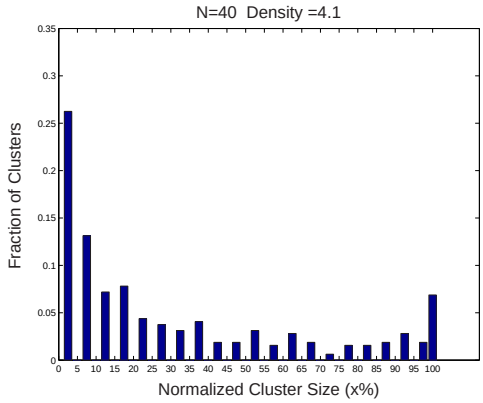

N=40 Density =3.6
Fraction of Clusters
Normalized Cluster Size (x%)

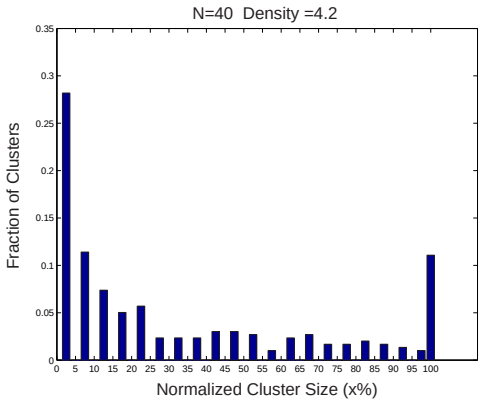

N=40 Density =3.7
Fraction of Clusters
Normalized Cluster Size (x%)

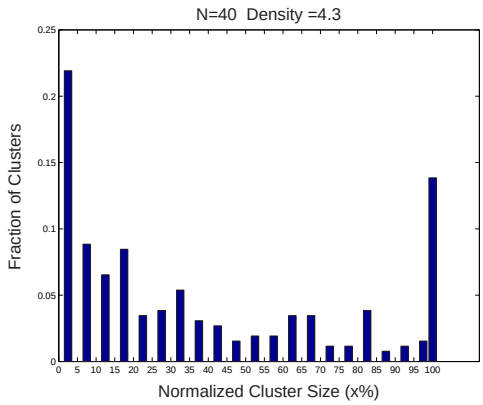

N=40 Density =3.8
Fraction of Clusters
Normalized Cluster Size (x%)

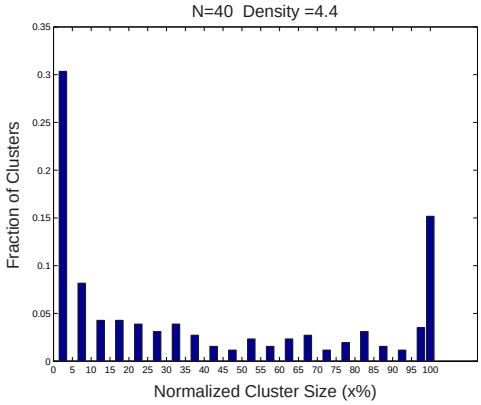

N=40 Density =3.9
Fraction of Clusters
Normalized Cluster Size (x%)

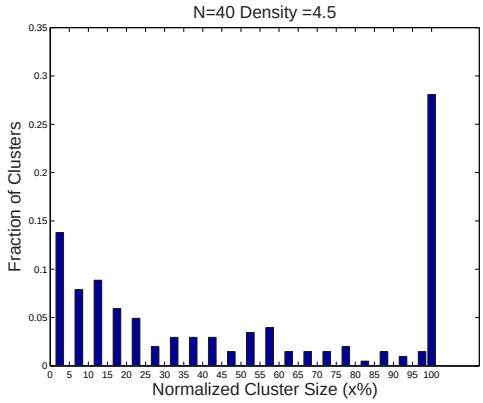

N=40 Density =4
Percentage of Clusters
Normalized Cluster Size (x%)

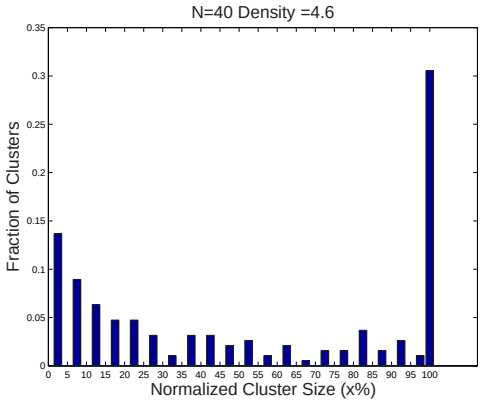

N=40 Density =4.1
Fraction of Clusters
Normalized Cluster Size (x%)

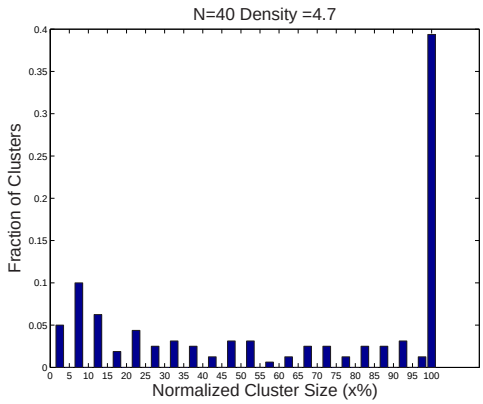

N=40 Density =4.2
Fraction of Clusters
Normalized Cluster Size (x%)

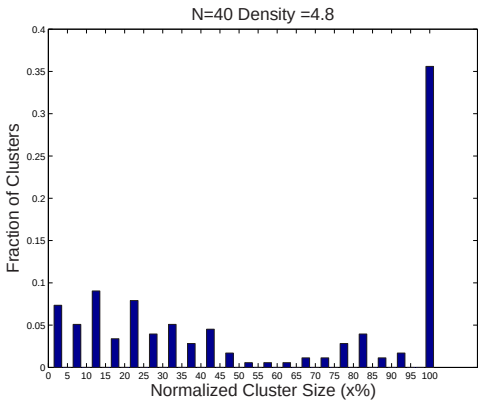

N=40 Density =4.3
Fraction of Clusters
Normalized Cluster Size (x%)


N=40 Density =4.4
Fraction of Clusters
Normalized Cluster Size (x%)

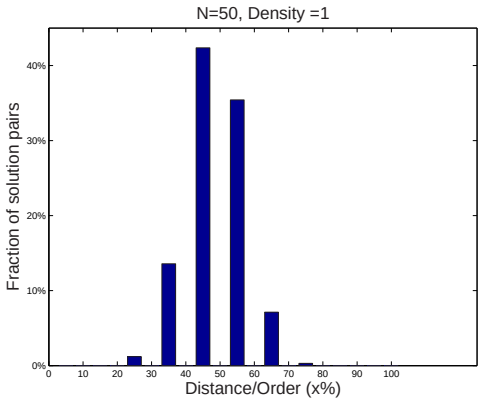

N=40 Density =4.5
Fraction of Clusters
Normalized Cluster Size (x%)

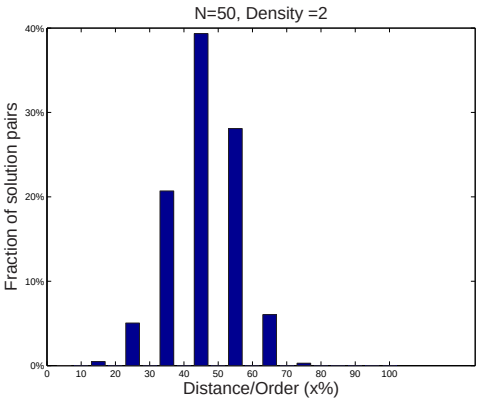

N=40 Density =4.6
Fraction of Clusters
Normalized Cluster Size (x%)

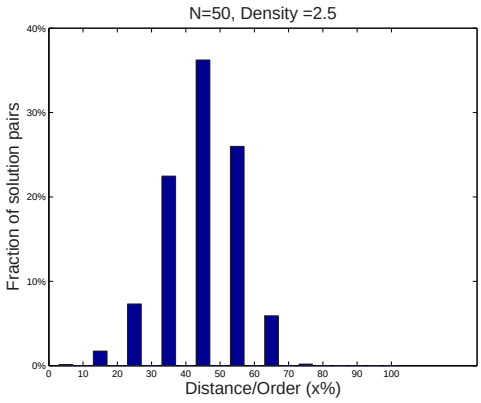

N=40 Density =4.7
Fraction of Clusters
Normalized Cluster Size (x%)

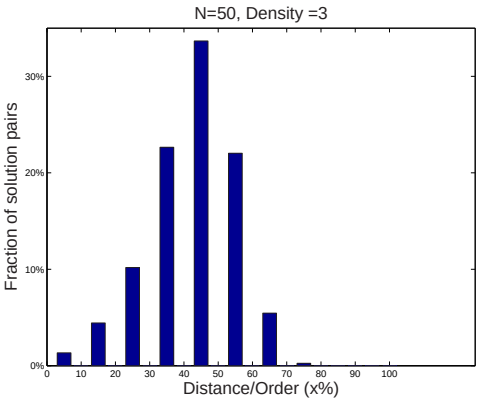

N=40 Density =4.8
Fraction of Clusters
Normalized Cluster Size (x%)

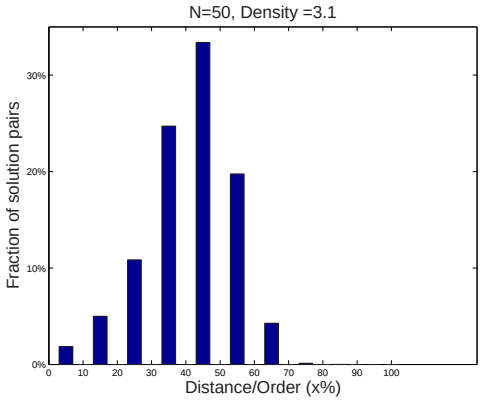
## A.2 The histogram of distance between solutions

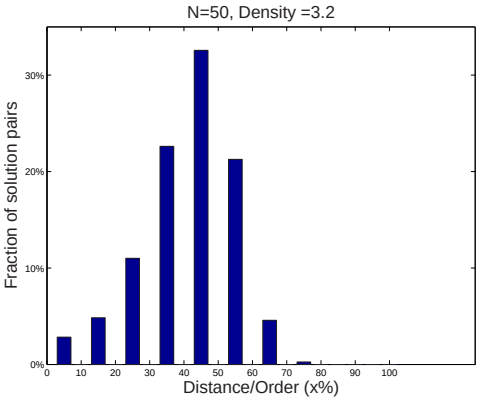


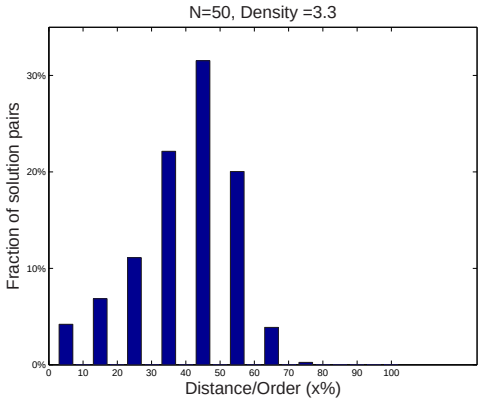



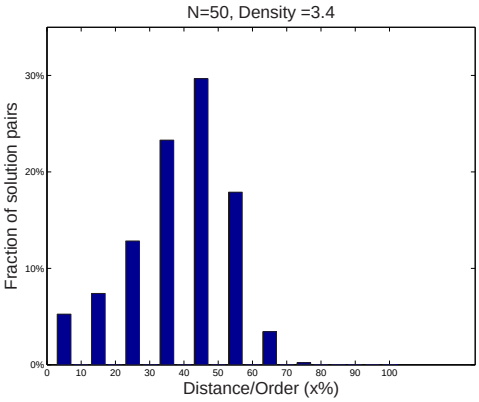



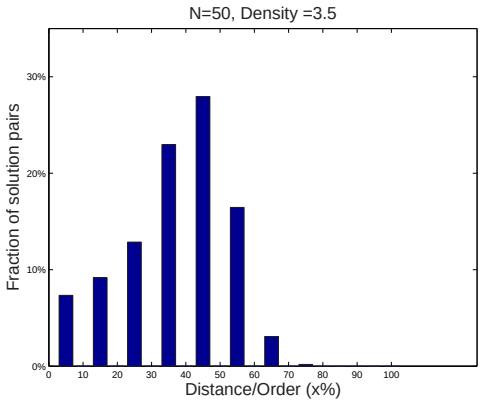



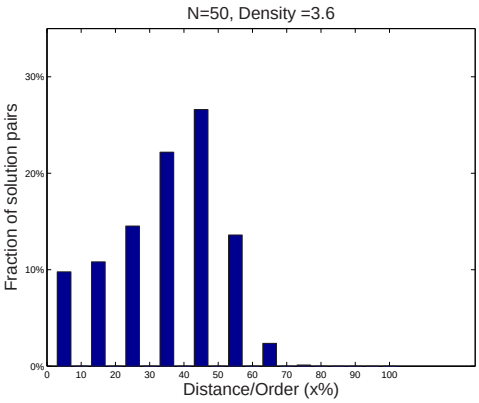

N=50, Density =3.1
Fraction of solution pairs
Distance/Order (x%)

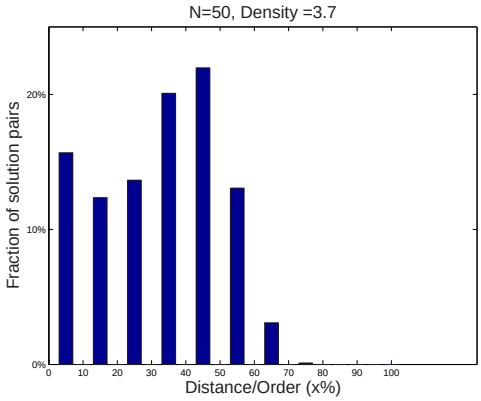

N=50, Density =3.2
Fraction of solution pairs
Distance/Order (x%)

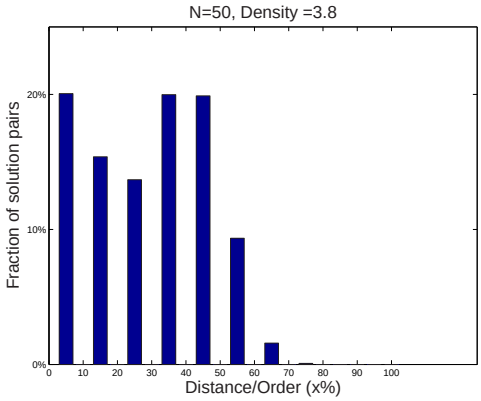

N=50, Density =3.3
Fraction of solution pairs
Distance/Order (x%)

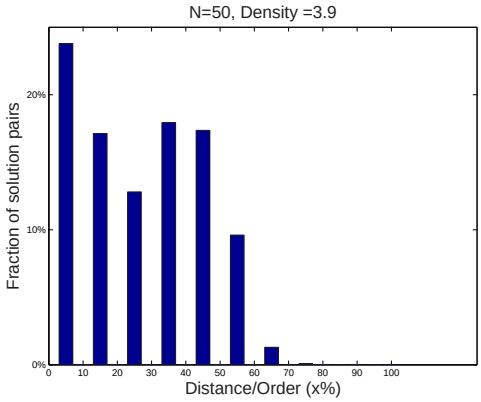

N=50, Density =3.4
Fraction of solution pairs
Distance/Order (x%)

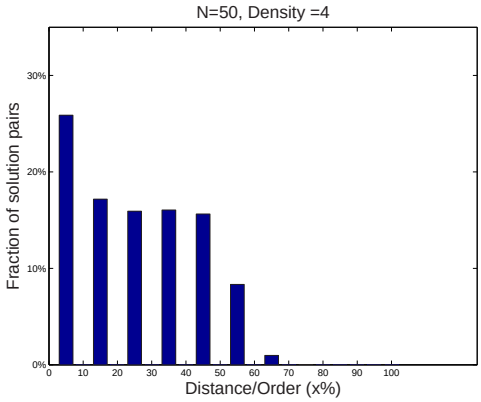

N=50, Density =3.5
Fraction of solution pairs
Distance/Order (x%)

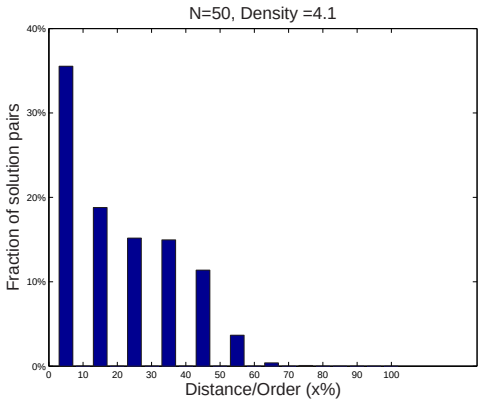

N=50, Density =3.6
Fraction of solution pairs
Distance/Order (x%)

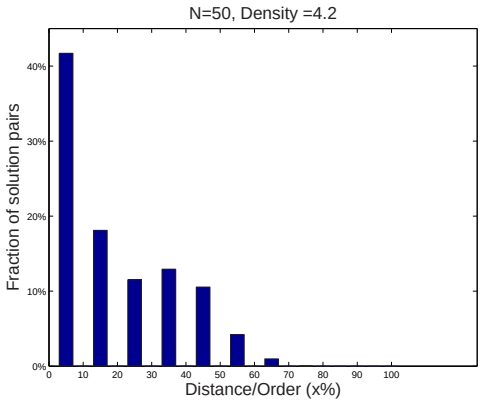

N=50, Density =3.7
Fraction of solution pairs
0%
10%
20%
0 10 20 30 40 50 60 70 80 90 100
Distance/Order (x%)

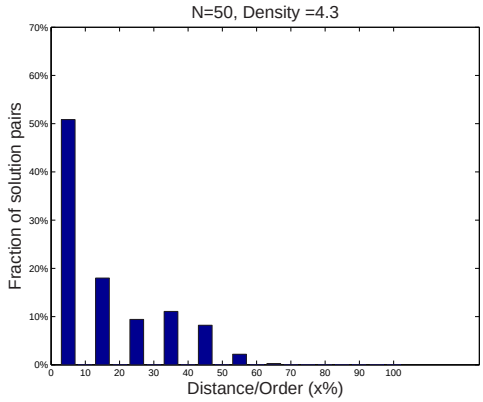

N=50, Density =3.8
Fraction of solution pairs
0%
10%
20%
0 10 20 30 40 50 60 70 80 90 100
Distance/Order (x%)


N=50, Density =3.9
Fraction of solution pairs
0%
10%
20%
0 10 20 30 40 50 60 70 80 90 100
Distance/Order (x%)

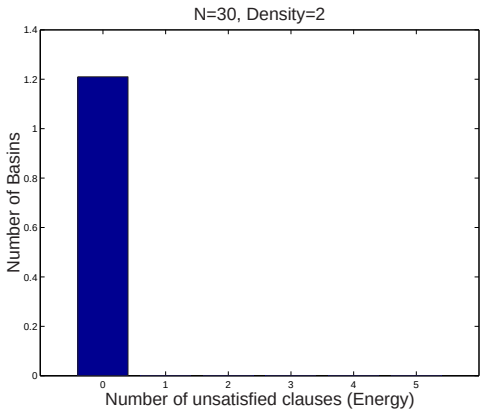

N=50, Density =4
Fraction of solution pairs
0%
10%
20%
30%
0 10 20 30 40 50 60 70 80 90 100
Distance/Order (x%)

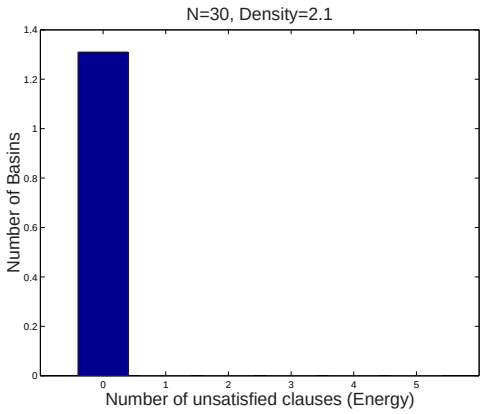

N=50, Density =4.1
Fraction of solution pairs
0%
10%
20%
30%
40%
0 10 20 30 40 50 60 70 80 90 100
Distance/Order (x%)

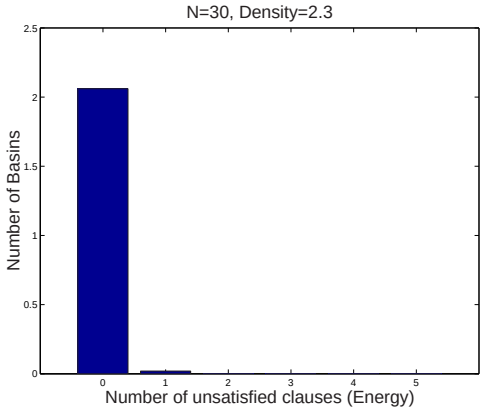

N=50, Density =4.2
Fraction of solution pairs
0%
10%
20%
30%
40%
0 10 20 30 40 50 60 70 80 90 100
Distance/Order (x%)

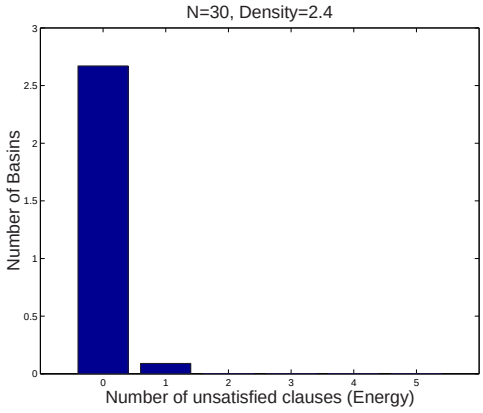

N=50, Density =4.3
Fraction of solution pairs
70%
60%
50%
40%
30%
20%
10%
0%
0
10
20
30
40
50
60
70
80
90
100
Distance/Order (x%)

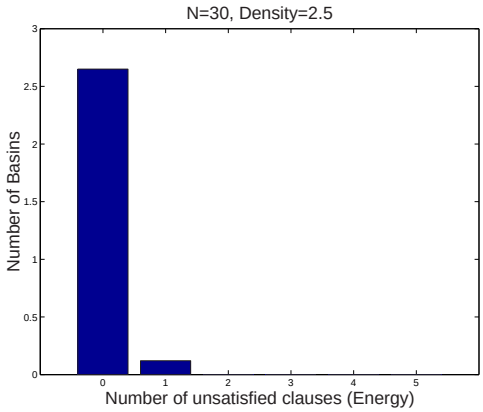
## A.3 The energy of basins

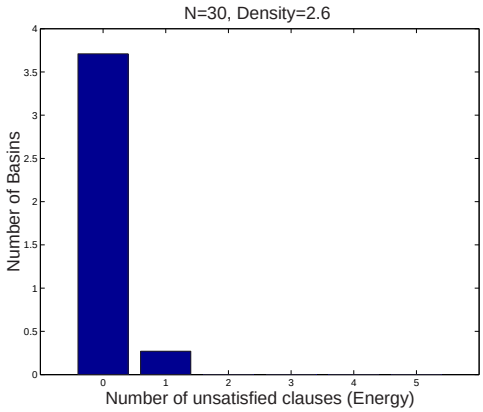


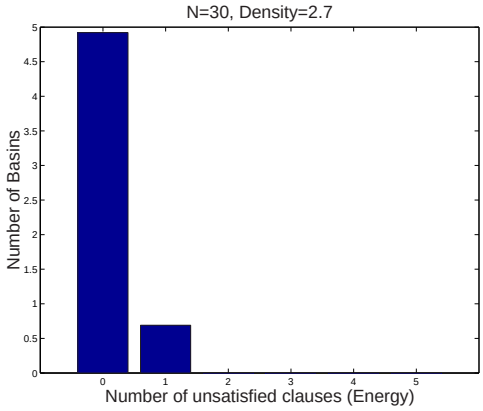



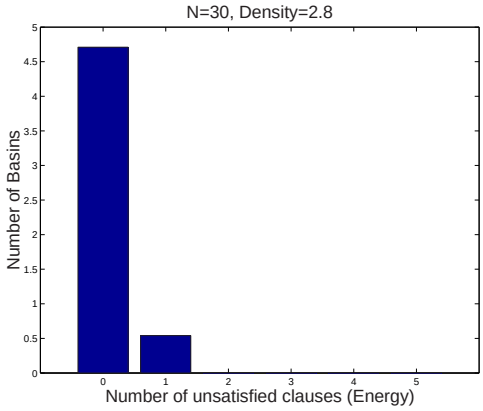



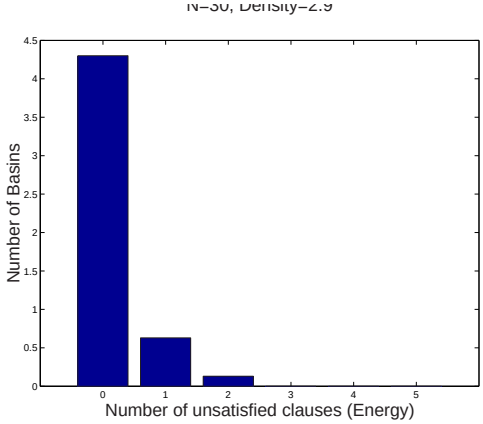



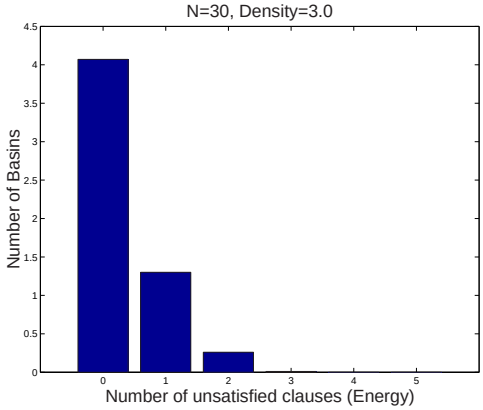

N=30, Density=2.5
Number of Basins
Number of unsatisfied clauses (Energy)

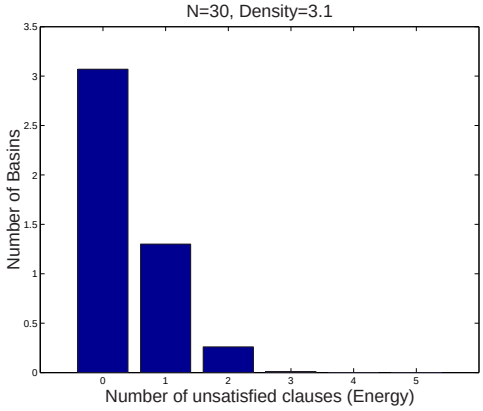

N=30, Density=2.6
Number of Basins
Number of unsatisfied clauses (Energy)

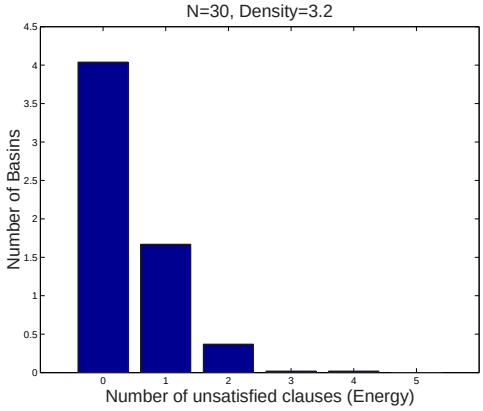

N=30, Density=2.7
Number of Basins
Number of unsatisfied clauses (Energy)

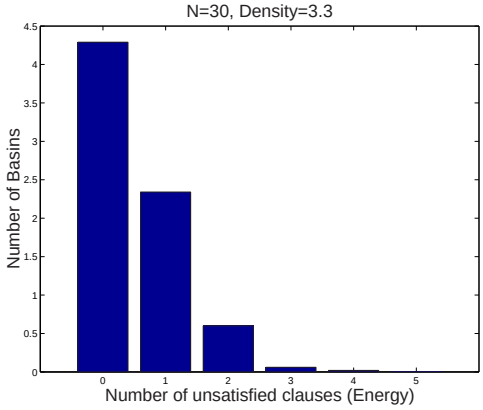

N=30, Density=2.8
Number of Basins
Number of unsatisfied clauses (Energy)

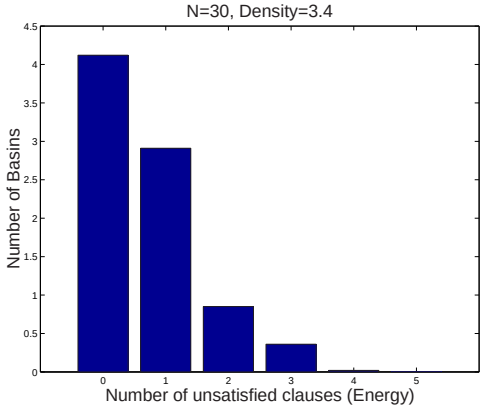

Number of Basins
Number of unsatisfied clauses (Energy)

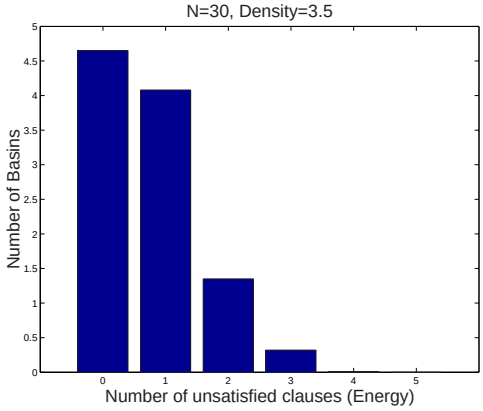

N=30, Density=3.0
Number of Basins
Number of unsatisfied clauses (Energy)

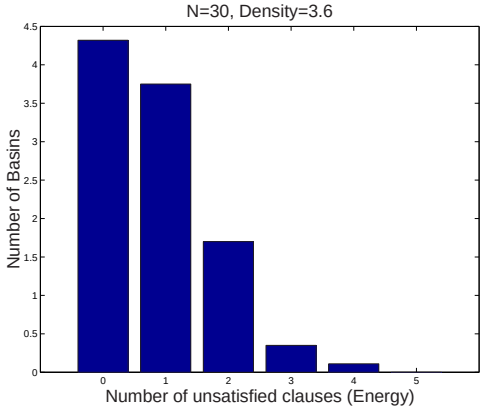

N=30, Density=3.1
Number of Basins
Number of unsatisfied clauses (Energy)

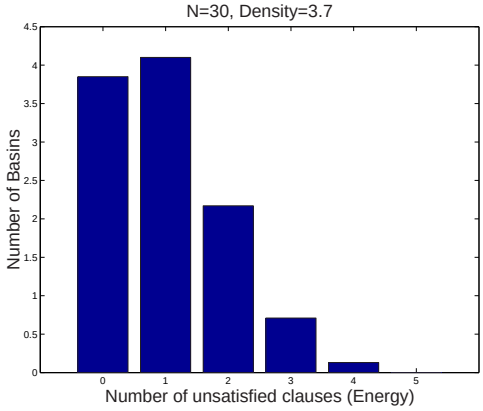

N=30, Density=3.2
Number of Basins
Number of unsatisfied clauses (Energy)

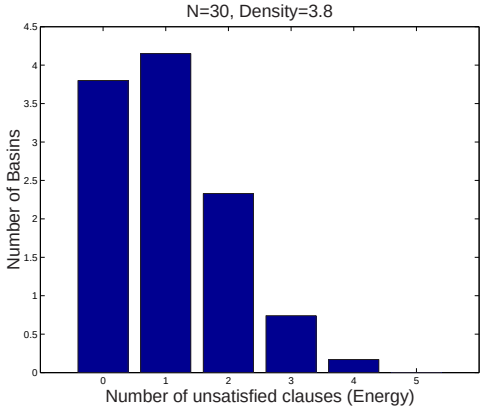

N=30, Density=3.3
Number of Basins
Number of unsatisfied clauses (Energy)

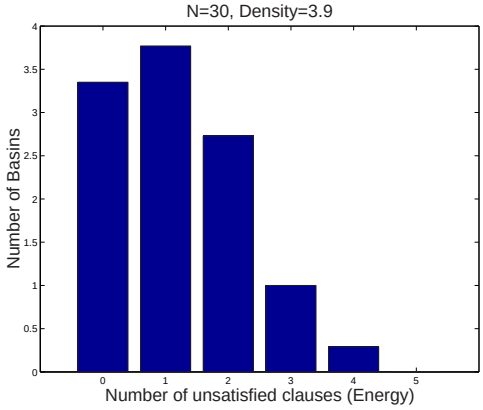

N=30, Density=3.4
Number of Basins
Number of unsatisfied clauses (Energy)

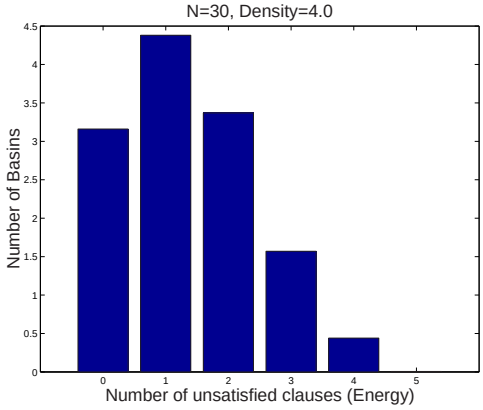

N=30, Density=3.5
Number of Basins
Number of unsatisfied clauses (Energy)

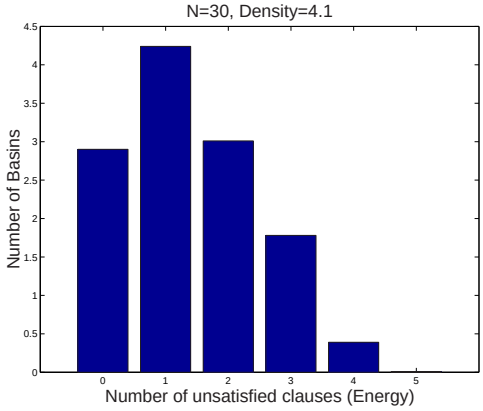

N=30, Density=3.6
Number of Basins
Number of unsatisfied clauses (Energy)

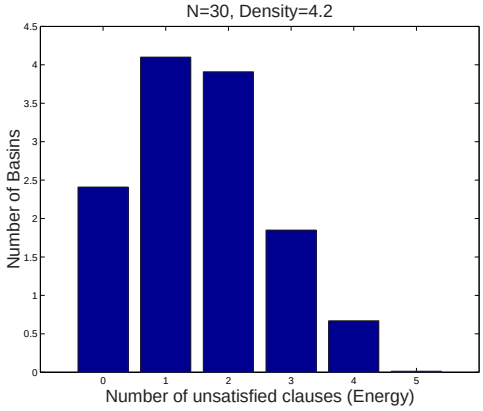

N=30, Density=3.7
Number of Basins
Number of unsatisfied clauses (Energy)

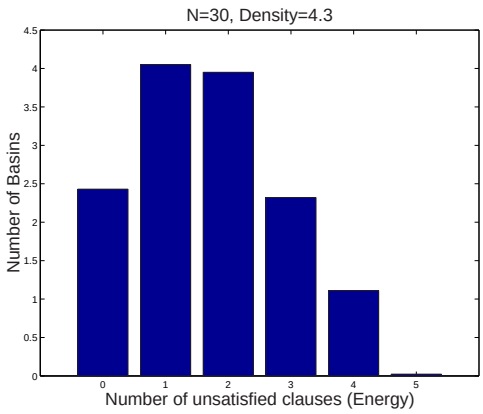

N=30, Density=3.8
Number of Basins
Number of unsatisfied clauses (Energy)

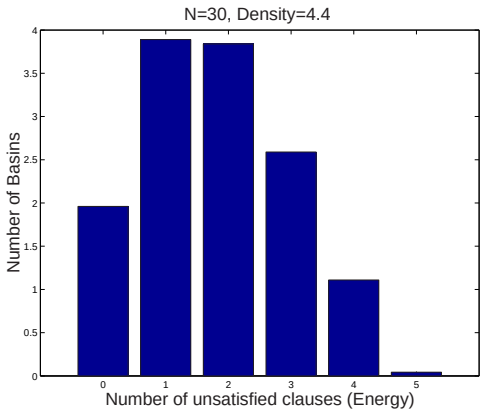

N=30, Density=3.9
Number of Basins
Number of unsatisfied clauses (Energy)


N=30, Density=4.0
Number of Basins
Number of unsatisfied clauses (Energy)


N=30, Density=4.1
Number of Basins
Number of unsatisfied clauses (Energy)


N=30, Density=4.2
Number of Basins
Number of unsatisfied clauses (Energy)


N=30, Density=4.3
Number of Basins
Number of unsatisfied clauses (Energy)


N=30, Density=4.4
Number of Basins
Number of unsatisfied clauses (Energy)

# References


1. A.S.M. Aguirre and M. Vardi. Random 3-sat and bdds: the plot thickens further. In *Proceeding of 7th International Conference on Principles and Practice of Constraint Programming (CP2001)*, page 121, 2001.

2. S.B. Akers. Binary decision diagrams. *IEEE Trans. on Computers*, vol.c-27, no.6:509–516, 1978.

3. H.R. Andersen. An introduction to binary decision diagrams. *Lecture notes for 49285 advanced algorithms E97*, 1998.

4. E. Angel. On the landscape ruggedness of the quadratic assignment. *Theoretical Computer Science*, pages 159–172, 2001.

5. A.M. Frieze A.Z. Broder and E. Upfal. On the satisfiability and maximum satisfiability of random 3-cnf formulas. In *Proc. 4th Annual ACM-SIAM Symp.on Discrete Algorithms*, page 322, 1993.

6. D.G. Mitchell B. Selman and H.J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):273–295, 1996.

7. H. Kautz B. Selman and B. Cohen. Local search strategies for satisfiability testing. In *Proceedings of 2nd DIMACS Challenge on Cliques, Coloring and Satisfiability*, pages 290–295, 1993.

8. H. Levesque B. Selman and D. Mitchell. Gsat - a new method for solving hard satisfiability problems. In *Proceedings AAAI-92*, pages 440–446, 1992.

9. H.A. Kautz B. Selman and B. Cohen. Noise strategies for improving local search. In *Proc. AAAI-94*, pages 337–343, 1994.

10. Y. Boufkhad and O. Dubois. Length of prime implicants and number of solutions of random cnf formulas. *Theoretical Computing Science*, 215:1–30, 1999.

11. R.E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

12. A.S. Miguel M. Vardi C. Coarfa, D.D. Demopoulos and D. Subramanian. Random 3-sat: the plot thickens. In *Proceedings 6th International Conference on Principles and Practice of Constraint Programming*, pages 143–159, Singapore, 2000.

13. R.G. Jeroslow C.E. Blair and J.K. Lowe. Some results and experiments in programming techniques for propositional logic. *Computers and Operations Research*, 5:633–645, 1986.

14. S.A. Cook. The complexity of theorem proving procedures. In *Proceeding of Third Annual ACM Symposium on the Theory of Computing*, pages 151–158, New york, 1971. ACM.

15. J.M. Crawford and L.D. Auton. Experimental results on the croossover point in random 3-sat. In *Proceeding of 11th National Conference on Articial Intelligence*, volume 81(1-2), pages 21–27, 1993.

16. B. Selman D. McAllester and H. Kautz. Evidence for invariants in local search. In *Proceeding of AAAI97*, pages 321–326, 1997.

17. B. Selman D. Mitchell and H. Levesque. Hard and easy distributions of sat problems. In AAAI Press/ MIT Press, editor, *Proceedings of the 10th National Conference on Artificial Intelli- gence*, pages 459–465, 1992.

18. M. Davis and H. Putnam. A computing procedure of quantification theory. *Journal of the ACM(JACM)*, Volume 7:201–215, 1960.

19. J. Erenrich and B. Selman. Sampling combinatorial spaces using biased random walks. In *IJCAI posters*, 2003.

20. R. Monasson G. Biroli and M. Weigt. A variational description of the ground state structure in random satisfiability problems. *Eur. Phys. J. B*, Vol.14, 2000.

21. Y. Gao. Threshold phenomena in nk landscapes. *Master Thesis, University of Alberta*, pages 21–25, 2001.

22. M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. New York: W.H.Freeman, 1983.

23. M. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence*, 1(25-46), 1993.

24. J. Gu. Local search for satisfiability (sat) problem. *IEEE Trans. on Systems, Man, and Cybernetics*, 24(4):709, Apr. 1994.

25. J.N. Hooker. A quantitative approach to logical inference. *Decision Support Systems*, 4:45–69, 1988.

26. J.N. Hooker and C. Fedjki. Branch-and-cut solution of inference problems in propositional logic. *Technical Report 77-89, GSIA, Carnegie Mellon University*, Aug. 1989.

27. H.H. Hoos. An adaptive noise mechanism for walksat. In *AAAI press*, pages 655–660, 2002.

28. K. Huang. *Statistical Mechanics*. Wiley, New York, 1967.

29. P. Prosser P. Shaw I.P. Gent, E. MacIntyre and T. Walsh. The constrainedness of arc consistency. In *Proc. CP-97*, pages 327–340, 1997.

30. J. Franco J. Gu, P.W. Purdom and B.W. Wah. Algorithms for the satisfiability(sat) problems: a survey. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society*, pages 19–152, 1997.

31. I.P. Gent J. Singer and Alan Smaill. Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, pages 235–270, 2000.

32. G. Johnson. Separateing insolvable and difficult. *The New York Times*, July 13, 1999.

33. K.L. McMillan J.R. Burch, E.M. Clarke and D.L. Dill. Sequential circuit verification using symbolic model checking. In *Proc. Design Automation Conf.*, 1990.

34. R.M. Karp. Reducibility among combinatorial problems. *In Complexity of Computer Computations*, New york: Plenum:85–103, 1972.

35. S.A. Kaufman. *The origins of order: self-organization and selection in evolution*, volume Chapter2. Oxford University Press, 1993.

36. S.A. Kaufman. *At home in the universe: the search for the laws of self-organization and complexity*, volume Chaper8. Oxford University Press, 1995.

37. V. Kumar. Algorithms for constraint satisfaction problems: a survey. *AI Magazine*, Vol.13, No.1:32–34, 1992.

38. C. Lee. Representation of switching cirtuits by binary decision diagrams. *Bell Systems Tech. Journal*, 38:985–999, 1959.

39. J.A. Ludwig and J.F. Reynolds. *Statistical ecology: a primer on methods and computing*. John Wiley and Sons Ltd, New York, 1988.

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

41. G. Parisi M. Mezard and M.A. Virasoro. *Spin glass theory and beyond*. World Scientific, Singapore, 1987.

42. G. Parisi M. Mezard and R. Zecchina. Analytic and algorithmic solutions of random satisfiability problems. *Science*, 297:812, 2002.

43. E. Mendelson. *Introduction to Mathematical Logic*. London: Chapman and Hall, page 30, 1997.

44. M. Mezard and G. Parisi. Replica field theory for random manifolds. *Journal of Physics*, pages 809–836, 1991.

45. M. Mezard and A.P. Young. Replica symmetry breaking in the random field ising model. *Europhys. Lett.*, 18:653, 1992.

46. R. Monasson and R. Zecchina. Statistical mechanics of the random k-sat problem. *Physcis Review*, E 56:1357–1361, 1997.

47. R. Monasson O.C. Martin and R. Zecchina. Statistical mechanics methods and phase transitions in optimization problems. *Theoretical Computer Science*, 265(1-2):3–67, 2001.

48. P. Prosser. Forward checking with backmarking. *Technical ReportAISI-48-93, Computer Science, Glasgow University*, 1993.

49. C.M.Gaona G.D.Hachtel E.Macii A.Pardo R. I.Bahar, E.A.Frohm and F.Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 188–191, 1993.

50. S. Kirkpatrick B. Selman R. Monasson, R. Zecchina and L.Troyansky. 2+p-sat: relation of typical-case complexity to the nature of the phase transition. *Random Structures and Algorithms*, (15):414–435, 1999.

51. S. Kirkpatrick B. Selman R. Monasson, R. Zecchina and L. Troyansky. Determining computational complexity from character. *Nature*, Vol.400(8):133–137, 1999.

52. B. Selman. new figures. *http://www.cs.cornell.edu/home/selman/*, cite on April, 2004.

53. B. Selman and S. Kirkpatrick. Critical behavior in the computational cost of satisfiability testing. *Artificial Intelligence*, 81(1-2):273–295, 1996.

54. F. Somenzi. Binary decision diagrams. *Calculational System Design*, 173 of NATO Science Series F: Computer and Systems Sciences:303–366, 1999.

55. F. Somenzi. *CUDD manual*. Department of Electrical and Computer Engineering,University of Colorado at Boulder, http://vlsi.colorado.edu/ fabio/CUDD/, 2004.

56. A. angan W. Zhang and M. Looks. Backbone guided local search for maximum satisfiability. In *Proceedings of 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 9–15, 2003.

57. E.D. Weinberger. Np completeness of kauffman's n-k model, a tuneable rugged fitness landscape. *http://ideas.repec.org/p/wop/safiwp/96-02-003.html*, 1996.

58. Z. Yang. Perfomance analysis of symbolic reachability algorithms in model checking. *Master Thesis, Rice University*, 1999.