UNIVERSITY OF SOUTHAMPTON

Clustering Solutions: A Novel Approach to Solving NP-Complete Problems

by

Mohamed Qasem

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Faculty of Engineering and Applied Science

School of Electronics and Computer Science

June 2010

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Mohamed Qasem

In this thesis, we introduce a novel approach to solving MAX-SAT problems. This algorithm clusters good solutions, and restarts the search from the closest feasible configuration to the centroid of each cluster. We call this method Clustered-Landscape Guided Hopping (CLGH). In addition, where clustering does not provide an advantage due to the non-clustered landscape configuration, we use Averaged-Landscape Guided Hopping (ALGH). CLGH is shown to be highly efficient for finding good solutions of large MAX-SAT problems. Systematic studies of the landscape are presented to show that the success of clustering is due to the learning of large-scale structure of the fitness landscape. Previous studies conducted by other researchers analysed the relationship between local and global minima and provided an insight into the configuration of the landscape. It was found that local minima formed clusters around global ones. We expanded these analyses to cover the relationship between clusters, and found that local minima form many correlated yet distant clusters. In addition, we show the existence of a relationship between the size of the problem and the distance between local minima.

To rule out other possibilities of this success we test several other population based algorithms, and compare their performances to clustering. In addition, we compare with solo-search algorithms. We show that this method is superior to all algorithms tested. CLGH produces results that might be produced by a solo-local search algorithm within 95% less time. However, this is not a standalone technique, and can be incorporated within other algorithms to further enhance their performance.

A further application of clustering is carried out on the Traveling Salesman Problem (TSP) in the discrete domain, and Artificial Neural Networks (ANN) using backpropagation for the purpose of data classification in the continuous domain. Since TSP does not show a clustered landscape configuration we find that ALGH is an effective method for improving search results. Preliminary results are shown indicating that extensions of the proposed algorithm can give similar improvements on these hard optimisation problems.

Contents

List of Figures

List of Tables

Nomenclature

C	A single clause
X	A variable in a clause
V	Variable assignment $V \in 0, 1$
m	the number of clauses
h	A const. in the exponent for the no. of tests required by exhaustive search
k	the number of variables per clause
n	the total number of variables
f	The expression composed of variables and clauses
$S(\cdot)$	the evaluation of the clause, either 1 or 0
$Z(\cdot)$	the inverse of $S(\cdot)$
w	the weight of the clause
ρ, c, σ	the cost of the expression
α	the ratio of clauses to variables
$P(\cdot)$	the probability of choosing a centre point
d	the length of an edge in a tour
π	a particular tour permutation
N	number of cities or nodes in the tour
x, y	an edge in the tour
R	the number of neighbouring nodes
μ	fittest individuals in EDA
$\sim$	A relation of equivalence
ρ	Momentum constant for weight updating
$T(n)$	Number of iterations with respect to n
s	Number of solutions
LGH	Landscape Guided Hopping
CLGH	Clustered Landscape Guided Hopping
ALGH	Averaged Landscape Guided Hopping
EDA	Estimation of Distribution Algorithms
SAT	Satisfiability
MAX-SAT	Maximum Satisfiability
ANN	Artificial Neural Network
UMDA	Univariate Marginal Distribution Algorithms

TSP	Traveling Salesman Problem
NP	Non-Deterministic Polynomial
GA	Genetic Algorithms
ACO	Ant Colony Optimization
QEA	Quantum-Inspired Evolutionary Algorithm
FCL	Fixed Clause Length model
UPI	Unary Prime Implicates
BHC	Basic Hill-Climber
PBIL	Population Based Incremental Learning

Acknowledgements

This part of the thesis has probably been the easiest to write, since I know with absolute certainty the people who have contributed towards the successful completion of my work. For his infinite patience, wisdom and insights over the past three years, I am forever indebted to my supervisor Adam Prügel-Bennett. I cannot remember a time when he would not set aside his work to listen to my half baked ideas and research rants. He carefully guided me into independent thought, and provided me with insights only a person with his knowledge would have had. (Adam, thank you so very much. You are brilliant!).

Also, Bassam Farran, a trustworthy colleague who has not only kept me sane throughout my Ph.D., but his invaluable and brainy discussions turned on many light bulbs over my head. These days will be missed. My heart felt thanks to all the people in the ISIS group. The discussions, fun, and sharing of information are unforgettable.

My parents who have constantly encouraged and motivated me to learn, ask, and think all my life, deserve my utmost honor and appreciation. Thanks to my brothers and sister for being there, and especially in dire times. I could always count on them for their support.

Last, but not in any way least, my wife, my support, my anchor and my centre. She has sacrificed many tens of thousands of pounds she could have earned while she was in Kuwait to be with me and care for my needs. She has. I am downright lost as to what words would best describe my gratitude for her. Notwithstanding, she is the reason I am.

To mom, dad, Aida, and my kids...

Chapter 1

Introduction

Propositional satisfiability is a well known and studied problem. The main body of literature consists of phase transition analyses (Mitchell *et al.*, 1992; Zhang, 2001; Crawford and Auton, 1996), landscape analyses (Parkes, 1997; Zhang *et al.*, 2003; Parkes, 2001b), and methods for solving them (Davis and Putnam, 1960; Crawford and Auton, 1993; Stephan *et al.*, 1996; Marques-Silva and Sakallah, 1999; Zhang *et al.*, 2001). The analysis started with the focus on a single characteristic of the satisfiability problem, i.e., phase transitions, and gradually progressed into more comprehensive landscape analysis. In this thesis, we shed some light on previous studies and attempt to broaden these analyses further. We also propose a novel approach for exploring the landscape of solutions of large scale satisfiability problems more efficiently.

Although our main contribution is a new search method, the landscape analyses are invaluable on their own. They provide a substantially more expansive picture than previously reported. A great effort went into discovering the configurations of solutions through hundreds of hours of empirical tests, and via the statistical analysis of hundreds of thousands of data points. Moreover, we made these analyses more concrete by modeling the data points. As a proof of concept we took advantage of our analysis by introducing a new method of search.

This new method, which we call Landscape Guided Hopping (LGH) capitalises on the structure of the solution space. We observed that local solutions of satisfiability problems group together forming clusters. Within these clusters, better and better solutions tend to gravitate towards the centre of these clusters. The most recent attempts at exploiting these structures looked at the solution space as a single cluster (Zhang *et al.*, 2003). It seems that the analysis carried by previous researchers did not identify these multiple clusters. However, even with one cluster they were able to obtain better results. We show that with the Clustered-Landscape Guided Hopping (CLGH) method we are able to outperform standard local search algorithms and the state of the art by a clear margin most notably on random MAX-3-SAT problems. To show this we performed a

comparison with over 30 different local search algorithms. We have outperformed them all in the quality of the solution and the time it takes to find a better solution. This work has been developed into a Windows based software that was placed on the Internet. It has been downloaded and used many researchers around the world. Not only do we provide the algorithm within this software, but we also developed the fastest GSAT and WALKSAT search algorithms, see sections B.1 B.2 in the Appendix.

To see if LGH applies to other problems it was applied to another NP-Hard Problem. The main problem we analysed was the well-known Traveling Salesman Problem (TSP) (Applegate *et al.*, 2006). Many methods were developed to solve this problem. One that stands out is offered by (Helsgaun, 2006). The goal was not to outperform these methods, but to explore the degree of effectiveness of clustering. The structural analysis performed on maximum satisfiability cannot be directly implemented on this problem. In addition, the implementation details are significantly different between the two. Although it did not appear that these two problems shared the same clustered structure of the satisfiability problem. However, the traveling salesman problem did appear to have a single cluster that can still be utilized to improve results using Averaged-Landscape Guided Hopping (ALGH).

The previously mentioned problems were combinatorial optimization problems. They are discrete in nature. As a final experiment of CLGH, we applied it to a continuous problem. The goal was to reduce the Mean Squared Error (MSE) of an Artificial Neural Network (Haykin, 1999) when classifying data sets by improving the weights of the network. Different methods have been developed to optimise the weights of an artificial neural network. We chose backpropagation (Rumelhart *et al.*, 1986) as a testing ground. Here too, we showed that the weight in some problems can be optimized using CLGH. However, examining one problem in the continuous domain is not sufficient to demonstrate that CLGH works. More problems need to be tested, and further landscape analysis should be performed. Despite this, it is a step in the right direction.

One of the most important aspects of LGH is the application of clustering or averaging after performing several local searches. The stage at which clustering or averaging is applied is crucial to the success of the method. Although the exact stage of application has not determined theoretically, some empirical analyses have been done to show its importance. This point is discussed more thoroughly in the conclusion and is left as an open question for further research.

The structure of the thesis is as follows: in the next chapter we introduce propositional satisfiability, discuss the NP-Complete concept, methods used for solving SAT and MAX-SAT problems, and the structure of SAT and backbones. This work is mostly based on a review of literature. In Chapter 3 we examine the number of flips required to reach local solutions, and the probability of finding local solutions using a local search procedure and complete-neighbourhood search. Moreover, we show the existence of a phase transition

even for local search. We compare these results with the phase transitions obtained by complete methods. The majority of papers were concerned with the effect of exact algorithms on satisfiability problems. Some tested local search at and around the phase transition (Parkes, 2001a, 1997). We shed some light on the relationship between the phase transitions and local search with a slightly different approach. We will show that local search behaves on maximum satisfiability problems in the same way exact algorithms do on satisfiability problems when searching for local optimums.

In Chapter 4 we take the analysis a step further. We perform a comprehensive study of the landscape looking for structures that we can exploit in the search. This is done on relatively small problem sizes. We analyse the relationship between quasi-global solutions, and local and quasi-global solutions. We model the solution landscape, and draw conclusions on its nature. We follow this with the application of the CLGH approach on large problem instances, chapter 5. The method we propose combines hill-climbing and K-means clustering to efficiently explore solutions. We also compare this new approach to several algorithms such as Genetic Algorithms and Univariate Estimation of Distribution Algorithms to show that averaging of solution clusters is very different from these methods. We also compare our approach to state of the art algorithms, and show that our algorithm is superior to them all. In comparison with Backbone guided search, CLGH achieves 14.5% improvements in the results. When CLGH is compared with the top local searchers, CLGH achieves same quality results in less than 95% of the time alloted for the local searcher. Not only does our algorithm outperforms other algorithms, but we will show that clustering of solutions can be used in conjunction with other local searchers to improve their results too.

In Chapter 6 and 7 we go a step further to test how well LGH applies to other problems. We start with the Traveling Salesman Problem as a discrete combinatorial optimization problem, and then move to Artificial Neural Networks in the continuous domain. These experiments will show that although CLGH is not as effective as in the case of Maximum Satisfiability, ALGH does provide a new way of solving these problems. We also discuss possible reasons for clustering in CLGH being not as effective in these cases.

Finally, we conclude and briefly discuss future work that needs to be done to understand how to maximise gains in other problems. This approach is new, and much is left to be studied. It needs further investigation of when and how to apply it with local search. More will also be discussed on the landscape of solutions in an effort to understand if clustering would be a preferred tool over other algorithms.

1.1 Contributions

This work has culminated into three publications:

- Qasem, M. and Prügel-Bennett, A. (2009). Learning the Large-Scale Structure of the MAXSAT Landscape Using Populations. Evolutionary Computation, IEEE Transactions on (In print).
- Qasem, M. and Prügel-Bennett, A. (2009). Improving Performance in Combinatorial Optimisation Using Averaging and Clustering. In EvoCOP, pages 180-191.
- Qasem, M. and Prügel-Bennett, A. Complexity of MAX-SAT Using Stochastic Algorithms, in Proc. Annu. Conf. Genetic Evol. Comput. (GECCO), 2008, pp. 615-616.

Chapter 2

Propositional Satisfiability

Propositional Satisfiability or Boolean Satisfiability is fundamental in solving many problems in the fields of Artificial Intelligence, mathematical logic, and combinatorial optimizations. Some of these areas are directly related to practical problems such as Electronic Design Automation (EDA) (Marques-Silva and A. Sakallah, 2000), which include Automatic Test Pattern Generation (Larrabee, 1992), path delay faults (Chen and Gupta, 1996), Field Programmable Gate Array routing (Gi-Joon *et al.*, 1999), crosstalk noise analysis (Chen and Keutzer, 1999), and functional vector generation (Fallah *et al.*, 1998). These types of problems, such as combinational logic circuits, are first reduced to SAT expressions using algebraic methods, and then a solution is found using different search methods (Marques-Silva and A. Sakallah, 2000).

2.1 The Satisfiability (SAT) Problem

A SAT expression f is composed from a set of m clauses in conjunction, $C_1 \wedge C_2 \wedge ... \wedge C_m$, where each clause C_i is formed by the disjunction of Boolean variables, $X_1, X_2, ..., X_n$ or their negation, where $X_i \in \{0, 1\}$. Here, we use 0 to represent false, and 1 to represent true. This form of the conjunction of clauses and the disjunction of literals (variables and their negation) is called Conjunctive Normal Form (CNF). The SAT problem asks if there exists a truth assignment $\boldsymbol{X}$ such that the expression is true. The vector notation $\boldsymbol{X}$ represents a string of 0 and 1 configurations. If the number of literals in each clause is k then the problem is called k-SAT. When $k \geq 3$, then k-SAT becomes NP-Complete.

Maximum Satisfiability, on the other hand, is a generalization of SAT. Where SAT is a decision problem, MAX-SAT is an optimization one. The goal in MAX-SAT is to find an assignment $\boldsymbol{X}$ such that the number of satisfied clauses is a maximum. MAX-k-SAT is NP-Hard for $k \geq 2$. The MAX-SAT problem is described in Equation 2.1.

$$\sigma = \max_{\boldsymbol{X} \in \{0,1\}^n} \sum_{i=1}^{m} S_i(\boldsymbol{X}) \tag{2.1}$$

Where σ is the cost, and S_i is a 1 if the clause is satisfied, and 0 if the clause is unsatisfied. Another way of looking at this is by expressing the cost of MAX-SAT as the minimum number of unsatisfied clause as in equation 2.2.

$$c = \min_{\boldsymbol{X} \in \{0,1\}^n} \sum_{i=1}^{m} Z_i(\boldsymbol{X}) \tag{2.2}$$

Where c is the cost, Z_i is a 1 if the clause is unsatisfied, and a 0 otherwise. Current literature uses the costs interchangeably in their interpretations of the results. We will allow ourselves the same level of flexibility when necessary.

Weighted MAX-SAT is yet another generalization of MAX-SAT. A weight w_i is associated with each clause in the satisfiability expression. While the goal of MAX-SAT is to find the maximum number of satisfying assignments, the goal in weighted MAX-SAT is to find the maximum sum of the weights for the satisfied clauses (Wah and Yi, 1997), Equation 2.3.

$$\rho = \max_{\boldsymbol{X} \in \{0,1\}^n} \sum_{i=1}^{m} w_i S_i(\boldsymbol{X}) \tag{2.3}$$

2.1.1 Non-Deterministic Polynomial Time Problems

Non-Deterministic Polynomial time (NP) refers to the set of decision problems that can be solved in polynomial time with a non-deterministic algorithm, e.g. see (Garey and Johnson, 1979). The polynomial time that is mentioned in this definition refers to the idea that if a solution was somehow offered, the verification of the solution can be done in polynomial time. Finding an optimal solution to this set of problems requires at the worst case an exhaustive search through the entire solution space using a deterministic algorithm. Unfortunately, as far as we know, to implement a non-deterministic algorithm using a deterministic one requires an exponential time in the worst case. An NP-Complete problem denotes a problem that is of the NP class, and all other problems in its class can be transformed to it in polytime. Also, finding a polynomial solution to one problem means polynomial solutions can be found for all NP-Complete problems.

Non-deterministic polynomial time hard (NP-Hard) problems are the class of problems that are at least as hard as the hardest problem in the NP-Complete class. Since MAX-SAT is not a decision problem it automatically falls outside the NP class. No longer is the problem asking if there exists a solution that satisfies the formula. Instead the

question is, what is the maximum number of satisfiable clauses? Compare a systematic search that traverses the 0 and 1 tree in an effort to solve a SAT problem with one that tries to solve a MAX-SAT problem. In SAT, we can prune the search tree as soon as single clause is not satisfied. Having a single clause unsatisfied in the MAX-SAT search does not necessarily amount to directly pruning the tree. Instead the condition would have to be that the number of unsatisfied clauses found so far is less than the one that was found previously. In practice, this makes MAX-SAT much harder than SAT.

SAT is known to be NP-Complete for $k \geq 3$ (Cook, 1971). When $k \leq 2$, the SAT problem is in P. However, MAX-k-SAT for $k \geq 2$ is NP-Hard. Having SAT in the NP-complete set of problem does not necessarily make all instances of it hard to solve. Assuming $P \neq NP$, it is just that some problems require a superpolynomial number of truth assignment tests to guarantee finding a solution. In other words, the complexity of exhaustive search algorithms that attempt to solve NP-Complete SAT problems require an order $2^{n/h}$ tests, where h is a constant.

The importance of the SAT problem not only stems from its relationship to real world problems, but due to its representational simplicity of NP-Complete problems it has received the attention of many researchers. Historically, it was the first problem proven to be NP-Complete (Cook, 1971).

2.2 Methods for Solving SAT and MAX-SAT

There are two types of approaches that are applied to solving SAT or MAX-SAT problems, and they fall into two categories: complete methods that find optimum solutions mainly via depth first search, and incomplete methods that find optimal or near optimal solutions mainly through local search methods. Both complete and incomplete methods use exact search methods (these that do not incorporate probabilistic heuristics, and which include some form of a systematic search algorithm), probabilistic heuristics or a combination of both. A list of solvers have been included in the a survey in (Gomes *et al.*, 2008). We will discuss some of these in here.

A fundamental complete procedure that guarantees finding solutions to SAT or MAX-SAT problems is the Davis-Putnam-Logemann-Loveland (DPLL) algorithm (Martin *et al.*, 1962). The method is also known as a modified form of Backtracking algorithm. In backtracking a search tree comprising of all combinations of the assignments of $\boldsymbol{X}$ is traversed. The individual values X_i of $\boldsymbol{X}$ are either assigned values of $X \in \{0, 1\}$, or they are unassigned. The initial cost of the expression f is considered only with the full assignment of $\boldsymbol{X}$. Once the first cost c_t has been determined using a full depth search, a chronological backtrack is performed, the subsequent cost is computed at every node, and the search tree is pruned if the new cost c_{t+1} is greater than the lowest cost found so far, c_t. This is how branch and bound is applied to MAX-SAT. With SAT, the tree

is pruned as soon as a clause is unsatisfied since the goal is to find $\boldsymbol{X}$ that satisfies f. DPLL, on the other hand, systematically searches every possible Boolean combination of $\boldsymbol{X}$ through backtracking while reducing the CNF expression by a process of elimination of unit clauses. Unit clauses are clauses that have all their variables assigned values except one. In MAX-SAT, unit clause elimination is no longer applicable making the problem considerably more difficult. The DPL procedure was introduced in 1960, and it was later modified, DPLL, in 1962 (Martin *et al.*, 1962; Davis and Putnam, 1960). To date, it is used as the foundation for the most effective complete algorithms.

Although DPLL is a clever method for finding solutions to satisfiability problems, its systematic traversal of the NP problem renders it ineffective with large problems. Many methods have been designed to incorporate a modified version of DPLL as part of their search engines, and, in fact, have been turned into applications that are used in the Electronic Design Automation (EDA) industry. Some such methods include Tableau (Crawford and Auton, 1993), TEGUS (Stephan *et al.*, 1996), BCP (Zabih and McAllester, 1988), GRASP (Marques-Silva and Sakallah, 1999), and zChaff (Zhang *et al.*, 2001).

Hybrid methods that use both exact and probabilistic methods to achieve complete solutions are abundant. A simple form takes a probabilistic procedure as a precursor to DPLL, e.g., the Two-Phased Exact algorithm (Borchers and Furman, 1999). In this method a lower bound to the number of satisfied clauses is found using GSAT (Selman *et al.*, 1992), a probabilistic procedure, in the first phase. In the second phase, DPLL is applied to find costs larger than the lower bound. Another more complicated method relies on three different procedures: unit propagation (as in DPLL) based on nonlinear integer programming, look ahead based on linear programming (LP) to estimate the largest number of satisfiable clauses and dynamic weight ordering (Zhao and Zhang, 2004).

Incomplete methods do not guarantee optimality. They rely on heuristic procedures to search for solutions in satisfiability problems. Although they are categorized as local searches, analysis of their effectiveness on small problems, i.e., problems with a relatively small number of variables, have shown that they almost always converge to global maxima rapidly (Selman *et al.*, 1994; Parkes and Walser, 1996; Gent and Walsh, 1993; Zhang *et al.*, 2003; Selman and Kautz, 1993).

The first inception of a local search for random 3-SAT came in the form of a greedy hill-climb[1] search called GSAT (Selman *et al.*, 1992). GSAT starts with a random assignment of $\boldsymbol{X}$, and changes the Boolean assignment of a variable X_i that yields the most number of satisfied clauses. This process is repeated a set number of times. In SAT, if a satisfying assignment is found, the procedure is stopped, otherwise it is restarted with another initial random assignment. However, in MAX-SAT the procedure is restarted an

[1]Most literature looks at SAT or MAX-SAT problems in terms of the minimum number of unsatisfied clauses as apposed to the maximum number of satisfied clauses. A hill-climber algorithm should more appropriately be called descent in this case.

unspecified number of times while keeping track of the best local optimum[2]. This greedy approach has been the building block for lots of other incomplete methods. Methods such as WALKSAT (Papadimitriou, 1991), WSat/G, WSat/SKC (Parkes and Walser, 1996), UnitWalk (Hirsch and Kojevnikov, 2005), BGWALKSAT (Zhang *et al.*, 2003), and Novelty (McAllester *et al.*, 1997) are examples of methods based on it. Variations of this greedy approach include random walks, unit clause elimination, and variable greediness.

Another stochastic local search approach that is quasi-greedy, yet not related to previously mentioned methods, is Simulated Annealing (SA) (Kirkpatrick *et al.*, 1983). SA is a modification of a hill-climber that allows the searcher to occasionally make a move which decreases the number of satisfied clauses. This allows the searcher to escape from local minima in which a hill-climber would normally get stuck. The probability of making such a move is controlled by the "annealing temperature" which is typically reduced over time. Consequently, SA initially behaves similarly to a random walk, allowing most moves, but over time reduces the probability of making moves that maximizes the cost until it resembles a hill-climber.

Another powerful method for finding solutions to satisfiability problems is Tabu search (TS). This method is based on two main concepts: adaptive memory and responsive exploration (Glover and Laguna, 2002). In contrast to memoryless systems such as hill-climbing algorithms or simulated annealing, TS uses memory to prevent the search from moving to a region that has already been explored. This, again, prevents the search algorithm from getting trapped in local minima. It differs from Backtracking also by adaptively storing information as opposed to inflexibly branching through all possibilities. Reactive Search (RS) method (Battiti and Protasi, 1997), and Iterated Robust TABU Search (IRoTS) (Smyth *et al.*, 2003), are two examples of TS, both are used for solving MAX-SAT problem.

2.3 Population Based Algorithms

The methods mentioned earlier are solo-search algorithms. Another category of search algorithms is based on populations. These include Genetic Algorithms (De Jong and Spears, 1989; Boughaci *et al.*, 2004), Particle Swarm Optimisation, Ant Colony Optimisation (Villagra and Barán, 2007), and Quantum Evolutionary Algorithms (Layeb and Saidouni, 2008; Xiaoyue *et al.*, 2008). All of these methods were applied to the satisfiability problem with limited success. Usually, they are applied to small problems, and even then they have not been shown to outperform solo-search algorithms on the

[2]The term "local optimum" refers to an assignment that is obtained by the search in a particular locality, and that this assignment cannot be improved upon by the search method any further. However, the "best local optimum" is the best cost local optimum that was obtained through several searches starting from different initial points and ending at different localities.

whole. In fact, the program that has consistently outperformed (or at least has been on equal footing with) all population search algorithms is WALKSAT. We believe that many researchers have abandoned this genre of research prematurely, because they have not achieved substantial success against even the most basic of the solo-search algorithms. This explains the dearth of published papers on population based algorithms with regards to satisfiability.

Genetic algorithms (GA) have been applied to find optimal solutions in satisfiability problems. Due to the simple binary representation that satisfiability problems possess, it makes it easy for researchers to apply GA for solving them. A GA algorithm can be applied to the problem via direct binary string representation of the assignment $\boldsymbol{X}$ with the normal crossover and mutation operators (De Jong and Spears, 1989). Others combine hill-climbing with GA to avoid having GAs prematurely converge to a local minima (Boughaci *et al.*, 2004). The procedure alternates between crossover in the population and hill-climbing over the members of the population.

Ant Colony Optimisation (ACO) was applied to MAX-SAT problems by (Villagra and Barán, 2007). This heuristic is inspired by the foraging behaviour of ant colonies. The ant or agents iteratively construct candidate solutions through the guidance of pheromone trails. With satisfiability problems, ACO cannot be directly implemented in its native form. Here, ants look for a minimum cost path in which they lay their pheromones. Appropriately, the problem is coded as fully connected construction graph where the variables and their assignments, $\langle X_i, V_i \rangle$, are the nodes, and the edges connect the variables that are in the clause. The desired optimum path is of length $m = \{\langle X_1, V_1 \rangle, \langle X_2, V_2 \rangle, ..., \langle X_n, V_n \rangle\}$ such that $\langle X_i, V_i \rangle \in m$ and $\langle X_i, V_j \rangle \notin m$ where $V_i \neq V_j$. Each ant constructs a path or model, and sets a pheromone. The pheromone evaporation procedure is simulated according to ACO rules. The results reported show that ACO is outperformed by WALKSAT in some problems, and is competitive in other problems. The authors of this paper claim that there has not been a successful ACO algorithm for MAX-SAT problem prior to their work.

A more recent method is based on the Quantum Mechanics is called, Quantum-Inspired Evolutionary Algorithm (QEA). It is based on quantum bits (qubits) and the superposition of states. This method is also an evolutionary based algorithm, and requires the processing of a number of quantum states in parallel. This algorithm was originally developed by Kuk-Hyun Han and jong-Hwan Kim (Kuk-Hyun and Jong-Hwan, 2002), and because it directly maps to satisfiability assignment it has been used for solving MAX-3-SAT using QSAT (Layeb and Saidouni, 2008). In QSAT, QEA was directly applied to the problem with the addition of a basic local search algorithm. The results reported provided very little improvement if any over GSAT even on problems with a maximum of 200 variables and 400 clauses. Another more enhanced version of QEA is the Improved QSAT (IQSAT) (Xiaoyue *et al.*, 2008). It was applied to 3-SAT. The authors cited the Benchmark satisfiability problems obtained from SATLIB (Hoos and

Stützle, 2007) for their tests, but there was no mention of the size of the problems used. Regardless, the largest benchmark problem they could have used would not have exceed 250 variables with 1065 clauses. Their results show that WALKSAT performed either equally as well or better.

One dominant feature of the majority of these population based algorithms is that they incorporated a form of solo-search algorithm to enhance the search. On their own they produce weak results. The main reason for employing evolutionary algorithms is because they provide a way to explore the search space, while the solo-search algorithm exploit outcomes. Doing away with solo-search algorithms makes the search impractical. We have done some experiments on GA and Estimation of distribution in sections 5.4 and 5.2. We will show that unless these algorithms are hybridised with a local-search method they become inefficient. Without local-search, the search becomes slow. This is mainly due to the population size, and hence would easily lose the race.

2.4 Problem Sizes

In the majority of the previous search techniques, the problems examined were small. Early research in this area was limited by less powerful computers with limited capacity. Performing analyses on large scale problems was not feasible then. However, with more powerful computers nowadays these obstacles should not exist. Despite the increase of computing power, the problems examined remain small. This is due to the newer methods relying on population based algorithms that still require a great of deal of computational power. Having a multiple number of large assignments, and applying the search operators on each assignment still requires more capacity and more computational power. In comparison, solo-search algorithms do not require as much power and capacity. As a consequence, comparisons can only be drawn between these types of algorithms on smaller problems.

Another problem we believe why researchers still perform tests on small problems is because most researchers still use the SATLIB benchmark library. It contains small random problem instances. It has not been updated since 11/8/2000. This library should be updated with larger problems to drive researchers into developing more sophisticated algorithms. The only experiments done on large scale problems was done on the Backbone-Guided algorithm (discussed later). In our research, we worked on large MAX-SAT problems, and we developed our own WinSATS application (talked about in section 5.9). It can generate very large problems. We have used those problems to test our novel approach. The problems tested here are much larger than what is reported in current literature. We will discuss these results and Zhang's results in Chapter 5.

2.5 The Structure of Satisfiability Problems

The focus of both complete and incomplete algorithms has been on two main issues. First, since satisfiability is an NP-Complete problem—hence time complexity for complete algorithms grows super-polynomially with the problem size—most methods concentrated on finding solutions rapidly. This is especially necessary in real world applications with thousands of variables and hundreds of thousands of clauses. Second, most of the literature investigated the structure of satisfiability problems to determine the most difficult regions. This provided scientists with an understanding of the relationship between the ratio of the number of variables to the number of clauses, and their effect on the complexity of the problem.

2.5.1 Generating Hard Satisfiability Problems

Studying the structure of satisfiability problems required the generation of random instances of CNF formulas for benchmarking purposes. The best known method for generating hard satisfiability problems is known as the *Fixed Clause Length* (FCL) model (Mitchell *et al.*, 1992). Prior to the introduction of the FCL model, the random CNF formulas that were generated were shown to be typically easy to solve (Mitchell *et al.*, 1992; Selman *et al.*, 1992). In contrast, the FCL model was capable of generating problems that are hard to solve using complete and incomplete methods if a certain criterion was met. In addition, FCL generated problems that are claimed to be representative of real world problems (Selman, 1995).

The generation of hard random k-SAT or MAX-k-SAT formulas using FCL is straightforward:

- Generate m clauses by randomly selecting k different variables from n variables for each clause.
- During the selection of the k variables, negate each variable with a probability of 0.5.
- Discard any duplicate clauses.

2.5.2 The Phase Transition

The phase transition is defined as the transition of the complexity of the satisfiability problem from easy to hard and then to easy again for SAT problems or from easy to hard for MAX-SAT problems. One caveat should be kept in mind with regards to this definition: all literature reviewed for this report has analyzed the phase transition using

depth first search algorithms. Applications of local search methods have been directed to solving problems at and around the transition point. In addition, some have tested for the existence of the phase transition (Parkes, 2001a, 1997). We will later extend this definition to include local search heuristics on local optimums.

The phase transition in most SAT papers has been associated with the point where 50% of the randomly generated formulas are satisfiable. At this point the ratio of the number of variables to number of clauses is $\alpha = m/n$ is 4.3. This was empirically shown by Mitchell and Selman (Mitchell *et al.*, 1992). Using emprical analysis they have demonstrated that SAT problems follow an easy-hard-easy transition about that point. Zhang, on the other hand, showed that with respect to MAX-SAT the problem follows an easy-hard transition (Zhang, 2001). A more comprehensive study of the phase transition was carried by Crawford and Auton (Crawford and Auton, 1996). Using millions of instances of CNF formulas they have found that the transition point asymptotically reaches 4.258.

There are two important points that have to be observed with respect to the transition phase. One, the term phase-transition denotes an abrupt change in the properties of a system. The transition phase in most SAT literature, however, has been associated with the 50% point where the randomly generated problems gradually progress from mostly satisfiable to mostly unsatisfiable, i.e., there is no immediate change. Yet the phase transition analogy has been applied to the satisfiability problem because it does appear to have a phase transition (in the abrupt sense) when the problem size becomes large enough. Second, the phase-transition of the hardness of the problem is associated with the 50% point, and that might not always be true. This association has been shown by empirical results, and might just be coincidental.

It should be noted that the easy-hard-easy or easy-hard transitions refer to the complexity of the problem. Researchers either considered the time by which an algorithm takes to solve problems or the number of bit flips required to reach a solution. It is at the transition point, the critically constrained region, that the problem becomes computationally prohibitive (Zhang *et al.*, 2003).

2.5.3 Backbone Structures

There are three separate definitions of backbones for SAT and MAX-SAT (Zhang *et al.*, 2003; Zhang, 2001; Kilby *et al.*, 2005; Parkes, 1997), but a more comprehensive definition is found in (Prugel-Bennett, 2007): A backbone is defined as the set of variables that remain fixed in all globally optimal solutions. Changing the value of one backbone variable will not allow for finding an optimal solution (Zhang, 2001). Parkes (Parkes, 1997) refers to the set of frozen/backbone variables as the Unary Prime Implicates (UPIs). These UPIs are the set of literals which are logically entailed by a satisfiability

expression such that every solution of the expression makes these literals true.

Parkes has shown that, slightly below the phase transition, as the number of variables increase, the percentage of the problems having UPIs decrease . The number of UPIs also increase abruptly at a certain point when α is increased. Furthermore, the empirical tests showed that the performance of WSAT is affected by the size of the backbone. Zhang (Zhang, 2001) showed that the size of the backbones increases abruptly around $\alpha = 3.6$, and he also showed that the transition from a structure with almost no backbone to one with a backbone correlates closely with the phase transition. The reason no backbone structure is found below the critically constrained region is because of the diverse number of optimal solutions found in such problems that diversify the configurations of the solution assignments. As α gets larger, and becomes over-constrained the number of solutions decrease, and the backbone becomes larger.

Zhang found that the number of optimal solutions decreased as α increased (Zhang, 2001). Then he related this to backbones, and he made the assumption that these few optimal solution are clustered in a small region. In our experiments (Chapter 4) we have shown that Zhang's assumption is not entirely true. Although a great majority of optimal solutions are clustered, there still exist optimal solutions that are further apart in Hamming distance, and hence have different backbone arrangements.

Although finding a backbone structure is NP-Complete as proven by (Kilby *et al.*, 2005), using a *pseudo-backbone* as an estimation of a *true-backbone* was valuable in solving over-constrained problems (Zhang *et al.*, 2003). The pseudo-backbone was determined by statistical analysis of many runs performed by a method based on WALKSAT called BGWALKSAT.

A recent publication (Prugel-Bennett, 2007) contrasts backbones with a new, related concept of Critical Backbones. A critical backbone is a subset of a backbone that lies within the basin of attraction of the global solution. Once a set of critical variables are found, finding the optimal solution becomes easy. This definition differentiates itself from the definition of backbones which are independent of the neighbouring structure. Using a toy problem and a hybrid-GA algorithm it was shown that the critical backbone served as a vessel for faster convergence towards the global optimum. The Hybrid-GA consisted of a hill-climbing algorithm coupled with selection and crossover.

Chapter 3

Phase Transition Analysis

The majority of the algorithms mentioned previously exploit hidden structures in SAT or MAX-SAT problems. Researchers gathered empirical information, analyzed, modeled, and improved their search procedures to obtain better results. Our work extends this complexity analysis to more difficult MAX-SAT problems. These problems consist of many variables and have a high clause to variable ratio. As a consequence of having many variables complete methods could not be used for collecting statistical data. They are not efficient enough to study large problems in a reasonable amount of time. Therefore, we used local search methods for our tests.

3.1 Local Search and Phase Transitions

The time complexity and distributions of SAT and MAX-SAT problems have been thoroughly studied using complete or systematic procedures (Monasson *et al.*, 1999; Mitchell *et al.*, 1992; Selman, 1995; Zhang, 2001; Crawford and Auton, 1996). However, the majority of phase transition experiments have been carried out using depth first or exhaustive search procedures. A natural question to ask is whether this phase transition exists for stochastic methods? Are phase transitions an inherent property of the structure of SAT and MAX-SAT problems or are they influenced by the type of depth first search algorithms as stated in (Monasson *et al.*, 1999)? We investigate the effect of both the number of variables and the phase transition on the performance of a simple stochastic procedure.

An important study by Parkes (Parkes, 2001a) relating the complexity of WSAT, WALKSAT, and WSAT(PAR), Parallel WALKSAT, before the phase transition of random 3-SAT problems. It has been shown that sequential WSAT solves problems in $O(n)$ time for $\alpha \leq 3.8$. It also shows that WSAT(PAR) requires $O(\log(n)^2)$ on average to solve instances at and below this region. We will attempt to look at and around the phase transition in search for local optimums using a simple hill-climber algorithm.

3.2 Local Search Procedure

To study the complexity of MAX-SAT using stochastic algorithms a simple *descent* (or *hill-climb*) algorithm was used to find local minima, and to guarantee that a local minima is reached an *complete-neighbourhood* search was implemented. The complete-neighbourhood search is used to check if there exists a local optimum around a particular assignment on a plateau. Although the search is exhaustive for points on the plateau, the search is not a complete one in that it does not exhaustively search every possible solution there is in the entire search space. It does not rise above the plateau increasing the cost. Figures A.1 and A.2 in Appendix A show these two algorithms.

In the descent algorithm, a randomly chosen bit in the bit assignment $\boldsymbol{X}$ with cost c is flipped yielding $\boldsymbol{X}'$, and if the cost c' (i.e., the number of unsatisfied clauses) is less than or equal to c then $\boldsymbol{X}'$ is kept. The descent is run until it finds the first assignment that has a cost c' which is less than or equal to c. After descent finds a better solution, the complete-neighbourhood search is run on the resultant $\boldsymbol{X}'$ until either a better or equal solution is found. If a better solution is found, then descent is run again until it finds this next cost that was found by the complete-neighbourhood search. Both descent and the complete-neighbourhood search are continuously run until the cost cannot be improved further. This ensures that descent reaches a local minimum. In effect, the complete-neighbourhood search acts as a explorer of solutions giving the hill-climber a hint to resume or stop the search.

The descent algorithm used in our analysis is simple in that it capitalizes on both equal and better costs. Other algorithms such as GSAT, WALKSAT, WSAT/G, WSAT/SKC, UNITWALKSAT are known to perform better on SAT problems, but due their very greedy design and their tendency to get stuck in local minima very quickly we avoided them in our tests (Although, most methods implemented a random walk to avoid being stuck in local minima). Furthermore, most other algorithms traverse the solution space from the point of view of the unsatisfied clauses rather than the assignment $\boldsymbol{X}$. They choose to flip the bit assignment of a variable that is found in an unsatisfied clause. In this simple descent we chose not to bias our bit flips on unsatisfied clauses. The only bias was if the bit-flip resulted in an equal or better cost.

Also, in most other stochastic algorithms the maximum number of flips is set to a fixed value. If an optimal solution was not reached before exhausting the maximum flips, $\boldsymbol{X}$ was reset with a new random assignment, and the process is restarted again. Our goal was to find the number of flips necessary to reach a local optimum, and to guarantee that a local optimum was reached a complete-neighbourhood search was used to ensure that. The decision to stop or continue is determined by the availability of solutions in the locality.

The complete-neighbourhood search behaves the same way as descent except that it

systematically searches through the entire neighbourhood of $\boldsymbol{X}'$ that was obtained from the hill-climber for better solutions. Neighbours are considered at a Hamming distance of 1 from each other. The complete-neighbourhood search starts with $\boldsymbol{X}'$, flips each bit in $\boldsymbol{X}'$, and computes the cost c'. If the cost c' is equal to c, then the new $\boldsymbol{X}'$ is stored into a stack so that the costs of all of its nearest neighbours are searched later. Every time $\boldsymbol{X}'$ is popped from the stack it is stored in to a hash table. This is done so that this permutation of $\boldsymbol{X}'$ is not revisited. If at any point c' was less than c, then the cost c' is returned so that the descent procedure can look for that particular solution or any other solution that is equal or better. If, on the other hand, a better solution cannot be found by the complete-neighbourhood search then $\boldsymbol{X}'$ is a local minimum. An illustration of the complete-neighbourhood search is found in Figure 3.1.

The reason for having the complete-neighbourhood search is because hill-climbing as described does not "know" if a local optimum is reached. Local search performs a blind random flip of a bit which either decreases cost or retains it. The hill-climber does not look a head to determine if there is a local optimum in the neighbourhood of solutions. This investigation is done by the complete-neighbourhood search. It determines whether there exists a better cost in the neighbourhood, and allows the hill-climber to look for it. Once all possible neighbours of good solutions are exhausted, the hill-climber is notified to stop.

3.3 Experimental Results

A great deal of the effort in analysing stochastic algorithms has been focused on SAT problems (Selman *et al.*, 1994; Parkes and Walser, 1996; Gent and Walsh, 1993; Zhang *et al.*, 2003; Selman and Kautz, 1993), and most of this work, although empirical in nature, has been either geared towards rapidly finding optimal solutions, or they have been limited in their scope to analysis around the phase transition of $\alpha = 4.3$. This was natural since the goal was to solve SAT problems efficiently, and the most troublesome region for complete algorithms is located around $\alpha = 4.3$. As a remedy for the phase transition problem, faster stochastic algorithms have been applied to discover solutions. We performed several experiments on random 3-SAT problems to study the phase transition in relation to hill-climbing.

The best study available devoted to understanding the effects of the phase transition on local search was done by (Parkes, 2002). The easy-hard-easy transitions of 3-SAT problems were determined using WalkSAT. In the easy-hard region a sequential WalkSAT was applied. The number of flips were determined for fully satisfiable instances. In the second easy region a parallel WalkSAT was applied with a target number of satisfied clauses. Parkes found that below the threshold $\alpha \approx 3.1$ the number of flips grow linearly, beyond this $\alpha \approx 3.1$ the flips grow super-polynomially. We aim to perform a similar

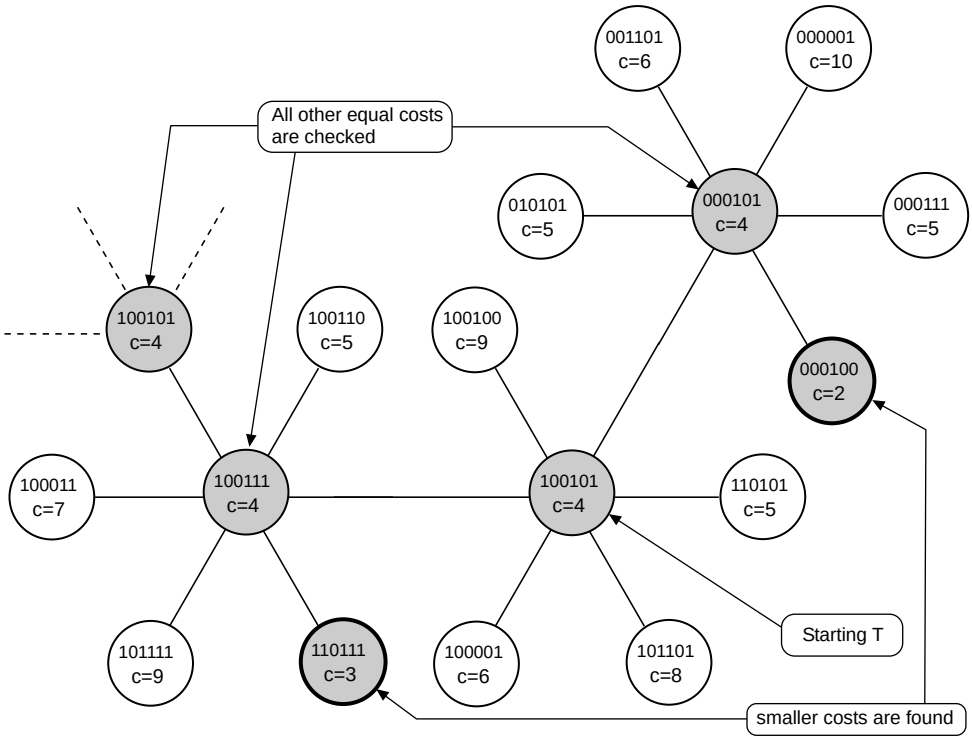

FIGURE 3.1: Complete-neighbourhood search example. All neighbours with a Hamming distance of 1 and cost of c are searched entirely. Starting from 100101, the two neighbours 100111 and 000101 would be put on the stack. The stack would be popped and the neighbour 000101 would be examined. Since one of the nearest neighbours was a lower cost then the search would be halted. Although two lower costs are shown in this illustration, the search is halted as soon as one of them is found.

tests with a slight difference. The number of flips will be determined for a range of clause to variable ratios without a target value. The stopping criteria for the search is if the local optimum was reached. This will be explained further next.

Using the FCL method, random formulas were generated for our analysis. We generated 20 random problem sets for each of the number of variables, starting from 20 to 150 variables in increments of 10. We also varied the number of clauses for each of the variable increments such that the ratio $\alpha = m/n$ was between 2.0 and 10.0 in increments of 1. Each problem set was searched for local minima with 1 000 different initial starting points, and the local minima were confirmed using the complete-neighbourhood search. In total, the experiment was performed on 2520 random formulas where the hill-climb and complete-neighbourhood searches were applied to each formula 1 000 times. Although this number of random instances appears to be small, yet performing the complete-neighbourhood search proved to be daunting both in CPU time and memory requirements (this will be explained later). Another test was run for 20, 30, and 40 variables. These tests ranged from $\alpha = 2$ to $\alpha = 10$ in increments of 0.1. The number of problem instances used in this experiment was 24 000.

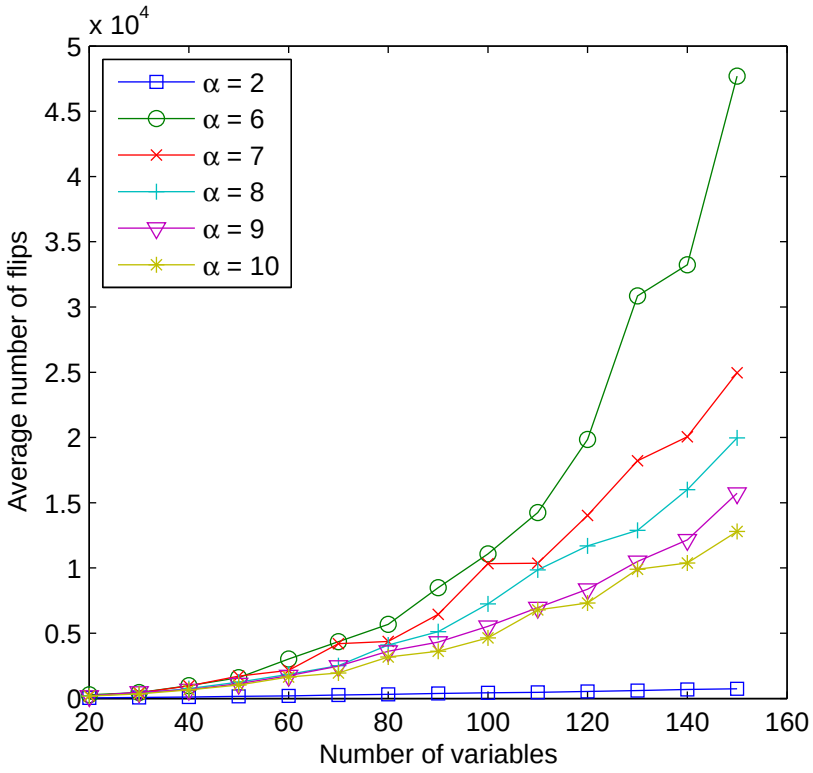

FIGURE 3.2: The average number of flips required to reach a local optimum. Note that the order of the lines is not regular.

3.4 Polynomial Number of Flips

Our first tests focused on the number of flips required to reach a locally optimum solution. As mentioned, the local solution that is found in every descent is guaranteed to be the best cost available in that region of search. Figure 3.2 shows the average and Figure 3.3 median number of flips necessary to reach a local minimum. For each of the results the number of variables was varied from 20 to 150, and the test was executed for different values of α. The values of α that are not shown in Figure 3.3 and 3.2 are $\alpha = 3$, 4 and 5. For these values, we were able to generate data for up to 90 variables. Beyond this point the memory requirements grew enormously despite the small amount of flips that reached the final solution. Since complete-neighbourhood search was not able to verify the best cost local minimum we omitted these results from these plots. However, we will visit these partial results later.

There is a very clear distinction in the way time complexity of complete and incomplete stochastic heuristics behave with regards to the problem, i.e., the former is non-polynomial, and the latter is polynomial in complexity in certain regions. Figures 3.2 and 3.3 show the growth of the number of flips with respect to the number of variables. It is clear that to reach a local minimum only a polynomial number of flips was required. In plotting a log-log graph of the averages and the medians, Figure 3.4 and 3.5, we de-

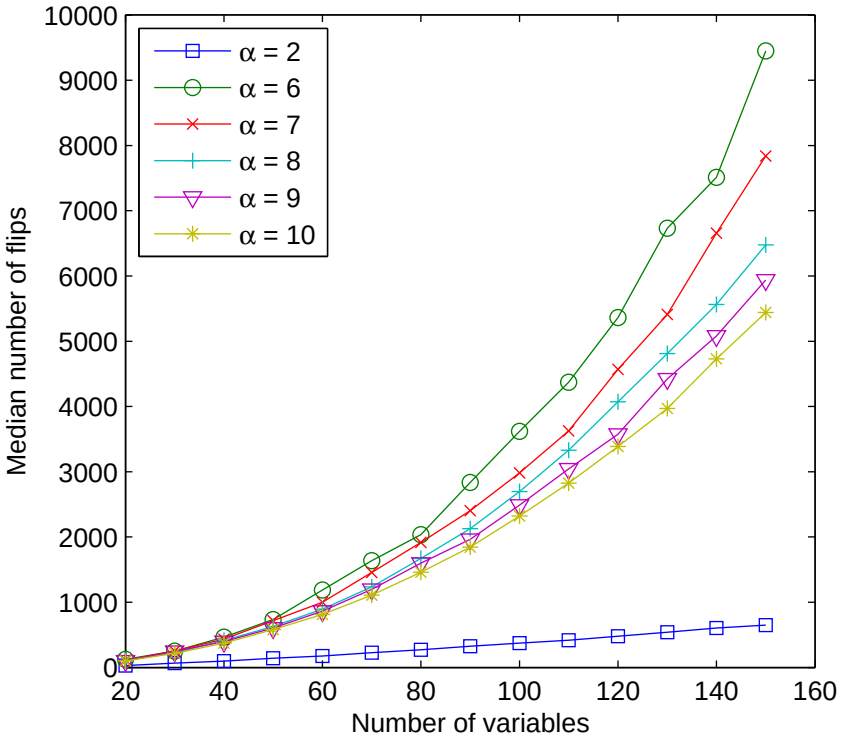

FIGURE 3.3: The median number of flips required to reach a local optimum. Due to the effect of the few very large flips on the averages, the median was plotted as well.

termined the degree of the polynomial to be approximately 2.64 for the average and approximately 2.18 for the median. This is an important result since the complexity of all depth-first search methods are equivalent in that they require $2^{n/h}$ where h is a constant.

It might not be reasonable to compare two results that are different in that one, the complete procedure finds an optimal solution, while the other finds a local solution[1]. This is true, and to give more meaning to these results, another measure is required. This measure is the probability of reaching the best local optimum given a polynomial search.

3.5 The Probability of Reaching Best Local Optimum

As a measure of confidence to determine how well a polynomial number of flips does with regards to the best local cost obtained, we have determined the probability of reaching such a solution with respect to the number of the variables. For 1 000 runs per problem

[1]We have performed tests on problems with up to 50 variable for different values of α using both hill-climbing, and the branch-and-bound algorithm. In every instance we have tested we found that the hill-climbing converged to global optima. Not only did we perform tests on random formulas generated by FCL, but we also tested a great deal of the SATLIB benchmarking problem instances.

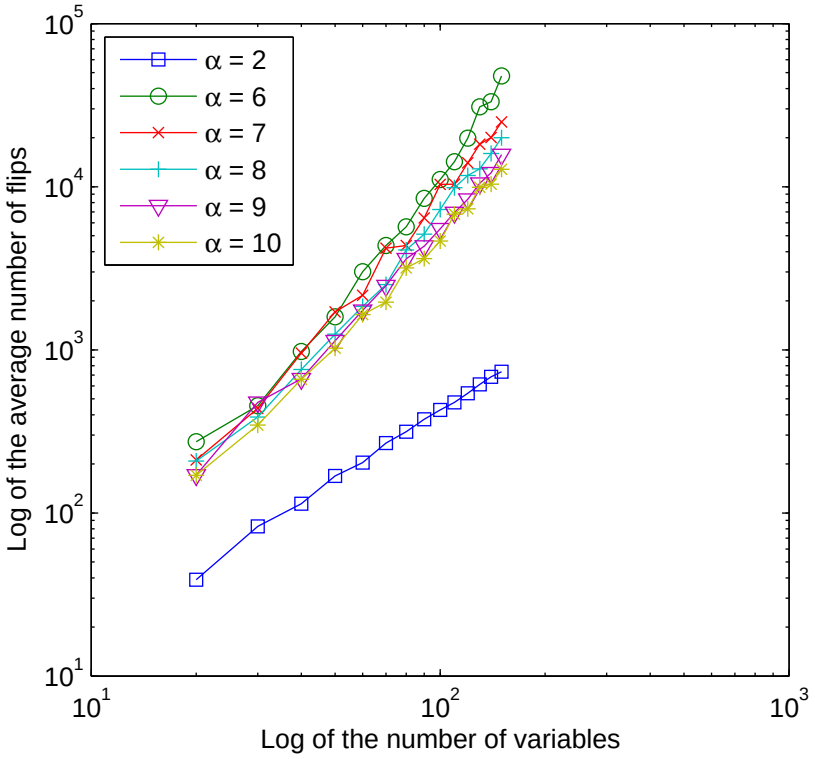

FIGURE 3.4: A log-log graph of the average number of flips required to reach a local solution.

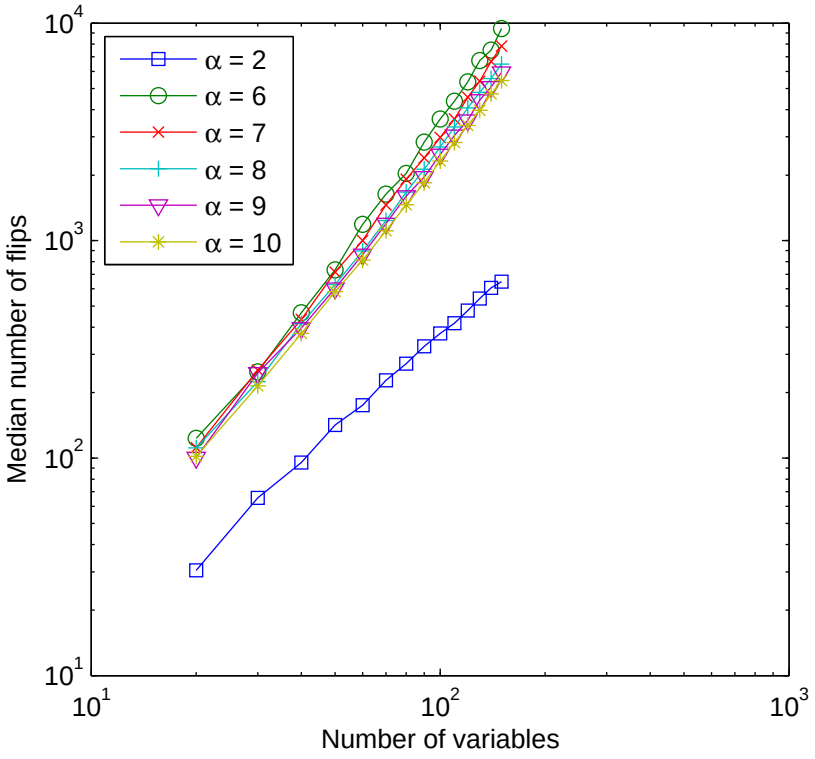

FIGURE 3.5: A log-log graph of median number of flips required to reach a local solution.

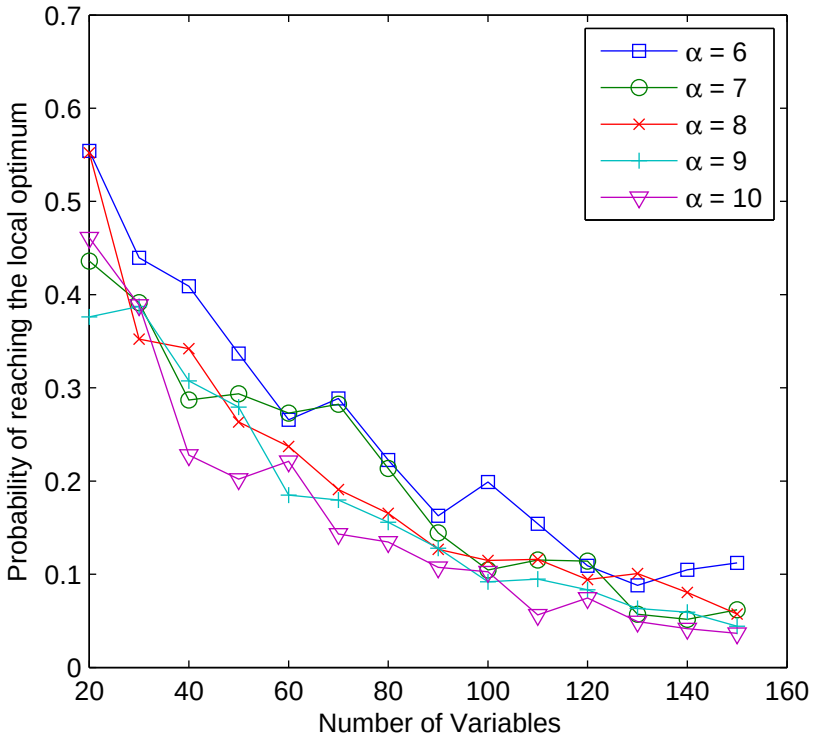

FIGURE 3.6: Probability of reaching the local optimum with respect to the number of variables.

instance, we acquired the best local solution, and calculated the probability of finding such minimum. The results of our experiments are shown graphically in Figure 3.6.

The graph shows a downward trend from a probability of 0.5 on average to a small probability of around 0.1. It is clear that the probability of getting to the best local solution decreases as the number variables increase; in problems with a small number of variable solutions can be found with a high degree of certainty. As the number of variables increases the best solution becomes harder to find. Notice that for different values of α these probabilities do not differ much. The graph shows a narrow band of probabilities for the different clause to variable ratio. The effect the number variables has on the probability is discouraging since it suggests that hill-climbing becomes less effective as the size of the problem grows via the number of variables. Yet in terms of the ratio α hill-climb does appear to perform very well regardless of how large α gets. This is true for large α's beyond the phase transition, but, as we will see, around the phase transition, descent will be affected in the same way as complete algorithms. Is this result describing the difficulty descent faces in locating better solutions, or does the structure of formulas change with the size of the problem such that the best solutions become scarce?

3.6 The Phase Transition and Local Search

In analyzing different clause to variable ratios, $\alpha = m/n$, it has been empirically established that at approximately $\alpha = 4.3$, the random formula becomes hard specially for complete procedures (Mitchell *et al.*, 1992; Selman, 1995; Zhang, 2001). What effect does the phase transition have on stochastic procedures? Are stochastic methods less susceptible to these regions, and therefore reliable in finding better solutions?

By looking back at Figures 3.2 and 3.3 we see that going back from $\alpha = 10$ to $\alpha = 6$ the complexity of the local search increases, and then returns to a low complexity when $\alpha = 2$. Also, the complexity around the phase transition becomes more pronounced as the number of the variables increase. The plots of the average number of flips and the median against the clause/variable ratio are illustrated in Figures 3.7 and 3.8. It is evident that around the phase transition the average and median number of flip increase, and even more so when the number of variables becomes larger. This illustrates the difficulty hill-climbing faces as the number of variables increase around the transition phase.

Unfortunately, we were not able to obtain data points for assignments of size 100 to 150 variables at $\alpha = 5$; for the value of $\alpha = 5$ the maximum we were able to reach was 90 variables. This is shown as a discontinuity in the curves at the α values of 4 and in some cases 5. Even though the average number of necessary flips were not very high (50 000 flips on average for 150 variables), yet the intense memory consumption by the complete-neighbourhood search limited the acquisition of more data points. The memory was depleted because the program saved each assignment that was tested. Saving the assignments was done for two reasons. The first is to make sure that the assignment is not visited again in cyclical fashion (through another assignment). The second is to allow the program to go through all the neighbours of each assignment. Imagine the same procedure being carried out using recursion. Except in this case we were using hash tables to do the tracking.

The reason for the severe memory constraint is because of the large landscape of nearest neighbours of equal costs. Mitchell (Mitchell *et al.*, 1992) and Zhang (Zhang, 2001) define the phase transition at the point where 50% of problems are satisfiable. Below this point, the probability of having a problem satisfiable is highly likely, and in the instances where a problem is fully satisfiable our local search stops on 0 unsatisfied clauses without further investigating the region with complete-neighbourhood search. However, in those problem instances that are not satisfiable, the number of local solutions of equal costs (usually one unsatisfiable clause) is prodigious. Since the complete-neighbourhood search attempts to find better solutions it stacks enormous number of solutions that eventually exhaust the memory.

Since we were able to perform tests for problems with 20, 30 or 40 literals we conducted

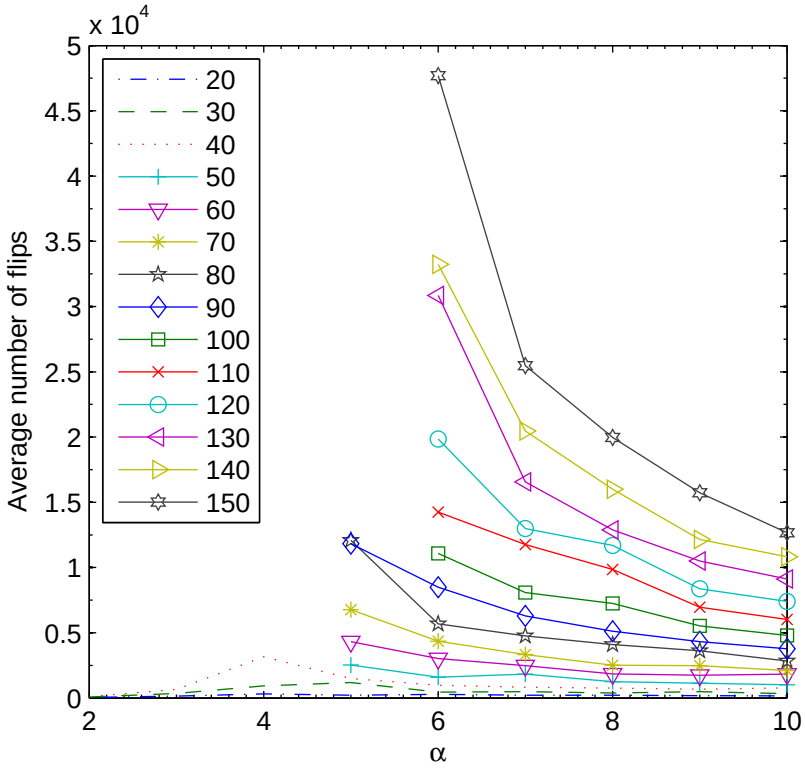

FIGURE 3.7: The average number of flips required to reach a local solution with respect to the α ratio.

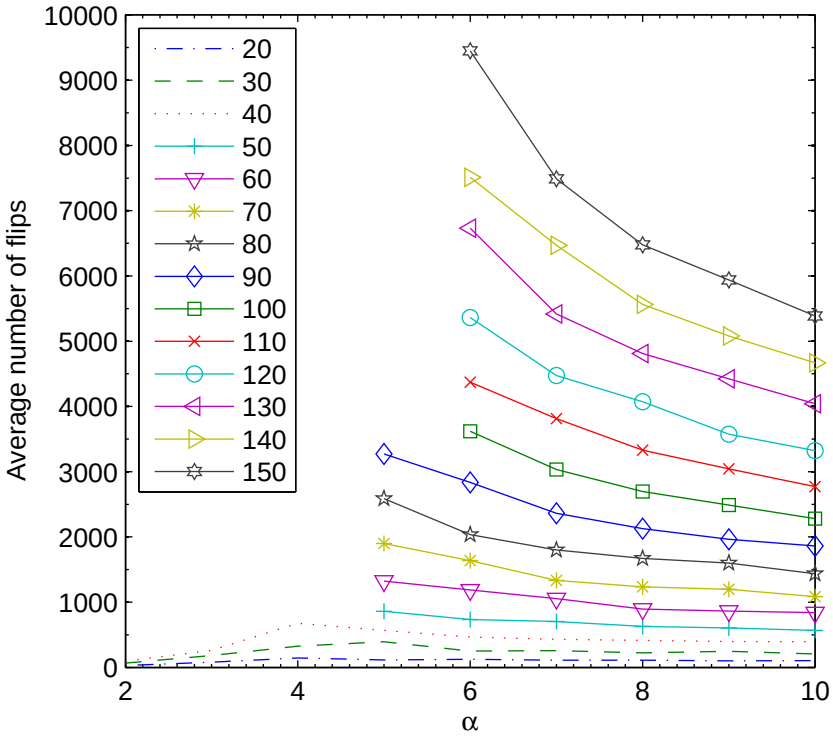

FIGURE 3.8: The median number of flips required to reach a local solution.

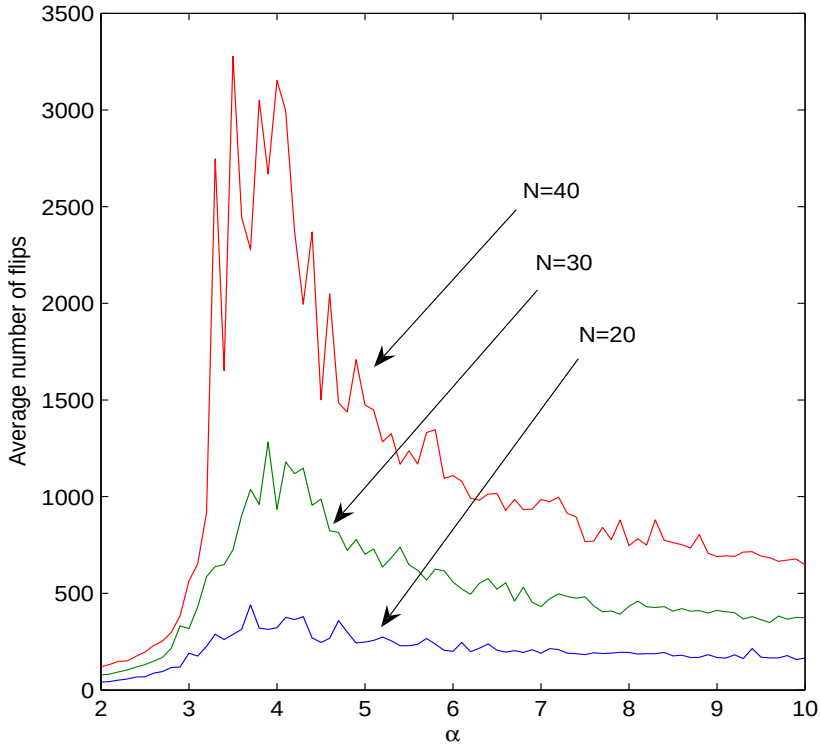

FIGURE 3.9: For 20, 30 and 40 variables the average number of flips were plotted against different values of α.

a more detailed test around the phase transition. These tests were conducted on varying values of α ranging from 2 to 10 in increments of 0.1 with 100 problem instances per increment. We obtained the graph in Figures 3.9 and 3.10. This important result demonstrates the behavior of local search around the transition point. It shows that even local search finds the transition point difficult. In fact, local search follows easy-hard-easy transition just as complete algorithms do.

Why is it that descent finds it difficult around the transition point? Descent starts by finding global minima very easily below the phase transition, because below this point there are large plateaus of global minima. As α is increased toward the phase transition, the plateaus begin to rise above the global minima while slightly breaking up. These broken up plateaus of local minima which are still relatively large compared with global minima present descent with more room for sideway moves before descending further. This greatly reduces the probability of finding better solutions, thus increasing the number of flips. As α moves away from the phase transition, the plateau breakup into even smaller regions creating more local minima. In this state, descent finds it easy to reach a local optimum again.

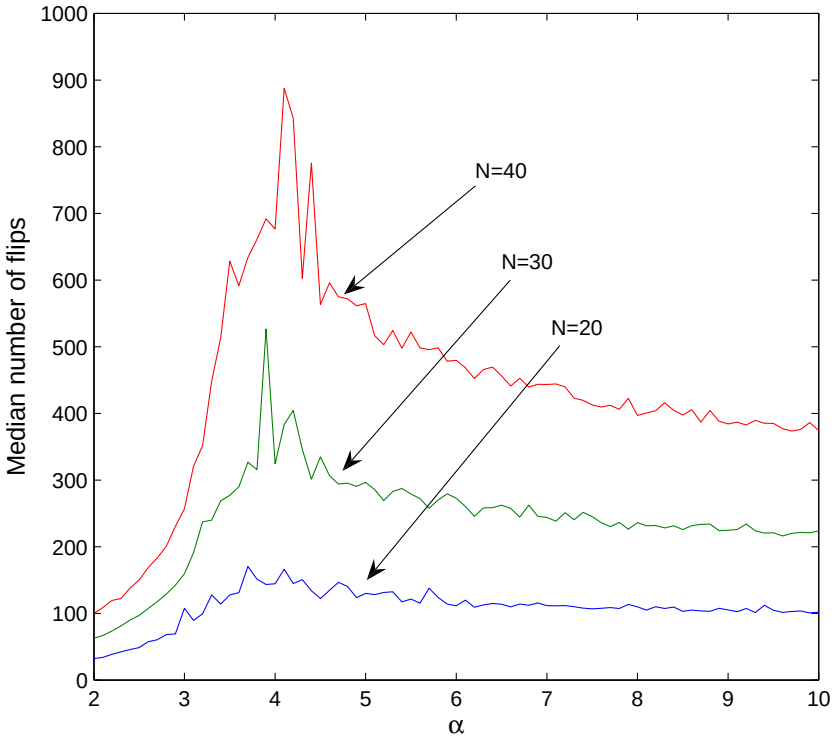

FIGURE 3.10: For 20, 30, and 40 variables the median number of flips where plotted against different values of α.

Chapter 4

Landscape Analysis

Can a population-based algorithm take advantage of global information about the fitness landscape provided by its members to help it solve an optimization problem in a way that cannot be achieved by a solo local search algorithm? This is a common assumption of many users of evolutionary algorithms, however, there is rarely strong evidence that this is the case. Of course, there may be other important ways in which a population might be beneficial. For example, using a population may be advantageous because, by searching different parts of search space, it may quickly find promising regions were it can concentrate its search effort. Also crossover may be beneficial as a macro-mutation which naturally anneals itself as the population converges. These other benefits may be very significant, but they are different to and arguably less exciting than the possibility that a population can learn about the large-scale structure of a problem and then exploit this information to find superior solutions. Although there are a few artificially constructed problems that demonstrate that populations can in principle gain a significant advantage by learning properties of the landscape (e.g. (Shapiro and Prügel-Bennett, 1997; Jansen and Wegener, 1999; Rogers and Prügel-Bennett, 2000, 2001; Watson, 2001; Watson and Jansen, 2007; Prugel-Bennett, 2007)), there has been little unambiguous evidence that this is the case for any naturally occurring optimization problem. We present an algorithm which we will argue does precisely this for one of the classic combinatorial optimization problems, MAX-SAT.

The algorithm we present here is a hybrid algorithm. We find many good solutions using a local neighbourhood search algorithm. The solutions are clustered using a K-means clustering algorithm. The configuration closest to the centroid of each cluster is then used as a starting position for applying a second round of the local neighbourhood search algorithm. The general framework by which multiple local searches are applied followed by clustering or averaging, and then followed by another round of local search is called Landscape Guided Hopping (LGH). The idea behind this method is that several local searches provide guidance to some clustering, Clustered-LGH (CLGH), or averaging, Averaged-LGH (CLGH), techniques, which then is used to hop into a point in the

solution space. This new point allows a local searcher to produce better results more efficiently avoiding intermediate steps. This very simple algorithm finds remarkably good solutions—we describe our tests of the algorithm in chapter 5.

Our interpretation of why LGH performs so well is that the fitness landscape (which we take to be the landscape using the Hamming distance as a metric) consists of a few global maxima (assignments of the variables that maximize the number of satisfied clauses) positioned at different locations in the search space. The global maxima are correlated, but not strongly. Around each global maximum there is a 'galaxy' of local maxima. The closer the local maxima are to a global maximum the more likely they are to have high fitness values. Although, these local maxima tend to be clustered around global maxima they can still be quite far from each other in Hamming distance. For example, they may well differ in 30–40% of their variables, which would mean in a 1 000 variable problems they may be at a Hamming distance of 300–400 from a global maximum. We postulate that our local search algorithm finds good solutions (close to, if not at, a local maximum). By clustering the solutions we pick out good solutions centred around a global maximum (or, at least, around some very good local maximum). By taking the configuration closest to the centroid we move close to the centre of the galaxy which has many high quality solutions in its vicinity. These centroid solutions usually are not very good, because the search space is extremely rugged. However, by performing a local neighbourhood search we end up finding a solution which is almost always superior to the previous solutions we found. The major contribution of this work is to present evidence to support this picture.

As part of our algorithm, we needed a fast method for finding good local solutions for relatively small problems. The classic local search algorithm for SAT and MAX-SAT problems is GSAT which performs hill-climbing by exhaustively searching the entire neighbourhood before choosing the neighbour that gives the best improvement in fitness. As this will get stuck in a local maximum, a number of variants have been suggested, most notably WALKSAT which alternates between GSAT moves and walk moves which alleviates the algorithm from getting trapped in a local maximum. We compared this algorithm with a basic hill-climber (BHC) where a neighbour is chosen at random, and a move is made if the fitness is greater than or equal to the current fitness, otherwise the current position is kept. This is shown to perform much faster than GSAT or WALKSAT on small problems. Intuitively this is not surprising as tracking the neighbourhood at each step is slightly more costly than flipping an assignment at random. We will show later that for larger problems we have developed the fastest GSAT and WALKSAT implementations.

In the next section, we briefly discuss MAX-SAT and describe how we generate the instances used in our tests. We then present our study of local neighbourhood search methods. This is followed in section 4.2 by a presentation of some studies on the landscape properties of small problems. Chapter 5 presents the results of a number of

different algorithms on much larger problem instances.

4.1 MAX-SAT

MAX-SAT is one of the best studied optimization problems—in part because of its association with SAT, which, besides from its theoretical importance, has a huge number of practical applications. Although the number of applications of MAX-SAT are small in comparison with SAT, nonetheless, it has been applied to design debugging of VLSI and SoC cyles(Safarpour *et al.*, 2007) and in Protein Interaction Inference (Ya *et al.*, 2005). A large amount of research has gone into characterizing the typical behavior of random instances. Here, we also concentrate on random *Fixed Clause Length* (FCL) instances (Mitchell *et al.*, 1992). These consist of a set of m clauses where the clauses consist of $k = 3$ literals (we take this to be a strict set rather than a multiset, so that no clause is repeated). The literals in any clause all involve different variables. Every allowable clause is chosen with equal probability. In practice we use the FCL method, section 2.5.1, to generate these problems.

The phase transition has been investigated using statistical mechanics approaches (Monasson *et al.*, 1999; Mezard and Zecchina, 2002). Although these are not rigorous, there is a region around the phase transition where the calculation is believed to be exact in the limit $n \to \infty$ (at least, it passes several stringent self-consistency tests and it gives prediction in agreement with carefully conducted simulations). For small values of $\alpha = m/n$, the problem has a simple landscape corresponding to one very large cluster of satisfied solutions which is easily reached by hill-climbing. Around the phase transition, the statistical mechanics calculation undergoes, so called, one-step replica-symmetry breaking that is a signal for the existence of many local maxima weakly correlated with each other. It is has statistically determined that solutions become clustered in the region $3.87 \leq \alpha \leq 8.29$ the solution space decomposes into clusters which are disconnected. Away from the phase transition, one-step replica symmetry breaking no longer holds and it is postulated that the system enters a state of full replica-symmetry breaking (Montanari *et al.*, 2004; Battaglia *et al.*, 2004). Although there is no analytic solution of the behavior in this region, full replica-symmetry breaking is taken to be an indication of complex clustering of the local optima (Mézard *et al.*, 2005). We focus on random instances with $\alpha = m/n = 8$, which is deep in the hard phase for MAX-3-SAT where full replica-symmetry breaking is believed to hold. To investigate the structure of the fitness landscape we have carried out extensive empirical studies.

4.2 Landscape of Random MAX-3-SAT

In this section, we present some empirical observations on the fitness landscape of random MAX-3-SAT for $\alpha = m/n = 8$. These were carried out as part of a broader investigation of the landscape of MAX-3-SAT, but here we only present results relevant to our thesis. We studied instances up to size 100 by finding many local maxima. To achieve this we used BHC starting from different, randomly-chosen, starting points. To ensure that we had found a local maxima, after running the hill-climber with no improvements in many attempts we switched to an complete-neighbourhood search method that checked all neighbours at the same cost as the current point, and then checked their neighbours repeatedly, until either a fitter solution was found or else all neighbours at the current cost had been searched, in which case we could be sure that we were at a local maximum. By performing multiple searches on the same instances, we were able to measure statistical properties of the local maxima. A common feature of all the instances that we investigated was that the higher the fitness of the local maximum the more likely we would find it. As a rule-of-thumb, we observed that the likelihood of finding a local maximum roughly doubled each time we satisfied one more clause. This result is not so surprising as it is easy to imagine why better local maxima could typically have larger basins of attraction than less fit local maxima.

What makes MAX-3-SAT instances hard is that there are many more local maxima than global maxima. Thus, even though the basin of attraction appears to be largest for the global maxima, nevertheless, we are more likely to get trapped in a lower-fitness local maximum because there are many more of them. The number of local maxima appears to increase exponentially with the size of the instances, which makes finding a global maxima increasingly less likely as the instance size becomes large. The exponent describing the exponential growth is, however, rather small so even for systems of size 100 finding a global maxima is not difficult. At least, for problems up to this size we were able to find the fittest local maxima multiple times. We postulate that these are the global maxima, since if there were even a single maxima fitter than those we found then we would expect to find it with high probability given the number of hill-climbs we made (unless it had a very atypically small basin of attraction). We call our best maxima found in this way, *quasi-global maxima* as we believe them to be the true global maxima, although we have no proof of this. (For small problems, $n \leq 50$, we could find the true global maxima using a branch-and-bound algorithm. In every case, the best solution found by performing multiple BHC were true global maxima. We also tested problems with $n = 100$ from SATLIB and in every case we were able to find the best solution for the problem using BHC). Note that if we were to look at much larger-sized problems, then we would find each best solution only once or a very few times, in which case we would have no grounds to argue that these are likely to be the global maxima. The fact that we believe we can find all global optima for relatively large instances makes this problem class very rich to study empirically.

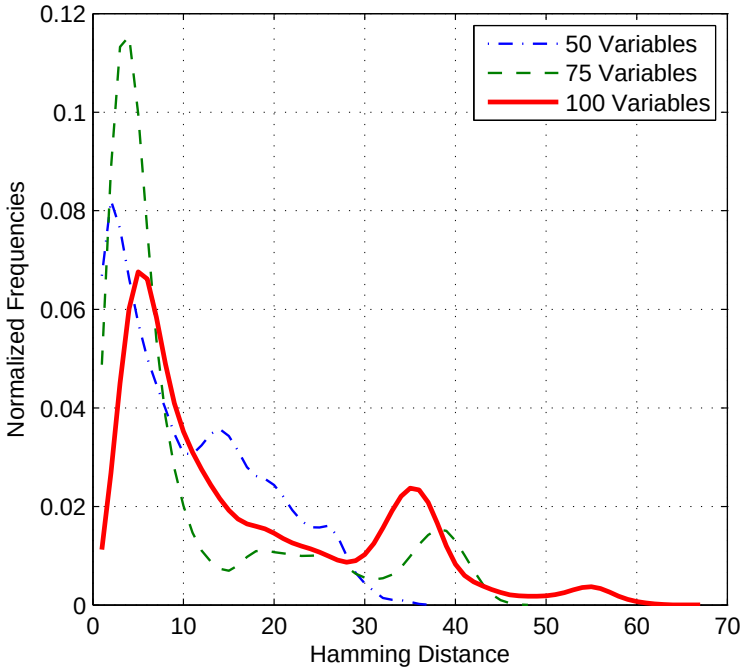

FIGURE 4.1: Shows a histogram of the Hamming distance between quasi-global maxima for 50 instances of size $n = 50$, 75 and 100 variables and with a clause to variable ratio of $\alpha = 8$. There is a cluster of very close global minima below a Hamming distance of 10. Also, a significant number of global minima are found at Hamming distances equivalent to 30–40% of the variables.

We investigated the distribution of quasi-global maxima by examining the frequencies of Hamming distances between all quasi-global maxima in an instance. In figure 4.1, we show these frequencies averaged over 300 problem instances. To find the set of quasi-global maxima we ran BHC followed by complete-neighbourhood search 5 000 times. The histogram has a large peak at a Hamming distance approximately equal to 5% of the total number of variables. This indicates a clustering of quasi-global maxima around each other. However, the histogram has a large tail with a second peak at a large Hamming distance away from the first. This is indicative of multiple clusters that are weakly correlated with each other (if there was no correlation then the clusters would be at a Hamming distance of $n/2$).

To demonstrate that the histogram is consistent with this picture. We generated clusters using the following procedure. We chose a centre $\boldsymbol{C} = (C_1, C_2, \ldots, C_n)$ where $C_i \in \{0, 1\}$, and a second centre $\boldsymbol{C'} = (C'_1, C'_2, \ldots, C'_n)$ was generated from the first by randomly changing k variables where k is a uniformly distributed integer between 0 and $3\,n/4$. Thus on average, $\boldsymbol{C}$ and $\boldsymbol{C'}$, are separated by a Hamming distance $3n/8$. We then generated between 20 and 220 random strings centred around each of the two centres at an average Hamming distance of $n/10$. We then computed the correlation between all pairs of randomly chosen strings. This was then averaged over 100 samples. The histogram of correlations is shown in figure 4.2. We observe a very strong similarity in

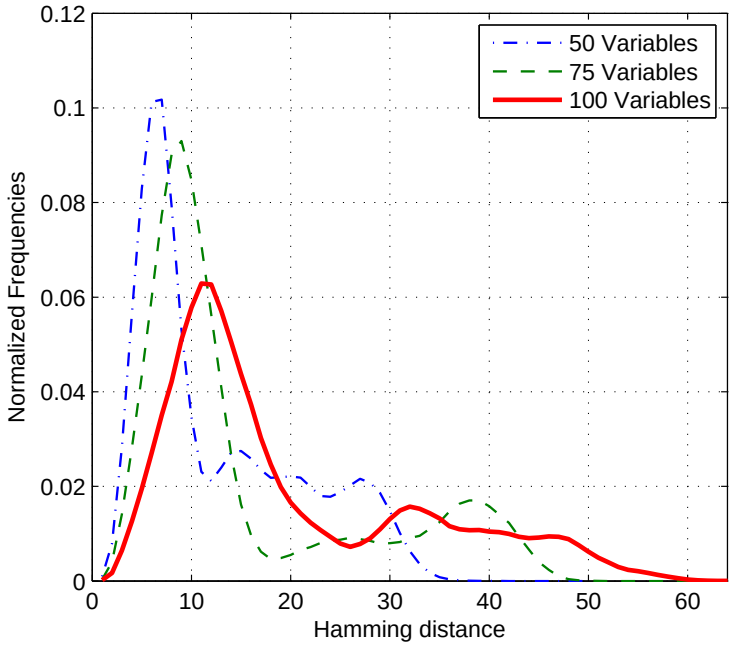

FIGURE 4.2: Shows a histogram the Hamming distance between randomly chosen points forming two clusters. The distance between each cluster is taken to be a uniformly chosen random variable between 0 and $3\,n/2$. The graph is generated by averaging over 100 samples. We note the strong similarity between this and Figure 4.1. This similarity is shown in having a large main peak on the left side of the frequency plot, and several other smaller peaks following the first.

the structure of this figure and figure 4.1 which lends support to the hypothesis that the quasi-global solutions are themselves clustered around a few centres.

To illustrate how the lower-cost local maxima are clustered relative to the quasi-global solutions, we measured the Hamming distance between the local maxima and the nearest quasi-global solution. Histograms of this Hamming distance are shown in figure 4.3. In these figures, we consider only those local optima at a cost of 4 and 8 away from the global-maximum cost. We note that the higher cost solutions are closer on average to a quasi-global maxima than lower cost solutions.

Figure 4.4 shows how the average Hamming distance between the local maxima and the nearest quasi-global maxima varies as a function of the difference in the cost between the local maxima and the quasi-global maxima. By scaling both axes by $1/n$ these curves appear to collapse onto a universal curve. It is easy to understand why higher cost solutions should be closely correlated on average with the quasi-global maxima as local-optimum solutions represent good ways of maximizing the number of satisfied clauses. Therefore, nearby solutions are also likely to satisfy many clauses. However, what is perhaps more surprising is that even the solutions whose cost differs by one from the quasi-global optima have a high average Hamming distance from any quasi-global optima. Even for relatively small problems with 100 variables this average Hamming

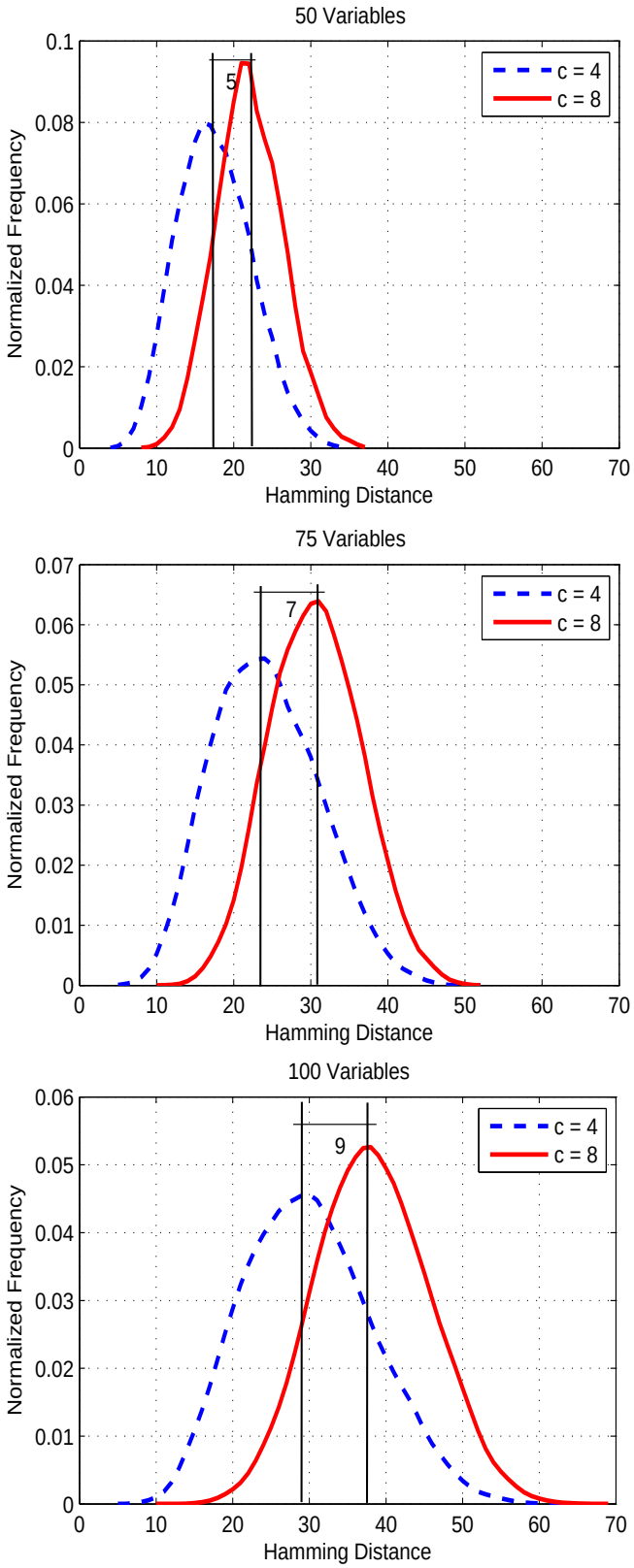

FIGURE 4.3: Shows a histogram of the Hamming distance between the quasi-global maxima and local maxima with fitnesses 4 and 8 below the quasi-global maxima. As the number of variables increase, the Hamming distance to the quasi-global maxima also increase.

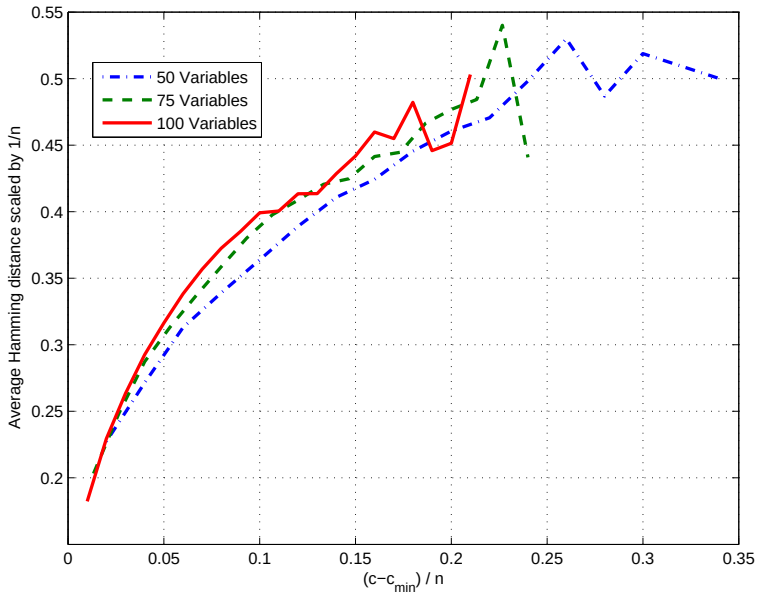

FIGURE 4.4: The average Hamming distance between the quasi-global maxima and the local maxima. As the gap in fitness between the quasi-global maxima and local maxima decreases so does the average distance to the quasi-global minimum.

distance is around 18 which is sufficiently large that the probability of a stochastic hill-climber reaching a global maximum from a local maximum is negligibly small. To be more explicit if the Hamming distance between a local maxima and a better solution is k, then a local search algorithm would typically have to explore every solution up to a Hamming distance k before finding a better solution. The number of solutions in a ball whose Hamming radius is strictly less than k is

$$\sum_{i=0}^{k-1} \binom{n}{i}$$

which for $n \gg k$ is $O(n^{k-1}/(k-1)!)$ (even for small instances with $n = 100$ and $k = 18$ this is approximately 8×10^{18}). Within this "Hamming ball of radius k" there will be no solution better than the current solution (since by assumption the closest solution is a Hamming distance k away). Thus there is no local gradient information to exploit. There may be solutions of the same cost in this Hamming ball which are closer to the global solution, but there is no way of knowing whether it is closer to or further from a better solution than the current solution.

Although it is always dangerous to rely on low-dimensional pictures to understand what happens in a high-dimensional space, nevertheless, we offer the following caricature of our fitness landscape. We imagine the search space as being points on a 'world' where the height of the points representing the fitness values. This is schematically illustrated in figure 4.5. The good solutions lie in mountain ranges. The mountain ranges have

hugely more foothills than high mountains. There are only a few mountain ranges in this world and they are slightly correlated (e.g. all the mountain ranges might lie in one hemisphere). The mountain ranges occupy only a very small proportion of the world. As with real mountain ranges, higher solutions tend to lie in the middle of the mountain ranges. Starting from a random position and hill-climbing we are likely to land up at a foothill, just because there are so many of them. Finding a good solution through hill-climbing alone will be very difficult. An alternative strategy is to perform a large number of hill-climbs starting from different randomly-chosen positions. We could then take the average of the solutions we find. This will put us in the centre of the hilly hemisphere. Although, we are unlikely to be at a peak, if we perform a hill-climb we are likely to find a superior solution than if we started from a random position. However, we can do even better by clustering the solutions we find after performing hill-climbing. If we are lucky, a cluster will correspond to a mountain range. The centres of the clusters corresponds to the regions with many high mountains so if we restart hill-climbing from the centre of a cluster we have a very good chance of finding a high quality solutions, Figure 4.6. Of course, this picture fails in many ways. The search space is not continuous, but is discrete. Furthermore, using a Hamming neighbourhood the topology of the search space is an n-dimensional hypercube. The high-dimensionality makes it harder for low-cost solutions to be local maxima since they have a large number of neighbours. Also the set of costs is discrete so that there is no gradient information. Nevertheless, as we will see an algorithm based on clustering seems to perform very well which suggests that this simple picture might not be too misleading.

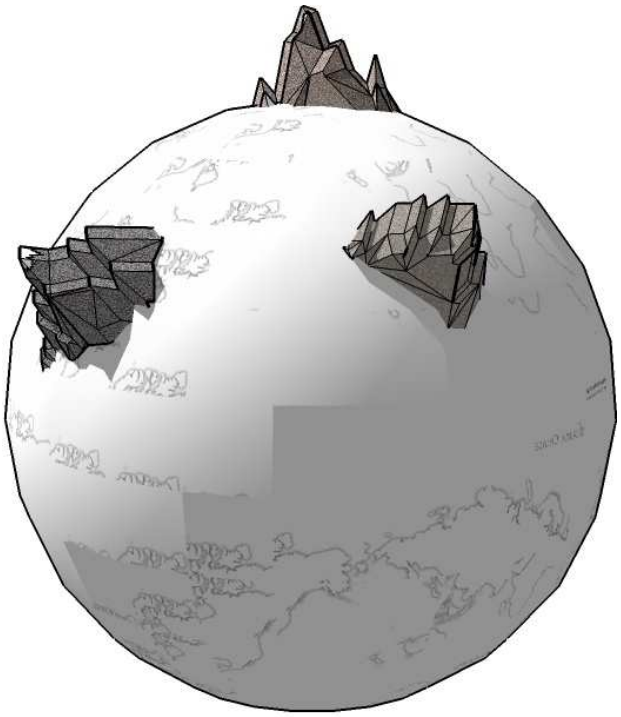

FIGURE 4.5: Caricature of the Fitness Landscape showing the clustering of good solutions.

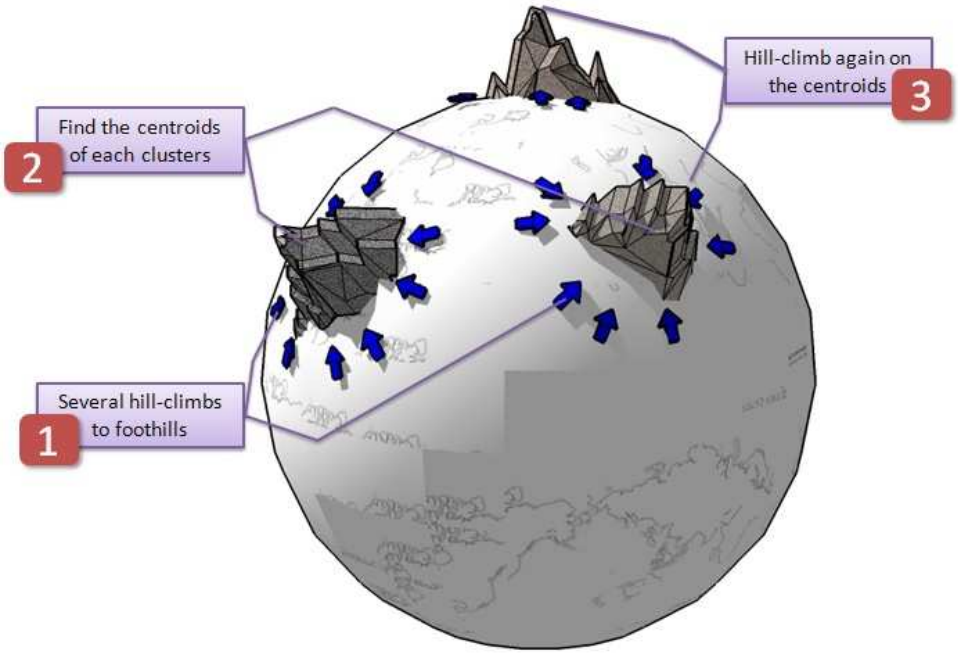

FIGURE 4.6: The concept of using K-means to finding high-quality solutions.

Chapter 5

CLGH: A Novel Approach to Solving Satisfiability

To exploit the structure of solutions presented in the previous chapter we use Clustered Landscape Guided Hopping (CLGH), and the mechanism used for clustering of solutions is the K-means algorithm (Hartigan and Wong, 1979). In K-means, a number of points s are divided into K clusters. In the typical case, when the n-dimensional points are in $\mathbb{R}^n$ space, the sum of the squares are minimised. In this case, the points are n-cubed in $\{0, 1\}$ space. The sum of the Hamming distances are minimised. The K centroids that are found for these clusters are used as starting points for Hill-Climbing.

We will see that these points provide good starting points in the search for solutions. This method is compared with a number of different solo-search and population based algorithms, and it is found that CLGH produces better solutions on large scale MAX-3-SAT problems. This approach is shown to be effective at different points around the phase transition. In addition, different ways of applying CLGH will be discussed to show that only a single application of clustering is required to find good starting points.

5.1 Experimental Setup

We have tested our proposed algorithm on large instances (6 000 to 18 000 variables) of MAX-3-SAT for $\alpha = m/n = 8$. We are unable to compare our algorithm with most other algorithms that appear in the literature since the other studies were performed on much smaller instances (typically around 100 variables). For such small instances we found that running the basic hill-climber a few times would almost always find a solution we were unable to improve on and which we believe to be the global optimum. This made CLGH approach redundant. The only work we are aware of which studied similar sized instances to those used here is by Zhang (Zhang, 2004). Our algorithm

substantially out-performs the results given in that paper.

To provide some comparators to the CLGH approach we ran a number of different algorithms. The main purpose of these comparators was to rule out other possible explanations of why the approach we are taking is successful. The CLGH algorithm we propose has not been highly optimized. By careful tuning we would expect that we can achieve better solutions. However, the purpose of our experiments is to demonstrate that a significant improvement in performance is obtained by clustering.

We generated random MAX-3-SAT instances using the method described in section 2.5.1. We considered problem instances ranging in size from 6 000 to 18 000 variables in increments of 2 000 variables, and with $\alpha = m/n = 8$. These are difficult problems since they are in the over-constrained region. For each increment we generated 100 problems instances.

In all tests we carried out we started by performing 1 000 hill-climbs starting from different random starting configurations. We used the basic hill-climbing strategy. The number of iterations used on a problems with n variables is $T(n) = 5\,n/2 + 5000$. That gives 20 000 iterations for 6 000 variables and 50 000 for 18 000 variables. The number of iterations were increased with the number of variables so that BHC would be given more opportunities to find better quality solutions. With the growth of the number of variables it becomes more difficult for a local search algorithm to reach local maxima, although the goal was not necessarily to reach a local maximum, but only to find a good solution. The best result for the 1 000 hill-climbs averaged over all 100 problem instances is shown in the second column of table 5.1.

We then tested a number of different strategies to boost the performance obtained from these initial 1 000 points. The testing procedure we carried out is shown schematically in figure 5.1. As a baseline we repeated the basic hill-climber for another $T(n)$ steps on all 1 000 search points. These results are shown in the third column of table 5.1. This second round of hill-climbing shows that the solutions found in the first round were still some way away from being locally optimal.

5.2 CLGH and ALGH

We next performed CLGH using the K-means clustering algorithm on the 1 000 search points found by the initial hill-climbing. These points were not tested for uniqueness. However, because the problems were large, i.e., 6 000 to 18 000 variables, the number of local solutions are also large, and hence the probability of landing in the same local optima is very small. This algorithm starts by assigning a random string on the n-cube to each of K initial "centres" (note that, in this section, K is used to denote the numbers of centres in K-means clustering and should not be confused with the number

TABLE 5.1: Comparison of different algorithms. Column 1 shows the problem size, while columns 2–6 give the lowest number of unsatisfied variables found by different algorithms. These are BHC, BHC+BHC (baseline), BHC+K-Means+BHC (CLGH), BHC+Averaging+BHC (ALGH), hybrid-GA and BHC+Perturb+BHC. Columns 7 and 8 show the increase in performance over the baseline achieved by using CLGH and ALGH respectively. The tests were carried out on random MAX-3-SAT problems with $\alpha = 8.0$. Each test was performed on 100 problem instances for each number of variables.

#Vars	**First BHC**	**Second BHC (1)**	**K-Means/ BHC (2)**	**Average/ BHC (3)**	**hybrid-GA**	**Perturb/ BHC**	**(2) - (1)**	**(3) - (1)**
6000	1971.77	1448.35	1370.61	1385.82	2429.5	1447.92	77.74	62.53
8000	2944.03	2037.26	1913.26	1943.38	3691.22	2038.78	124	93.88
10000	3464.7	2614.65	2456.67	2507.56	4908.87	2617.19	157.98	107.09
12000	4235.8	3247.74	3051.09	3125.79	6218.57	3247.4	196.65	121.95
14000	4999.14	3892.06	3652.23	3761.51	7533.33	3895.38	239.77	130.55
16000	5711.81	4496.69	4226.15	4368.23	N/A	N/A	270.54	128.46
18000	6551.83	5256.28	4932.41	5129.12	N/A	N/A	323.87	127.16

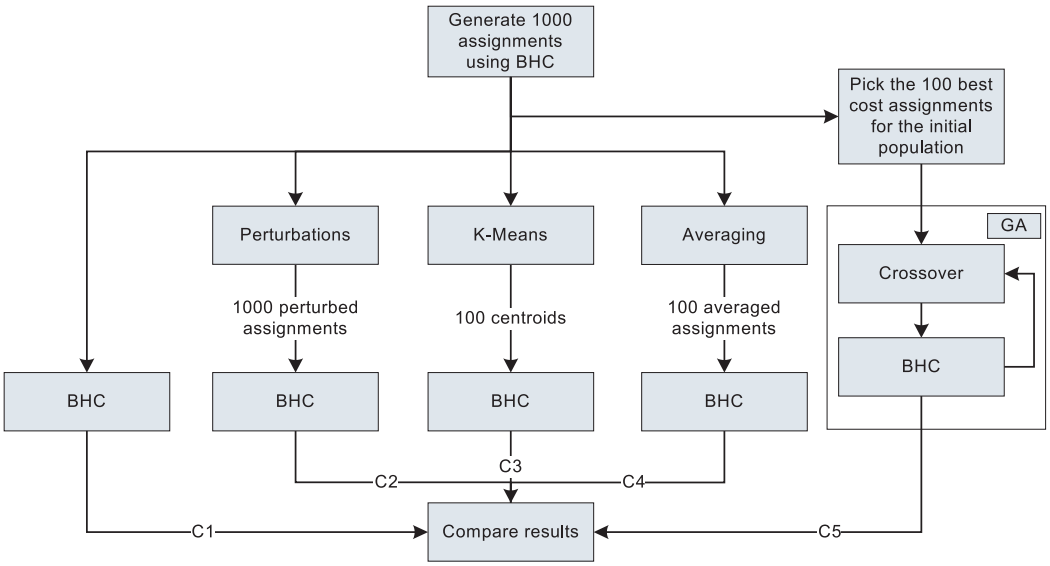

FIGURE 5.1: Schematic diagram of the set of tests carried out and reported in table 5.1.

of variables in each clause). Each of the 1 000 points is then assigned to the cluster with the nearest centre. The centres are then updated to be the configuration which best represents the points in the cluster, in the sense that it minimises the mean Hamming distances to the set of points in the cluster, $\mathcal{C}$, i.e.

$$\boldsymbol{X} = \underset{\boldsymbol{X}}{\operatorname{argmin}} \frac{1}{|C|} \sum_{\boldsymbol{Y} \in C} H(\boldsymbol{X}, \boldsymbol{Y}) \tag{5.1}$$

where $H(\boldsymbol{X}, \boldsymbol{Y})$ is the Hamming distance between configurations. This is equivalent to rounding the average bit value of each site up to 1 or down to 0. The points are reassigned to the nearest centroid and the process is repeated until there are no changes. This usually happens after five to ten iterations. The computational cost of K-means clustering is small compared with the time required to do hill-climbing. Once the centroids have been computed, a new starting point is found by rounding each component of the centroid to obtain a feasible solution.

In the results we summarise in table 5.1, we used $K = 100$ clusters. This was decided after a small amount of experimentation. This is probably not optimal, but fits with our decision not to fine tune CLGH. A second round of hill-climbing is carried out from the solutions obtained from the 100 centroids. The results obtained after this procedure are shown in the forth column of table 5.1. In every case there is a considerable gain in performance compared to the baseline, even though the baseline involved considerably more work (because the second round of hill-climbing reported in column 3 of table 5.1 was carried out on all 1 000 points rather than 100 used in the K-means clustering algorithm). The gain in performance compared to the baseline is shown in column 8 of

table 5.1.

We have compared CLGH with ALGH, where we randomly selected 10 points and find the centroid of the group (for this problem the centroid can be found by taking the average assignment of each variable and rounding). This was repeated 100 times to give 100 centroids so as to give a fair comparison with the CLGH method. A second cycle of hill-climbing is then carried out. The results are shown in the fifth column of table 5.1. This again produced a substantial gain in performance compared with the baseline (the gain is shown in the last column of table 5.1), however, these gains are smaller than those obtained by CLGH, particularly for large number of variables. This provides further empirical support for the claim that the global maxima are clustered (although we have shown clustering for instances of size 100, these results are for much larger instances). It also shows that even the mean of all the good solutions provides a much better starting point than a random starting point.

We want to show that these results are *not* due to clustering in CLGH or averaging in ALGH acting as a macro-mutation which allows the search to escape out of local maxima. To do so we applied random perturbations of 0.1%, 1%, 2%, 5% and 10% of the variables, and then repeating hill-climbing. We found that doing this gave us worse performance than the baseline algorithm. Even with 0.1% the random perturbation appears slightly detrimental (see column 7 of table 5.1). These results are not so surprising, since it is clear from comparing the results of the baseline algorithm with the results after the first hill-climb (columns 2 and 1 respectively) that we are far from being stuck in a local maximum.

As a final test, we compared our algorithm against a hybrid genetic algorithm. The hybrid genetic algorithm combined hill-climbing with selection and two-parent crossover. A population of 100 individuals was used. We used Boltzmann selection where we chose each member of the population with a probability proportional to $\exp(-\beta F_i/\sigma)$ where F_i is the fitness of individual, i; σ is the standard deviation of the fitness values in the population; and β controls the selection strength. Various values of β were tried, but this did not strongly affect the results. Uniform, single-point and multi-point crossovers were tried. The best results were obtained with single-point crossover. Column 6 of table 5.1 shows the best results we were able to obtain using a GA. Although we do not claim that all the parameters were optimally chosen, the results obtained by the hybrid-GA are disappointing compared to the other algorithms. The reason for this is, in part, due to the fact that the GA was not given sufficient time to converge. In the next subsection, we analyse the performance of CLGH when it is run for longer times. Even then, we will see that the CLGH approach has a considerable advantage over a GA.

This may seem surprising as two-parent crossover might superficially appear to be doing something similar to averaging, however it is important to appreciate the difference. This

is easily seen by considering a simple example. Consider a unitation problem consisting of a binary string $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$ with $X_i \in \{0, 1\}$, where the fitness is a function of the number of 1's in the string. Defining the proportion of ones as $m = \sum_{i=1}^{n} X_i/n$, the fitness is given by

$$F(x) = \begin{cases} m & m < m_1 \\ m_1 & m_1 \leq m < m_2 \\ m - m_2 + m_1 & m_2 \leq m \end{cases}$$

This is shown in figure 5.2 for the case when $m_1 = 0.75$ and $m_2 = 0.95$.

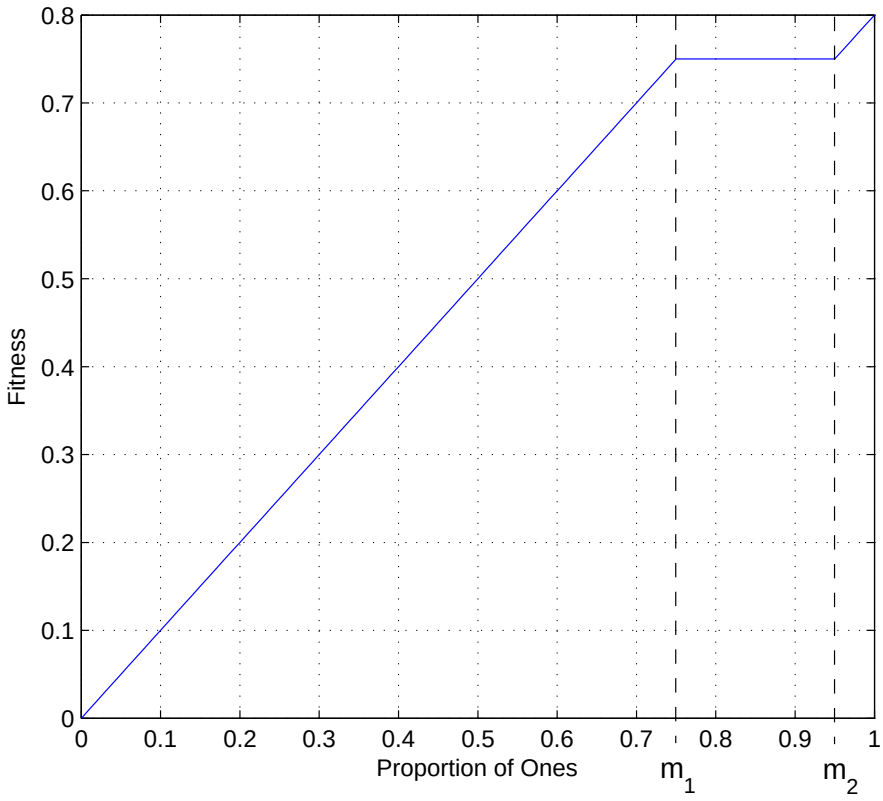

FIGURE 5.2: Fitness function of the Iceberg problem. This problem is easily solved by averaging good solutions, but very hard for a hill-climber or genetic algorithm.

We call this the Iceberg problem because the configurations with $m > m_2$ can be viewed as a small iceberg in a large ocean of solutions with cost m_1. For large n a hill-climber starting from a random string with fitness close to 0.5 will climb the slope until it reaches a state where 75% of the variables are 1's. When it reaches the plateau it has no heuristic information. As the density of states falls off very fast as a function of the number of 1's the hill-climber will, with high probability, lie close to the edge of the plateau with approximately 75% of the variables equal to 1. A population of hill-climbers will also lie very close to the edge of the plateau. If we were now to perform crossover on two

individuals then again with high probability the child would have approximately 75% of its variables equal to 1. This would be a slightly more efficient way to explore the plateau than hill-climbing alone as most random mutation moves will, on average, move away from the all 1's string (since there are more 1's than 0's a random mutation is more likely to attempt to change a 1 to a 0 rather than a 0 to a 1). Crossover, by contrast does not change the number of 1's on average. This concentration effect of crossover has been discussed in closely related models previously (Shapiro and Prügel-Bennett, 1997; Jansen and Wegener, 1999; Rogers and Prügel-Bennett, 2000, 2001). Nevertheless, it still takes an exponential amount of time to find the global maximum using crossover. In contrast if we average a population of say 100 individuals that have undergone hill-climbing and round up to 0 or 1, then, for any reasonable size problem, the resulting solution will, with overwhelming probability, consist of the all 1's string.

Clearly, this is a contrived problem, its purpose is to demonstrate that averaging is very different to crossover. This is true even if we used multi-parent crossover (Syswerda, 1993) or a uni-variate estimation of distribution algorithm (EDA), as we will see in section 5.4, where, despite averaging, the expected time to solve the problem shown in figure 5.2 would still grow exponentially. Clearly, the landscape of MAX-SAT is much more rugged on short length scales than the Iceberg problem. On very large length scales, the landscape of MAX-SAT differs because it possesses multiple global maxima some distance apart. However, on some intermediate scale this model appears to capture some important properties of the landscape of MAX-SAT–that is, the globally optimal solutions lie at the centre of more easily found local optima.

5.3 Temporal Behavior

In the section above the behavior of the hybrid genetic algorithm was particularly poor. This was due to the limited number of BHCs allowed for each algorithm. When given a longer time the hybrid-GA performs considerably better. In figures 5.3 and 5.4 we show the average performance of parallel-BHC, CLGH and the hybrid-GA. Each algorithm was run for 3 minutes and the results were averaged over 100 instances of randomly generated MAX-3-SAT instances with 6 000 and 10 000 variables at $\alpha = 8$. In parallel-BHC, we run 10 BHCs in parallel and show the best of these (ten runs were chosen as it appeared to give good performance in preliminary tests). CLGH was run starting with an initial population of 100 where we performed 27 000 BHCs before performing K-means clustering with $K = 10$ clusters and then running BHC starting from the 10 centroids after they had been rounded to the nearest feasible solution. No tuning was performed on the K-means clustering algorithm. Finally we tested a hybrid-GA with a population of size 10 where we performed uniform crossover, Boltzmann selection with a selection strength of $\beta = 0.1$ and BHC. The parameters for the hybrid-GA were chosen after performing a large number of preliminary tests. As can be seen the GA

outperforms BHC given enough time, but does not beat CLGH on average, (although in some instances it does).

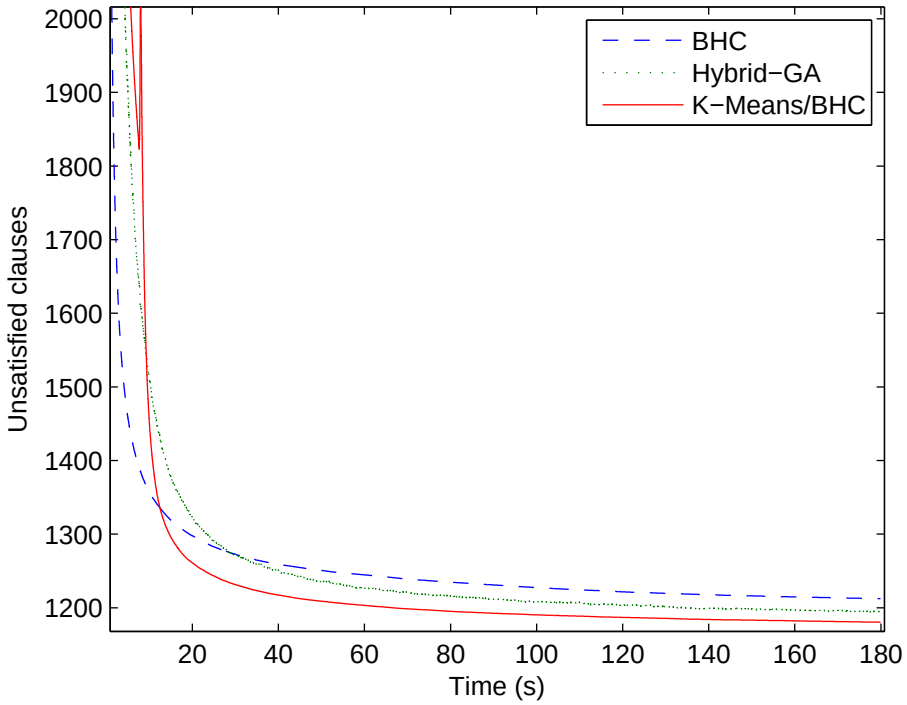

FIGURE 5.3: Comparison of BHC, genetic algorithms and CLGH as a function of CPU time run on 100 randomly generated MAX-3-SAT instances with 6 000 variables at $\alpha = 8$. The large jop in fitness in the CLGH method after around 10 seconds marks the point where K-means clustering is carried out.

For larger problem instances the speed of CLGH becomes more pronounced so that for problems with 18 000 variables run for 5 minutes CLGH gave better performance than a hybrid-GA on every one of 50 instances that was tested. The average performance of the three methods are plotted in Figure 5.5. These results demonstrate that the benefit of performing CLGH persists even after some time. We attribute this to the fact that K-means has moved the searcher to a part of the search space where there are more high quality solutions.

5.4 Comparison with the Estimation of Distribution Algorithm (EDA)

One concern that could be raised with regards to way we used ALGH with MAX-SAT is that it closely resembles the way Univariate Estimation of Distribution Algorithms (UEDA) operate. In UEDA a population of λ individuals are randomly generated. The fitness of these individuals is calculated. Then, via a particular selection criteria, a set

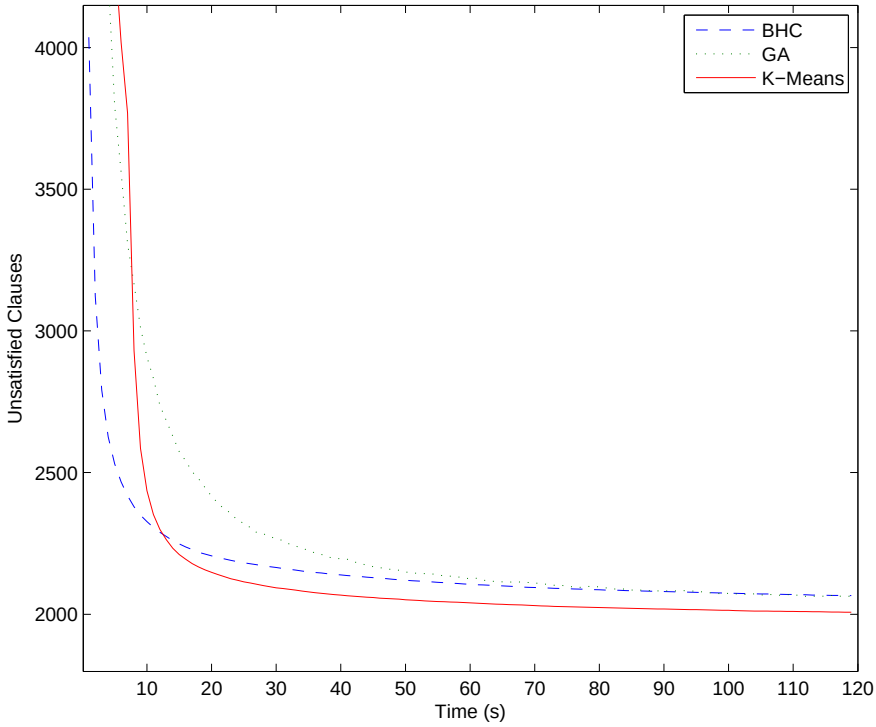

FIGURE 5.4: Comparison of BHC, genetic algorithms and CLGH as a function of CPU time run on 100 randomly generated MAX-3-SAT instances with 10 000 variables at $\alpha = 8$. The large hop in fitness in the CLGH method after around 10 seconds marks the point where K-means clustering is carried out.

of μ individuals are chosen from the population. The probability distribution for these μ individuals is found, and either some or all of population is replaced by generating new individuals from the μ individuals. Applying this algorithm to MAX-SAT would give the impression that UEDA and ALGH are the same. However, we will show by empirical evidence that they work differently.

Estimation of Distribution algorithms come in different forms. In general they are like GAs in that they are population based. However, EDAs do not employ the crossover or mutation operators in general. The only resemblance they have with GA is that they incorporate a selection criteria based on the fitness of the individuals in the population. Conversely, EDA is different from GA because it utilizes a statistical paradigm that allows for the production of the next generation of individuals (Qingfu, 2004).

We use the Univariate Marginal Distribution Algorithm with hill-climbing also known as the Population Based Incremental Learning (PBIL) (Baluja and Caruana, 1995) in our experiments. The mechanism of how these algorithms work is simple. Using μ individuals, a probability vector is created based on the number of 1s and 0s across the individuals. This generalizes the information obtained from the population. This

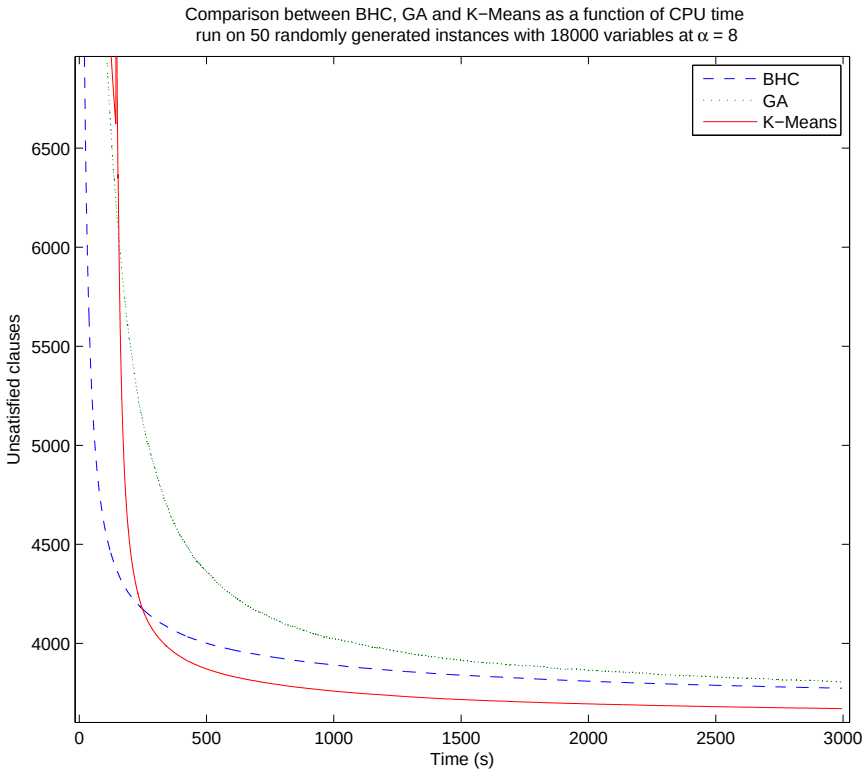

FIGURE 5.5: Comparison of BHC, genetic algorithms and CLGH as a function of CPU time run on 50 randomly generated MAX-3-SAT instances with 18 000 variables at $\alpha = 8$. The large hop in fitness in the CLGH method after around 10 seconds marks the point where K-means clustering is carried out.

probability vector can then be used to set variables of the new individuals based on the proportions of 1s to 0s or their probabilities. Each of the variables in the individuals is considered independent from the other and hence univariate. This makes the computations much simpler especially since we are avoiding the cumbersome computations of the joint distributions of variables (Conzalez *et al.*, 2000).

In principle, Multivariate Marginal Distributions Algorithms (MMDA) could solve MAX-SAT Problems more efficiently than UMDA. However, tests on small problems were performed using hierarchical Bayesian Optimisation Algorithm (hBOA) hybridised with GSAT (hBOA+GSAT) (Pelikan and Goldberg, 2003). Hybrid-hBOA was compared with GSAT and WALKSAT on random MAX-3-SAT problems obtained from SATLIB. The authors report that WALKSAT performed slightly better than hBOA+GSAT on random problems, and solved all instances of Graph colouring MAX-SAT problems of which none was solved by WALKSAT.

One of the main problems of GAs and UMDA is premature convergence. This happens because the population gradually seeks a particular optimum. However, solutions have been proposed to diversify the search. Some of which are based on multiple populations

interacting with each other occasionally (Baluja and Caruana, 1995). This, however, imposes a computational burden on the search algorithm having to work on different populations. Our K-means clustering algorithm in CLGH does precisely this with minimum computational overhead, especially since it is a one time operation that directs the search into the most promising regions of the solutions space.

We performed experiments that compare the performance of Hill-Climbing, GA, EDA and CLGH. In this test, we used the BPIL algorithm. We start with a population of 20 individuals, then we apply a round of hill-climbing to each member of the population. After which, we evaluate the fitness of the individuals, and select the top $\mu = 5$ individuals. We average the variables across the individuals to obtain the probability of a 1 for each variable. consequently, an entirely new population is created by setting the variables of the new individuals based on the probability vector. Finally, another round of hill-climbing is performed, and the process is repeated.

After experimenting with several trials we have adjusted the taking-of-turns between PBIL and hill-climbing to obtain the best performance possible. We show from these results that in the case of MAX-SAT, there are varying performance levels of PBIL impacted by varying values of α. Figures 5.6, 5.7, 5.8, and 5.9 show this comparison for 6 000 variables with values of $\alpha = 4, 6, 8,$ and 10. We see that as α increases from 4 to 10 the performance of PBIL becomes better than GA (around the phase the phase transition). As α increases, the performance of PBIL lags behind to GA (The more noisy line represents the performance of GA). In all cases, CLGH offers the best performance level in comparison.

Although the graphs show that there is a clear advantage to using CLGH, it could be argued that the single application of K-means followed by hill-climbing could have give it this advantage. We could, for example, generate a new population in EDA after the first round of hill-climbing, and follow that with another local search. We argue that the decisive jump that we obtain from K-means cannot be replicated via the application of a single selection of individuals and the creation of new individuals via the probability distribution. We have tested for this, and found that the performance becomes even worse. We attribute this to the way the next population is created. In CLGH, when the individuals (using the same terminology used in EDA) are clustered and averaged they are then rounded to the nearest 0 or 1. Here, we are precisely hopping very near the centre of each cluster. In contrast to EDA, by creating several individuals for the next generation from the probability vector, the probability of generating the centre point becomes smaller as n grows larger. The probability of generating the centre point of the cluster follows equation 5.2.

$$P(\boldsymbol{X}) = \prod_{i=1}^{n} P(X_i) \tag{5.2}$$

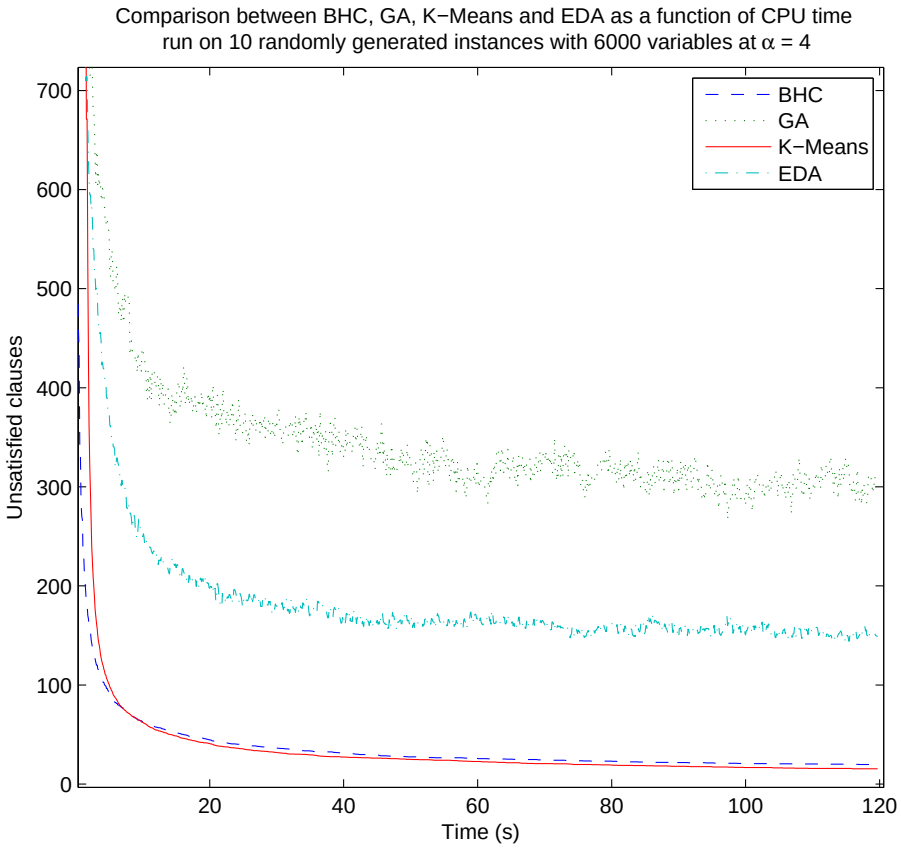

FIGURE 5.6: Supplementary results all for $N = 6\,000$ and 10 problem instances at different $\alpha = 4$ values.

Given that $X_1, X_2, ..., X_n$ are independent. Where X is a variable, and n is the number of variables. The probability of getting the best starting point for the next round of hill-climbing becomes exceedingly small as the number of variables becomes large.

Another problem with EDA is the selection criteria. Two main ways are implemented in the selection process. Either the top fittest μ individuals are selected, or the individuals are selected with a probability proportional to the fitness based on Boltzman selection. In both cases, the fittest individuals have more chances of being selected. This, as we have seen in the case of GA, is not always an efficient procedure for combining individuals. The main problem with this picture is that the individuals chosen might belong to several regions of the solution space with very different variable-configurations. This would raise the prospect of having the new population land far away from where the best solutions tend to be.

In CLGH it is quite a different story. Instead of mixing the population to create the next starting points regardless of the arrangement of the solutions as in the case of GA and EDA, CLGH groups similar solutions with each other to create individuals from each group within the group, hence lead to better solutions. In the beginning of the

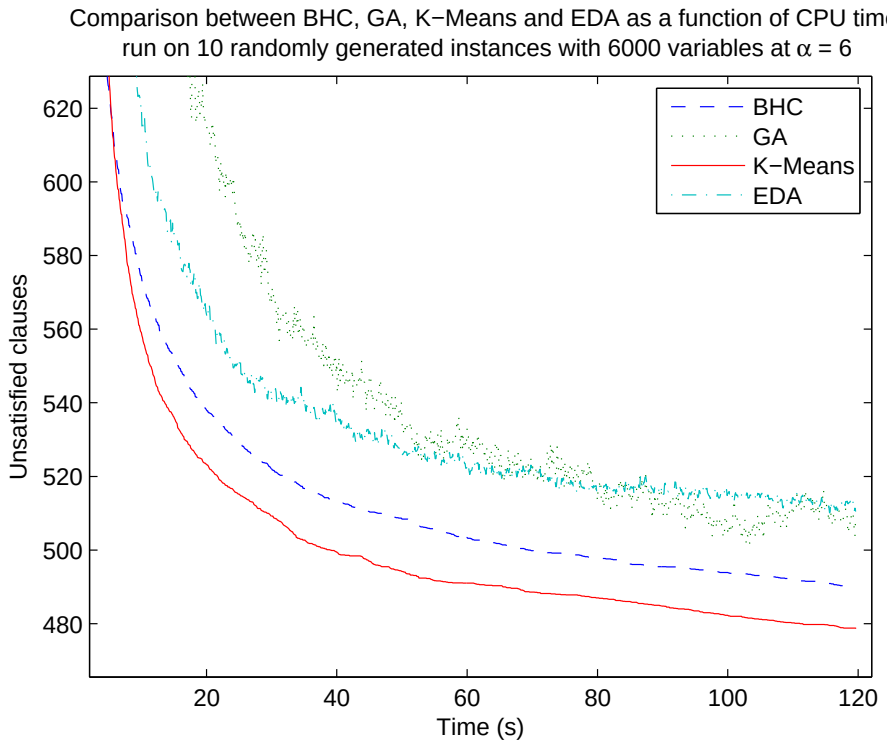

FIGURE 5.7: Supplementary results all for $N = 6\,000$ and 10 problem instances at different $\alpha = 6$ values.

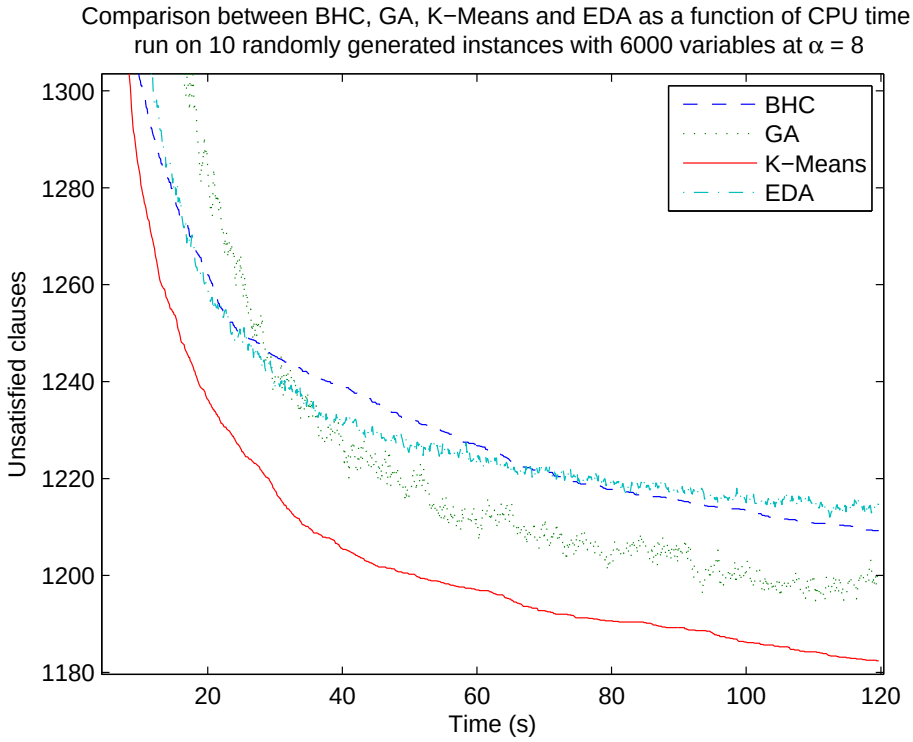

FIGURE 5.8: Supplementary results all for $N = 6\,000$ and 10 problem instances at different $\alpha = 8$ values.

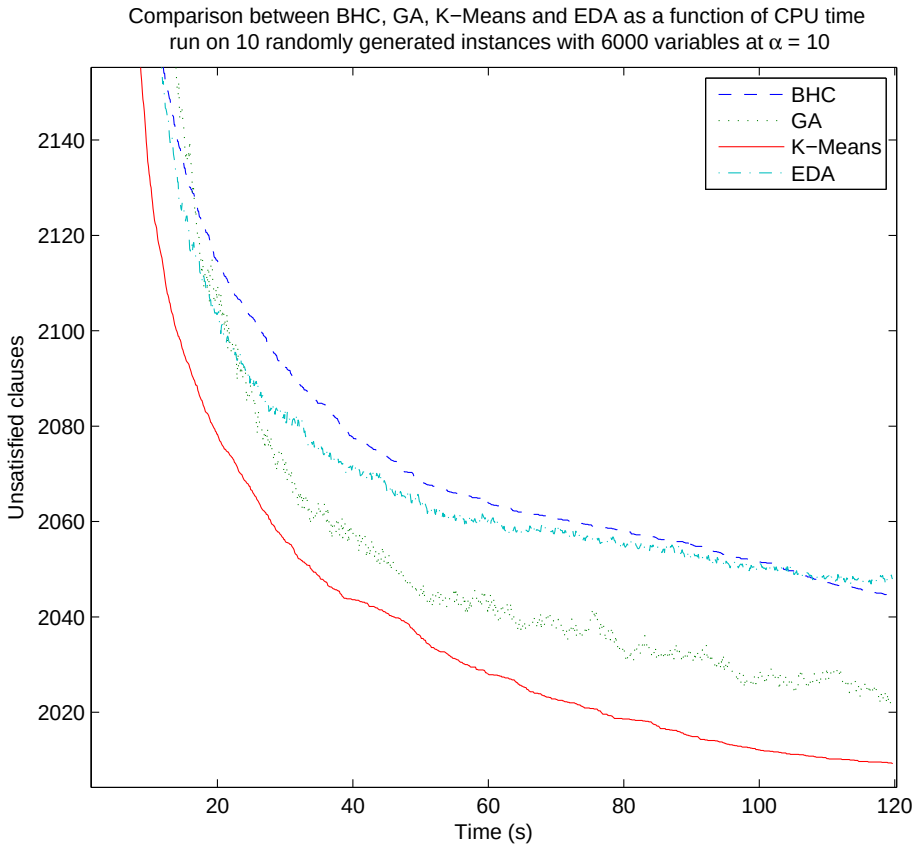

FIGURE 5.9: Supplementary results all for $N = 6\,000$ and 10 problem instances at different $\alpha = 10$ values.

search, EDA and GA could mix individuals from different regions of the solution space. As the search draws to an end, both algorithms usually converge to a particular region of space.

5.5 Peculiarities in the Results

It can be easily seen that there are inconsistencies between the performance of the Hybrid-GA as reported in Table 5.1 and in Figure 5.3. While the tables report results that show poor performance for Hybrid-GA, the graph shows its results to be competitive with CLGH. The difference lies in the starting point of each. Despite having the hybrid-GA use the best 100 assignments obtained from an initial hill-climb, it did not perform better than when it started from a random genepool of assignments. One would think that the former would outperform the latter due to the hill-climb advantage.

Upon observing the behavior of the Hybrid-GA in both cases, we found that although it seems advantageous to have the GA start with good assignments it is quite indifferent to either. When several high cost assignments are obtained via hill-climbing they might

differ in as much as 40% of their variable makeup. With the crossover of the first generation, these differences are amplified, and hence worse costs. In fact, the results of the first generation crossover are so weak that they return the assignments to costs that are on average as bad as a random assignment. After the initial disruption caused by the first crossover, the Hybrid-GA slowly improves on the results by focusing the assignments to a particular region of space. It is as if the Hybrid-GA started from random assignments (though they were subjected to hill-climbing) while the CLGH started right after the first round of hill-climbing.

The rate at which the results are improved is dependent upon the number of hill-climbs between each crossover. In the first case, only several thousand hill-climbs were performed which gave the Hybrid-GA less of a chance to improve on the results. Contrast this with K-means in CLGH which focuses the assignments such that the next hill-climb improves the results drastically; after clustering only a few thousand hill-climbs were required to enhance the solutions. In the second, however, millions of hill-climbs took place giving the GA much more room to adjust the assignments.

5.6 Comparison with Local Search Results

Our algorithm performs well on large problem instances. This makes it difficult to compare with previous results reported in the literature, which tend to concentrate on small instances. The only work we are aware of which studied similar sized instances statistically (i.e. gave results of multiple runs on multiple problem instances) where those given by Zhang (Zhang, 2004). Our algorithm substantially out-performs the results obtained by Zhang. As an example, using BG-Dyna-WALKSAT, Zhang reports that for 6000 variables and $\alpha = 8$ he reduced the number of unsatisfied clauses to 1 597.36. In comparison, we have obtained 1 370.61 unsatisfied clauses using CLGH. This is a 14.2% improvement over his results. Also, to provide a comparison with state-of-the-art local search algorithms we have compared our algorithms to those implemented in UBCSAT[1], which provides a fast implementation of a range of modern algorithms for MAX-SAT.

We tested all algorithms provided by UBCSAT using their default settings. Results are given on 5 randomly drawn MAX-SAT instances with 18 000 variable and $\alpha = 8$. Each algorithm is run five times. We report results for the five best algorithms from UBCSAT on these instances. These algorithms were steepest ascent-mildest descend (SAMD), iterated robust TABU search (IROTS), history with random walk GSAT (HWSAT), greedy SAT with random walk (GWSAT), and GSAT. In each case, we run for 5 minutes. The performance is compared with BHC run for 3.5 minutes and two CLGH runs. In both cases we run 100 BHCs for 200 000 iterations and then used K-means to find 5 centroids (again the choice of 5 centroids was chosen after some preliminary experi-

[1] `http://www.satlib.org/ubcsat/`

mentation). This stage took no more that 15 seconds. We then run a hill-climber on each centroid for 30 seconds (thus the total amount of time spent by these algorithms was 2.75 minutes). The difference between the two tests was that in the first one we used our BHC algorithm after K-means while in the second test we used the GSAT from UBCSAT. The results are shown in table 5.2. As can be seen, CLGH substantially out-performs all other algorithms in UBCSAT despite giving them more time. Note that in table 5.2 we report the specific methods used in the CLGH framework. Here we used BHC/K-Means/BHC, and BHC/K-Means/GSAT.

Algorithm	Overall average	Time (minutes)
SAMD	3696.4	5
IROTS	3583.2	5
HWSAT	3678.2	5
GWSAT	3636.1	5
GSAT	3667.4	5
BHC	3667.1	3.5
BHC/K-Means/BHC (CLGH)	3572.6	2.75
BHC/K-Means/GSAT (CLGH)	3527.8	2.75

TABLE 5.2: Average performance of different search strategies on 5 random instances of 18 000 variable MAX-SAT problem with $\alpha = 8$.

Most of the algorithms seem to plateau after 5 minutes. However, GWSAT (a fast implementation of WALKSAT) continues to find good solutions. We found that it gave similar solutions as CLGH using BHC/K-means/GSAT which were obtained in 2.75 minutes if it was run for around 1 hour. We can see from table 5.2, that the best CLGH improves the cost by 1.55% over the best local searcher IROTS even in 55% of the time given. However, if we take into account the actual time it would take GWSAT to reach the same quality of solution reached by CLGH (which is 1 hour), then CLGH reaches the solutions in 95.4% less time. No doubt some of the algorithms we have tried may have run faster had we optimised their parameters. We have tried to compensate for this by allowing the other algorithms more time. Furthermore, we have not attempted to fine tune the parameters of our own algorithm. The fact that we have obtained such good performance, provides support for our contention that the CLGH method explores the landscape in a fundamentally different way to existing algorithms.

As a final set of tests we have performed longer runs on larger problem instances, $n = 20\,000$ and $50\,000$, at $\alpha = 6$, 8 and 10. In these experiments, we used GSAT followed by K-means followed by WALKSAT. We run $1\,000\,000$ GSAT 200 times. We then performed K-means clustering with $K = 5$. This was followed by $40\,000\,000$ WALKSAT moves on each of the 5 centroids. We report the best result of the centroid. Although we take the same number of GSAT and WALKSAT moves, the majority of time is spent performing WALKSAT, which takes considerably more time to complete a move than GSAT. We used our own implementation of GSAT and WALKSAT. We compare

this with UBCSAT's GWSAT run for 100 000 000 moves and our own implementation of WALKSAT for 1 000 000 000 steps. our WALKSAT appears to have the same performance as UBCSAT's GWSAT, but is considerably faster. We give timings for the algorithms run on an Intel Core 2 Quad Q6600 with 4 GB RAM running Windows Vista. We also compare with UBCSAT's IROTS run for 30 000 000 steps. These results are shown in table 5.3. We observe that CLGH, despite being given considerably less time, out-performs 10^9 iterations of WALKSAT, which in turn outperforms the two top UBCSAT algorithms.

For the two top performing algorithms CLGH with (GSAT/K-Means/WALKSAT) and Our WALKSAT, the number of unsatisfied clauses is reduced by 3.4% in 60% less flips for 20 000 variables with $\alpha = 6$, 0.7% unsatisfied clause at $\alpha = 8$ and 1.07% unsatisified clauses at $\alpha = 8$. For 50 000 variables and $\alpha = 6$, the number of unsatisfied clauses was reduced down by 1.2%, 0.2% unsatisfied clauses at $\alpha = 8$, and 1.5% unsatisfied clauses at $\alpha = 10$. This was not an average over many problems, but they were example tests. We reiterate that these small differences in unsatisfied clauses are not to be taken lightly. To satisfy these very few clauses takes many more flips by a solo-local search algorithm.

Code running our algorithm is publicly available in a package WinSATS[2]. We have also made available the random instances we used in the experiments reported above. These can be found from the link given.

It is important to emphasise that even though there seems to be a small difference in the number of unsatisfied clauses between CLGH and WALKSAT in some experiments, even satisfying this many clauses requires an enormous amounts of flips by WALKSAT or GSAT alone. We reported that it took an hour for GWSAT to reach the results reached by CLGH using BHC/K-means/GSAT in 2.75 minutes. The reason for this is that as the search plateaus it becomes all the more difficult to find the proper variables to flip to improve the result further. We can see this more clearly from Table 5.3 when we applied our WALKSAT with almost an order or magnitude more flips, yet in all the cases shown, the results are still far from those achieved by K-means. True, GWSAT will ultimately find these solutions, but at the expense of many more flips.

5.7 Room for Improvement

In no way have we attempted to improve the performance of the search by improving K-means. We have used K-means in its simplest form to test if it does anything to improve search results. In its simplest form, K-means does not always find the best grouping of assignments. This depends on the choice of the number of centroids that represent the clusters, the initial starting point for each of the centroids and the algorithm that maps the centroids to the clusters. Figure 5.10 shows an example of a centroid in between two

[2] `http://users.ecs.soton.ac.uk/mqq06r/winsat/`

n	α	Algorithm	Total Number of Flips	Time	UNSAT
20 000	6	GSAT/K-Means/WALKSAT	2×10^8(GSAT) + 2×10^8 (WALKSAT)	17.8 min	1 539
20 000	6	UBCSAT GWSAT	10^8	51.0 min	1 602
20 000	6	Our WALKSAT	10^9	1.08 hours	1 593
20 000	6	UBCSAT IROTS	3×10^7	52.3 min	1 631
20 000	8	GSAT/K-Means/WALKSAT	2×10^8(GSAT) + 2×10^8 (WALKSAT)	24.1 min	3 916
20 000	8	UBCSAT GWSAT	10^8	51.9 min	3 953
20 000	8	Our WALKSAT	10^9	1.56 hours	3 944
20 000	8	UBCSAT IROTS	3×10^7	50.8 min	4 049
20 000	10	GSAT/K-Means/WALKSAT	2×10^8(GSAT) + 2×10^8(WALKSAT)	31.2 min	6 621
20 000	10	UBCSAT GWSAT	10^8	53.1 min	6 722
20 000	10	Our WALKSAT	10^9	1.94 hours	6 693
20 000	10	UBCSAT IROTS	3×10^7	50.3 min	6 699
50 000	6	GSAT/K-Means/WALKSAT	2×10^8(GSAT) + 2×10^8 (WALKSAT)	30.0 min	8 684
50 000	6	UBCSAT GWSAT	10^8	1.98 hours	8 853
50 000	6	Our WALKSAT	10^9	1.82 hours	8 789
50 000	6	UBCSAT IROTS	3×10^7	2.15 hours	8 821
50 000	8	GSAT/K-Means/WALKSAT	2×10^8(GSAT) + 2×10^8 (WALKSAT)	36.0 min	15 955
50 000	8	UBCSAT GWSAT	10^8	2.04 hours	19 194
50 000	8	Our WALKSAT	10^9	2.20 hours	15 992
50 000	8	UBCSAT IROTS	3×10^7	1.96 hours	16 321
50 000	10	GSAT/K-Means/WALKSAT	2×10^8(GSAT) + 2×10^8 (WALKSAT)	47.0 min	23 838
50 000	10	UBCSAT GWSAT	10^8	2.11 hours	24 206
50 000	10	Our WALKSAT	10^9	2.83 hours	24 075
50 000	10	UBCSAT IROTS	3×10^7	2.23 hours	24 384

TABLE 5.3: Performance of Algorithms for n = 20 000, 50 000 and α = 6, 8 and 10. UNSAT is the number of unsatisfied clauses in the assignment found by the algorithms.

clusters. Starting the search from this centroid is unlikely to produce good solutions. We believe that the results of the search can be significantly improved if each cluster is made more homogeneous. A plethora of literature is devoted to optimizing clusters (Arthur and Vassilvitskii, 2006; Pelleg and Moore, 2000; Wagsta *et al.*, 2001; Pea *et al.*, 1999; Ding and He, 2004).

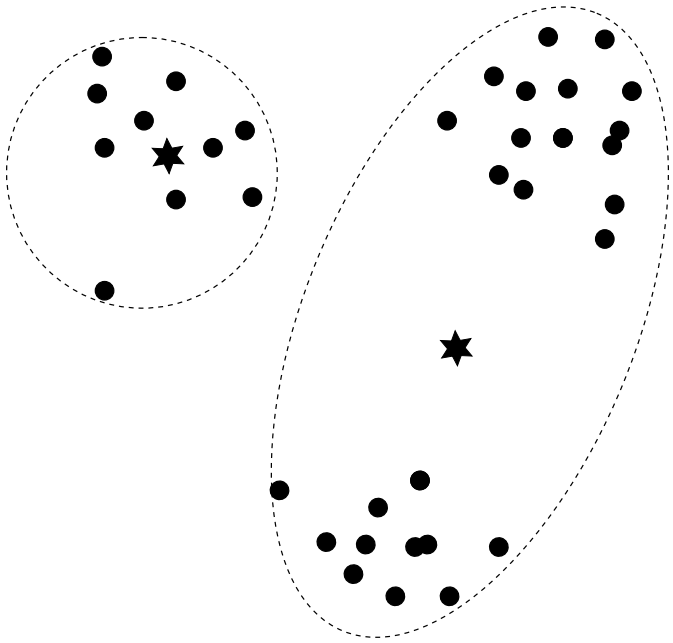

FIGURE 5.10: An illustration improper clustering of solutions. The solid circles represent the solutions, and the stars are the centroids. One possible way of clustering where the centroid would not land within a cluster, and hence would not provide for a good starting point for the search. Note: K-means divides the space into Voronoi regions (Du *et al.*, 1999). The clusters are encased with dashed ellipses for simplicity.

Having stated that, care must be taken in choosing the point at which the K-means is applied. K-means yields the best results if the assignments have been sufficiently improved by hill-climbing. Having them less or more mature will render K-means drastically less effective. If the initial random assignments were subjected to a few hill-climbs they would have very little beneficial information to use in clustering. In the analogy of the fitness landscape in Figure 4.6 a few hill-climbs would amount to moving on the ground, and never reaching the foothills. Clustering these points would likely have the centroids on the ground also. On the other hand, performing a large number of hill-climbs on each assignment simply saturates the assignments such that averaging a cluster, with these highly correlated points, creates centroids within the vicinity of the local optima. There would not be a whole lot of differences to exploit. It would be like having the assignments very close to each other on one side of a mountain. In our tests we have found that the best point to apply K-means is when the Hill-Climb begins to slows down, and before reaching a plateau, Figure 5.11.

We call this point of application the "Sweet Spot". Although there is a range of points

where the K-means can be applied to produce good results, the application of K-means at the sweet spot produces the best results. Outside this range, we obtain poor results. This is akin to the audible experience obtained in the cinema. Outside the cinema room no sound is heard. Inside, the sound is heard at varying levels. However, the best location for a more intense listening experience is the sweet spot where all sound waves produced by the speakers meet in perfect synchronisation.

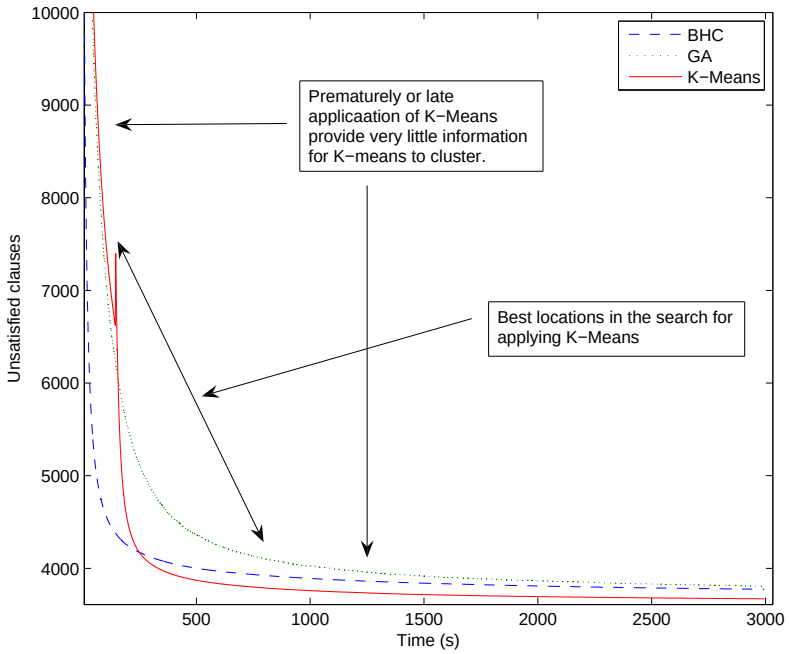

FIGURE 5.11: The best locations for the application of K-means is before after the maturity of the search, and before saturation.

Tables 5.4 and 5.5 show the result of applying K-means before and after the sweet spot (before and after the range more specifically). These are individual tests run on problems with 4 000 variables at $\alpha = 8$. Initially, BHC is applied for 10 000 iterations starting from 100 different random assignments, then K-means or averaging is used, and finally, another round of BHC with 20 000 iterations is applied to 10 centroids. Table 5.4 shows the results when K-means is applied slightly before sweet spot. The best results are shown in bold. We observe that for the 20 problems examined, CLGH performs better than ALGH in 65% of the problems. In Table 5.5, we report results of applying K-means after the sweet spot. Here, CLGH outperfomed ALGH on only 55% of the problems. By comparing these results with ones we obtained before the sweet spot, we find that in all tested problems, CLGH outperfomed ALGH.

Another test was performed was on a relatively small problem. We worked on these problems to allow BHC to easily reach a plateau. We applied K-means long after reaching the lowest cost obtained by BHC. We find that the GLCH sometimes produces

First BHC	CLGH	**ALGH**
1479	**830**	840
1474	**820**	846
1452	814	**798**
1456	**828**	832
1474	**808**	836
1459	**815**	818
1461	834	**833**
1451	**807**	811
1454	**819**	830
1450	831	**830**
1495	**857**	860
1454	827	**826**
1444	811	**808**
1457	**810**	814
1453	**801**	814
1469	826	826
1449	**821**	824
1452	**817**	821
1499	873	**852**
1470	**822**	839

TABLE 5.4: The performance of CLGH in comparison with ALGH applied slightly before the sweet spot in the cost vs time plot. The results show that there is very little difference between CLGH and ALGH of randomly selected solution points.

worse results than the initial search. This is shown in table 5.6

5.8 Other Experiments

Two other investigations centred on different ways of applying CLGH to the solution space to determine if we can produce finer results. From these experiments we gained further insight into how CLGH works. We will see that these experiments set CLGH apart from conventional evolutionary algorithms. One experiment, and somewhat related to EDA, iteratively focuses and defocuses centroids, and the other relied on clustering of only the fittest solutions.

5.8.1 Focusing/Defocusing

One of the ways we can think of K-means is as a focusing operator. When averaging a cluster, we are finding the focal point of that cluster. Before rounding each of the variables to 0 or 1 we have a probability vector. Just as in UMDA, we use this centroid to generate new starting points for the search. Using the probability of the variables

First BHC	CLGH	**ALGH**
1129	1038	**992**
1128	1011	**1010**
1152	**1057**	1079
1131	**996**	1005
1127	**1020**	1023
1119	988	988
1143	**1009**	1052
1114	1001	**991**
1141	1023	**1000**
1163	1038	**1026**
1106	1002	**993**
1121	**985**	999
1127	**989**	1020
1127	**1005**	1015
1143	1077	**1048**
1132	**993**	1003
1131	**1015**	1016
1154	**973**	1040
1123	**1006**	1032
1158	1060	**1056**

TABLE 5.5: The performance of CLGH in comparison with ALGH applied slightly after the curvature in the cost vs time plot.

First BHC	**GLCH**	**ALCH**
204	**201**	205
193	195	195
216	215	214
204	**197**	207
201	**200**	206
192	201	201
202	**207**	209
216	218	211
206	207	204
203	**206**	212

TABLE 5.6: When applying K-means after the BHC had plateaued the results worsen again. The worst results are denoted in bold font.

we create several new solutions by setting the variables to either 1 or 0. This in effect, defocuses the centroid to many different solutions around it. An illustration of this is shown in Figure 5.12. Once we generate different solutions, we restart a local search on these solutions. Unlike EDA, however, we apply another round of hill-climbing, then K-means is applied again to these guided solutions, then we repeat this process over. This is like focusing and defocussing of the results.

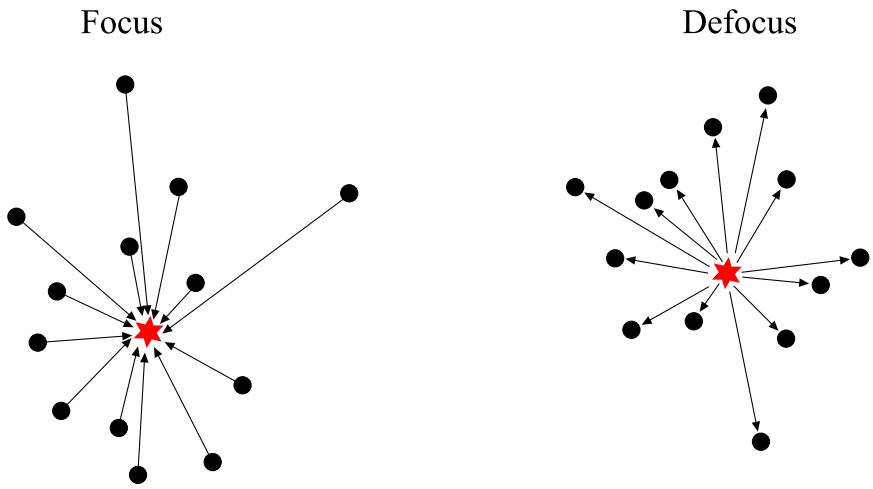

FIGURE 5.12: Illustrates the K-means as a focusing operator. Using the solution points we can focus in centroid. From the centroids we can defocus the into new points in space.

We ran this test over many problems. The results of the search did not improve at all over a single application of K-means. It seems that the centroid or the focal point of the first search provides the best starting point for the search. This shows how powerful K-means in CLGH is as a one time operation, and it demonstrates that the focal points of each cluster provides a solution that does not require any restarts or iterations. Compare this with other evolutionary algorithms. Evolutionary algorithms rely on an iterative process to gradually improve the results. In previous cases, when GA or EDA were applied, they were hybridised with hill-climbing. After every few steps of hill-climbing, GA and EDA operators were applied. With CLGH it is only necessary to find the centroids once. These points turn out to be best known points for starting a new search.

5.8.2 Clustering Fittest Solutions

An interesting result we obtained was when we clustered only the fittest solutions instead of relying solely on hamming distance. Prior to this experiment, we clustered all solutions. K-means did not distinguish between fitter an less fit assignments. It grouped the solutions based entirely on Hamming distance. We added another requirement to the clustering in this test. We pick the top μ solutions, and then cluster these.

We have found that *mu*-CLGH was less effective in this case than averaging of random solutions, and at times less effective than solo-BHC. This goes against the conventional view that fitter solutions provide more advantage than less fit ones as in GA and EDA. We know from Figure 4.4 that on average, the Hamming distance of the higher cost solutions are closer to the quasi-global solutions. This, however, is the average behaviour of the Hamming distance versus cost. This means that some less-fit solutions are close in Hamming distance, and some more-fit solutions are further away from the quasi-global solutions. In effect, those solutions whose average Hamming distance is closer to the quasi-global solutions, which do not necessarily have a higher cost, could contribute to the cluster, and ultimately to the formation of better centroids.

This is yet another distinguishing aspect of CLGH in comparison with other evolutionary algorithms. Where it seems advantageous to use the fittest solutions for the creation of the next individuals in the population in both GA and EDA, CLGH looks for structure in the arrangement of solutions space without strict emphasis on choosing the fittest solutions. Clearly, we cannot apply CLGH to randomly generated solutions before driving them into better costs. In spite of that, the cost do not have to be optimal. In some experiments we achieved the opposite effect. Optimum solutions did not provide a clustered structure that achieve good centroids. More on this in Chapter 8.2.

5.9 WinSATS Application

In a parallel track to our work we have developed a Windows based SAT Solver application. Although the majority of the tests for this work were performed using the Borland C++ compiler in the command line environment. We have developed this application using the GUI based Borland turbo C++. The GUI provides a simple interface for user interaction. In addition, the program was developed in a way that would allow for easy reporting of results (in a spreadsheet view). Also, since one of our goals was to include graphs that would report the performance of local search algorithms, Windows seemed as good a candidate as any other operating system for this task.

The application is capable of loading CNF files or generating random CNF problem instances, it provides several search methods including our own approach, and generates a results report sheet, see Appendix B.1. This application has been tested against UBCSAT which contains some of the most advanced and up-to-date local search algorithms. We have shown that our GSAT and WALKSAT perform an order magnitude or more flips in the same time than the implementations found in UBCSAT for large instances. In addition, we have shown that using CLGH we are able to outperform every algorithm there is in UBCSAT even with the use of the basic hill-climbing algorithms such as BHC, GSAT and WALKSAT.

The main difference between WinSATS and UBCSAT is in the way they were designed

to handle future algorithms. While UBCSAT was designed to include only solo-search algorithms such as GSAT, WSAT, IROTS, HWSAT, and so on, the WinSATS application was developed to include both solo-search algorithms and algorithms that are based on populations. This makes it all the more comprehensive. Although we currently have K-means as the only populations based method incorporated into WinSATS, we will add other algorithms such as GA, EDA and particle swarm and Quantum Evolutionary Algorithm (QEA) in the future.

The current version of WinSATS is 2.04. The application has been downloaded by 664 users from all around the world at the point of writing this thesis. We have not had a single bug reported by any of the users. We hope that the WinSATS application becomes a benchmark tool that researchers use. In addition to the WinSATS application we have developed a web page describing the satisfiability problem for other researchers, Appendix B.3. Along with this page, we have also included the user manual for WinSATS, benchmarks, and comparisons. It is our goal to eventually include the source code of the application to have it developed by the SAT community.

Chapter 6

The Applications of LGH to the TSP Problem

It would be interesting to see if the application of LGH extends to other NP-Complete problems. Is LGH generic enough that it can refine the search in other problems as well? We could ask whether the landscape of other NP-Complete problems is similar to that of MAX-SAT, and hence LGH could be applied to them equally well. Do other NP-Complete problem share the same clustered structure of solutions? It is natural to assume that real problems (opposed to artificially constructed problems such as toy problems) would tend to group good solutions in close proximity. A multiple clusters of solutions is a possible configuration of the solution landscape where CLGH could work naturally. However, another possible configuration is that although solutions group, they could very well form a single cluster. In which case ALGH would also work.

We have a plethora of NP-Complete problems with which we can test LGH. The most famous of these is the Traveling Salesman Problem (TSP). Just as in MAX-SAT problems, exact and non-exact methods have been developed to solve it. The best of these methods rely on local optimisation techniques. Our goal here is to see the effect of the application of LGH on TSP, and although we intentionally did not perform rigorous analyses on the structure of solutions for this problem, we will supplement this work with a few experiments. We shall see in the next section that it is not clear whether the solution space of the traveling salesman problem follows a many-clustered view.

6.1 The Traveling Salesman Problem

The traveling salesman (TSP) problem is defined as the shortest round trip traversing a number of cities such that each city is visited once. This combinatorial optimization problem is well known to be NP-Hard. It has been studied extensively, and a prodigious

amount of algorithms were designed to solve it, some, with fantastic efficiency (Helsgaun, 2006; Applegate *et al.*, 2006). However, it remains computationally difficult, and as with satisfiability, the size of the problem greatly affects the efficiency of finding optimal solutions. Nowadays, TSP problems are large enough that they require either more computing power or more efficient algorithms.

Despite TSP being computationally hard to solve, from 1950 up to 2006 the TSP challenge problems that have been solved grew significantly in size (Applegate *et al.*, 2006). With the increase of computer power and, more importantly, improved search techniques these large problems have been solved to optimality. The largest problem that was solved in 2006 contained 85,900 cities. The next challenging problem has as many as 1,904,711 cities, and although no exact solutions have been found for this problem to date, the closest approximation is 0.058% away from the Held-Karp lower bound (Valenzuela and Jones, 1997). It is remarkable that such large and rapidly growing problems have been solved considering that they belong to the NP-Hard category.

TSP has a wide range of real world applications which makes it more desirable to solve. It directly maps to drilling circuits on microprocessors boards, and Vehicle Routing Problem (VRP) (Toth and Vigo, 2002) (While I was working for FedEx in the US during my summer breaks as an undergrad I was working with large databases of truck routes. We were trying to find shorter distances for the drivers. It never occurred to me then that I was working on the Traveling Salesman Problem). With modifications, it can be applied to a plethora of other problems such as planning, guiding lasers for crystal art, Aiming telescopes and X-Rays, logistics, and Gene sequencing (Applegate *et al.*, 2006).

TSP is formally defined as a set of cities $c_1, c_2, ..., c_N$ or nodes with the distance $d(c_i, c_j)$ between each two distinct cities or edge. The permutation of the cities generates a set of edges that establish a Hamiltonian cycle through the cities. It is required to find the permutation with the minimum tour length (David, 1990),

$$\min \sum_{i=1}^{N-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(N)}, c_{\pi(1)}) \tag{6.1}$$

The distance measure $d(c_i, c_j)$ can be Euclidean in 2 or 3 dimensions, geographical, Manhattan, or it can be determined using a special function. In addition, the problem can be either symmetric or non-symmetric. That is, in a symmetric problem the distance $d(c_i, c_j) = d(c_j, c_i)$, while in the non-symmetric case $d(c_i, c_j) \neq d(c_j, c_i)$ is allowed. For our purposes we used symmetric problems with 2-D euclidean distance measure, which is the easiest form of distance measure to start with. We provide an example of a TSP problem in Figure 6.1. This is the rat783 problem solved to optimality. This problem is a symmetric 2-D problem.

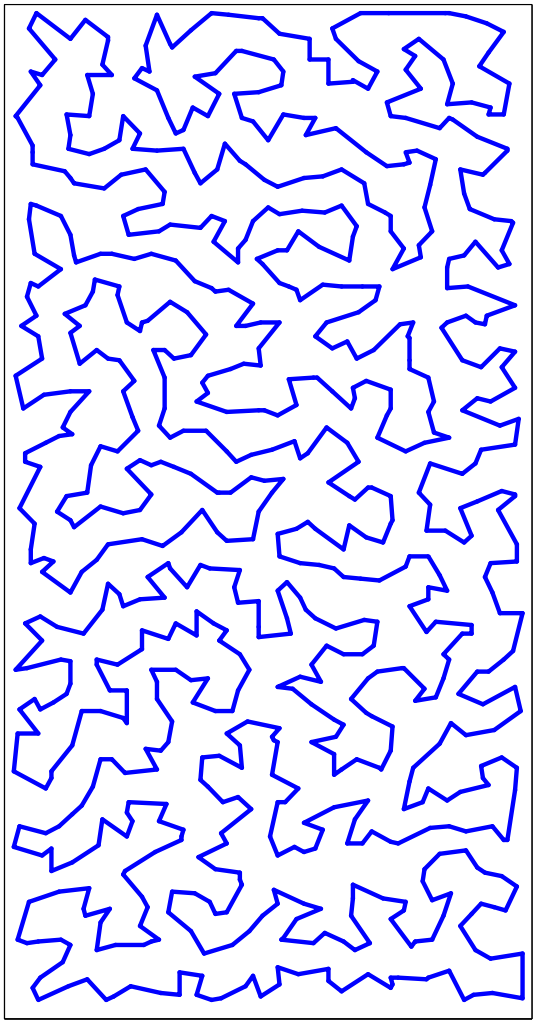

FIGURE 6.1: The rat783 problem with 783 nodes. The tour that is shown is optimal.

The goal of our tests that follow is not to compete with algorithms that have been gradually and systematically developed over the past two decades. This would be a futile exercise as this problem has been tackled by a prodigious number of researchers over the years, and the algorithms that have been developed to solve it have been made extraordinarily efficient. Here, we aim to investigate the effects of applying LGH to TSP, and to broaden our understanding of the performance of LGH on it.

We have seen that applying CLGH to the satisfiability problem produces exceptional results compared with many stochastic algorithms. We have shown it to gives a clear edge even over the state of the art methods. In spite of these results the application of clustering or averaging in LGH are not stand-alone techniques. They can be used to enhance the performance of other search algorithms. This justifies our goal for not going after maximum efficiency since we can in principle, if the K-means or averaging exploit the the solution structure, if it exists at all, utilize them within well known stochastic

algorithms.

This view is not without merits. By taking a set of tours, and considering each edge that appears most frequently in them as the set of edges that form a backbone one can use these edges to form a good starting point for a local search algorithm (Zhang and Looks, 2005). However, the edges are not suspended from the search by locking them in. Rather, the frequency of the appearance of the edge is given a probability that makes it malleable. The backbone or rather pseudo-backbone information that is extracted from a multiple number of solutions cannot by itself be used to solve TSP problems. It must be incorporated into a local search algorithm that would benefit from it to bias the search. Zhang's backbone method (Zhang and Looks, 2005) has been successfully incorporated into one of the most well known search TSP heuristics, the Lin-Kernighan-Helsgaun (LKH) method (Helsgaun, 2006). It has been shown to give state-of-the-art results in VLSI problems. As a stand alone algorithm the backbone guided method would not work; it would have to be included with other local search algorithms.

6.2 Local and Constructive Search Algorithms

Several local and constructive search algorithms have been proposed to solve the TSP problem with varying degrees of effectiveness. *Nearest Neighbour* is a simple greedy constructive heuristic, whereby a city c_i is chosen arbitrary as the starting city, then the closest city c_j is chosen as the next city in the tour, and then the next closest, c_k, and so on. This is repeated for all the cities until a full cycle is completed. Although this greedy method might yield good solutions, it usually produces suboptimal solutions, and in fact might even produce unique worst tours (Bang-Jensen *et al.*, 2004).

Another method is the *Nearest Insertion*. In this method we start with two cities that are closest to each other. We form the initial tour $c_i - c_j - c_i$. The next step is to find the city c_k that is closest to any node in the sub tour c_i or c_j. Insert that city into the subtour to create the shortest distance possible. The same steps are repeated until the tour is completed. Another variation of this algorithm is the *Farthest Insertion*. Start with cities c_i and c_j that are furthest from each other. Create a sub tour $c_i - c_j - c_i$. Then find the furthest city c_k from both of c_i or c_j, and insert it into the subtour. Repeat for all the cities until a full tour is create (Golden *et al.*, 1980).

There are other variations such as Cheapest Insertion, Arbitrary Insertion, Convex Hull (Golden *et al.*, 1980) and simulated annealing (David, 1990). Most of these early algorithms have become obsolete, or they have been infused with other algorithms, but on their own they have been used for their illustrative value rather than their practicality. They are used less today on their own in favor of more robust algorithms. One of which is the Lin-Kernighan Algorithm. This method has become the basis for some of the most sophisticated modern TSP solvers.

More recent solvers are population based. Some use genetic algorithms (Kaur and Murugappan, 2008; Baraglia *et al.*, 2001; Huai-Kuang *et al.*, 2004), other methods use Ant Colony optimization (Dorigo and Gambardella, 1997). Population based algorithms usually require high execution times by design. Nevertheless, the biggest obstacle population based methods face is the representation problem. We too had our share of troubles in determining the best way to represent tours in such way that would fit into our clustering mechanism.

6.3 *k*-Opt and Lin-Kernighan

One local search method that is applied to TSP is the k-Opt method. We start with a random tour, then a systematic exchange of k edges with another k edges is performed such that the result of each exchange yields a shorter tour. Suppose that we have 9 cities with the edges denoted with $x_1, x_2, ..., x_9$ as in figure 6.2. With 2-Opt we replace the edges x_1 and x_2 with the edges y_1 and y_2 given that $y_1 + y_2 < x_1 + x_2$. This process is performed until all possible 2-Opt exchanges are exhausted. This yields either a globally or locally optimal solution (Lin and Kernighan, 1973). As k becomes larger, the complexity of the search also becomes large, since the permutations of finding the proper k exchanges increase in number. Therefore, the number of k exchanges are restricted to 2-Opt or 3-Opt in most implementation of the k-Opt algorithms (Helsgaun, 2006).

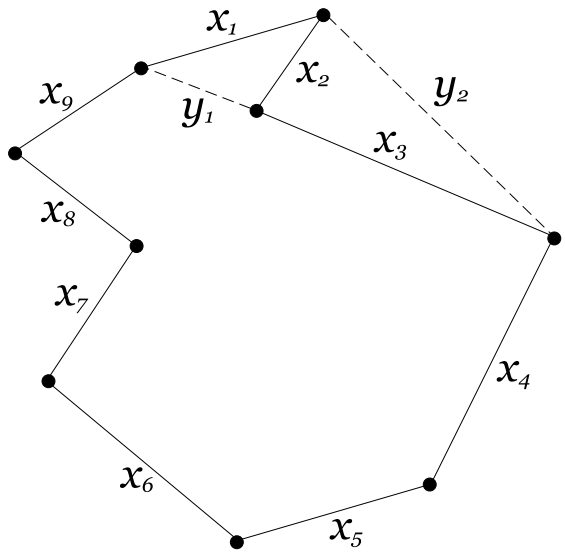

FIGURE 6.2: Applying 2-Opt to a tour by exchanging two edges for another two edges such that the resulting tour length is smaller.

The total number of exchanges performed in k-Opt is $E = 2^{k-1}(k-1)!$ where E is the number of exchanges (Helsgaun, 2006). Figure 6.3 shows some of the possible 4-Opt exchanges for 4 edges. Just for a particular set of four edges in the tour, the number of possible exchanges is $E = 48$. Here, the solid lines signify a partial tour starting

with a particular city and ending with another. There could be other cities that are in between, but they are not included for clarity. The dashed lines represent edges going from one city to the next with no other cities in between. Each partial tour is labeled with a number from 1 through 4. The sequence of labels below each exchange in the figure shows the order of the partial tours going clockwise. We can think of swapping two edges in terms of flipping an entire partial tour. If the tour is to remain unflipped, then it is assigned a positive sign, (e.g. +1). On the other hand, if the partial tour is flipped it is assigned a negative sign. This makes the process of 4-Opt clearer than if we had to think of swapping the edges themselves.

It should be made clear that despite the number of exchanges for a 4-Opt is small, $E = 48$, the number of 4-exchanges that must be done is very large when we have to consider taking all possible combinations of 4 edges in a tour. That is, not only do we have to perform exchanges between 4 edges, we also have to do this for every possible 4 edges in the tour against all other 4 edges. Therefore, instead of performing these exchanges on all possible edges, only a subset of the edges are considered. One way to limit the number of edges is to only apply 4-Opt to neighbouring cities, since it is clear that adjacent edges are more likely to be swapped with each other than ones on the opposite side of a tour. We will talk more about that in section 6.6.

Lin-Kernighan (Lin and Kernighan, 1973) proposed a heuristic that would allow for variable k exchanges where $2 \leq k \leq N$. The LK algorithm starts with a random non-optimal tour. Then, an edge by edge exchange is performed as long as the total length of all the edges exchanged produce a shorter tour. The process starts with an edges x_i, and it is exchanged with another edge y_i such that this would maximize the improvement. If initial exchange was successful, then the process is repeated with next two edges. This is repeated for $i = 1, 2, ...k$ until there is no more room for improvement. This removes the restriction that k be set a priori, since it is not known beforehand how many edges are to be exchanged. In addition, it alleviates the remarkably large number of permutations required by testing each and every k exchanges. Although the number of k-exchanges can in principle extend to any number, however, because of the other computational costs, a top limit has been set for k. In the case of LKH, empirical tests have been carried on the effect of increasing k up to 8, and it has been shown that the CPU time increases exponentially with k (Helsgaun, 2006).

The LK algorithm has become an integral part of the most successful algorithms. The LK method as it stands has been known to find solutions of the traveling salesman problem within 1-2% of the optimum (Helsgaun, 2006). Moreover, current implementations such as LKH are ever more effective in solving large TSP problems within reasonable times. LKH removed some of the restrictions imposed by Lin-Kerninghan, freed the algorithm from costly computations, and improved solutions. We will not delve into the details of the numerous improvements and additional implementations made to the original LK algorithm as this has no bearing on the work we have done. What we have done

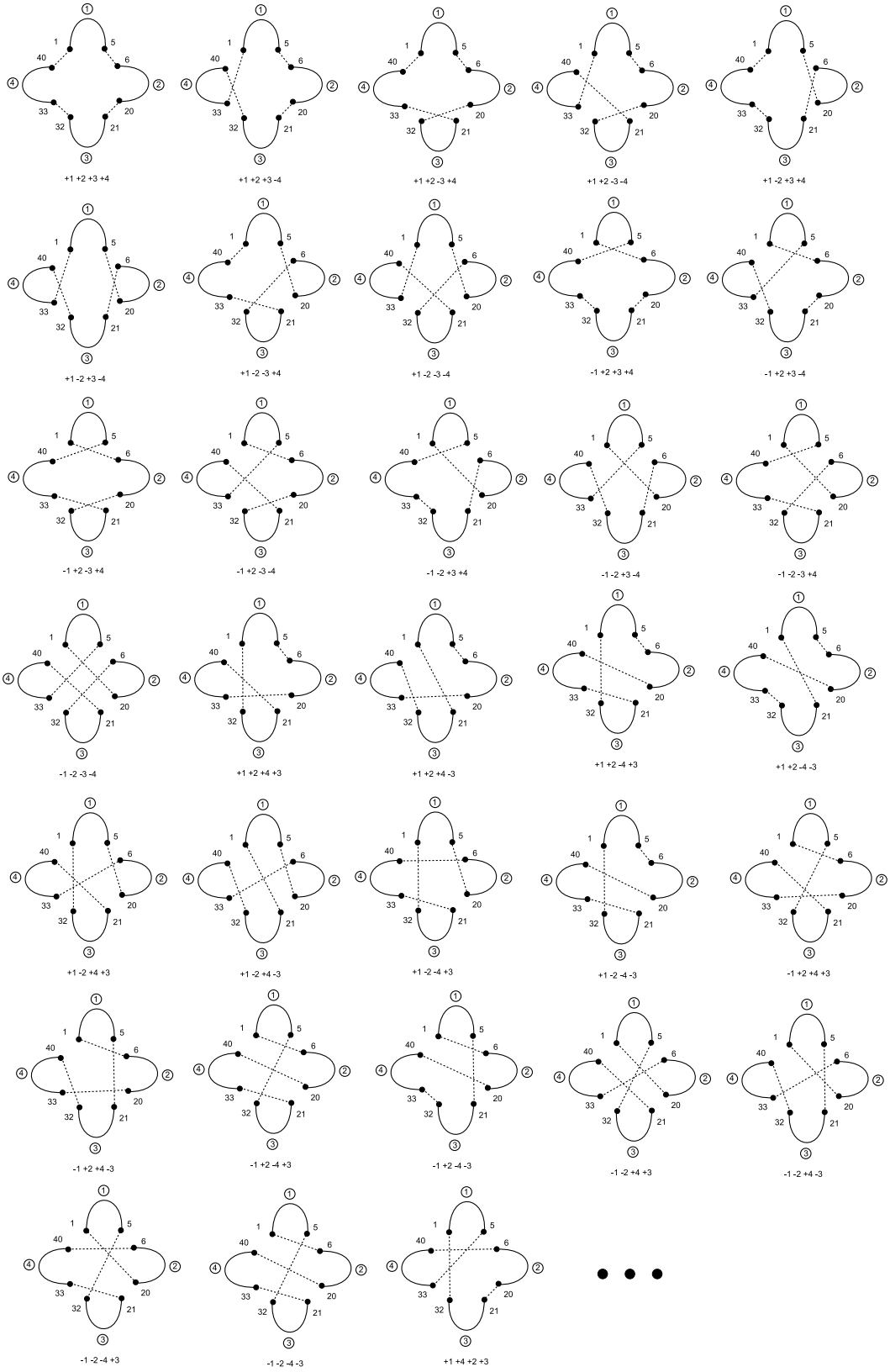

FIGURE 6.3: Shows some of the possible combinations of a 4-Opt exchanges. This operation shows enormous number of permutations when exchanging 4 edges. This number becomes amplified with we consider that this is done for all $\binom{N}{4}$ subset of edges.

was to apply only 2-Opt moves within the framework of an Iterated Local Search (ILS) algorithm, and performed comparisons within this framework. We will discuss this in the section 6.6.

The LKH algorithm is an algorithm that uses a number of different concepts to solve the traveling salesman problem efficiently. It contains several state of the art methods. A method that has been added recently to the LKH is Zhang's Backbone Guided Local Search method (Zhang and Looks, 2005). It has been shown to be particularly effective with VLSI TSP problems. Just as in the SAT problem, backbone information is gathered from multiple LK searches to build backbone information. Since the LK method reaches local solutions that are within a small percentage of the optimum tour, the frequency of the edges over several locally optimum tours is accumulated.

These frequencies are turned into the probability p of an edge being the globally optimal solution. The more an edges appears in the different tours the higher the probability that the edge is part of the optimal tour and vice versa. However, once the backbone information is gathered, Zhang does not fix these edges based on their probabilities. Rather, the distance between two nodes is made elastic based on the probability of the edge being an element of the optimal tour. If the length of the edge was l_i, then by computing the length of the tour, the adjusted length becomes $l'_i = l_i \cdot (1 - p_i)$. This morphs the lengths of edges such that the edges with higher frequencies are reduced in size so as to preserve them in an exchange, and the less frequent ones are elongated to allow them to be exchanged more readily.

6.4 LGH and TSP

The application of LGH in TSP is a little more involved than the way it is applied to MAX-SAT. This process requires that we look the at the sequence of edges in a tour instead of the sequence of cities. This was proposed by (Peter and Bernd, 1999). They found that the correlation between tours represented by their city-sequence is lower than those represented by edge-sequence. In city-sequence representation, each tour is represented by a sequence of cities as they are presented in the tour. They are labeled from 1 through n. Figure 6.4(a) illustrates a tour sequence of 20 cities. As a consequence of having this representation, the sequence can be rotated cyclically while maintaining the same tour length, Figure 6.4(b).

Because the same tour can be represented in many different ways, using the city-sequence representation for clustering would be impractical. This is because even if two solution-tours are close to each other in terms of the order of the cities (having very close fitnesses), but are rotated differently, their points in the solution space will be in two far apart locations. In this case, clustering would not be able to group these two similar solutions.

4	1	20	3	6	19	12	5	10	8	9	2	16	7	11	15	17	18	13	15

(a) Tour sequence

6	19	12	5	10	8	9	2	16	7	11	15	17	18	13	15	4	1	20	3

(b) Same tour sequence rotated

FIGURE 6.4: The same tour sequence can be represented differently by cyclically rotating the sequence of cities.

Two studies, the Maximal Preservative Crossover (MPX) (Mathias and Whitley, 1992) and Distance Preserving Crossover (DPX) (Freisleben and Merz, 1996), used crossover of parent individuals in a population in GA. Both of these methods used the city-sequence string to represent the individuals in the population. In another study, edge recombination was used (Whitley *et al.*, 1991). We adopt a string of the edges between each city and the next. In this case each tour of length n cities would be represented by a string of edges that is $n(n+1)/2$ edges long. An example is shown in Figure 6.5. When an edge exists between two cities the edges string is set to 1 otherwise 0 for these two cities. This fits very well with K-means as we shall see next.

FIGURE 6.5: Representing a tour by its edges. Instead of using the sequence of cities vector, edges between cities can be stored in a vector. With this, we avoid the cyclical problem.

The next step is to apply CLGH to these strings of edges. Again, we use K-means in our CLGH. K-means groups similar edge configurations with each other into K different clusters. With this simple representation we can directly use the Hamming distance as a measure of similarity between a tour and the centroid. To average each cluster, we count the frequency of the appearance of the each edge in all of the suboptimal tours in each group. The frequency count of the edges is used to generate the centroid tour. These centroids do not necessarily create a valid tour. Hence, we reconstruct a valid tour from this centroid.

6.5 Tour Reconstruction

We use the centroid in two ways. First, we reconstruct a valid tour from it by keeping the edges with the highest frequencies while maintaining a proper tour. At times, more than two edges are connected to a single city appear with a high frequency. We make sure that no invalid tour is created even though we obtained unusual centroids from averaging.

Second, we preserve the frequency information of the edges. This information is crucial for the second round of the search. We use this information to apply Zhang's method.

Constructing the centroid is a laborious work. It requires several steps, but because it is a one time operation, the overhead of processing the proper tour is minimal. The first step in reconstructing the tour, is to sort the edges in the order of their frequency counts. There are two important points that need to be checked when building a proper tour. First, centroids might have several edges connected to one city. The number of edges connected to a particular city is dictated by the frequency of the appearance these edges. This invalidates the tour, and in the process of reconstruction, the extra edges are discarded. The second, and also important step, is to make sure that as we add edges we do not create a partial closed tour, Figure 6.6. As the edges are accumulated based on their frequencies, their might be two cities that close the loop before all the edges are added.

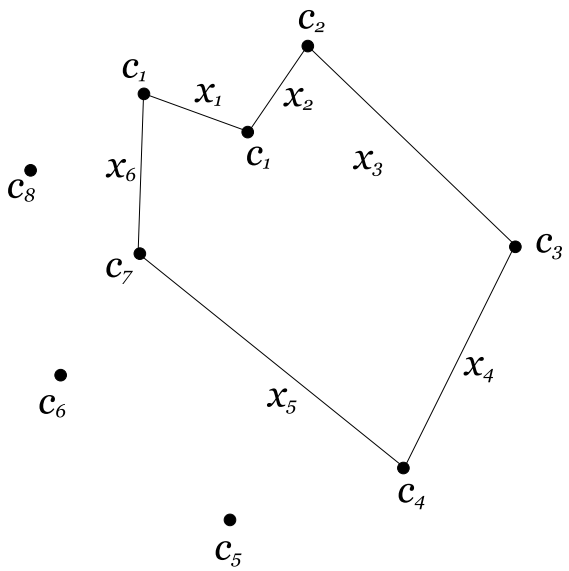

FIGURE 6.6: Reconstructing a tour from a centroid might create a partially closed tour. This is because a centroid is not always a proper tour.

Using disjoint sets, which is an efficient data structure to solve the equivalence problem, we ensure that a tour is not closed prematurely (Weiss, 2007). When a pair of elements (a, b) are related by some relation, $a \sim b$ such that $a, b \in S$ where S is set, and $\sim$ is relation on the set S, then we can call this relationship an equivalence relationship if it satisfies the following three properties:

- Reflexivity: $a \sim a$
- Symmetry: $a \sim b$
- Transitivity: $a \sim b$ and $b \sim c$ then $a \sim c$

All these properties apply to the cities in a TSP problem where the edge is the relation, and the cities are the elements.

After sorting the edges by their frequencies, we take each edge in the order of its appearance in the ordered list, and insert it in the set of edges. Initially, all the edges will be in different sets, and no two sets have a relation, i.e., $S_i \cap S_j = \emptyset$. When an edges is inserted via the union of the two cities, c_i and c_j that are connected to it, two cities are assigned to the same set, S_i. As we add more edges, more relations are added to the sets. Sets are merged into $S_k = S_i \cup S_j$ if two cities $c_i \in S_i$ and $c_j \in S_j$ are connected by an edge. Before adding new edges, it can be determined if the two cities are in the same set. If they are, then we have a closed tour. This situation is avoided before the inclusion of the new edges.

There is another very important verification that is carried before inserting an edge into the sets. It must be checked that the edge is not connected to a city that has two edges already connected to it. The disjoint set algorithm does not check for this.

The final step in the reconstruction of the tour is to check if there any orphaned cities. This happens when some sets are disjoint. This means that the tour is incomplete, and there would be orphaned cities. The reason this happens is because the centroids do not usually produce proper tours. They are a mixture of edges from different tours with their respective frequencies of appearance. Building a tour from the highest cost edges will likely produce partial tours. We fix this by inserting edges between the tour fragments. These tours might not be best selection of edges, however, they will be corrected for in the next round of the local search.

6.6 Experimental Results

Developing a TSP search engine that would make use of CLGH requires 3 stages. First, an initial population of random tours is searched for locally optimal solutions. Second, these tours are clustered and each cluster is averaged to create centroids. Finally, the centroids are put into a second round local-search with the use of the probabilities of the edges. For the local search algorithm we have used Iterated Local Search (ILS) (Ramalhinho *et al.*, 2000) with 2-Opt. The local search algorithm works as in Appendix A Figure A.5.

The local search that is incorporated into this algorithm is a basic 2-Opt. We perform a systematic 2-Opt on all the cities to ensure that no two edges are overlapping. When there are no more edges that can be swapped, the tour is a locally optimal solution. Initially, we take two edges, then we swap their ends. If the resulting tour is shorter, we keep the exchange. We perform this task on all the edges in the tour until there are no edges that can be swapped. More improvements can be obtained for the tour via the

ILS algorithm, where perturbations and 2-Opts are performed iteratively.

If this process is carried out on the entire set of edges against all other edges in the tour it would be costly. The number of possible edges that can be exchanged is enormous in large problems. We, therefore, restricted the 2-Opt operations to neighbouring cities. In the initial phase of the algorithm, we find the neighbours by looking at every city in the problem, and choose the closest R cities. In this case, we have chosen $R = 30$ cities as we have found this to be a reasonable number of neighbours. We performed a variety of tests on multiple problems to come to this judgement.

Figure 6.7 shows the a280 TSP problem with a few examples cities and their neighbours. Each city encompasses its neighbours within a circle (the circle is just for illustration). Each city has a variable size circle. We are not concerned with the size of the circle. Instead we concentrate on the number of neighbours. Once we have these neighbours we can apply 2-Opt within the locality of city, and this reduces the search space greatly.

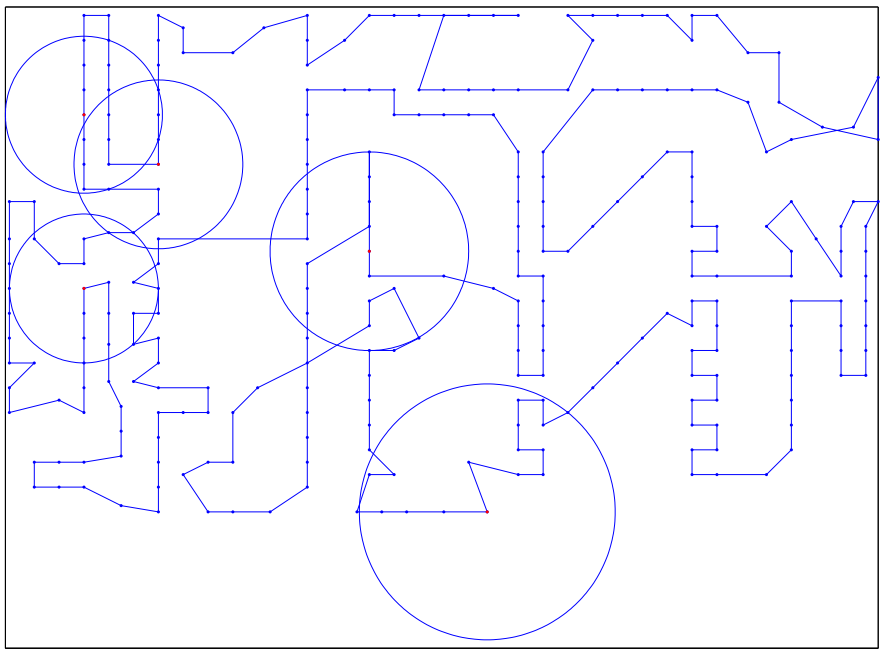

FIGURE 6.7: Limiting the search space by apply k-Opt only to neightbouring cities. To limit the search space we have only applied the search to the nearest 30 neigbours. This figure illustrates this concept with 5 cities each having 15 neighbours. The size of radius for each point is variable, and it ensures that each point has 15 neighbours associated with it.

Choosing neighbours by selecting the closest R neighbours has a serious shortcoming. If the cities are uniformly distributed in space, then this problem would not be obvious. However, if the cities are clustered (this should not be confused with the solution space. We are discussing the position of cities in 2-D Euclidean space), then choosing R neighbours in this direct fashion limits the search for an optimal tour. Take for example figure 6.8 which was generated using DIMACS TSP Challenge code. The cities in the tour

are randomly generate in clusters. If R is less than the number of cities in each cluster, then all the neighbours would naturally be chosen from within the cluster. This limits the exchange of edges to within the cluster itself, neglecting the neighbouring clusters, and thus leading the search to suboptimal results. In our experiments we avoided using clustered cities even though this does not mitigate the problem entirely.

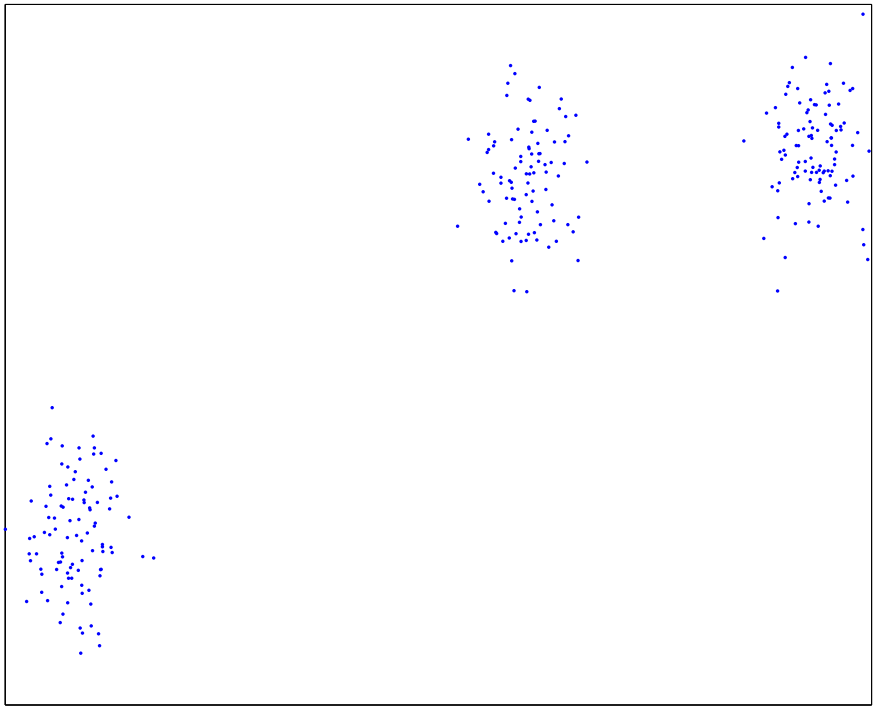

FIGURE 6.8: Randomly generated clustered tour. If R is chosen to be smaller the number of cities in a cluster, then every city in the cluster would have its neighbours form within the cluster. This might produce suboptimal solutions.

Initially we start by creating several random initial starting tours. For our experiments, we created 100 random initial tours, then we applied 2-Opt to each tour until no more improvements were possible. After that, we applied ILS to further enhance the tours. We next applied K-means with 5 clusters. Once the centroid-tours are assembled, they are put through another round of 2-Opt/ILS. We tried different ways of performing the random perturbations. Additionally, we tried varying the number of cities that are be affected by the perturbations. We have found that some perturbations work better than others. Once the tours are perturbed, we adjust the edges using 2-Opt again. If the resulting tour is shorter, it is kept. Otherwise, we revert back to the tour before the perturbation, and we repeat the ILS and 2-Opt.

In the second phase of the search, we used the frequency information collected from the clusters. By obtaining the mean of the frequency values, we assign probabilities to the edges that give them a degree of importance. The closer the probability of an edge is to 1 the more important it is to keep it, and the closer to 0 the less important it is. However, we use the probabilities of the edges in a different way. If the length of the

edges is l, and the probability of the edges is p, then we modify the length of the edges to be $l' = l \cdot (1 - p)$. What this effectively does is to give the edge a weight based on the proportion of its frequency. In the extreme case when $p = 0$, the edge is longest, and thus making it less likely to be kept in a exchange, and when $p = 1$, the length becomes l' becomes 0 and thus would be more likely to be kept after the exchange. We will call this method, Edge-Smoothing, since it is related to a similar method developed by (Gu and Huang, 1994). The edge-smoothing method was used in the backbone guided method (Zhang and Looks, 2005), and it was found to be extremely effective for improving the solutions to many TSP problems.

Another very important detail that should be mentioned about the way we chose to keep an exchange of edges. After the edges are perturbed, and the exchange of the edges is performed based on edge-smoothing, the acceptance for exchange is based on the actual tour length rather than l'. To be more precise, when a tour is perturbed, then it is perturbed on a number of neighbouring cities. Correcting for the perturbation is carried using smoothing, and while smoothing allows for the acceptance of the exchange locally, the exchange might not necessarily produce shorter tours. This is the reason for checking the actual full tour length via the actual edge-length, l. This was done after all the exchanges were applied. Once it is found that the full tour is shorter, all the edge exchanges are kept.

The first comparison we made was CLGH against solo-ILS. Figure 6.9 shows results of averaging of 100 simulations on the fnl4461 problem. It can be seen that CLGH produces better results than solo-ILS. With 100 initial starting points, 5 centroids, we show the results of best centroid out of 5. For the solo-ILS, after the initial 100 search we chose the top 5 performing tours, and carry another search on each. The graph shows the result of the best of the top 5 tours.

To see if CLGH took advantage of some form of clustering we compared it to ALGH on randomly selected tours. We ran the same tests on three different problems: rat783, fnl4461 and pr1003. CLGH was applied exactly the same manner it was in MAX-SAT. In ALGH, we selected 5 tours at random from the 100 tours. We obtained similar performance levels between CLGH and ALGH. These tests can be seen in Figure 6.10, 6.11, and 6.12.

6.7 Understanding the Results

What we have seen in these experiments is that CLGH and ALGH provide an advantage over solo-ILS. However, there was no clear difference between CLGH and ALGH. There could be several factors that are affecting the results. First, if we assume that there is a clustered structure to the solutions in TSP, then the reason would probably be that we did not apply K-means in CLGH at the sweet spot. This was our experience in

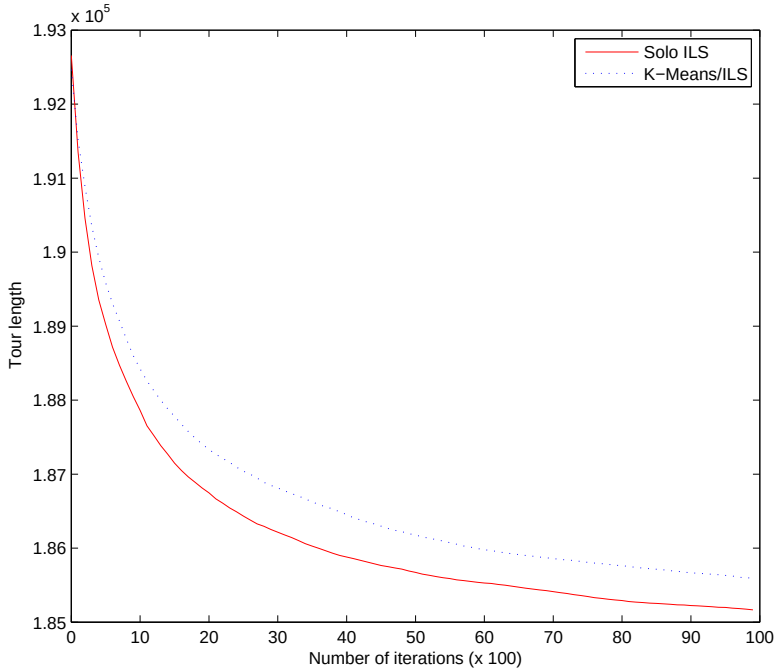

FIGURE 6.9: Comparison between solo-ILS and CLGH. CLGH produces better results than solo-ILS

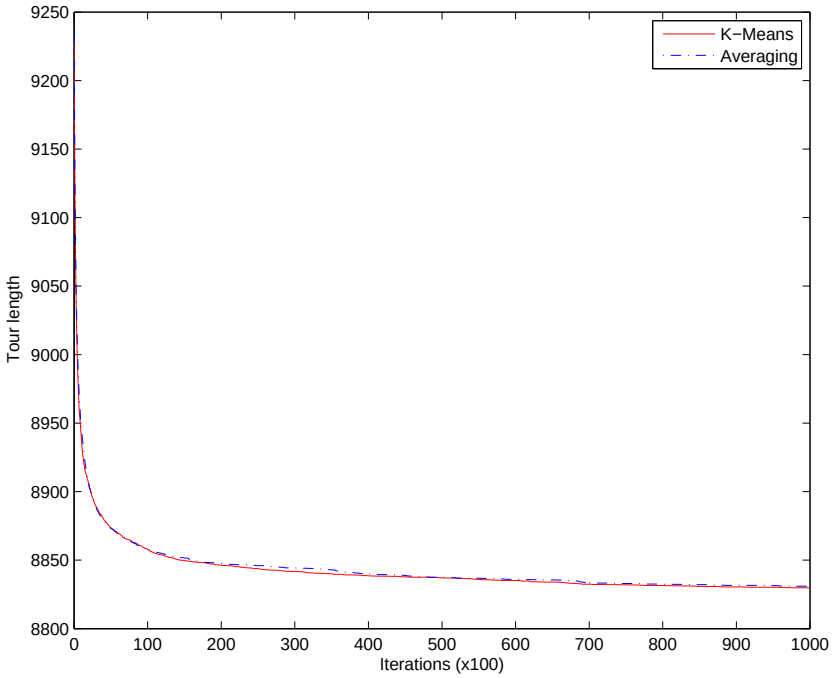

FIGURE 6.10: Comparison between the performance of CLGH and ALGH on the rat783 TSP problem. The number of initial random tours that were created were 100, clustered into 5 clusters. The result shown for CLGH is based on the best of the 5 centroid. With ALGH, 20 tours were selected at random from the 100 tours, and averaged. This plot shows the best result out of 5 different centroids.

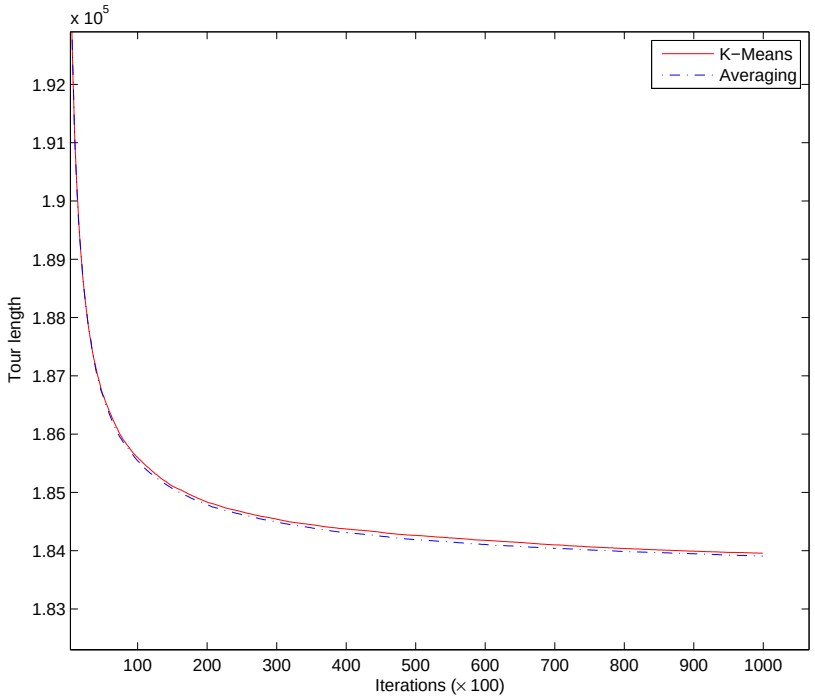

FIGURE 6.11: Comparison between the performance of CLGH and ALGH on the fnl4461 TSP problem. The number of initial random tours that were created were 100, clustered into 5 clusters. The result shown for CLGH is based on the best of the 5 centroid. With ALGH, 20 tours were selected at random from the 100 tours, and averaged. This plot shows the best result out of 5 different centroids.

MAX-SAT. Before finding the sweet spot where K-means is applied, the results showed that CLGH and ALGH to be the same.

In MAX-SAT we found that the best location to cluster was right around the maximum curvature in cost versus time graph 5.11, the sweet spot. We have verified this with a large number of experiments, and for all the experiments we carried, this location seems to apply to all the problems we tested. In TSP, we applied K-means at around the same point, and it was effective against solo-ILS. Notwithstanding, to over come ALGH it could be that the sweet spot is somewhere else. We will see in Chapter 7 that this is the case for some problems in Artificial Neural Networks. We get better solutions after the search plateaus much farther away from the sweet spot found for MAX-SAT.

Here, we could not find a sweet spot which provides CLGH with an edge over ALGH in our experiments. We tested for it before and after the maximum curvature, but there was no success in finding it. We tested for it in different locations along the search in steps with gaps in between. We could have missed it, since it could have been confined to a more a narrower region of the search. The smallest problem that we considered to be somewhat difficult, and a good starting point for our experiments was the rat783 problem. With each test consisting of at least 50 runs, each experiment required around 4 days to be completed. We could have performed more tests, but it required many

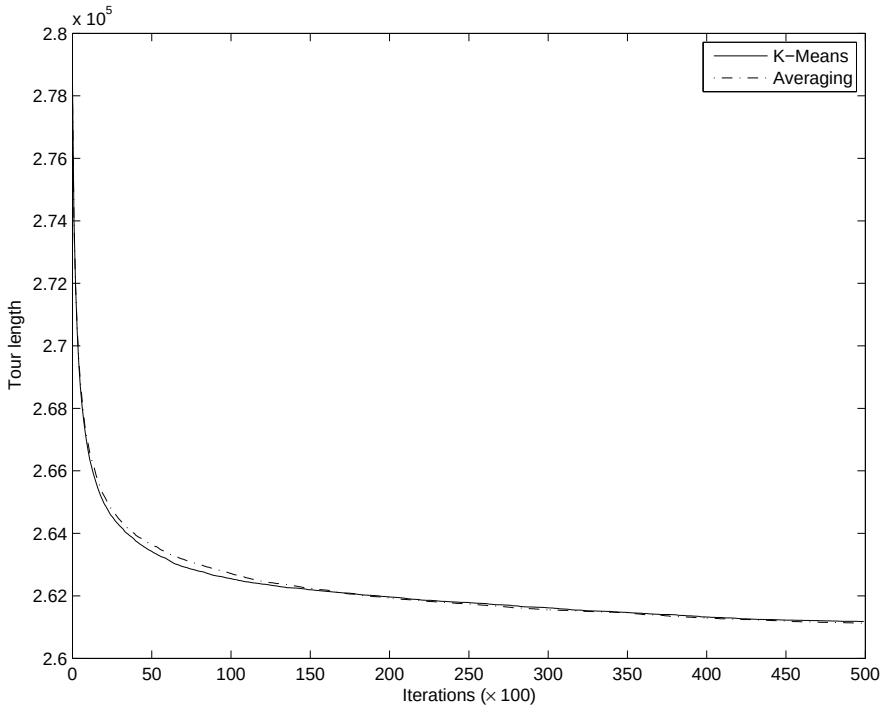

FIGURE 6.12: Comparison between the performance of CLGH and ALGH on the randomly generated pr1003. The number of initial random tours that were created were 100, clustered into 5 clusters. The result shown for CLGH is based on the best of the 5 centroid. With ALGH, 20 tours were selected at random from the 100 tours, and averaged. This plot shows the best result out of 5 different centroids.

more months of testing. The point of application is an open question. We leave the discussion of this point for the chapter on Future Work.

A second reason that could account for similarity in performance is that there really is not a clear structure of clusters in TSP problems. In our investigation of MAX-SAT, we generated thousands of random problems, and composed a picture of the landscape. The tests we ran on these hard random problems were indicative of the how our algorithm would behave with DIMACS structured benchmark problems (Selman, 1995). In TSP, even though researchers such as (Mühlenbein, 1992; Bianchi *et al.*, 2002) tested their algorithms on randomly generated problem, there is no clear evidence that these problems provide a similar structure to real world problems. Hence we avoided the generation of random problems for this specific task.

We could not have studied the structure of TSP in the same way as MAX-SAT. TSP is quite different from MAX-SAT. In MAX-SAT finding local solutions was dependent on single flips of a variable assignment. Measuring the relationship between two solutions was direct. We used the Hamming distance, and it gave a clear indication of the resemblance between two solutions. In TSP, we found solutions using 2-Opt, or by changing two edges at a time. Purely applying 2-Opt led to local solutions that were quite far from the optimum. We could, on the other hand, find better solutions with 3-Opt, or

even better ones with 4-Opt, and so on. However, the computational complexity increases enormously, prohibiting us from performing structural analysis with our current computational power.

We used ILS which perturbed the tour, and corrected the perturbation with 2-Opt. With this method we were able to find better solutions. This though still does not solve the problem of finding a structure. In MAX-SAT we used BHC for hill-climbing. When applying BHC, we moved to local optima with ease. After which the search would get stuck. BHC did not include any random walks. This meant that the search would constantly move towards a local or global optimum. When the solutions reached by 2-Opt are perturbed in TSP, it moved the search away from local solutions, and in to or out of local solutions. It was not clear what these solutions signified. Were these local optima? In MAX-SAT we used local optima to compare with global optima. With this, and the relationship between global optima the structure was exposed. This was not possible with TSP.

We therefore tried different measures to understand the landscape. One method was to apply Principle Component Analysis (PCA) to reduce the dimensionality of the solution points. Using the first 5 components we plotted the different 3-dimensional figures using different combinations of solutions for the Berlin52 problem, Figures 6.13 and 6.14. We hoped to get a general outlook of the landscape. We did not find a clear indication that there was clustering. With the first three components in Figure 6.13, we find that there seems to be clustering. However, as the other components are viewed, clustering information is reduced as can be seen for the components 2, 4, and 5 in figure 6.14.

In addition, the plot of the first three components show that there is a relationship between the cost and the Hamming distance to the optimal solution. As the length of the tour gets smaller, the distance between better solutions and the global optimum gets smaller. The spectrum of colors expresses the cost. The red colors show the worse solutions, and the blue colors show the better solutions. However for the components 3, 4 and 5 in Figure 6.14, the relationship is almost non-existent.

In fact, PCA could not have produced good results since most of the other components, beyond 5, that were removed were just as important in showing the rest of the picture. It should be noted that each solution point was $n(n-1)/2$ long. They represent the tour edges. With Berlin52, which had 52 cities, the number of possible edges where 1 326. Although the representation was sparse, reducing it to 5-dimensions becomes less meaningful. However, even this much of a glimpse showed us that there was no clear evidence of a clustered structure, or a clear cost versus Hamming distance relationship.

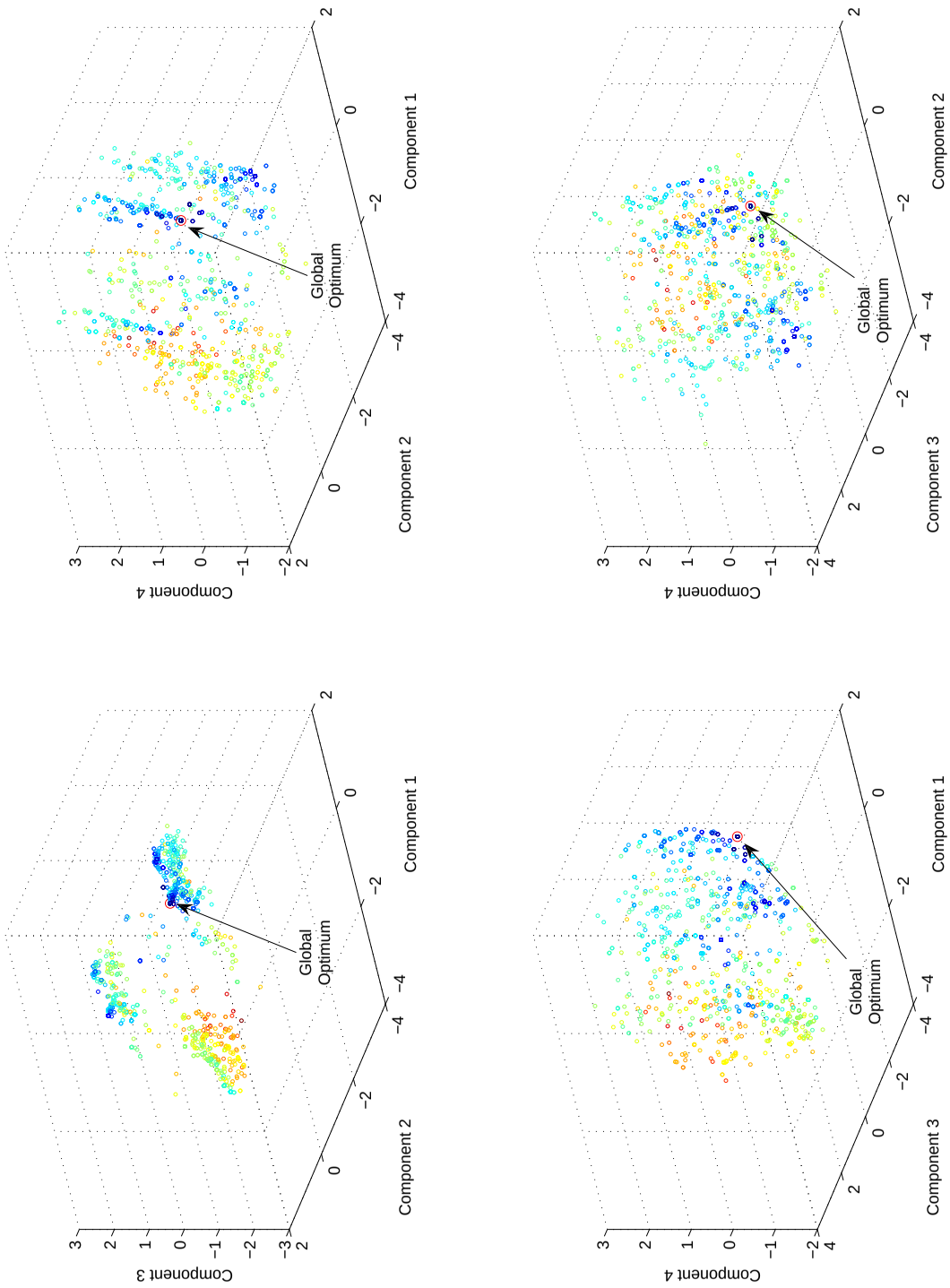

FIGURE 6.13: PCA plot in 3-dimensions

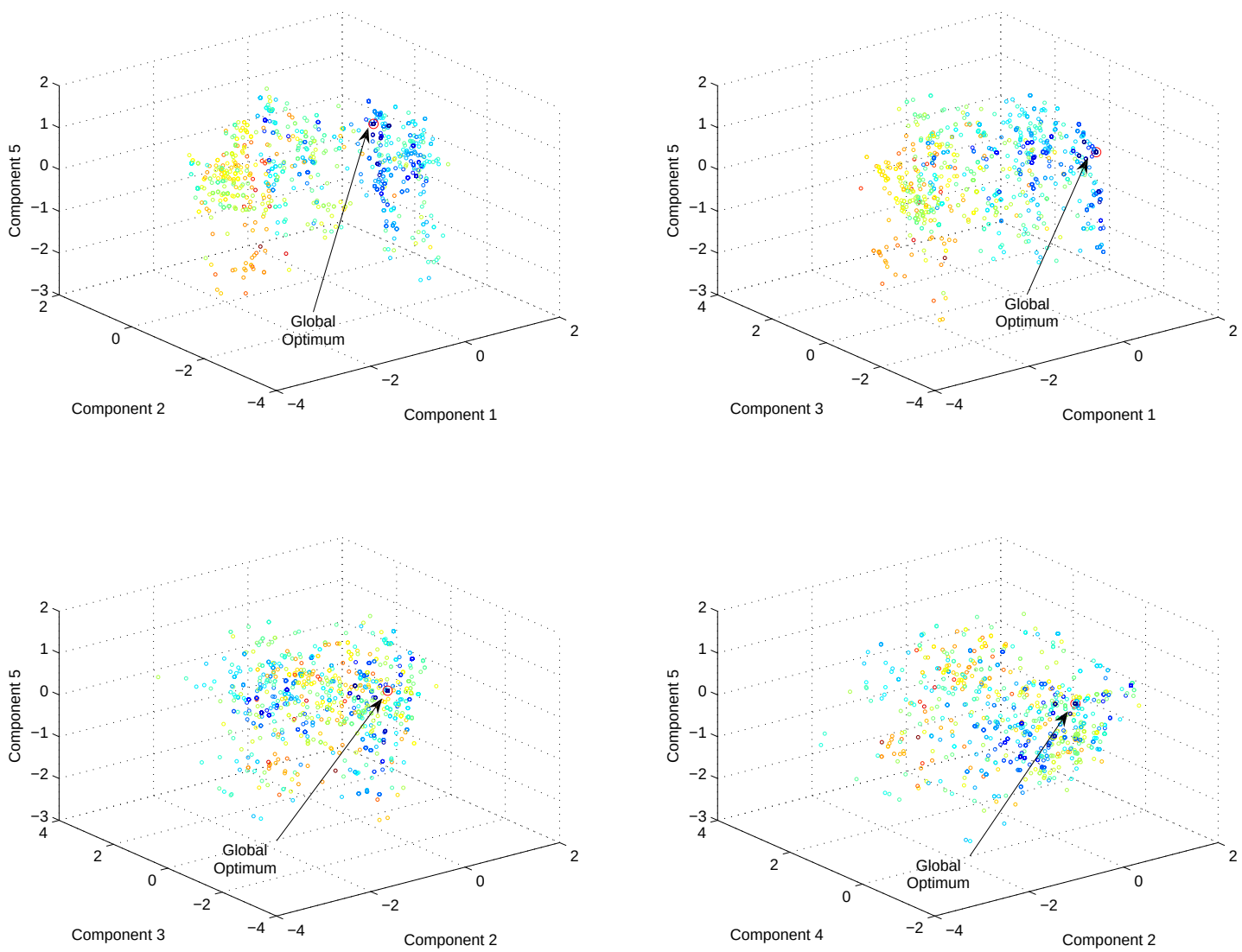

FIGURE 6.14: PCA plot in 3-dimensions

We also found the frequency distribution of the edges in 20 different tours. In this experiment we generated 20 different random tours, and applied ILS on each tour until we obtained no improvements. We ran this experiment 100 times for each of the different problems, C1K1, fnl4461, pr102, and rat783 tsp problems. We compared the edges of the tours to each other in each problem, and determined how many times an edge appeared once, twice, and so on until 20. An example of this is shown in Figure 6.15. If we have 5 cities, and 3 different tours, then the edges are stored in an array of edges. We count the occurrence of an edge in all the 3 tours. After that we find the frequency of having an edge appear once, twice and so on.

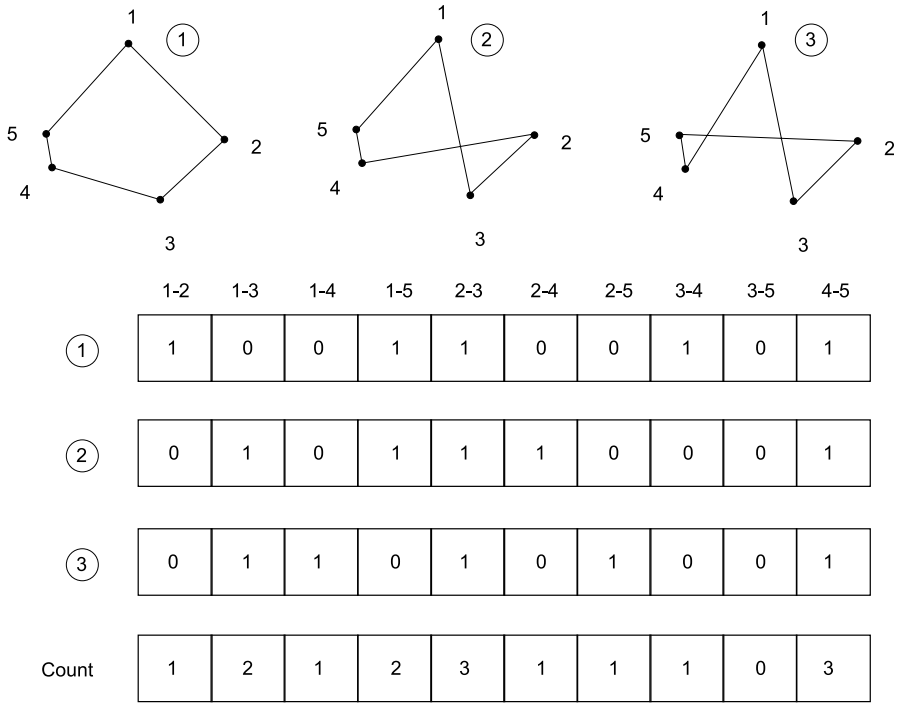

FIGURE 6.15: An example of a tour with 5 cities. The edges of the different configurations accumulated. Then the sum is used to determine the frequency of the appearance of each edge in the tours.

Figure 6.16 shows the normalized frequency for the results for 20 tours in each of the problem mentioned earlier. A great number of the edges appear only once. As the number of occurrences of the edges increases, the frequency of these decrease. The tail end of the edges increase in frequency again. This shows that the tours have very dissimilar arrangements, and this could account for suboptimal solutions that are quite far apart in space.

At the tail end of the plot we see that some edges appeared to have been fixed for all tours. That is the reason for the increase in their frequency even though the tendency for frequencies of the edges to go down. We can use the example in Figure 6.15 to explain this. The smallest edge is between cities 4 and 5. This edge would likely stay fixed amongst different tour configurations because it cannot be replaced by a better one. Of

course this does not mean that optimal solutions would necessarily have to contain these shorter edges.

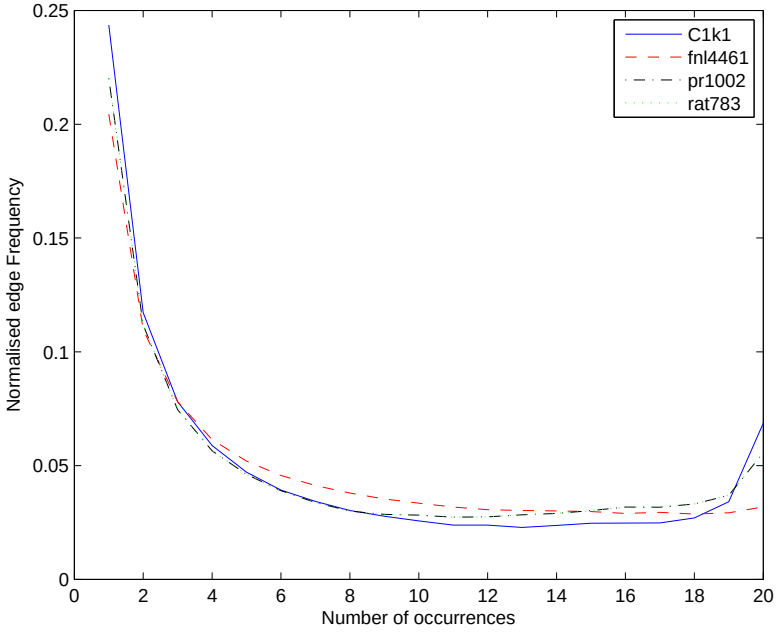

FIGURE 6.16: The frequency distribution of common edges between 20 local tour-solutions over 100 runs each. Most tours contain dissimilar edge configurations. Very few edges appear in most tours. The tail end of the graph shows some edges persist over many tours.

We also obtained the mean and the standard deviation for the number of shared edges between any 2, 3, 4 and 5 tours amongst 30 suboptimal tours in the Rat783 problem. Table 6.1 shows that the more tours are included in the comparison, the fewer edges these tours share. With 783 edges for each tour, the average number of shared edges between two tours is 489.821 edges. This is only 62.6% edges in common. This average is reduced even more as the number of compared tours are increased. For 5 tours, the average number of shared edges becomes 274.182, which is 35.02% shared edges amongst the 5 tours. This suggests that locally optimal tours are a long distance apart with little evidence of clustering.

# of compared tours	Average	Std
2	489.821	13.8363
3	377.104	13.052
4	314.679	11.8763
5	274.182	10.7783

TABLE 6.1: The average number of shared edges between 2, 3, 4 and 5 tours using 30 suboptimal tours in the Rat783 problem.

The only study that claimed that there was a structure to the solutions, and considered

landscape of TSP problems to have a globally convex or a big valley characteristic is found by (Boese, 1995). This study found that lower cost solutions tend to be in the vicinity of other better solutions and the optimal solution. It also showed that the optimal tour is located centrally within good solutions. While we had hoped to find this picture, their study only took into account a single problem with 532 cities, and it cannot be generalised over all problems.

Another study on the cost versus distance correlation (Peter and Bernd, 1999) showed that by using the LK algorithm the solutions showed less correlation than when using differential greedy algorithms. In fact for some problems, such as Cat5252 (Peter and Bernd, 1999), it seems that the is no correlation between the local and global optima when using LK algorithm. Although, LK heuristic performed better than the differential search heuristic. Both these tests show that the cost versus distance relationship is different for different search methods. More tests are needed to render a comprehensive description of the TSP landscape.

Finally, we have shown success in producing results using CLGH or ALGH. Averaging of solutions in this way is akin to Zhang's backbone guided search method with a small difference. With the backbone guided method, the improvements obtained by gathering backbone information were not only used for building the backbone, they were also used to generate more feasible starting solutions. On the other hand, K-means or averaging is sandwiched between two search heuristics. After its application the results are used to go further into the search without reintialisation.

Chapter 7

LGH and Continuous Problem Spaces

We have so far, in this thesis, applied LGH to discrete problems. We have shown that CLGH works very well with MAX-SAT problems, and we have also shown that it works effectively with the TSP problem, even though there was no difference between CLGH and ALGH in the TSP case. All the problems we have tested LGH with were combinatorial optimization problems. So far, we have not applied LGH to problems in the continuous domain. The next step in the analysis is to see what performance gains we could obtain in the continuous domain.

A good test bed for LGH is the Artificial Neural Networks. Here, we will only apply LGH to test its feasibility (no landscape analysis will be offered for this problem). To examine LGH, backpropagation is used. Here, we will attempt to cluster the weights in a fully connected feedforward Artificial Neural Network with one hidden layer. Then we will apply the K-means clustering, which will be somewhat different than in the case of MAX-SAT and TSP, and that is due to representation issues. We will see the difference in representation later in section 7.4.

From these tests we will show that CLGH works very well for difficult classification problems, and produces equal results on less challenging ones. When CLGH is successful, the search rapidly finds very good results. This will show that LGH establishes itself as a more generic form of local search that can be applied in the continuous problem space too.

7.1 Artificial Neural Networks

An Artificial Neural Networks (ANN) (Haykin, 1999) is a very simple representation of the brain's neural networks. It captures the nonlinearity of the way the brain works

and the parallel processing of neural networks (although most implementations in programming have been sequential). Despite ANN's simple representation they have been shown to be very effective in numerous applications, too numerous to list. We only include some categories in which it was applied. It has been applied across many disciplines. Some examples are Artificial Intelligence, Vision recognition and simulation, voice recognition, data classification, stock market prediction, and so on.

There are several ANN learning paradigms. They are classified as supervised learning, unsupervised learning, and reinforcement learning. Supervised learning is the process of adapting a network to produce specific output patterns given specific input patterns (Reed and II, 1999). In learning it is meant that the networks finds a subtle relationship between the input and output by incrementally and gradually optimising for it. The supervision means that both the input and the output are provided for the networks so that it can learn the relationship. An example of the supervised learning algorithms is Back-propagation (Rumelhart *et al.*, 1986).

In unsupervised or self organised learning there is no sense in the learning of input and output relationship. In this case, the training data is not labeled, and the targets are not defined. The goal of an unsupervised ANN network is to find patterns and regularities in the data guided by the implicit rules in the design. Reinforcement learning is a mixture of supervised and unsupervised learning. The input output relationship is defined with less rigor or in a more abstract fashion (Reed and II, 1999).

7.2 Back-Propagation

We have chosen backpropagation (Hecht-Nielsen, 1989) in the supervised learning class as a test case. It is one of the most widely used neural network architectures, and it is highly studied. The feedfoward multi-layered neural network is a fully connected network with an input layer, an output layer and h number of hidden layers. Figure 7.1 shows an example of a fully connected neural net with one hidden layer. Each layer contains nodes that are interconnected with every other node in the other layers via weights $\underline{\mathbf{w}}$. Given a set of training examples $(\mathbf{t_1}, \mathbf{o_1}), (\mathbf{t_2}, \mathbf{o_2}), ..., (\mathbf{t_u}, \mathbf{o_u})$ where $\mathbf{t_i}$ is a vector of the training input pattern, and $\mathbf{o_i}$ is a vector of the desired output pattern, it is required that if the input $\mathbf{t_i}$ is shown to the network, the output $\mathbf{o_i}$ is produced.

The inputs $\underline{\mathbf{t}} = (\mathbf{t_1}, \mathbf{t_2}, ..., \mathbf{t_u})$ are shown to the network, and they are propagated by multiplying each input by the corresponding weights and then summed, $\mu_j = \sum t_i w_{ij}$. This sum is then passed through a bounded monotonic function such as the sigmoid function $f(\mu)$ shown in equation 7.1. The output of the function $f(\mu)$ is carried through the network performing the same operations throughout, the result $\underline{\widetilde{\mathbf{o}}} = (\widetilde{\mathbf{o}}_\mathbf{1}, \widetilde{\mathbf{o}}_\mathbf{2}, ..., \widetilde{\mathbf{o}}_\mathbf{k})$ is produced at the end of the network.

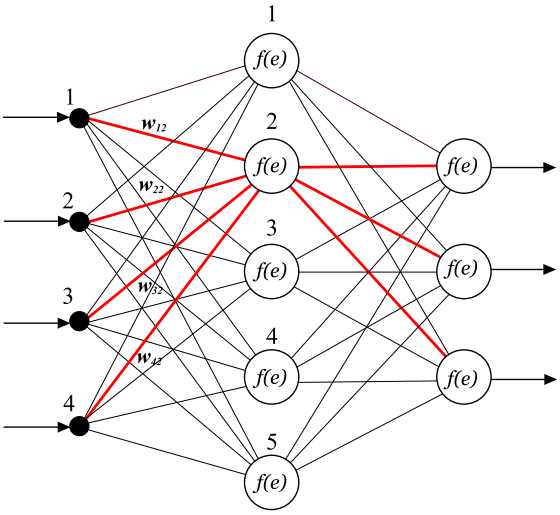

FIGURE 7.1: An example of a fully connected neural network with the input layer, one hidden layer and the output layer.

$$f(\mu) = \frac{1}{1 + e^{-\mu}} \tag{7.1}$$

The process by which the network is trained to produce $\mathbf{o_i}$ from $\mathbf{t_i}$ is by feeding the inputs $\mathbf{t_i}$ and propagating them through the network in a forward fashion. The outputs $\widetilde{\mathbf{o}}_\mathbf{i}$ of the network are compared with the desired values $\mathbf{o_i}$. Then the error difference between the desired and actual outputs are propagated backwards in the network, and the weights of the network are adjusted to minimize the error. This process is repeated until the error get small enough. This is called, Backpropagation.

The set of weights that connect each of the nodes i in the previous layer to the node j in the next layer, w_{ij}, are adjusted by equation 7.2. This equation basically performs a simple gradient descent (Riedmiller and Braun, 1993) by adjusting the weights with the partial derivative of the Error, E, with respect to the weight w_{ij}.

$$w(t+1) = w(t) - \eta \frac{\partial E}{\partial w_{ij}} \tag{7.2}$$

Where η is the learning rate that takes on values $0 \leq \eta \leq 1$.

Backpropagation has limitations. It has been shown to get stuck in local minima both empirically and theoretically before learning the entire training set (Gori and Tesi, 1992). Backpropagation is also known to be slow in optimising the set of weights in the network with current computational power (Hamm *et al.*, 2007). Several local search algorithm are proposed for reducing the effects of these problems.

7.3 Local Optimization of the Weights

Many different methods have been proposed to enhance the performance of neural networks. Aside from improving the performance of reaching an optimum for the weights by improving the backpropagation method (Trejo and Sandoval, 1995; Magoulas *et al.*, 1999), there are many other local search methods that do not fall into this classical scheme. The reason for using local search methods is because methods that rely on gradient descent are slower. In the time that it takes a gradient descent algorithm to reach a local minimum, a local search algorithm can be restarted many times, and hence could provide better solutions (Hamm *et al.*, 2007).

Some of these methods use genetic algorithms (Montana and Davis, 1989), Particle Swarm optimization (Eberhart, 1995), and simulated annealing (Sexton *et al.*, 1999). Not only do some algorithms develop values for the weights for a fixed network topology, some also develop the topology of the network entirely relieving the designer from trying different node and connection configurations. These techniques have been called evolving neural network algorithms (Maniezzo, 1994).

7.4 Experimental Results

One of the main stumbling blocks of applying CLGH to the weights of a multilayered neural network is representation. We have tested several ways of creating the weight vectors for the purpose of clustering. One way is to take the entire set of weights in an ANN and represent them as a vector, or we could separate the layers and take the set of weights of each layer independently. However, both of these ways are susceptible to the hidden layer permutation problem (Radcliffe, 1990; Hancock, 1992), also known as the competing conventions problem. Because of the interchangeability of the hidden nodes, if the weights are set as a vector without taking this into account, we introduce symmetric regions into space that would be detrimental to clustering. This is because the same vector of weights can be reordered in many different ways. Although the permutation problem does not affect the networks input to output relationship, the entire set of weights, if represented as a vector, can be ordered differently.

The permutation problem is shown in Figures 7.2 and 7.3. All the set of weights in Figure 7.2 can be represented as a vector as shown in Figure 7.2 (a). If the hidden nodes 1 and 3 are interchanged, then the vector representation would change as shown in Figure 7.2 (b). The change in the network does not affect the results, but the string would place two identical solutions in two different regions of the space. This sort of problem will arise with multiple runs of backpropagation starting from random initial weights. As gradient descent corrects for the weights, the group of weights connected to a hidden node could be exchanged in the next run.

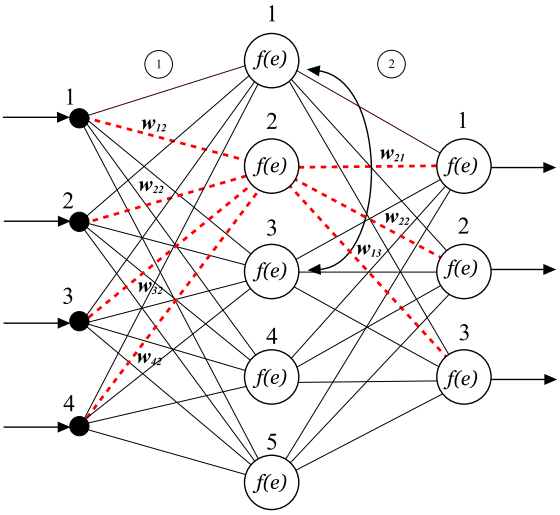

FIGURE 7.2: Shows the permutation problem. This example shows that nodes 1 and 3, in the hidden layer, can be swapped without affecting the input/output relationship.

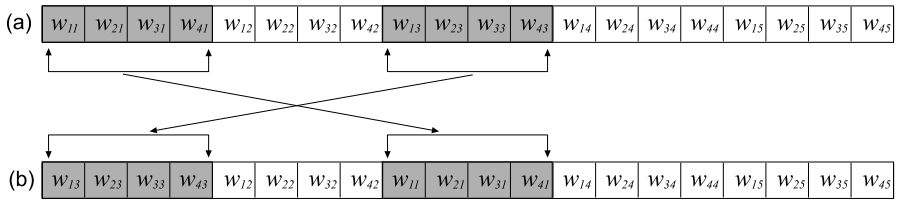

FIGURE 7.3: Shows the effect of the permutation problem on weight representation. The nodes in the neural network can be swapped where this would not have any impact on the network. However, the string representation of the weights are entirely different.

To mitigate the problem of the hidden layer permutations we worked with one hidden layer, we took all the weights that are connected to each node in the hidden layer, and considered each set of weights as a vector. An example of the weights selected for one node is shown as the thick dotted lines in Figure 7.2. In this example, we would have 5 vectors each being 7 elements long. Figure 7.4 shows one of these nodes. This is because the number of weights connected to each node in the hidden layer is 7 (4 on the left side, and 3 on the right). With this method we avoid having to worry about the interchangeability of the hidden layer nodes. We can reconstruct the network by simply reassigning the weight vectors to any of the hidden nodes without regard to order.

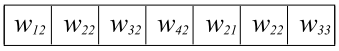

FIGURE 7.4: To avoid the permutation problem, the weights connected to each hidden node are stored in separate vectors.

If we run backpropagation multiple times, and have similar weight values that are assigned to different hidden nodes, then weight vectors will form clusters with close proximity in the weight-space. With this representation, we cluster the solutions in a very

different way. In previous problems such as MAX-SAT we would run the local search algorithm several times, and then decided how many clusters were needed, then K-means is applied. Here, we cannot apply the same concept. The number of clusters is dictated by the number of hidden nodes. Once the weight vectors are created, we set the number of clusters to the number of nodes in the hidden layer. Then K-means is applied, and the centroid of each cluster is found. The centroids can be assigned to the hidden layer arbitrarily. We then proceed with another round of backpropagation.

Figure 7.5 shows the comparison between CLGH and backpropagation alone for the Iris (Fisher, 1988a) problem. This problem is multivariate classification problems with 3 classes. Each class refers to a type of an Iris plant. The three classes are not linearly separable which makes classification difficult. It contains 150 instances with 4 attributes. The Iris dataset is one of the most used in pattern recognition literature.

We create a network with 5 inputs, 1 output, and 20 nodes in the hidden layer. Then the network is trained for 80 seconds with 50 different initial random weights. For the solo-backpropagation, the best of 50 runs is chosen. The best set of weights is used for the second round of backpropagation. As for CLGH, the weights are clustered with 20 clusters. We used all of the centroids for the weights in the second round of backpropagation. The results are averaged over 50 simulations. Figure 7.5 shows that the CLGH method yields better results than those obtained by solo-backpropagation even after the training is run for a total of 200 seconds. The plot also shows the common sharp spike found previously in MAX-SAT problems, followed by a steep drop in the error.

Another experiment is performed on the multivariate Forest Fires dataset. This problem is also claimed to be a difficult regression task (Fisher, 1988b). It contains 517 instances with 13 attributes. The networks is trained with 10 initial random weights. After which, K-means was applied at the first second of the search. Figure 7.6 shows the performance boost obtained by CLGH averaged over 20 simulations.

Despite achieving better results for the Iris and Forest problems, not all problems tested gave similar performance levels. In some cases CLGH was only equally as good as solo-backpropagation. As an example, the test is performed on the Wine problem. This problem has 178 instances with 13 attributes. This problem, according to the donors is not a very challenging one (Fisher, 1988c). In this instance, the results of CLGH are as good as solo-backpropagation. The same performance is obtained for the Breast Cancer classification data, Figure 7.8. This problem has 569 instances and 32 attributes. The reason for this equal performance level is because both problems are easy. Both methods reach optimum solutions very easily. This occurred with MAX-SAT problems also. The enhancements CLGH provides can be easily seen in more difficult problems.

Even in the instances where CLGH worked well, the sweet spot was hard to find. This was especially the case with the Forest Fire dataset. Clustering was applied at the first

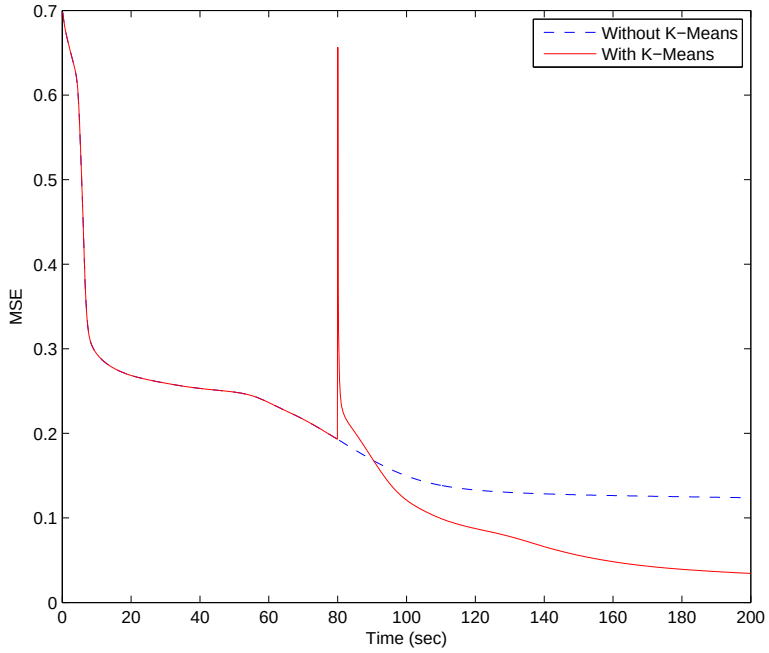

FIGURE 7.5: Shows the performance of solo-backpropagation against CLGH for the Iris problem. We can see that there is a significant difference in applying CLGH in reducing the MSE.

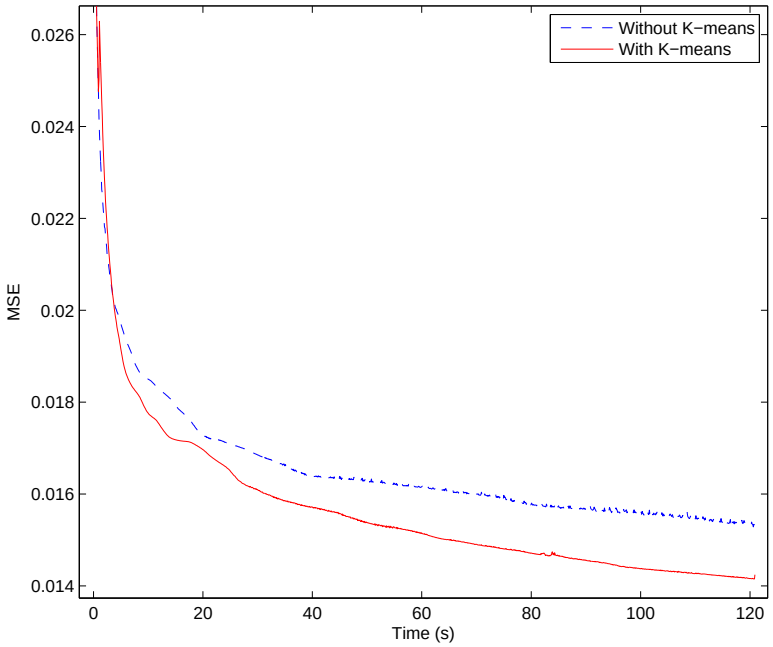

FIGURE 7.6: Shows the performance of solo-backpropagation against CLGH for the Forest Fires classification problem problem. K-means was applied after the first second of backpropagation. Here, we also get a performance boost via CLGH

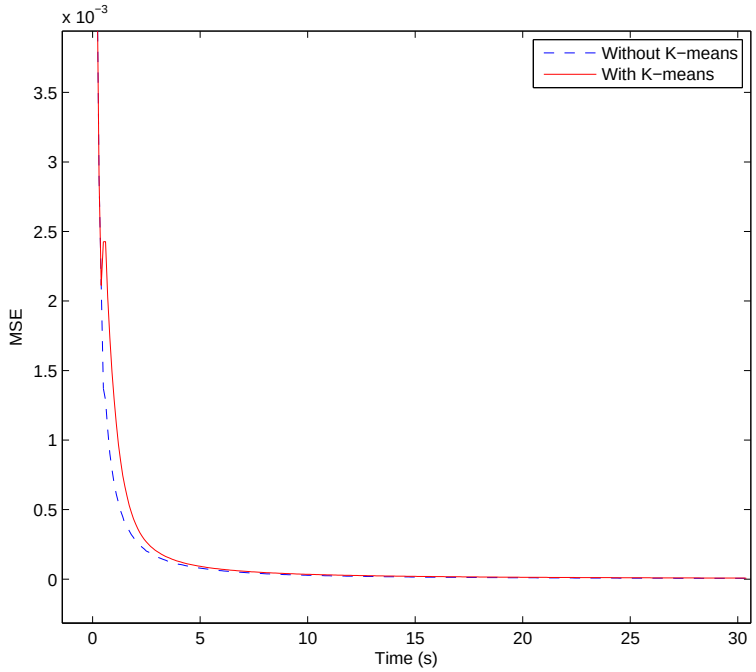

FIGURE 7.7: Shows CLGH performing slightly worse than solo-backpropagation on the Wine data.

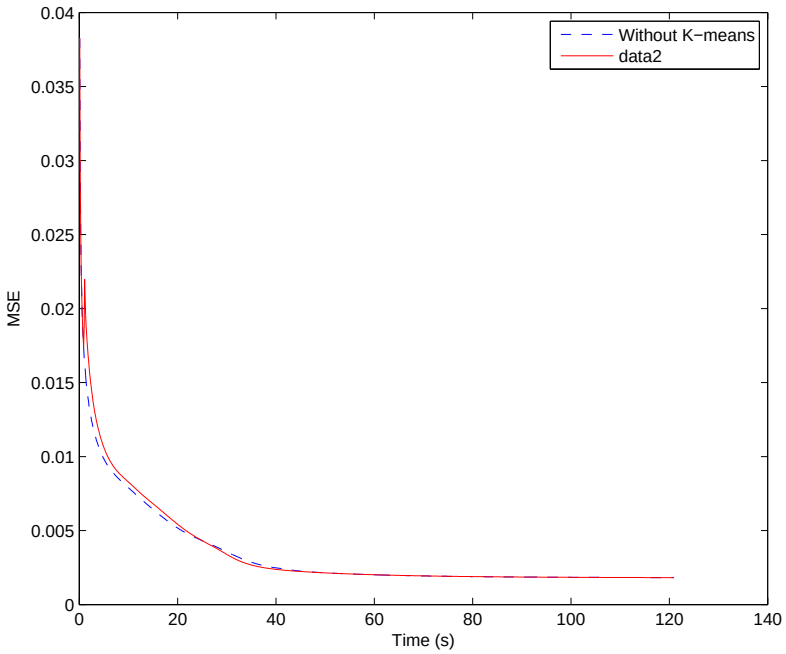

FIGURE 7.8: Shows CLGH performing slightly worse than solo-backpropagation on the breast-cancer data.

second in the search. If it was applied slightly less or more than one second; even by as much as 0.25 seconds earlier or later, the results would not have been as good. With ANNs the number of local minima is enormous (Hamm *et al.*, 2007). We could have the centroids close to a global minimum, but if they were surrounded by many local minima, applying backpropagation would drive the solution into a local minimum. This is also true of other local search techniques being applied to neural networks (Hamm *et al.*, 2007).

We tried different hidden nodes to see what effect this would have on CLGH, but the results did not differ much. Although the overall performance of solo-backpropagation was affected by varying the number of hidden nodes, but overall the performance CLGH was better. Also, we did not apply the weight decay term, 7.3, which speeds-up the learning process.

$$w(t+1) = \rho w(t) - \eta \frac{\partial E}{\partial w_{ij}} \tag{7.3}$$

This improvement would have enhanced backpropagation, but it is not crucial to our experiments.

Our approach to clustering in ANN was not same as that of MAX-SAT or TSP. Although we used K-means to cluster solutions, the clustering was in effect performing a single average for each of the nodes in the hidden layer. We can see from these experiments that CLGH works well with ANN problems. In these cases when CLGH performed at the same level of solo-backpropagation, the problem were known to be easy. When backpropagation finds it hard to solve a problem, this is when CLGH seems to offer better results.

Chapter 8

Conclusions & Future work

8.1 Conclusions

We have discussed the satisfiability problem in brief, described some complete and incomplete solutions, and looked at the structure of the space in which these problems reside. The majority of phase transition analyses provided by researches were performed on SAT and MAX-SAT problems using depth first search algorithms. However, a few researchers performed the same analysis on MAX-SAT problems using a local search algorithm. We also performed phase transition analyses using a local search algorithm with a slight difference. The difference is that the previous results were done on fully satisfiable problems below the phase transition, and a targeted number of satisfied clauses were set beyond the phase transition. We performed our tests on random 3-SAT problems through the easy-hard-easy regions with focus on local optima. This was important. It gave us an initial intuitive insight into the solution landscape with regards to the local and global optima.

Most researchers use stochastic methods to find solutions of SAT and MAX-SAT problems, and in fact most of them apply these methods to the most difficult regions. Our analyses show that these solutions might still behave poorly around the phase transition even for local optima, and the difficulties start to be evident when problems grow in size. From the phase transition analysis, we were able to form a picture of the landscape of solutions, and we were able to relate this picture to the easy-hard-easy transition.

The phase transition analyses were followed by a thorough analyses of the structure of the solution landscape in order to provide a more comprehensive and expansive view. The empirical evidence described in this report provides strong support for the hypothesis that we are able to learn about the large-scale structure of the landscape for large MAX-3-SAT problems. By doing so we have produced a general framework, LGH, by which clustering (CLGH) or averaging (ALGH) can applied. With LGH we substantially outperforms more conventional algorithms. It was not the intention of the paper to tune

our algorithm and so we believe these results to be due to the substantive difference in approach not superficial reasons.

Interestingly a genetic algorithm using conventional crossover does not learn this large-scale structure nearly as well as clustering or averaging. Averaging, or using the centroid of a cluster, breaks the metaphor of natural selection. Averaging acts like blending inheritance in that it reduces the diversity in the population. Since the modern synthesis, it has been understood that an essential component of natural selection is the particulate nature of genes such that crossover exchanges genes rather than average them. As a consequence, the kind of averaging we have undertaken has been little explored, yet, as we have argued, this averaging allows the landscape to be explored in a very different way to conventional crossover. Perhaps some of the algorithms which come closest to exploring landscapes in an analogous way to the CLGH and ALGH proposed here are the univariate estimation of distribution algorithms. Yet as we have shown even these do not perform moves similar to averaging.

We chose to study the random MAX-3-SAT problem as this is one of the best understood NP-Hard problems. An important question is whether we can obtain similar performance on other NP-Hard problems? Clearly, this can only be determined empirically. For this we explored TSP problem. In TSP, we showed that averaging of solutions provides a centroid that can be used as a better starting point for the search. However, there was little evidence that the solution space was clustered. The limited number of analyses we were able to perform (due to many complications intrinsic to TSP) confirmed this view. Instead, as researchers have pointed out, there might be a single cluster. Hence, CLGH and ALGH produced similar results.

We are however optimistic that the LGH approach taken here should be applicable to many other problems. The essential features of the landscape which made our approach work was that good solutions are, at least weakly, correlated with global optimal solutions. This seems to be a property of many NP-Hard problems, which suggests they may be amenable to a similar approach.

With ANNs, the results were problem dependent. In some problems, there was a clear advantage to using CLGH. The results showed a significant gain in performance. In others, CLGH performed just as well as solo-backpropagation. We have not studied the landscape of this problem, nor have we compared it with state-of-the-art solvers. Nevertheless, there is an indication that CLGH can be used to improve the results in some cases.

The studies carried out in this paper have lead to a number of significant observations about MAX-3-SAT, TSP and ANN. The three most striking observations were the following

- Very significant improvements can be achieved by clustering good solutions and

restarting a hill-climber from the feasible configuration closest to the centroids of the cluster.

- Taking the average of a set of solutions is radically different and, for this problem, substantially better than crossover.
- Finally, to maximise the benefits of clustering it should be applied at very specific points in the search.

In retrospect, the application of CLGH looks like a straightforward way of solving difficult problems. Indeed, we started with the proposition that clustering of solutions is a natural way for solving satisfiability problems. There were two reasons why we came to this conclusion. First, we have already seen the limited study of the structure of solutions by Zhang. It was limited to showing the solution space as a single cluster. His research showed that better-cost solutions shared common traits with the global optima. Second, we have shown from our cluster-analysis that the correlation between local and global optima increases as the cost of the optima get better, there might be multiple global solutions, and that the better-cost solutions are clustered around the global ones. This automatically led us to believe that clustering would natively work with this picture. The implementation of a clustering technique to solve satisfiability problems did not work directly.

What we initially found was that directly executing K-means on solutions obtained from hill-climbing was not beneficial. In fact, the results from the search were either as good as averaging of random solution points or worse. These unexpected results went against the way the solutions described the space in which they reside. After many experiments we found the reason for this lack of performance. It was the point of application of K-means on the solutions obtained from the initial search. We can neither apply K-means before or after a certain point in the search. We believe that this simple and direct approach was not used previously by researchers due to this fact. Here is the central question that needs to be answered, at which points should we try to find the centroids?

K-means has been used on an incredible number of problems. It is used in vector quantisation in image compression (P. C. Cosman *et al.*, 1993), Classification (Li *et al.*, 2002; Wagsta *et al.*, 2001; Wang *et al.*, 2002), intrusion detection (Laskov *et al.*, 2005) and more. In all these cases, the data that is clustered is available beforehand. It is clear what needs to be clustered. The goal is to find centroids that best represent each cluster. For example, if an image is to be compressed using vector quantization (which uses K-means or what is commonly referred to as the LBG algorithm (Linde *et al.*, 1980) in the image compression world), then an image is segmented into blocks. Each block is turned into a vector in an N-dimensional space. Then these vectors are clustered to get the centroids that best represent the cluster. It is the same with classification. Individual centroids represent particular classes. Image data, intrusion data, or even data to be classified have an inherent structure. K-means exploits this existing structure.

Here, what exactly are we trying to better represent? We argued that we are trying to represent the solution space. However, which points in the solution space are these? Indeed, we are generating these points based on a fitness function. This function describes the solution space, but does not render its shape. For the fitness function to be useful, it must be applied to different points. We can think of the fitness function as an abstract description of the solution space, and the points as the actual physical or concrete description. Another way we could think of the function is as an implicitly defined landscape, and the point as explicit points or samples representing part of that implicit landscape.

We obviously cannot classify randomly generated solutions. Random solutions tell very little about the landscape. Instead we generate random points, then search the space starting with these points, and improve their costs. These points coat important regions. We recognise that this too is not enough. If the points were under-searched or over-searched they loose structure. They simply do not contribute information to create a useful centroids. Figure 8.1 shows an illustration of 3 levels of the application of local search. The first, A, is the randomly generated points in space. clustering these points does not offer any advantage to the local search algorithm, because the points are simply scattered in space. If we perform a local search moving from the A to C directly, then we have moved the solutions closer to local optima. Suppose that the global solutions are denoted in stars, as positioned in C, then clustering might provide a centroid that is closer to the one of the global optima, but it is more likely that it might not. The solutions are highly correlated. The best point in clustering is somewhere in between, B. We referred to this point as the sweet spot.

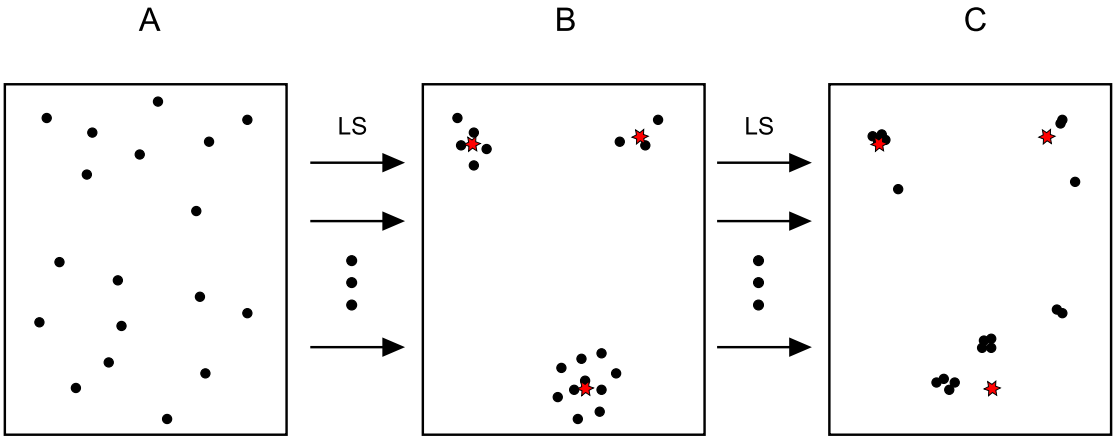

FIGURE 8.1: An illustration of the solution landscape in 2-dimensions. A proposed location for the application of K-means. There are 3 stages of the search. The first is the random initialisations of assignments. The second stage is locating good solutions. The third is over-fitting of the solutions. The best location to apply K-means is at the mid stage. This is because at this stage these solutions expose structure.

Evidence of this behaviour can be seen when we compare K-means clustering with averaging of random points. When we initially compare the two at the wrong point

in the search we discover that averaging performed sometimes better and other times worse. Only when we applied K-means at the right point in the local search did it beat averaging. This was evident for MAX-SAT. However, this can be seen more clearly in TSP. CLGH and ALGH perform equally well. It could be that either there are no clusters we can exploit, or the sweet spot was not found.

We have been successful in using LGH in MAX-SAT, TSP and to some degree ANNs. CLGH is even more successful than the state of the art algorithm, BGWALKSAT, and we have shown it to outperform every other local-search algorithm in its genre.

8.2 Future Work

We believe that this success can be extended to other optimization problems especially if the problem follows the same sort of structure found in satisfiability. More difficult problems should be tested for the viability of LGH search. Initially, the structure of many problems should be studied. However, this is not enough. More should be done to understand what solutions contribute to creating representative points or centroids.

Also, more should be done to understand the structure of solution of other problems. We have seen how highly advantageous it is to understand the clustered MAX-SAT solutions. We have seen that global solutions were separated in space, and they resided within a cluster of similar less optimal solutions. This study should be applied to TSP, ANNs and other problems to see if clustering is even appropriate to begin with. We have not done enough to understand these other problems. We attempted to implement CLGH to these problems based on the assumption that they too have a structure. Although this might be the case, only the analysis could prove that to be so.

Finally, many other clustering models beside K-means are available in literature. They should be studied and applied. Our simple use of K-means to cluster solutions certainly applied quite well to MAX-SAT. It yielded great results. We propose that these solutions can be further enhanced if more advanced methods were applied. Currently, we initialise the centroids randomly. Then they are iteratively improved. These centroid might converge to cluster or each might represent more than one cluster. The choice of the initial starting points is important. Also, choosing the number of centroids to match the number of clusters is crucial. This too will help assign each centroid to each cluster properly.

Appendix A

Algorithms

```
1: Procedure BasicHillClimb(X̄)
2: for i ← 1 to MaxTries do
3:     X̄ ← random Boolean assignment
4:     c′ ← cost(X̄)
5:     for j ← 1 to iterations do
6:         indexToFlip ← random(1, 2, ..., N)
7:         X̄′ ← flip(X̄, indexToFlip)
8:         c ← cost(X̄′)
9:         if (c = 0) then
10:            return X̄
11:        else if (c ≤ c′) then
12:            c′ ← c
13:            X̄ ← X̄′
14:        end if
15:    end for
16: end for
17: return X̄
```

FIGURE A.1: The Basic Hill-Climber Algorithm

```
1: Procedure Exhaustive(X̄)
2: c ← cost(X̄)
3: Save X̄ into Hash
4: Push X̄ into Stack
5: while (Stack is not empty) do
6:     X̄ ← Pop Stack
7:     for i ← 1 to size of X̄ do
8:         X̄′ ← flip(X̄, i)
9:         c′ ← cost(X̄′)
10:        if (c′ = c) then
11:            if (X̄′ is not in Hash) then
12:                Save X̄′ into Hash
13:                Push X̄′ into Stack
14:            end if
15:        else if (c′ < c) then
16:            clear Stack
17:            clear Hash
18:            c ← c′
19:            Save X̄ into Hash
20:            Push X̄ into Stack
21:        end if
22:    end for
23: end while
24: return Hash
25: End Procedure
```

FIGURE A.2: Exhaustive search algorithms

```
1: Procedure K-MeansSearch(X̄)
2: X̄ ← GenerateRandomAssignments(N)
3: for i ← 1 to N do
4:     X̄′_i ← BasicHillClimb(X̄_i)
5: end for
6: C̄ ← K-Means(X̄′, N, nCentroids)
7: for i ← 1 to nCentroids do
8:     C̄_i ← HillClimb(C̄_i)
9: end for
10: End Procedure
```

FIGURE A.3: K-Means algorithm

```
1: Procedure K-Means(X̄, N, nCentroids)
2: C̄ ← GenerateRandomAssignments(nCentroids)
3: repeat
4:     CreateZeroed(clusterCount)
5:     CreateZeroed(C̄′)
6:     for i ← 1 to N do
7:         d′ ← Hamming(X̄_i, C̄_1)
8:         index ← 1
9:         for j ← 2 to nCentroids do
10:            d ← Hamming(X̄_i, C̄_j)
11:            if (d < d′) then
12:                d′ ← d
13:                index ← j
14:            end if
15:            C̄′_index ← C̄′_index + X̄_i
16:            clusterCount_index ← clusterCount_index + 1
17:        end for
18:    end for
19:    for j ← 1 to nCentroids do
20:        C̄_i ← C̄_i/clusterCount_i
21:    end for
22: until No Change between current C̄ and previous C̄
23: return C̄
24: End Procedure
```

FIGURE A.4: K-Means search algorithm

```
1: procedure ITERATE LOCAL SEARCH
2:     T_0 ← GenerateRandomTour()
3:     T ← LocalSearch(T_0)
4:     for i ← 1 to MaxTries do
5:         T′ ← Perturbation(T, history)
6:         T" ← LocalSearch(T′)
7:         T ← AcceptanceCriteria(T, T", history)
8:     end for
9: end procedure
```

FIGURE A.5: Iterated Local Search

Appendix B

Extension

B.1 WinSATS Application

In an effort to support the SAT research community we have developed the WinSATS application. It is a robust Windows based application that is friendly and easy to use. Its primary use is to help researchers run different algorithms on SAT problems, and provide results that can help in their analyses. Currently, the program is in its second version. It contains several stochastic search algorithms such as Basic Hill-Climbing, GSAT and WalkSAT, and it accommodates search enhancers such as the CLGH method and perturbations.

WinSATS allows users to open CNF files, and apply different search methods on them. It can be used to cascade the different search methods one after the other, loop through them several times and observe the results at each stage of the search. The search can be started from a user specified number of assignments. The application is capable of creating random instances using the FCL model for a range of k values, variables and ratios, α. The main panel of the application is shown in Figure B.1.

There are many different settings that can be applied to each of the search methods. Examples of these settings are, the number of iterations the method is run before the search is stopped, the probability of the random walk in WalkSat, or the number of centroids K-Means algorithm creates. An example of these settings Figure B.2 shows the settings that are applicable to K-means.

Once WinSATS is run, it stores the results of the search for each of the assignments in a sheet. This information contains the problem description along with the cost (the number of unsatisfied clauses) of each assignment that was searched. Each new problem that is searched will have its results recorded in the sheet separately, Figure B.3. From the results sheet the CNF file that was loaded or randomly created for test can be

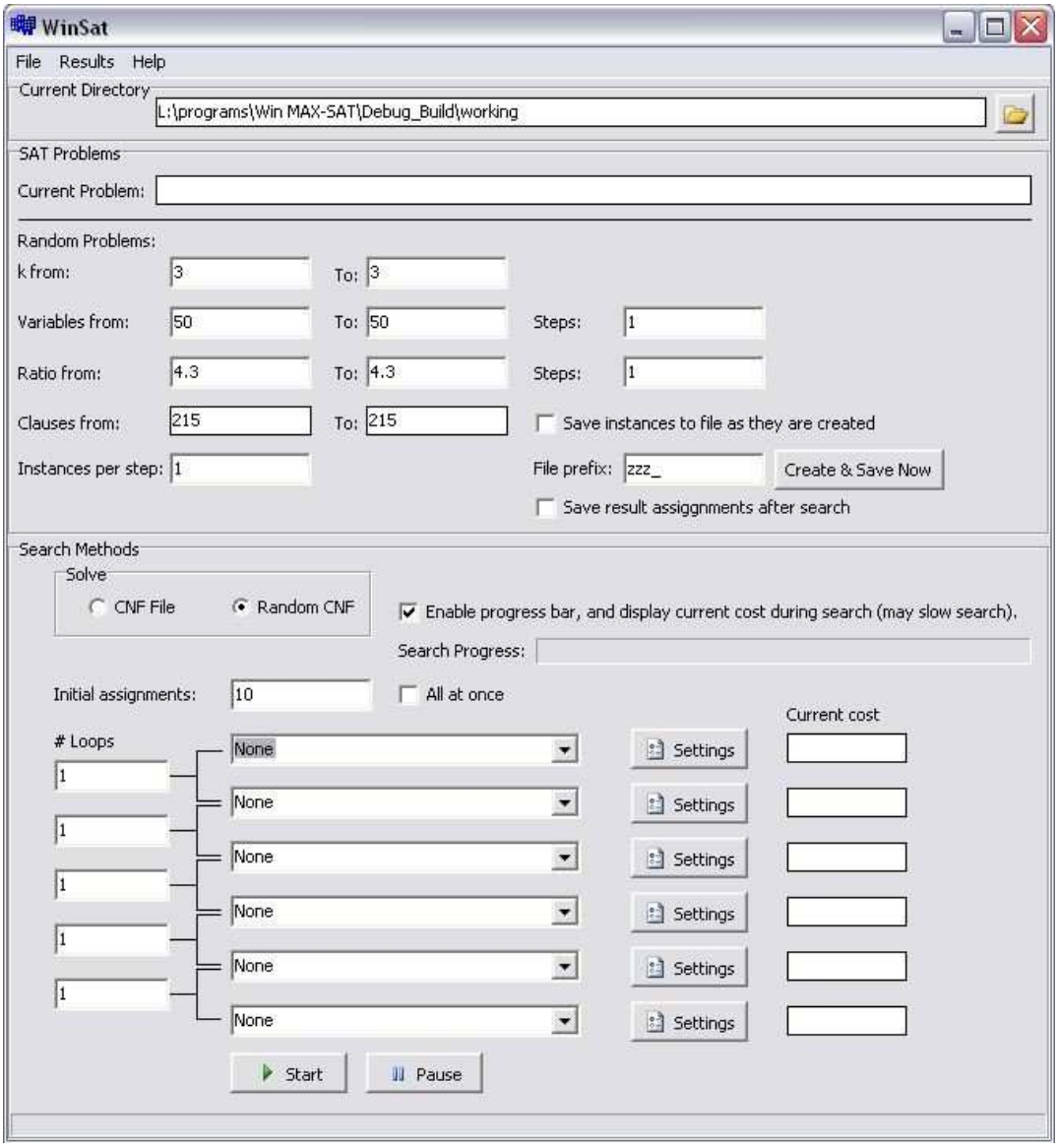

FIGURE B.1: The main control panel of the application WinSATS

viewed. In addition, the resultant true/false assignments, represented by 1s and 0s, can be viewed through the results sheet.

There is much more to WinSATS that is left out of this document. However the complete documentation for this application and the application itself can be found and downloaded at: http://users.ecs.soton.ac.uk/mqq06r/winsat/.

It is our goal to include many more of the well known complete and incomplete methods in this application. Ultimately, we hope to release the code to the research community to implement their own methods into it without having to worry about opening CNF files, creating random instances or even finding the cost of assignments.

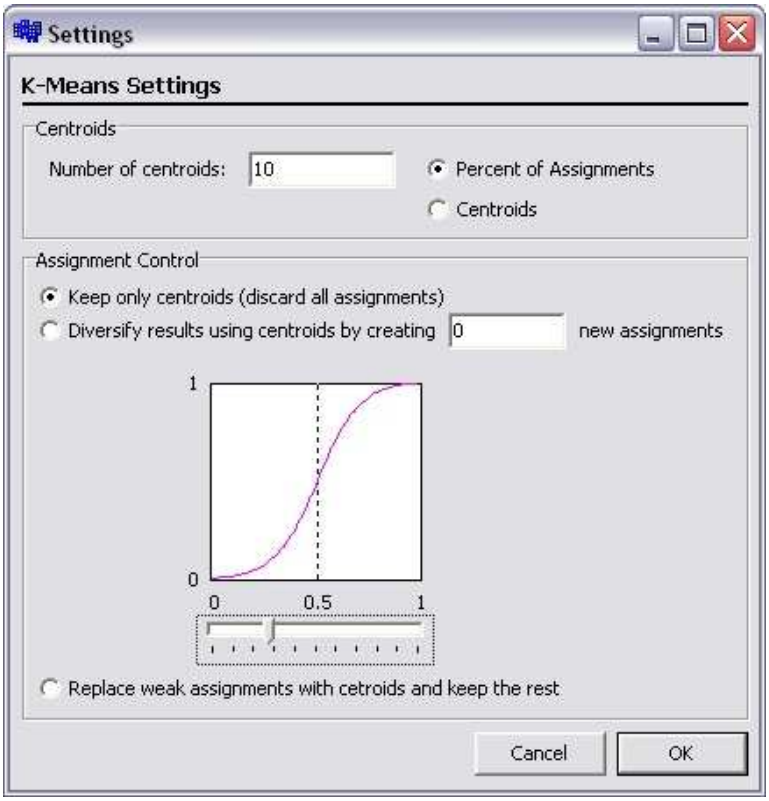

FIGURE B.2: This dialogbox contains the settings that can be applied to K-Means as an example.

B.2 A fast implementation of GSAT and WalkSAT

In WinSATS we improved the GSAT and WalkSAT methods such that we were able to compete with UBCSAT's state of the art implementation of these algorithms. We believe that our implementation is similar to UBCSAT's with one exception. We improved the speed at point when the search plateaus. This gave WinSATS the advantage, and provided speed-ups up to an order of magnitude more over UBCSAT on the large scale problems we have tested.

Initially, we implemented GSAT and WalkSAT in such a way that was much slower than BHC. The way GSAT works is by flipping the bit of a variable in an initial random assignment that gives the maximum number of satisfied clauses. This process is repeated until the search plateaus. By running many experiments we have realised that a boolean assignment comes to a point where flipping any variable provides either no improvement on the cost, or makes the cost worse. In our early implementation of the GSAT and WalkSAT algorithms, we searched the assignment by testing every variable to determine what cost improvement it would yield. After performing a full check on all the variables, we would chose the best flip. That meant that if there are n variables, then for every flip there was n checks. This was a long process which slowed the algorithm considerably. In BHC, we picked a variable at random, checked whether it improved the cost, and flipped

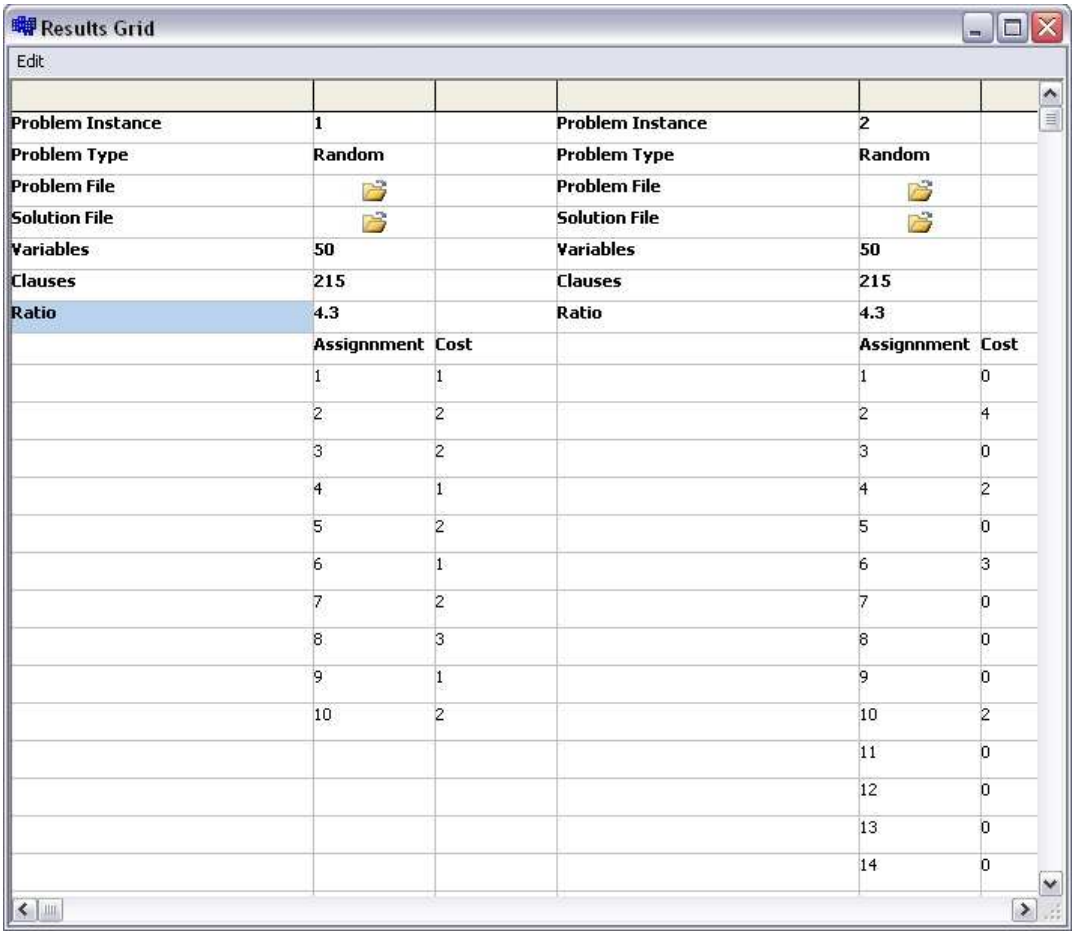

FIGURE B.3: The results of each problem searched is stored in the Results Grid

it if did. The BHC algorithm performed extremely well on relatively large problems in comparison.

The improvement we introduced was to constantly track variables as the they are flipped. When the initial assignment is generated randomly, a full sweep of the cost of each variable, $x_1, x_2, ..., x_n$ flip is found. This overhead is computed only once, and we store this information in a list. Now that we have a record of the costs of all the variables, we flip the variable x_i with the highest cost. After each flip, we find the effect of flipping x_i variable on each of the other variables it is in. We apply these new changes to the list. In this list, we store the cost of the flip and the index of the variable to be flipped. We perform the same for the next variable and so on. This we believe is how it was also implemented in the UBCSAT.

For this, we developed a specialised fast set data structure which involves two arrays. The first array is a simple list of the elements in the set. The second array indexes the elements in the first array where an index of -1 indicates the element is not in the array. Note that the size of the index array is the total number of elements that can be put in the set. This set allows $O(1)$ insertion, deletion, checking whether the element

is contained in the set, and choosing a random element from the set. This provides a considerable speed-up over a conventional binary tree structure which is order $O(log(n))$ (A hash set is impractical as it takes $O(n)$ to generate random numbers).

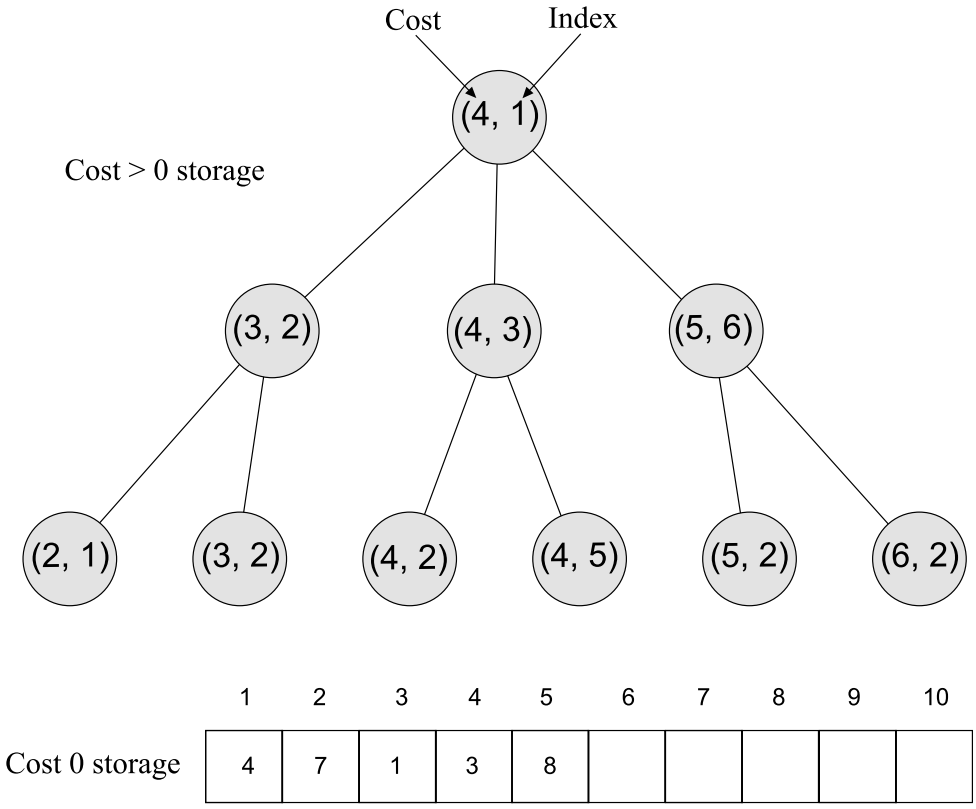

FIGURE B.4: Two levels of storage are utilized to speed up the GSAT and WALKSAT algorithms. The first is tree structured list, and the second is a simple array. The first structure stores costs greater than 0, and the second stores costs equal to 0. Since the search spends most of its time in the 0 cost flips, we gain a great deal of speed by allowing for this second level of storage.

B.3 SAT and MAX-SAT for the Lay-Researcher

In addition to the WinSATS application, we have created a web page that introduces satisfiability and maximum satisfiability for starting researchers. It was made simple using an informal language. This page can be found at: http://users.ecs.soton.ac.uk/mqq06r/sat/. Both this page and the WinSATS application page were posted on the Wikipedia Satisfiability page by us, and the link was reinserted into the Wikipedia Maximum Satisfiability page by others. The research community has shown interest in the page by visiting it at more than 5000 times since its inception.

Bibliography

Applegate, D. L., Bixby, R. E., Chvatal, V., and Cook, W. J. (2006). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

Arthur, D. and Vassilvitskii, S. (2006). K-means++: The Advantages of Careful Seeding. Technical Report 2006-13, Stanford InfoLab.

Baluja, S. and Caruana, R. (1995). Removing the Genetics from the Standard Genetic Algorithm. In *The Proceedings of the 12th Annual Conference on Machine Learning*, pages 38 – 46.

Bang-Jensen, J., Gutin, G., and Yeo, A. (2004). When the Greedy Algorithm Fails. *Elsevier, Discrete Optimization*, (1), 121–127.

Baraglia, R., Hidalgo, J. I., and Perego, R. (2001). A Hybrid Heuristic for the Traveling Salesman Problem. *Evolutionary Computation, IEEE Transactions on*, **5**(6), 613–622. 1089-778X.

Battaglia, D., Kolár, M., and Zecchina, R. (2004). Minimizing Energy Below the Glass Thresholds. *Phys. Rev. E*, **70**(3), 036107.

Battiti, R. and Protasi, M. (1997). Reactive search, a History-Sensitive Heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, **2**, 2.

Bianchi, L., Gambardella, L. M., and Dorigo, M. (2002). An Ant Colony Optimization Approach to the Probabilistic Traveling Salesman Problem. In *Proceedings of PPSN-VII, seventh international conference on parallel problem solving from nature*, volume 2439/2002, pages 883–892. Springer Berlin / Heidelberg.

Boese, K. D. (1995). Cost Versus Distance in the Traveling Salesman Problem. Technical Report TR-90018, UCLS Computer Science Dept., Lost Angeles, CA.

Borchers, B. and Furman, J. (1999). A Two-Phase Exact Algorithm for MAX-SAT and Weighted MAX-SAT Problems. *Journal of Combinatorial Optimization*, **2**(4), 299–306.

Boughaci, D., Drias, H., and Benhamou, B. (2004). Solving MAX-SAT Problems Using a Memetic Evolutionary Meta-Heuristic. volume 1, pages 480–484.

Chen, C.-A. and Gupta, S. K. (1996). A Satisfiability-Based Test Generator for Path Delay Faults in Combinational Circuits. In *Proceedings of the 33rd annual Design Automation Conference*, pages 209–214, Las Vegas, Nevada, United States. ACM.

Chen, P. and Keutzer, K. (1999). Towards True Crosstalk Noise Analysis. In *ICCAD '99: Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pages 132–138, Piscataway, NJ, USA. IEEE Press.

Conzalez, C., Lozano, J. A., and Larranaga, P. (2000). Analyzing the PBIL Algorithm by Means of Discrete Dynamical Systems. *Complex Systems*, **12**, 456–479.

Cook, S. A. (1971). The Complexity of Theorem-Proving Procedures. Proceedings of the 3rd annual ACM symposium on theory of computing, pages 151–8, Shaker Heights, OH, USA. Assoc. Computing Machinery.

Crawford, J. M. and Auton, L. D. (1993). Experimental Results on the Crossover Point in Satisfiability Problems. Proceedings of the Eleventh National Conference on Artificial Intelligence, pages 21–7, Washington, DC, USA. AAAI Press.

Crawford, J. M. and Auton, L. D. (1996). Experimental Results on the Crossover Point in Random 3-SAT. *Artificial Intelligence*, **81**(1-2), 31–57.

David, S. J. (1990). Local Optimization and the Traveling Salesman Problem. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, pages 446 – 461. Springer-Verlag.

Davis, M. and Putnam, H. (1960). A Computing Procedure for Quantification Theory. *Journal of the ACM (JACM)*, **7**(3), 201–215.

De Jong, A. K. and Spears, M. S. (1989). Using Genetic Algorithms to Solve NP-Complete Problems. In *Proceedings of The Third International Conference on Genetic Algorithms*, volume 93172, pages 124–132, George Mason university, United States. Morgan kaufmann Publishers Inc. San Francisco, CA, USA.

Ding, C. and He, X. (2004). K-means Clustering via Principal Component Anaylsis. In *Proceedings of the twenty-first international conference on Machine learning*, volume 69, page 29, Banff, Alberta, Canada. ACM International Conference Proceeding Series.

Dorigo, M. and Gambardella, L. M. (1997). Ant Colony System: a Cooperative Learning Approach to the Traveling Salesman Problem. *Evolutionary Computation, IEEE Transactions on*, **1**(1), 53–66. 1089-778X.

Du, Q., Faber, V., and Gunzburger, M. (1999). Centroidal Voronoi Tessellations: Applications and Algorithms. *SIAM Review*, **41**(4), 637–676.

Eberhart, R. (1995). A New Optimizer Using Particle Swarm Theory. In *Sixth International Symposium on Micro Machine and Human Science*, pages 39–43. IEEE Serivce Cneter, Piscataway, NJ.

Fallah, F., Devadas, S., and Keutzer, K. (1998). Functional Vector Generation for HDL Models Using Linear Programming and 3-Satisfiability. In *DAC '98: Proceedings of the 35th annual Design Automation Conference*, pages 528–533, New York, NY, USA. ACM.

Fisher, R. A. (1988a). Machine Leanring Repository. Center for Machine Learning and Intelligent Systems. http://archive.ics.uci.edu/ml/datasets/Iris.

Fisher, R. A. (1988b). Machine Leanring Repository. Center for Machine Learning and Intelligent Systems. http://archive.ics.uci.edu/ml/datasets/Forest+Fires.

Fisher, R. A. (1988c). Machine Leanring Repository. Center for Machine Learning and Intelligent Systems. http://archive.ics.uci.edu/ml/datasets/Wine.

Freisleben, B. and Merz, P. (1996). New Genetic Local Search Operators for the Traveling Salesman Problem. In *Parallel Problem Solving from Nature PPSN IV*, pages 890–899. Springer Berlin / Heidelberg.

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

Gent, I. P. and Walsh, T. (1993). An Empirical Analysis of Search in GSAT. *J. Artif. Int. Res.*, **1**(1), 47–59.

Gi-Joon, N., Karem, A. S., and Rob, A. R. (1999). Satisfiability-Based Layout Revisited: Detailed Routing of Complex FPGAs via Search-Based Boolean SAT. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, pages 167–175, Monterey, California, United States. ACM.

Glover, F. and Laguna, M. (2002). *Tabu Search*. Kluwer Academic Publishers, fifth edition.

Golden, B., Bodin, L., Doyle, T., and W. Stewart, J. (1980). Approximate Traveling Salesman Algorithms. *Operations Research*, **28**(3), 694–711.

Gomes, C. P., Kautz, H., Sabharwal, A., and Selman, B. (2008). Satisfiability solvers.

Gori, M. and Tesi, A. (1992). On the Problem of Local Minima in Backpropagation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **14**(1), 76–86. 0162-8828.

Gu, J. and Huang, X. (1994). Efficient Local Search with Search Space Smoothing: A Case Study of the Traveling Salesman Problem (TSP). *IEEE Trans. SMC,*, **24**, 728–735.

Hamm, L., Brorsen, B. W., and Hagan, M. T. (2007). Comparison of Stochastic Global Optimization Methods to Estimate Neural Network Weights. *Neural Process. Lett.*, **26**(3), 145–158.

Hancock, P. J. (1992). Genetic Algorithms and Permutation Problems: a Comparison of Recombination Operators for Neural Net Structure Specification. In *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 109–122, Baltimore. IEEE.

Hartigan, J. A. and Wong, M. A. (1979). A K-means Clustering Algorithm. *Applied Statistics*, **28**(1), 100–8.

Haykin, S. (1999). *Neural Networks A Comprehensive Foundation*. Prentice-Hall, Inc, second edition.

Hecht-Nielsen, R. (1989). Theory of the Backpropagation Neural Network. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, volume 1, pages 593–605.

Helsgaun, K. (2006). An Effective Implementation of K-Opt Moves for the Lin-Kernighan TSP Heuristic. Technical report, Roskilde University. (Revised Nov. 2007).

Hirsch, E. A. and Kojevnikov, A. (2005). UnitWalk: A New SAT Solver that Uses Local Search Guided by Unit Clause Elimination. *Annals of Mathematics and Artificial Intelligence*, **43**(1-4), 91–111.

Hoos, H. H. and Stützle, T. (accessed auguest 2007). SATLIB - Benchmark Problems. Computer Science Department of the University of British Columbia in Vancouver (Canada). http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html.

Huai-Kuang, T., Jinn-Moon, Y., Yuan-Fang, T., and Cheng-Yan, K. (2004). An Evolutionary Algorithm for Large Traveling Salesman Problems. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, **34**(4), 1718–1729. 1083-4419.

Jansen, T. and Wegener, I. (1999). On the Analysis of Evolutionary Algorithms — a Proof that Crossover can Really Help. In J. Nešetřil, editor, *Proceedings of the 7th Annual European Symposium on Algorithms (ESA'99)*, pages 184–193, Berlin. Springer.

Kaur, D. and Murugappan, M. M. (2008). Performance Enhancement in Solving Traveling Salesman Problem Using Hybrid Genetic Algorithm. In *Fuzzy Information Processing Society, 2008. NAFIPS 2008. Annual Meeting of the North American*, pages 1–6.

Kilby, P., Slaney, J., Thiebaux, S., and Walsh, T. (2005). Backbones and Backdoors in Satisfiability. volume 3 of *Proceedings of the National Conference on Artificial*

Intelligence, pages 1368–1373, Pittsburgh, PA, United States. American Association for Artificial Intelligence, Menlo Park, CA 94025-3496, United States.

Kirkpatrick, S., Gelatt, C. D., Jr., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, **220**, 671–680.

Kuk-Hyun, H. and Jong-Hwan, K. (2002). Quantum-Inspired Evolutionary Algorithm for a Class of Combinatorial Optimization. *Evolutionary Computation, IEEE Transactions on*, **6**(6), 580–593. 1089-778X.

Larrabee, T. (1992). *Test Pattern Generation Using Boolean Statisfiability*, volume 11. Institute of Electrical and Electronics Engineers, New York, NY, ETATS-UNIS.

Laskov, P., Dssel, P., Schfer, C., and Rieck, K. (2005). Learning Intrusion Detection: Supervised or Unsupervised? In *Image Analysis and Processing ICIAP 2005*, volume 3617/2005, pages 50–57. Springer Berlin / Heidelberg.

Layeb, A. and Saidouni, D.-E. (2008). A New Quantum Evolutionary Local Search Algorithm for MAX 3-SAT Problem. In *in Proceedings of International Conference on Intelligent Computing (ICIC2008)*, volume 5271/2008, pages 172–179, China. Springer Berlin / Heidelberg.

Li, D., Wong, K. D., Hu, Y. H., and Sayeed, A. M. (2002). Detection, Classification and Tracking of Targets in Distributed Sensor Networks. In *IEEE Signal Processing Magazine*, pages 17–29.

Lin, S. and Kernighan, W. (1973). An Effective Heuristic Algorithm for the Traveling Salesman Problem. *Operations Research*, **21**, 498–516.

Linde, Y., Buzo, A., and Gray, R. M. (1980). An Algorithm for Vector Quantization Design. *IEEE Transactions on Communications*, **28**(1), 84–94.

Magoulas, G. D., Vrahatis, M. N., and Androulakis, G. S. (1999). Improving the Convergence of the Backpropagation Algorithm Using Learning Rate Adaptation Methods. *Neural Comput.*, **11**(7), 1769–1796.

Maniezzo, V. (1994). Genetic Evolution of the Topology and Weight Distribution of Neural Networks. *IEEE Transactions on Neural Networks*, **5**(1), 39–53.

Marques-Silva, J. P. and A. Sakallah, K. (2000). Boolean Satisfiability in Electronic Design Automation.

Marques-Silva, J. P. and Sakallah, K. A. (1999). GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers*, **48**(5), 506–21.

Martin, D., George, L., and Donald, L. (1962). A Machine Program for Theorem-Proving.

Mathias, K. and Whitley, D. (1992). Genetic Operators, the Fitness Landscape and the Traveling Salesman Problem. In *Parallel Problem Solving from Nature*, pages 219–228. Elsevier Science Publishers.

McAllester, D., Selman, B., and Kautz, H. (1997). Evidence for Invariants in Local Search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 321–326. American Association for Artificial Intelligence.

Mezard, M. and Zecchina, R. (2002). The Random K-Satisfiability Problem: From an Analytic Solution to an Efficient Algorithm. *Physical Review E*, **66**, 056126.

Mézard, M., Mora, T., and Zecchina, R. (2005). Clustering of Solutions in the Random Satisfiability Problem. *Phys. Rev. Letts.*, **94**, 197205.

Mitchell, D., Selman, B., and Levesque, H. (1992). Hard and Easy Distributions of SAT Problems. pages 459–465, San Jose, CA, USA. Publ by AAAI, Menlo Park, CA, USA.

Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., and Troyansky, L. (1999). Determining Computational Complexity from Characteristic 'Phase Transitions'. *Nature*, **400**(6740), 133–7.

Montana, D. J. and Davis, L. (1989). Training Feedforward Neural Networks Using Genetic Algorithms. In *IJCAI'89: Proceedings of the 11th international joint conference on Artificial intelligence*, pages 762–767, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Montanari, A., Parisi, G., and Ricci-Tersenghi, F. (2004). Instability of One-Step Replica-Symmetry-Broken Phase in Satisfiability Problems. *Journal of Physics A: Mathematical and General*, **37**(6), 2073–2091.

Mühlenbein, H. (1992). Parallel Genetic Algorithms in Combinatorial Optimization. In *Computer Science and Operations Research*, pages 441–456. Pergamon Press.

P. C. Cosman, a., Oehler, K. L., Riskin, E. A., and Gray, R. M. (1993). Using Vector Quantization for Image Processing. In *Proceedings of IEEE*, volume 81, pages 1326–1341.

Papadimitriou, C. H. (1991). On Selecting a Satisfying Truth Assignment (Extended Abstract). In *SFCS '91: Proceedings of the 32nd annual symposium on Foundations of computer science*, pages 163–169, Washington, DC, USA. IEEE Computer Society.

Parkes, A. J. (1997). Clustering at the Phase Transition. In *In Proc. of the 14th Nat. Conf. on AI*, pages 340–345. AAAI Press / The MIT Press.

Parkes, A. J. (2001a). Distributed Local Search, Phase Transitions and Polylog Time. In *In Proceedings of the workshop on Stochastic Search Algorithms, held at Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01*.

Parkes, A. J. (2001b). Exploiting Solution Clusters for Coarse-Grained Distributed Search. In *In Proceedings of the workshop on Distributed Constraint Reasoning, held at Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01*.

Parkes, A. J. (2002). Easy Predictions for the Easy-Hard-easy Transition. In *Eighteenth national conference on Artificial intelligence*, pages 688–694, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Parkes, A. J. and Walser, J. P. (1996). Tuning Local Search for Satisfiability Testing. volume 1 of *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 356–62, Portland, OR, USA. MIT Press.

Pea, J. M., Lozano, J. A., and Larraaga, P. (1999). An Empirical Comparison of Four Initialization Methods for the K-Means Algorithm. *Pattern Recognition Letters*, **20**(10), 1027–1040.

Pelikan, M. and Goldberg, D. E. (2003). Hierarchical BOA Solves Ising Spin Glasses and MAXSAT. In *Genetic and Evolutionary Computation GECCO 2003*, volume 2724/2003, page 213. Springer Berlin / Heidelberg.

Pelleg, D. and Moore, A. (2000). X-means: Extending K-means with Efficient Estiamtion of the Number of Clusters. In *Proceedings of the Seventheenth International Conference on Machine Learning*, pages 727–734.

Peter, M. and Bernd, F. (1999). Fitness Landscapes and Memetic Algorithm Design. In *New ideas in optimization*, pages 245–260. McGraw-Hill Ltd., UK. 329081.

Prugel-Bennett, A. (2007). Finding Critical Backbone Structures with Genetic Algorithms. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1343 – 1348, London, England. ACM Press.

Qingfu, Z. (2004). On Stability of Fixed Points of Limit Models of Univariate Marginal Distribution Algorithm and Factorized Distribution Algorithm. *Evolutionary Computation, IEEE Transactions on*, **8**(1), 80–93. 1089-778X.

Radcliffe, N. J. (1990). *Generic Neural Networks on MIMD Computers*. Ph.D. thesis, University of Edinburgh.

Ramalhinho, H., Martin, O. C., and Sttzle, T. (2000). Iterated Local Search. Technical Report Economics and Business Working Papers Series; 513.

Reed, R. D. and II, R. J. M. (1999). *Neural Smiting: Supervised Learning in Feedforward Artificial Neural Networks*. Massachusetts Institute of Technology, first edition.

Riedmiller, M. and Braun, H. (1993). A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 586 – 591, San Francisco, CA.

Rogers, A. and Prügel-Bennett, A. (2000). The Dynamics of a Genetic Algorithm on a Model Hard Optimization Problem. *Complex Systems*, **11**(6), 437–464.

Rogers, A. and Prügel-Bennett, A. (2001). A Solvable Model of a Hard Optimization Problem. In L. Kallel, B. Naudts, and A. Rogers, editors, *Theoretical Aspects of Evolutionary Computing*, Natural Computing, pages 207–221, Berlin. Springer.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Representations by Back-Propagating Errors. *Nature*, **323**, 533 – 536.

Safarpour, S., Safarpour, S., Mangassarian, H., Veneris, A., Liffiton, M. H. A. L. M. H., and Sakallah, K. A. A. S. K. A. (2007). Improved design debugging using maximum satisfiability improved design debugging using maximum satisfiability. In H. Mangassarian, editor, *Formal Methods in Computer Aided Design*, pages 13–19.

Selman, B. (1995). Stochastic Search and Phase Transitions: AI Meets Physics. volume 1 of *IJCAI-95. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 998–1002, Montreal, Que., Canada. Morgan Kaufmann Publishers.

Selman, B. and Kautz, H. A. (1993). An Empirical Study of Greedy Local Search for Satisfiability Testing. Proceedings of the Eleventh National Conference on Artificial Intelligence, pages 46–51, Washington, DC, USA. AAAI Press.

Selman, B., Levesque, H., and Mitchell, D. (1992). A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, San Jose, CA, USA. Publ by AAAI, Menlo Park, CA, USA.

Selman, B., Kautz, H. A., and Cohen, B. (1994). Noise Strategies for Improving Local Search. volume 1 of *Proceeding of the Twelfth National Conference on Artificial Intelligence*, pages 337–43, Seattle, WA, USA. MIT Press.

Sexton, R. S., Dorsey, R. E., and Johnson, J. D. (1999). Beyond Back Propagation: Using Simulated Annealing for Training Neural Networks. *J. End User Comput.*, **11**(3), 3–10.

Shapiro, J. L. and Prügel-Bennett, A. (1997). Genetic Algorithms Dynamics in Two-Well Potentials with Basins and Barriers. In R. K. Belew and M. D. Vose, editors, *Foundations of Genetic Algorithms 4*, pages 101–116, San Francisco. Morgan Kaufmann.

Smyth, K., Hoos, H. H., and Stützle, T. (2003). Iterated Robust TABU Search for MAX-SAT. In *Proceedings of the Sixteenth Conference of the Canadian Society for Computational Studies of Intelligence (AI-03)*, volume 2671 of *Lecture Notes in Artificial Intelligence*, pages 129–144.

Stephan, P., Brayton, R. K., and Sangiovanni-Vincentelli, A. L. (1996). Combinational Test Generation Using Satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **15**(9), 1167–76.

Syswerda, G. (1993). Simulated Crossover in Genetic Algorithms. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, San Mateo. Morgan Kaufmann.

Toth, P. and Vigo, D. (2002). *The Vehicle Routing Problem (Monographs on Discrete Mathematics and Applications)*. SIAM.

Trejo, L. A. and Sandoval, C. (1995). Improving Back-Propagation: Epsilon-Back-Propagation. In *IWANN '96: Proceedings of the International Workshop on Artificial Neural Networks*, pages 427–432, London, UK. Springer-Verlag.

Valenzuela, C. L. and Jones, A. J. (1997). Estimating the Held-Karp Lower Bound for the Geometric TSP. *European Journal of Operational Research*, **102**, 157–175.

Villagra, M. and Barán, B. (2007). Ant Colony Optimization with Adaptive Fitness Function for Satisfiability Testing. In *Proceedings on Logic, Language, Information and Computation*, volume 4576/2007, pages 352–361. Springer Berlin / Heidelberg.

Wagsta, K., Cardie, C., and Schroedl, S. (2001). Constrained K-means Clustering with Background Knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 577–584. Morgan Kanfmann, San Francisco, CA.

Wah, B. W. and Yi, S. (1997). Discrete Lagrangian-Based Search for Solving MAX-SAT Problems. volume 1 of *IJCAI-97. Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 378–83.

Wang, S., Zhang, W. W., and Wang, Y. S. (2002). Fingerprint Classification by Directional Fields. page 395, Proceedings of the 4th IEEE International Conference on Multimodal Interfaces. International Conference on Multimodal Interfaces.

Watson, R. A. (2001). Analysis of Recombinative Algorithms on a Non-Separable Building-Block Problem. In W. N. Martin and W. M. Spears, editors, *Foundations of Genetic Algroithms (FOGA-6)*, pages 69–89, San Francisco. Morgan Kaufmann.

Watson, R. A. and Jansen, T. (2007). A Building-Block Royal Road Where Crossover is Provably Essential. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, pages 1452–1459.

Weiss, m. A. (2007). *Data Structures and Algorithm Analysis in Java*. Pearson Education, Inc., second edition.

Whitley, D., Starkweather, T., and Shaner, D. (1991). The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination. pages 350–372. Van Nostrand Reinhold, l. davis, ed edition.

Xiaoyue, F., Blanzieri, E., and Yanchun, L. (2008). Improved Quantum-Inspired Evolutionary Algorithm and its Application to 3-SAT Problems. In *Computer Science and Software Engineering, 2008 International Conference on*, volume 1, pages 333–336.

Ya, Z., Ya, Z., Hongyuan, Z., Chao-Hisen, C., and Xiang Ji, A. X. J. (2005). Protein interaction inference as a max-sat problem protein interaction inference as a max-sat problem. In Z. Hongyuan, editor, *Computer Vision and Pattern Recognition, 2005 IEEE Computer Society Conference on*, volume 3, pages 146–146.

Zabih, R. and McAllester, D. (1988). A Rearrangement Search Strategy for Determining Propositional Satisfiability. AAAI 88. Seventh National Conference on Artificial Intelligence, pages 155–60, Saint Paul, MN, USA. Morgan Kaufmann.

Zhang, L., Madigan, C. F., Moskewicz, M. H., and Malik, S. (2001). Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. In *ICCAD '01: Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 279–285, Piscataway, NJ, USA. IEEE Press.

Zhang, W. (2001). Phase Transitions and Backbones of 3-SAT and Maximum 3-SAT. volume 2239 of *Principles and Practice of Constraint Programming - CP 2002. 7th International Conference, CP 2001. Proceedings (Lecture Notes in Computer Science)*, pages 153–67, Paphos, Cyprus. Springer-Verlag.

Zhang, W. (2004). Configuration Landscape Analysis and Backbone Guided Local Search. Part 1: Satisfiability and Maximum Satisfiability. *Artificial Intelligence*, **158**(1), 1–26.

Zhang, W. and Looks, M. (2005). A Novel Local Search Algorithm for the Traveling Salesman Problem that Exploits Backbones. In *Proceedings of the 19th International Joint Conference on Artificial Intellegence*, pages 343–348, Edinburgh, Scotland.

Zhang, W., Rangan, A., and Looks, M. (2003). Backbone Guided Local Search for Maximum Satisfiability. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, volume IJCAI'03, pages 1179–1186.

Zhao, X. and Zhang, W. (2004). Efficient Strategies for (Weighted) Maximum Satisfiability. volume 3258 of *Principles and Practice of Constraint Programming - CP 2004. 10th International Conference, CP 2004. Proceedings (Lecture Notes in Comput. Sci.)*, pages 690–705, Toronto, Ont., Canada. Springer-Verlag.