Empirical Analysis of Local Search Algorithms and Problem Difficulty in Satisfiability

by

Dave Tae Shik Yoon

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Mechanical and Industrial Engineering
University of Toronto

Abstract

Empirical Analysis of Local Search Algorithms and Problem Difficulty in Satisfiability

Dave Tae Shik Yoon
Master of Applied Science
Graduate Department of Mechanical and Industrial Engineering
University of Toronto
2006

The central thesis of this dissertation is that an empirical analysis leads to a deeper understanding of local search methods and problem difficulty in satisfiability (SAT) and that the understanding can form a foundation for better algorithm designs. The investigation of problem difficulty for local search algorithms has received much attention from researchers, and this work is a continuation of such effort.

We provide evidence that the decrease in the local search cost in the over-constrained region for satisfiable instances is largely due to the effects of less extensive high-quality local minima compared to the under- and critically-constrained region. We also show that a backbone-guided local search algorithm works well for over-constrained instances because of its robustness to backbone estimation. Finally, our results from problem difficulty lead to the integration of path relinking with existing local search algorithms, which is demonstrated to be effective on various problem domains.

Acknowledgements

This work would not have been possible without the help of many people, notably:

My parents for their unconditional love and support. You are simply the best.

My supervisor Professor Beck for his insight, patience, and zest for research, which inspired and encouraged me to plow ahead. Also, thank you for taking a chance on a guy with little experience in the field.

Dr. Jean-Paul Watson for providing me with the unbelievable opportunity to work at Sandia National Laboratories, as well as his expertise and comments, which contributed to a significant portion of the thesis.

Dave Tompkins, Makoto Yokoo, Zhaohui Fu, Professor Bacchus, and Tom Carchrae for their helpful comments.

My friends for encouragement, laughs, and for occasionally asking me about my research.

Katie. You are always there for me.

I would also like to thank Sandia National Laboratories and Ontario Graduate Scholarship in Science and Technology for their financial support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The central thesis of this dissertation is that an empirical analysis leads to a deeper understanding of local search methods and problem difficulty in satisfiability (SAT) and that the understanding can form a foundation for better algorithm designs. The investigation of problem difficulty for local search algorithms has received much attention from researchers [51, 67, 57, 61], and this work is a continuation of such effort. Our results in problem difficulty provide the motivation to apply an existing metaheuristic from the Operations Research literature to local search algorithms for SAT.

In particular, in this dissertation:

- we identify a major contributing factor for the easy-hard-easy pattern in problem difficulty seen across the phase transition region for SAT local search algorithms.
- we characterize backbones for SAT and job-shop scheduling problems and identify the reasons for the success of a backbone-guided local search algorithm for SAT.
- motivated from the problem difficulty analysis, we apply path relinking, which has been successfully integrated in the state-of-the-art job-shop scheduling algorithm, to local search algorithms for SAT.

1.1 Background Information on Satisfiability

The *Satisfiability (SAT)* problem in propositional logic is an important class of problems in Artificial Intelligence (AI). Because of its status as the prototypical NP-complete problem, it is especially critical in the area of complexity theory [29]. In addition to its prominence in theoretical domain, SAT has shown its usefulness in many application problems. Many combinatorial decision/optimization problems can be effectively solved using SAT, and its range of applications is growing with the development of SAT-encoding techniques [48, 31] and powerful solvers [47, 25, 24, 50].

A satisfiability problem is conceptually very simple: it is a combinatorial decision problem, where the goal is to determine whether there exists a set of assignments to Boolean variables such that a given *formula* can be *satisfied*. The formula consists of Boolean variables joined using Boolean operators AND, OR, and NOT. If there exists a set of assignments that satisfy all the constraints (*clauses* in SAT terminology), the formula is said to be satisfiable, and in the opposite case, it is unsatisfiable (see Chapter 2 for a more detailed description of SAT).

There are specialized algorithms for SAT that attempt to find a set of assignments that will satisfy a given formula (prove satisfiability for some algorithms). There are two types of SAT solvers: one is called a constructive or systematic search method, which considers *all* the possible combinations of variable assignments (explicitly or implicitly) to determine whether a formula is satisfiable. The other type is a local search method, which starts with a set of random assignments and tries to incrementally improve (minimize) its objective function by changing its variable assignment one at a time. Generally, the objective function for a local search algorithm is the number of unsatisfied clauses.

1.2 Motivations

This thesis is motivated by the need for a better understanding for local search algorithms, particularly for SAT. Not only does the research have academic merit in terms of advancing the research in the area, but also, it has practical implications as it enables better algorithm designs and potentially allows wider range of real-world problems to be solved through satisfiability. Our work is specifically motivated by the following:

Problem difficulty for local search Search cost for algorithms in general is sensitive to the tightness of the constraints for a given problem. For constructive search algorithms, for example, there is a level of constrainedness that makes a problem especially hard [7, 10]. In such a case, the problem is neither under-constrained, which would have abundance of satisfying solutions, or over-constrained, which would allow quick pruning of unpromising search space. Thus, generally, for constructive search algorithms, problem difficulty follows the easy-hard-easy pattern across the under-, critically-, and over-constrained regions [7, 10]. Interestingly, such a pattern in problem difficulty is observed in SAT local search algorithms as well. We are interested in understanding why such pattern exists when there is no notion of pruning or search space reduction for local search algorithms. Among the notable literature in this area of research, we base our work on Parkes et al. [51], Yokoo [67], and Singer et al. [57] for SAT domain and Watson et al. [61] for another, non-SAT domain.

Backbones in local search A backbone variable is one that has the same truth assignment in all optimal solutions to a given problem [51, 17]. Recently, algorithms that take advantage of the estimates of membership in the set of backbone variables were introduced [68, 17]. This type of backbone-guidance biases the underlying algorithms such that they concentrate on correctly setting the variables that are likely backbones before the non-backbone variables. However, whether these algorithms really take advantage of the backbone information is not well known. The goal here is to identify how well the backbone-guided algorithms estimate the backbone variables and test if the backbone variables are indeed responsible for their success.

Long-term memory in local search Long-term memory is an important feature in various search algorithms such as Ant Colony Optimization [16], Genetic Algorithm [26], and

Backbone-Guided search [68]. Based on a Tabu Search, i-TSAB [49] is one of the state-of-the-art algorithms for job-shop scheduling problems [62]; this algorithm also uses long-term memory in the form of *path relinking*. Motivated by these successful algorithms and our problem difficulty results, we apply long-term memory to local search algorithms in SAT.

Constructive versus local search On problems where the proof of satisfiability is not important but the essence is in finding a satisfying solution, a local search algorithm is often as good or better than a constructive search algorithm [56, 53]. Due to the differences in their approaches, however, their performance varies for various problem domains and even for individual instances in the same domain. We would like to identify the strengths and weaknesses of both algorithms on various domains. Further, we investigate whether those factors known to affect local search cost will also affect constructive search cost.

Applications of SAT In addition to the critical role it assumes in complexity theory, satisfiability can be applied to practical problems as a solution technique. It has been applied to graph colouring problems [56], scheduling problems [9], logistics planning problems [41], and most recently to formal verification problems [1, 5]. Water Network Security problem is a real-life application problem from Sandia National Laboratories.[1] Being mostly a binary decision problem, it is well-suited to be solved as SAT. Although there are other methods such as Integer Programming and Greedy heuristic, the former is susceptible to the tractability issue while the latter does not guarantee optimality. Here, the goal is to evaluate the feasibility of applying SAT to the Water Network Security problem.

1.3 Outline

The outline of the thesis is as follows:

Chapter 2 provides background information for the thesis and looks at the literature relevant to the area of our research interest. A few of the well-known algorithms for both constructive and local search methods are presented as are some popular techniques found in modern SAT solvers. We also extend the literature review to local search solvers and techniques outside of satisfiability. Such AI domains as Constraint Programming and Scheduling share many similarities with SAT in how the problems are approached and the solvers are designed. Techniques used in those domains are examined with a particular focus on the use of memory. Literature on the problem difficulty in local search is also examined. We present many suggested reasons behind varying local search cost with different levels of constrainedness. Finally, literature on applications of SAT is discussed.

Chapter 3 presents an application problem that is solved as a satisfiability problem. Aside from the problem itself, it also serves as a motivation for analyzing SAT. In Water Network Security problem, we are interested in being able to locate the source of terrorist attack in a given water distribution network. Here, we study the feasibility of solving the problem as satisfiability and compare the results to the existing methods.

[1]In New Mexico, USA.

Chapter 4 compares two of the most prominent search algorithms for constructive and local search methods. zChaff [47] and Walksat [55] are run on various domains of SAT ranging from random k-SAT to highly structured problems. We also apply some of the factors that are known to affect local search cost to the constructive search method and see how the performance differs for the two types of algorithms.

Chapter 5 presents detailed analyses of problem difficulty for local search algorithms. Local search cost varies with respect to the levels of constrainedness, which is typically measured by the ratio of the number of clauses and the number of variables. Around the region where a problem instance is "critically-constrained", local search cost shows unexpected behaviours. We investigate some of the factors that affect the search cost by verifying Yokoo [67] and Singer et al.'s [57] work, and suggest a unifying explanation for the phenomenon seen around the critically-constrained region. We also examine Watson et al.'s [61] work on the problem difficulty on job-shop scheduling problems (JSP) and test specific conjectures made with respect to SAT problems.

Chapter 6 examines the idea of backbone and its usage in SAT solvers. Motivated from Watson et al.'s work on JSP, the relationship between the number of optimal solutions and the backbone size is examined. We introduce a new technique for measuring the effect of non-backbone variables and explain the different characteristics observed from SAT and JSP. Also, Zhang's [68] backbone-guided local search algorithm is analyzed to verify the backbone's influence on the algorithm. To this end, we conduct an experiment that evaluates how the different levels of prior backbone information affect the algorithm's performance. Further, we independently test how accurately the algorithm estimates backbone values.

Chapter 7 discusses how the idea of long-term memory and elite pool maintenance can be incorporated as a metaheuristic for local search algorithms for SAT. In particular, *path relinking*, which has been successfully applied to a tabu search for job-shop scheduling problems in i-TSAB [49], is integrated with Walksat and Novelty+ [44, 30] using a design methodology similar to that used for the tabu search [62]. Their performance is compared with their respective underlying algorithms, as well as Zhang's backbone-guided algorithm.

Chapter 8 provides the conclusions and highlights the major contributions of the work. It ends with suggestions for further research stemming from the thesis.

1.4 Summary of Contributions

The contributions of the thesis are as follows:

- We extend the previous work done on the problem difficulty for local search algorithms. Confirming Singer et al.'s [57] work by different methods, we identify the extensiveness of local minima to be a major factor for local search cost.

- We identify the driving force behind a backbone-guided local search algorithm and the reason for its success on certain instances. The key is in the way the backbone estimates can guide the underlying algorithm without requiring a very accurate backbone information.

- A new way of measuring the global effect of a non-backbone variable for a given instance, namely *degree of implication*, is introduced. It is a measure of how tightly the variables are linked to one another. A study on SAT and job-shop scheduling problems shows that non-backbone variables in SAT have much greater degree of implication on average, explaining the weaker correlation between the number of optimal solutions and the backbone size in SAT compared to JSP.

- Long-term memory in the form of path relinking is adopted to local search algorithms for SAT. By taking advantage of the diversification mechanism, the algorithms with path relinking show promising results. Further, through empirical testing, the importance of diversity in the elite pool is shown for algorithms with long-term memory.

- The performance of a local search algorithm is compared against a constructive search algorithm on various problem domains. We confirm the belief that a local search algorithm is superior on random instances while a constructive search algorithm is better on structured instances. Further, some of the factors that affect local search cost do not influence constructive search cost to the same degree.

- The Water Network Security problem is encoded as SAT. With further research, there is potential for incorporating SAT as a verification tool for this type of problem.

Chapter 2

Literature Review

In this chapter, we look at the previous work done on satisfiability (SAT) and problem difficulty in local search algorithms.[1] First, various SAT solvers and SAT solving techniques are discussed. Here, we identify two major techniques for solving SAT problems as well as meta-heuristics that can be used to drive these solvers. We also look at local search algorithms in other combinatorial optimization problems and see how the features in the problems may be useful to SAT solvers. Secondly, we examine the work done on problem difficulty in SAT. These include the strengths and weaknesses of local search and constructive search algorithms for various problem types and possible explanations for the variance in their performance. Finally, some applications of SAT are presented.

2.1 Satisfiability

Satisfiability (SAT) is a combinatorial decision problem that plays a prominent role in complexity theory and Artificial Intelligence [34]. The goal of satisfiability problems is to show whether the given logical formula is satisfiable or not. Typically they are presented as conjunctive normal form (CNF) formulas. A CNF formula is made up of *clauses*, which are made up of *literals*. Clauses are disjunction of literals, and literals are signed variables, positive or negative. CNF formula is satisfiable if and only if all of its clauses can be satisfied with all the variables assigned to a particular value, namely $true$ or $false$. In turn, a clause is satisfiable if and only if at least one literal in the clause is true. A CNF formula is shown below as an example:

$$F = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_3 \vee x_1 \vee \neg x_2)$$

All literals inside the clause (a bracket of literals) are joined by or's ($\vee$) and each clause is joined by and's ($\wedge$). Again, if F can be made $true$, the formula is satisfiable, and if not, the formula is unsatisfiable. In this instance, setting x_1 to $true$ and x_3 to $true$ will satisfy all three clauses (x_2 can be either $true$ or $false$).

[1]Some of the material that is specific only to certain chapters are presented in the respective chapters.

Maximum Satisfiability Maximum satisfiability, or MAX-SAT, can be seen as a generalization of SAT in which, instead of satisfying all clauses of a given formula F, the objective is to satisfy as many clauses as possible [34]. An optimal solution to a MAX-SAT instance is a variable assignment that satisfies a maximal number of clauses in F. In SAT, an optimal solution is equivalent to a satisfying solution.

2.2 Constructive Search Methods

There are two major methods for solving a SAT problem: one is a *constructive search*, also known as a *backtracking* or *systematic search*, where every possible combination of variable assignment is tried (either explicitly or implicitly). Another solving method is *local search*, which starts from a random set of assignments for all variables and makes incremental moves towards minimizing the number of unsatisfied clauses. This section examines the mechanics of a constructive search method and introduces various techniques that have been created over the years to improve its performance.

The most well-known constructive search algorithm is *Davis-Putnam, Logemann, and Loveland (DPLL)* [12] procedure [14]. Most of the state-of-the-art constructive SAT solvers are based on DPLL. The procedure for DPLL can be seen in Figure 2.1.

A DPLL algorithm chooses a variable to assign (if no unit clauses are available at first) a value according to some heuristic. This decision and the level with respect to the search tree is recorded. Based on this assignment, clauses containing the literal corresponding to the assigned variable can be deleted, and the opposing literals of the assigned variable can be deleted from their respective clauses. In the example CNF formula in Section 2.1, if the heuristic decides to assign $x_1 = false$, the second clause will be deleted from F since it is satisfied and no longer needs to be considered (unless there is backtracking, which we will get into later). Further, the literal x_1 in the first and third clause will be deleted from its respective clauses. This leads to a simplified formula $F' = (x_2) \wedge (\neg x_3 \vee \neg x_2)$.

At this point, *unit propagation*, a powerful technique that drives DPLL, takes place. Here, unit clauses (those clauses with only one literal) are identified and set to their correct values. In the example above, the unit propagation will set $x_2 = true$ in order to satisfy the first clause, leading to yet another simplified formula $F'' = (\neg x_3)$, which can be satisfied with $x_3 = false$.

If there is a conflict in variable assignments (thus an empty clause found), the search can undo (*i.e.* backtrack) all the unit propagations made up to the last point where a heuristic decision was made to a variable Q (see Figure 2.1). If Q has not been tried with both values, it can be flipped and the unit propagations can take place again. If Q has been tried with both values, the algorithm further backtracks until the last heuristic decision was made. If no such decision exists, the formula F is unsatisfiable. On the other hand, whenever there are no more clauses left in F (from clause deletion resulting from variable assignments), the corresponding set of assignments is a satisfying solution.

2.2.1 Improvement of DPLL

DPLL-based algorithms have evolved considerably since their introduction in 1962. Although DPLL-based algorithms have not changed very much in terms of its core propagation tech-

```
DPLL $(F)$
Input: a CNF formula $F$
Output: a decision of whether $F$ is satisfiable
unit_propagation($F$)
if an empty clause is generated, return 'unsatisfiable'
else if all variables are assigned a value, return 'satisfiable'
else
    $Q :=$ some unassigned variable
    return DPLL $(F \wedge Q)$ $OR$ DPLL $(F \wedge \neg Q)$
```

Figure 2.1: DPLL pseudo-code [14].

nique, through the use of clever heuristics and *conflict learning* among other techniques, significant improvement in performance has been observed. Here, we introduce some of the more popular mechanisms.

Conflict Learning The idea with the conflict learning is to avoid the same mistakes that were made earlier in the search. Thus, when a contradiction is encountered, the set of assignments that caused the contradiction (the *conflict set*) is identified and recorded as a clause such that the same set of assignments is not made later in the search.

Consider the following example by Dixon et al. [15]. Suppose that a partial assignment contains $\{a, \neg b, d, \neg e\}$ and that our problem contains the following two clauses:

$$(\neg a \vee b \vee c \vee \neg d \vee e) \tag{2.1}$$

$$(\neg c \vee \neg d) \tag{2.2}$$

Based on the partial assignment, the first clause allows us to conclude c, while the second clause allow us to conclude $\neg c$. From this contradiction, we form a *nogood*, a new clause to prevent us from assigning the same partial assignment:

$$(\neg a \vee b \vee \neg d \vee e) \tag{2.3}$$

With chronological backtracking, if a search finds a dead-end, it will simply go back up to the previous decision variable to flip its value. However, that previous decision variable may not be directly in conflict with the current variable we just instantiated. What we would rather do is to identify which of the decision variables had the conflict(s) with the current variable. This is the idea behind *backjumping*. By storing some information about the nogoods discovered during the search, we can backjump to the deepest variable that conflicts with the current variable assignments.

Branching Heuristics Branching heuristics come into play when a constructive search algorithm needs to choose a variable (Q in Figure 2.1) to assign a value. The primary criterion for the selection of a branching variable is to pick one that will enable a cascade of unit propagations [15]. Based on this idea, *MOMS rule* branches on the variable that has the *Maximum Occurrences in clauses of Minimum Size* [37, 27, 11]. This heuristic is based on the intuition that shorter clauses are more important than longer ones. The *two-sided Jeroslow-Wang(J-W)* rule works in a similar fashion [66].

The *unit propagation rule* [11] computes the exact number of propagations that would be caused by a branching choice [15]. Compared to a more approximate method such as MOMS, this method is considerably more expensive computationally and thus is not often implemented.

Another strategy that works well is branching on the variables that are likely to be in the *backbone* [17]. A backbone variable is one that has the same value in all optimal solutions to a given problem. Again, from Dixon et al.'s work [15], given a problem C and a partial assignment P, the backbone heuristic attempts to branch on variables that are in the backbone of the subset of those clauses in C that are satisfied by P; the likelihood that any particular variable is backbone is approximated by counting the appearances of the literal in the satisfied clauses in C. This heuristic is known to outperform the previous branching heuristics mentioned here [15].

One branching heuristic that works well with the conflict learning is the *variable state independent decaying sum (VSIDS)* used in *zChaff* [47]. This method keeps a count of the number of times each literal occurs in a formula. This includes those literals from the new clauses added from conflict learning. VSIDS then selects a variable with the highest count. Meanwhile, all the counts are divided by a constant factor periodically such that the information from the recent conflicts are favoured [47, 15].

Watches Moskewicz et al. [47] developed a more efficient way to detect unit clauses that is now standard in DPLL implementations. This method chooses two arbitrary literals from each clause to be *watched*. By having these *watched literals* for each clause, the only time a clause needs to be checked for whether it is a unit or not is when one of the watched literals is falsified. At this point, there is either another literal (distinct from the remaining watched literal) that can be watched, or the only uninstantiated literal left in the clause is the remaining watched literal, at which point, unit propagation can take place. Considering that DPLL algorithms spend the majority of its time in unit propagation, this technique brings tremendous speedup over conventional DPLL algorithms.

2.3 Local Search Methods

The other end of the spectrum for SAT solving techniques is a local search method. Ever since the introduction of GSAT [56], local search algorithms have continuously gained popularity. Despite the inherent incompleteness, their ability to solve hard satisfiable problems coupled with the simplicity of their implementation have drawn many researchers into the field [53].

2.3.1 GSAT

As one of the first local search algorithms, GSAT showed tremendous scalability and speed on various types of hard satisfiable instances. Among these were random 3-SAT, graph-colouring, N-queens encodings, and Boolean induction problems [56]. The basic procedure for GSAT is simple: it starts with a random truth assignment to all the variables. If the assignment does not satisfy the given CNF formula, a variable that would lead to the largest increase in the total number of satisfied clauses is flipped. The number of clauses made satisfied by the change in a variable is denoted by $make$, and the number of clauses made unsatisfied is denoted by $break$. Such procedure is repeated until a satisfying solution is found or it has reached the maximum number of flips. This process repeats itself as necessary for a specified maximum number of tries. See Figure 2.2 for the pseudo-code.

2.3.2 Noise Strategies and Walksat

A big improvement in the performance of local search came with the idea of adding *noise* to the algorithm. Because GSAT always picks the variable that will yield the largest increase in the number of satisfied clause, the algorithm is largely deterministic. While this is advantageous in the early phase of the search, such a strategy can often lead to local minima, where the algorithm can be trapped indefinitely. In order to avoid this, noise was introduced to the procedure: instead of always choosing the variable with the highest $make - break$ count, with some probability p, the algorithm randomly chooses a variable v from the variables that appear in the unsatisfied clauses. With probability $1 - p$, the greedy heuristic is used as before. The algorithm with this modification is known as GWSAT [55].

With a few additional subtle changes to GWSAT, Selman et al. [55] created the powerful Walksat [55] that has been the algorithm of choice for much ensuing research in this area. The pseudo-code for Walksat can be found in Figure 2.3.

There are two subtle differences between GWSAT and Walksat. First, whereas GWSAT picks a variable v from the list of variables (with no duplicates) that appear in unsatisfied clauses, Walksat employs a two-stage process for picking v: it first picks a clause c randomly from all the unsatisfied clauses, and then a variable from c is selected. This favours those variables that appear in *many* unsatisfied clauses for Walksat while in GWSAT's case, all the variables that appear in unsatisfied clauses have equal likelihood for being picked [55].

Secondly, when Walksat greedily picks a variable to flip, it only looks at the $break$ count while GWSAT considers $make - break$ count. To our knowledge, there is no clear evidence as to which counting method works better.

2.3.3 Further Strategies for Improvements

Since the introduction of Walksat, there have been incremental improvements on the performance of local search. Novelty [44] works in the same general framework as GSAT except that it avoids picking the same variable repeatedly by keeping track of the variables that were flipped last. If the variable with the best $make - break$ count is the most recently flipped one, it is flipped with probability $1 - p$ [30]. Otherwise, the second best variable is flipped. Novelty+

```
**Input:** a CNF formula $F$, Max-Flips, Max-Tries
**Output:** a satisfying truth assignment if found
for $i := 1$ to Max-Tries
    $T :=$ a randomly generated truth assignment
    for $j := 1$ to Max-Flips
        if $T$ satisfies $F$ then return $T$, ‘satisfiable’
        $v :=$ variable with the largest $make - break$
        $T := T$ with $v$’s value flipped
    end for
end for
return ‘no satisfying assignment found’
```

Figure 2.2: GSAT pseudo-code [56].

[30] is an extension of Novelty with a random walk. In each search step, with probability wp, the variable to be flipped is randomly picked from the selected clause.

Adaptive noise mechanism for Walksat is a way to vary the noise value p dynamically during the search [32]. Intuitively, during the early stages of the search when the number of unsatisfied clauses is decreasing at a high rate, we want to make Walksat more deterministic by having a smaller p. However, when the search stagnates, we want to increase the noise value to guide the search out of local minima. The improvement in performance from the adaptive noise mechanism is substantial in certain problem instances while it rarely hurts the performance [68].

A clause-weighting scheme dynamically varies the weights of unsatisfied clauses such that the ones that are harder to satisfy are given higher priority to satisfy than others [59]. Examples of dynamic local search algorithms for SAT include Morris’ Breakout Method [46], GSAT with clause weighting [54, 18], Discrete Lagrangian Method (DLM) [65], Smoothed Descent and Flood (SDF) and Exponentiated Sub-Gradient (ESG) [53], and Scaling and Probabilistic Smoothing (SAPS) algorithm [36]. All these algorithms dynamically vary the weights of the clauses to adjust the search space that they focus on.

2.4 Use of Memory in Local Search

So far the discussion of local search methods was in the context of SAT. However, local search methods are broadly used for other combinatorial problems including constraint satisfaction problems, travelling salesman problems, quadratic assignment problems, and job-shop scheduling problems [29]. Naturally, there are many similarities in the design of local search methods for SAT and other combinatorial problems. In this section, we examine a key fea-

```
**Input:** a CNF formula $F$, Max-Flips, Max-Tries, $p$
**Output:** a satisfying truth assignment if found
for $i := 1$ to Max-Tries
    $T :=$ a randomly generated truth assignment
    for $j := 1$ to Max-Flips
        if $T$ satisfies $F$ then return $T$, ‘satisfiable’
        $c :=$ an unsatisfied clause of $F$, selected at random
        if there exists a variable in $c$ with $break = 0$
            $v :=$ such a variable
        else
            with probability $1 - p$
                $v :=$ a variable in $c$ with minimal $break$
            with probability $p$
                $v :=$ randomly selected variable from $c$
        $T := T$ with $v$’s value flipped
    end for
end for
return ‘no satisfying assignment found’
```

Figure 2.3: Walksat pseudo-code [55, 57].

ture, *memory*, in non-SAT local search methods and see how it is applied to SAT local search methods.

Memory has been integral part of local search methods. The use of memory has been of particular importance for *tabu search*, as the main mechanism for guiding the search is defined by the memory. Tabu search is a metaheuristic that can be superimposed on other procedures to prevent them from becoming trapped at locally optimal solutions [21]. This inner procedure is often a greedy heuristic that moves from one solution[2] to another based on some evaluation function.

Similar to most other local search methods, tabu search begins with a solution. This can be from a random assignment or from some initialization. Then tabu search generates a list of candidates to move to from the current solution's *neighbours* (further defined in Section 2.4.1). From the list, tabu search moves to the "best" candidate based on some criteria, forming a new solution. This process of finding a list of candidates and moving to a new solution repeats until some termination criteria (often the number of iterations or aspiration criteria) is met.

Central to tabu search is a *tabu list*, which keeps a list of moves that may not be performed. Thus, when generating a list of candidates from a current solution, some neighbouring solutions cannot be added to the list. The tabu list serves to insure that the moves in the list are not reversed thus preventing previous moves from being repeated. Criteria for moves entering the tabu list can be defined in many ways (discussed below), and similar criteria exist for moves to be off the tabu list. The length of the stay for a given move is called the tabu tenure.

2.4.1 Memory in Tabu Search

Memory in tabu search may be viewed as a way to modify the neighbourhood $N(x)$ of the current solution x. When tabu search looks for the next move, it considers the current solution's neighbourhood as the next candidates. Memory modifies the neighbourhood such that the candidates are from $N^*(x) \subseteq N(x)$ [23]. Depending on how memory is defined and what aspects of memory is used, the resulting subset, $N^*(x)$, can significantly vary. For instance, with a long tabu list and tabu tenure, the search has only limited choice of moves as most of the neighbouring moves are tabu. The opposite is true with a short tabu list and tabu tenure.

Glover and Laguna [23] identify four dimensions to memory: recency, frequency, quality, and influence. Each dimension uniquely defines the criteria by which moves enter and exit the tabu list. In particular, recency and frequency complement each other and form the basis for the short-term and long-term memory as discussed in the following.

2.4.1.1 Short-term Memory

The most commonly used short-term memory is in the form of *recency-based* memory, where the underlying algorithm keeps track of solution attributes that have changed in the recent past. Selected attributes of recently visited solutions are recorded as tabu (and stored in tabu list), and this will prevent the search from revisiting the same solution [23]. Tabu tenure is critical to

[2]In the Artificial Intelligence community, a "solution" is just a *state* (with all variables fully instantiated) in search space, not necessarily an optimal solution.

the performance of tabu search as appropriate tenure length will ensure that the same solution is not revisited while not ruling out too many possible solutions.

2.4.1.2 Long-term Memory

Although in some applications the use of short-term memory suffices, in most applications, significant performance improvement can be found by integrating long-term memory. In long-term memory, instead of relying on an individual solution for generating a modified neighbourhood, an algorithm may generate a pool of *elite solutions* (usually a collection of local minima visited along the search) or some attributes from past visits in the search space [23]. For example, from a pool of elite solutions, one can use the information on how often certain common attributes among solutions occur or how often attributes change as a basis to form the neighbourhood. Another way is to combine the elite solutions using some kind of weight that indicates the quality of the solutions.

Two important components of long-term memory are *intensification* and *diversification*. Intensification encourages moves in the search space that have historically been good. The idea behind intensification is that one should explore more thoroughly the portions of the search space that seem promising in order to make sure that the best solutions in these areas are indeed found.

The purpose of diversification is to prevent the search process from concentrating in one area of search space for too long since an optimal solution may be in another part of the search space. Diversification can be as simple as randomization in the search process. Partial or full restart is another way to achieve some diversity in the search. Also, diversity can be used as a criterion for elite pool replacement strategy: by not allowing a solution that is too close to another elite solution, the elite pool can maintain its diversity.

Path Relinking One way to exploit an elite pool is to use the individual elite solutions to generate another solution. The idea behind *path relinking* is to generate combinations of a set of elite solutions, thereby creating paths between these solutions. Along the path, there will be a new set of solution(s) in a neighbourhood space that will share significant subset of attributes contained in the parent solutions [22]. Further, path relinking encourages a different neighbourhood structure to be used than in the standard search phase. For example, moves that are allowed in path relinking may be excluded normally due to infeasibility or the hill-descending nature of the move.

To generate the desired path, two solutions from the elite set can be chosen (by some heuristic or randomly). Then, starting from one of the two solutions, a series of moves can be made toward the other solution. This is shown in Figure 2.4, where x' is path-relinked with x''. The motivation behind this is to isolate assignments that frequently and influentially occur in high quality solutions, and then introduce compatible subsets of these assignments into other solutions that are generated. If any of the solutions visited during the path relinking stage is better than either of the two parent solutions, a parent solution can be replaced, and the process can be repeated. Often the details of the implementation are problem domain specific.

One of the successful applications of this method is Nowicki and Smutnicki's i-TSAB algorithm [49]. i-TSAB, based on tabu search, uses path relinking as a means of diversification,

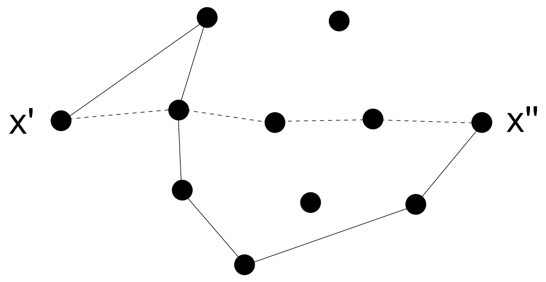


Figure 2.4: Two different paths going from solution x' to x''. Original path shown by heavy line; relinked path shown by dotted line [22].

and as it stands, i-TSAB is the current state-of-the-art local search algorithm on the makespan-minimization form of the classic job-shop scheduling problem [62].

2.4.2 Use of Memory in SAT Solvers

In this section, we look at how the memory mechanisms discussed above are applied to SAT solvers. In the SAT domain, use of memory is rather limited and simple. Most popular implementations come in the form of tabu list, where an algorithm keeps recently flipped variables to prevent them from being reversed too quickly. Also, there is some usage of long-term memory via clause-weighting.

2.4.2.1 Tabu List

The use of memory in SAT was introduced in HSAT [20]. Built based on the architecture of GSAT, it remembers when variables are flipped the last. Thus, when HSAT is offered a choice of variables, it always picks the one that was flipped the longest ago. The addition of the short-term memory significantly reduces the number of flips required to find a satisfying solution compared to GSAT. Similar ideas have been implemented on Walksat by McAllester et al. [44], where an explicit tabu list is kept of variables that have been flipped in the last t flips. The algorithm flips the variable from an unsatisfied clause only if it is not on the tabu list. One of the most successful local search algorithm, Novelty, also keeps track of recently flipped variables in order to avoid repeatedly flipping the most recently flipped variable [44].

2.4.2.2 Clause-Weighting

Another way of using memory in local search methods is more implicit. In clause weighting schemes, clauses that are unsatisfied more frequently than others are assigned higher weights such that they get a higher priority in being satisfied than other clauses, thus, in effect, warping the search space. There are many variations in how the weights are assigned and how often they are updated [54, 18]. Frank's work [18] indicated that the weighting scheme acts as a short-term memory that is only relevant at the existing search space but not necessarily at the new space.

Scaling and Probabilistic Smoothing (SAPS) algorithm by Tompkins [59] claimed the use of short- and long-term memory via dynamic clause-weighting. It noted that the superior performance of SAPS does not necessarily come from "warping" the landscape (via clause-weighting) as commonly believed, but rather, due to the diversification effect from the scaling (i.e. clause-weighting) and smoothing. The scaling helps the algorithm to escape local optima (short-term memory) while the smoothing reduces the clause-weighting effects once the search has left the trap (long-term memory). While Frank viewed the decay of previous weightings as lack of long-term memory, Tompkins considered it as active averaging of the weights, hence long-term memory.

Another algorithm that uses clause-weighting is Pure Additive Weighting Scheme (PAWS), augmented by *Usual Suspect* heuristic [38]. Instead of the usual contextual perturbation of clause weights (short-term memory), the algorithm keeps the weights for an extended period of time to provide long-term effects that channel its effort on clauses that are hardest to satisfy. PAWS+US provided notable (though not substantial) improvement in performance over PAWS in about a third of all instances tested.

2.4.2.3 Backbone-Guided Search

One of the more recent adaptations of long-term memory for SAT solvers is through the use of elite solutions. Zhang's *Backbone-Guided (BG)* Walksat algorithm [68] uses long-term memory to gather some information about the problem instance at hand and make "educated" decisions when flipping variables. The BG algorithm is based on the idea of *backbone* variables (those variables that have the same truth assignment in all optimal solutions). Thus, if we know which variables are in the backbone, we can focus on setting those variables to correct values, and the resulting set of clauses should be easier to solve. However, since it is impossible to know the backbone variables without knowing all the optimal solutions in advance, Zhang estimates the backbone variables by running the base algorithm repeatedly for a short (relative to the entire run) period of time until a pool of *elite solutions* is collected. From this, he can estimate which variables are in the backbone based on how often one literal appears compared to its negation. Based on these estimates, the BG algorithm makes biased decisions when choosing unsatisfied clauses and flipping variables in order to try to set the backbone variables to correct literal values. A more detailed description of the BG local search is provided in Section 6.3.1.

When compared to its base algorithm Walksat [55], Zhang's algorithm is particularly strong on over-constrained MAX-SAT problems and some structured problems. However, on critically- and under-constrained problems (see Section 2.5), the performance actually deteriorated with the use of backbone guidance.

One of the key differences between a backbone-guided search and path relinking is that, in a backbone-guided search, all the elite solutions are combined such that the search makes decisions based on the aggregated information from the elite solutions. However, path relinking influences the decision-making using the features of individual elite solutions. We will further address this difference in Chapter 7.

2.5 Problem Difficulty for Local Search

Along with the research to advance the speed of SAT solvers and their scalability, another related research area is in identifying difficult satisfiability problems and their key characteristics. Because there is so much variance in problem difficulty even for problems of the same size in a specific problem domain, being able to identify and predict whether an instance will be difficult is of practical interest as well as theoretical [57]. There are number of factors that have been shown to affect the local search cost to different degrees, including the number of optimal solutions, backbone size, number of local optima, and the extensiveness of local minima.

Clauses-to-Variables Ratio Clauses-to-variables c/v ratio is known as one of the most important features in predicting problem difficulty in satisfiability problems. In under-constrained regions with relatively low c/v ratio, almost all instances are satisfiable, making it easy to find an optimal solution. In over-constrained regions with high c/v ratio, almost all instances are unsatisfiable. Here, for constructive search algorithms, proving instances to be unsatisfiable is easy since backtracking search can cut off potential solution paths early in the search [7]. The critically-constrained region, which has c/v ratio of 4.3, is known to have the hardest problems. This region coincides with the *phase transition* region, where about half the instances are satisfiable and the rest are unsatisfiable [45, 10]. The phase transition region for 3-SAT, 100-variable instances can be seen in Figure 2.5. Though it is not the focus of our work, there has been substantial work in understanding the phase transition [64, 69, 43].

While the decrease in the search cost with increasing c/v beyond the critically-constrained region makes sense for systematic solvers, such observed phenomenon for local search algorithms is less intuitive. More specifically, for satisfiable instances, local search cost falls past the phase transition region despite the decreasing number of satisfying solutions. Thus, there must be a competing factor that offsets the effects from the number of satisfying solutions in order to decrease the local search cost beyond the phase transition region. The following are some of the possible factors.

2.5.1 Number of Optimal Solutions and Backbone Size

With all things equal, the more optimal solutions an instance has, the easier it should be to solve. For local search algorithms, the number of satisfying solutions is especially important due to their stochastic nature. Past studies have confirmed this: Clark et al. [8] reported a relatively strong negative $log - log$ correlation between the local search cost and the number of satisfying solutions. Thus, the search cost rises going from the under-constrained region to the critically-constrained region. However, as the number of satisfying solutions continues to fall with increasing c/v ratio past the critical region, the search cost actually decreases for satisfiable instances.

The same peak in local search cost was observed by Parkes [51]. Parkes claimed that the search cost peak at the critically-constrained region is from the rapid emergence of large-backbone instances at the critical c/v ratio. Further, Parkes noted that for any given size of backbone, the cost is actually higher for instances from the under-constrained region.

Singer et al. [57] further refined the relationship between the local search cost and the number of optimal solutions by showing that a strong correlation between search cost and

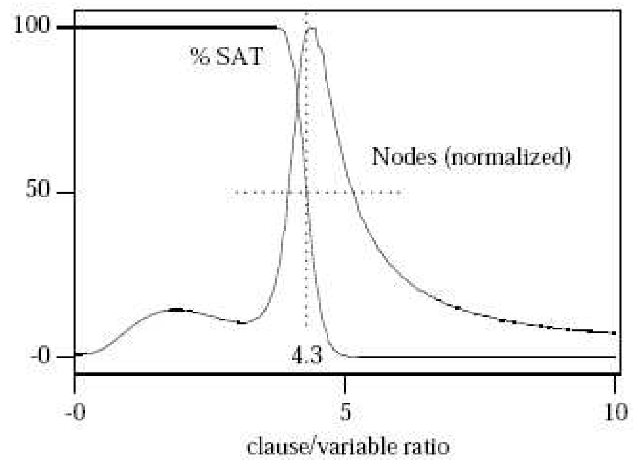


Figure 2.5: Percentage of satisfiable instances and size of the search tree for 3-SAT, 100 variables from Crawford et al. [10].

the number of optimal solutions exists only for those instances with small backbones. For these problems, they claimed that finding the backbone was easy and the main difficulty was encountering an optimal solution once the backbone has been established. The correlation between the two factors was not as strong in larger backbone sizes because the main difficulty of finding an optimal solution was in identifying the cluster of optimal solutions (backbones).

2.5.2 Extensiveness of Local Optima

A part of the explanation for decreasing search cost past the critical c/v ratio can be addressed by the distance between local optima and their nearest global optimum. Singer et al. [57] observed that the Hamming distance between a quasi-solution[3] and its nearest optimal solution decreases with c/v for fixed backbone sizes. This result indicates that at higher c/v ratio, although it may take a little longer to find a quasi-solution, once found, it is fairly close to a global optimal. However, in critically-constrained region, the study claims that there are vast number of high-quality solutions spread throughout the search space that they may mislead the search.

Singer et al. further reported that the correlation between the search cost and Hamming distance between a quasi-solution and its nearest optimal is especially high for small-backbone instances. The Hamming distance between a quasi-solution and its nearest optimal is a good indicator of how *extensive* or spread-out the quasi-solution area is. The more extensive this area (larger Hamming distance), the harder the instance gets since there are more attractive solutions that do not move the search closer to a globally optimal solution. Hence the strong positive correlation makes sense. For further discussion, readers may refer to Section 5.3.

[3]A quasi-solution was defined as a solution with 5 unsatisfied clauses for 100 variable instances

Backbone Fragility Singer et al. [57] introduced yet another reason for the decreasing search cost beyond the critically-constrained region suggesting *backbone fragility* as the possible explanation. They defined *backbone-fragile* instances as those that have a large reduction in the backbones size on average upon random removal of small set of clauses. *Backbone robustness* was defined as the opposite of backbone fragility.

According to the study, if an instance is backbone-fragile, we can expect that the quasi-solutions, where only the removed clauses are unsatisfied, will be attractive to local search algorithm and possibly Hamming-distant from the nearest global optimal. Thus, backbone fragility, which is an intrinsic property of an instance, approximately corresponds to how extensive the quasi-solution area is. The hypothesis was supported by strong negative correlation between the search cost and backbone robustness in large backbone problems. However, the correlation was not as strong for small-backbone instances; they claimed that the low correlations was from the fact that these instances were generally easier to solve and backbone did not affect them as much. Thus, the typical local search cost peaks in the critically-constrained region because of the appearance of many large-backbone instances which are moderately backbone-fragile. Further addition of clauses will make the instance more backbone-robust as is the case in the over-constrained region.

2.5.3 Number of Local Minima

Yokoo [67] claimed that the decreasing local search cost past the phase transition is from the decreasing number of local minima with the increasing c/v ratio. Abundance of local minima distracts the local search algorithms from getting closer to the optimal solution. The study showed that the number of local minima indeed falls with respect to the increasing c/v ratio, thus making the search easier past the phase transition region. He claimed that this factor along with the number of optimal solutions cause the search cost peak at the phase-transition region. This is further discussed in Section 5.2.

2.6 Applications of Satisfiability

In addition to the prominent role SAT plays in the theoretical research for complexity theory, there are many areas of its applications. In particular, AI planning problems, scheduling, constraint satisfaction problems, cryptographic key search, and inductive inference can be easily encoded into SAT and be solved with regular SAT solver [34]. More recently, SAT has been applied to hardware design and verification problems with success. Here, three areas of applications are discussed:

Planning as Satisfiability The term planning in AI is used to describe the construction of a sequence of actions that will achieve a goal [52]. A planning domain consists of a set of operators or action types. Each operator may be executed only in some particular set of states (its preconditions), and has some particular set of effects on its state (its effects). A planning problem consists of a planning domain together with an initial state and a desired goal state (or set of goal states). The planning problem is solved by producing a sequence of actions (operator instances) that takes the initial state to a goal state.

In 1992, Kautz and Selman proposed a way to solve planning problem as satisfiability [41]. Compared to the previous method of deduction, it allows more flexible and accurate modeling of planning problems. In terms of performance, directly SAT-encoded problems that are solved using Walksat outperformed Graphplan, which at the time was the state-of-the-art algorithm for planning problems [39].

Scheduling Problems as Satisfiability Scheduling problems are ubiquitous; in manufacturing environment, a series of jobs must be scheduled such that a set of machines or tools are used efficiently. In hospitals, one may have to assemble a number of surgical teams using a variety of specialists subject to a set of constraints on consecutive numbers of hours worked, availability of operating rooms, etc [9].

Crawford and Baker [9] examined the application of satisfiability for job-shop scheduling problems (JSP). In a $n \times m$ job-shop scheduling problem, n jobs must be processed exactly once on each of m machines. Each job i $(1 \leq i \leq n)$ is routed through each of the m machines in some pre-defined order. They found that the problems get very large quickly when they are encoded into SAT; the size of the problems made it hard for SAT solvers be competitive with the existing methods such as the slack-based heuristics used by Smith and Cheng [58]. They further noted that a DPLL-based algorithm outperformed a local search method, primarily due to the abundance of unit propagations available in JSP.

Formal Hardware Verification with Satisfiability An active area of research for SAT application is in formal hardware verification. Formal verification means *proving* that a property holds for a model of a design [35]. It is involved with the development of methods to analyze and determine whether a given implementation of a system conforms with its specification. With the exponential growth in system complexity, verification has become a true bottleneck in the development of software and hardware systems. In fact, over 50% of the resources invested in developing systems are reportedly spent on verification [1].

Along with *automatic test pattern generation*, SAT is is one of the most efficient ways to address formal hardware verification [5]. Since the introduction of *bounded model checking* (BMC) by Biere et al. [4], using SAT as a hardware verification tool has proven to be very efficient. By taking advantage of the depth-first nature of SAT search procedures, the BMC technique is both faster and more compact than existing Binary Decision Diagram based approaches [4].

2.7 Conclusions

The work discussed above serves as a motivation and a starting point for our work from here on. We believe that there are still many unanswered questions in the areas of problem difficulty for local search algorithms and also, in the concept and applications of backbone. Also, in general, we found that there is not much literature that compares the performance of constructive search and local search algorithms on a wide range of problems. All these issues will be discussed in the upcoming chapters. Further, based on the literature from non-SAT, combinatorial optimization domains, we implement a metaheuristic to a local search algorithm for SAT that uses long-term memory.

Chapter 3

Water Network Security Problem as Satisfiability

Along with the aforementioned areas of application of Satisfiability (SAT), problems with discrete valued variables may be solved as SAT. Binary integer programming problems are particularly well-suited for SAT due to the natural 1-to-1 mapping of the variables into the SAT literals. In this chapter, we give a concrete example of such case by solving a real application problem using SAT and comparing the results with other existing methods to demonstrate the potential usage of SAT in this class of problems and also serve as a motivation for further research. To our knowledge, this kind of comparison between SAT and integer programming (IP) has never been done although similar work exist for comparisons between SAT and CSP [31, 40, 2].

3.1 Problem Description

Water Network Security problem is a real-world problem faced by the researchers at Sandia National Laboratories.[1] Here, we try to minimize the public's health risk in the event that the integrity of a water distribution network is compromised by an outside source (*i.e.* terrorist attack, ground water leakage, contamination, etc.). There are two sub-problems that arise from the application: one is the sensor placement optimization, and the other is *source inversion*. Source inversion, which is this chapter's focus, looks at identifying the location of attacks given that there are contaminant concentration readings from one or more sensors. By identifying the location of the attacks, the water flow can be re-routed or contained to the immediate area such that the damage is minimized and the clean-up team can be placed in correct locations.

Integer Programming Formulation In order to be able to locate the point of attack, an attack from any of the possible finite attack points need to be *distinguishable* from another attack point. More rigorously, attack i causes certain nodes to observe nonzero concentrations of the contaminant at certain time. Given a sensor placement s, two attacks i and j are considered

[1]In New Mexico, USA.

distinguishable ($d_{ij} = 1$) if and only if there exists a vertex v such that v has a sensor ($s_v = 1$) and exactly one of i,j causes a nonzero concentration at vertex v.

We can illustrate the concept of distinguishability with a simple example seen in Figure 3.1. Here, in order to distinguish attack A from B, a sensor must be either at node 1 or 2, thus $D_{AB} = \{1, 2\}$. Similarly, we have $D_{BC} = \{2, 3, 4\}$ and $D_{AC} = \{1, 3, 4\}$.

Set D_{ij} of nodes v that can distinguish a pair i and j are pre-computed. In other words, if any one of the nodes from a set D_{ij} has a sensor, (i, j) attack pair is covered.

The problem with n attacks is most naturally modeled as a binary integer programming problem as follows:

Maximize distinguishability:

$$\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} d_{ij}$$

subject to:

$$\sum_{v} s_v \leq maxNumSensors \tag{3.1}$$

$$\sum_{v \in D_{ij}} s_v \geq d_{ij} \tag{3.2}$$

where,
$d_{ij} = 1$ if two attacks i,j are distinguishable, otherwise $d_{ij} = 0$,
D_{ij} represent the set of nodes that can distinguish attack i and j,
$s_v = 1$ if node v has a sensor, otherwise $s_v = 0$.

Constraint 3.1 stipulates that the number of sensors used must not exceed the sensor budget. Constraint 3.2 says that if any one of the nodes v for the given attack pair from the set D_{ij} has a sensor ($s_v = 1$), the attack pair is covered ($d_{ij} = 1$).

There are two typical approaches to this type of integer programming (IP) problem: One is to use standard IP solvers such as ILOG CPLEX and the other is to use some heuristic method.

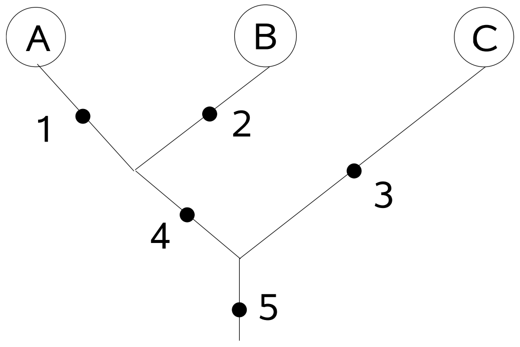


Figure 3.1: An example of a simple water network with 3 attacks and 5 nodes.

While CPLEX can solve IP problems to optimality, its scalability may be an issue. As with many other combinatorial optimization problems, the search space grows exponentially with the problem size, and quickly, it can become infeasible to solve using this method.

The other approach is to treat the problem as a set-cover problem and use a greedy heuristic. By re-arranging D_{ij}, we can form a set E_v of a pair of attack (i, j) that can determine which attack pairs can be distinguished by each node v. The heuristic selects one node at a time for sensor placements starting from the node that will cover the most number of attack pairs. Following a sensor placement, all the attack pairs covered by the node are removed from all sets, and subsequent sensor placement is made. This is repeated until the maximum number of sensors have been used. This method can solve larger problems than the integer programming method, but its drawback is that it does not guarantee the optimality.

3.2 Water Network Security Problem as Satisfiability

In this section, we propose a method to encode the binary integer programming problem as a SAT problem. One of the advantages of encoding the problem as satisfiability is that we can exploit powerful SAT solvers. There have been other practical problems such as planning and hardware verification problems that have been encoded as SAT in order to take advantage of the speed of SAT solvers. What makes this problem particularly appealing to re-formulate as SAT is the fact that most of the variables and constraints are binary. Due to the “natural” translation to SAT and the availability of highly optimized SAT solvers, this approach has the potential to be faster and more scalable than the IP method while being able to prove optimality.

Solving the Water Network problem as SAT is a two-step process: First, the CNF (Conjunctive Normal Form) formula must be generated using appropriate encoding method, and secondly, the resulting CNF formula must be solved using a SAT solver.

3.2.1 SAT Encoding of IP

The IP formulation discussed in Section 3.1 is used as the basis of CNF generation. Constraint 3.2 from the IP formulation says that each attack pair i, j will be covered if any one of the nodes in D_{ij} has a sensor. Thus, each attack pair D_{ij} can be represented as a clause:

$$(s_p \vee s_q \vee s_r \vee ...)$$

where $s_p, s_q, s_r, ... \in D_{ij}$

Encoding Constraint 3.1 of the IP formulation as CNF is non-trivial due to the comparison of the binary variables to a non-negative integer (constant). To get around this, a dummy variable $t_{k,h}$ is introduced and has the following definition:

Let $t_{k,h} = 1$ if and only if $\sum_{v=1}^{k} s_v = h$ where k represents the current node.

For each node k, there will be a set of clauses to enforce that the total number of sensors used so far will be less than or equal to the maximum allowed number of sensors. Then using this variable, every $k = [1, .., mn]$, where mn is the maximum number of nodes, can be expressed explicitly using recursion as follows:

For node k,

$$[(\neg t_{k,h}) \oplus (t_{k-1,h-1} \wedge s_k) \oplus (t_{k-1,h} \wedge \neg s_k)]$$

$$\wedge[(\neg t_{k,h-1}) \oplus (t_{k-1,h-2} \wedge s_k) \oplus (t_{k-1,h-1} \wedge \neg s_k)]$$

...

$$\wedge[(\neg t_{k,0}) \oplus (t_{k-1,0} \wedge \neg s_k)]$$

where $\oplus$ represents XOR operator.

The first line represents the case when there are h sensors used when counting from 1 to k nodes ($t_{k,h} = 1$). In such a case, either the second or the third round bracket *must* be true in order to satisfy the first line. The second round bracket is true if the current node k is used ($s_k = 1$) and thus, $h - 1$ sensors are used when counting up to $k - 1$; the third bracket is true if the current node k is not used. All the subsequent lines will be automatically satisfied by the negation of the dummy variable in the first round bracket of each line. All of these lines must be converted to the conjunctive normal from by expanding out the XOR's and manipulating the resultants using De Morgan's law. In addition to these clauses for each sensor, a boundary condition must be set such that the maximum number of nodes are used. For mn total number of nodes:

$$(t_{n,maxNumSensors})$$

Modeling the constraints as above makes the entire formula satisfiable if and only if *all* the attack pairs are covered. Thus, if not all the attack pairs can be covered with the given number of sensors, SAT solver will simply return "unsatisfiable". However, we need to know what the maximum number of attack pairs that can be covered given the sensor budget. To overcome this problem, we can adjust the formulation such that it can be solved as a *Weighted Maximum-SAT* problem, which is a superset of SAT problems. This is discussed in the following section.

The implementation of the SAT encoding is done using Python. The encoding translator takes in the input file and converts each constraint using the method above, and then outputs file with CNF formulae.

3.2.2 Solving as Satisfiability

Once the CNF formula is generated, it can be solved using a SAT solver. A popular way of solving the SAT problem is to use a constructive SAT solver such as *zChaff* [47]. Local search algorithms such as *Walksat* [55] have been tried as well, but based on preliminary results, constructive search algorithms are far superior on this problem due to its rich structure. Further, zChaff has the added benefit of being able to *prove* satisfiability. As mentioned earlier, one of the major drawbacks of solving this as SAT is that the solver will only show whether the formula is satisfiable or not. For satisfiable cases, we know that all the attack pairs can be distinguished by placing the sensors as directed by the SAT solution. However, if the formula is unsatisfiable, the SAT solver will simply state this and not provide us with any other information.

To address this problem, we can reformulate the CNF formula as Weighted Maximum-Satisfiability (MAX-SAT) problem. In MAX-SAT problems, given a formula F, one tries to maximize the number of satisfied clauses. Further, *Weighted* MAX-SAT problems have weights associated with each clause in order to signify the "importance" of each clause. Here, one tries to maximize the weighted sum of the satisfied clauses (or minimize the weighted sum of the unsatisfied clauses). MAX-SAT is a special case of weighted MAX-SAT with all the weights set equal to 1, and similarly, SAT is a special case of MAX-SAT with 0 unsatisfied clauses allowed.

The set of clauses that *must* be satisfied is the one that describes the maximum number of sensors (Constraint 3.1 in the original IP). Thus, we assign a large, arbitrary positive weight (say 1 million) for all the clauses that help enforce the maximum number of sensors. The other set of clauses that determines which attack pairs are distinguishable can be thought of as "soft" clauses. We would like to maximally satisfy these clauses, but the results will be still valid even if some are violated. They are assigned a small, arbitrary weight (say 1). Again, these weights will enforce the MAX-SAT solver to fully satisfy the "hard" constraints while satisfying as many "soft" constraints as possible.

3.3 Results and Discussion

Results from three different approaches to the Water Network problem are compared: solving the integer programming problem using ILOG CPLEX 10 (translated using AMPL), using the greedy heuristic (implemented in C++), and solving the IP as SAT problem. Ideally, for the last method, the problem can be solved as MAX-SAT using *maxsat* [6] or a similar MAX-SAT solver, but at this point, the technique is not competitive with the other two methods. Thus, a SAT solver zChaff was used to determine the satisfiability of a given SAT formula.

Table 3.1 shows the results for four different problem sizes. The IP formulation was run on a machine with 3.6GHz Intel Xeon and 8GB of RAM, and the greedy heuristic and SAT was run on a machine with 2.8GHz Intel Pentium 4 and 512MB of RAM. The results from the first machine was scaled to the first machine using a speed-up factor of 1.427.[2] The run-time for SAT is the time to generate the CNF formulas plus the time to solve them.

The results show that the heuristic-based model is very competitive with the IP model in terms of getting the optimal answers. For the cases with 250 and 500 attacks, the two methods, in fact, found identical answers, and for the case with 100 attacks, the two answers were nearly identical. However, there is quite a significant difference in the coverage for the largest case of 1000 attacks, where CPLEX clearly outperformed the greedy heuristic. The sudden deterioration in the greedy heuristic's quality in the 1000 attack case was unexpected.

[2]This was based on empirical results run on both machines for a few simple problems. For memory-intensive problems such as the problems at hand, we believe that the speed-up will be far greater.

Data	Compact100			Inversion250			Inversion500			Inversion1000		
No. Pairs	4950			31125			124750			499500		
No. Attacks	100			250			500			1000		
No. Nodes	3358			3358			3358			3358		
Method	IP	Greedy	SAT	IP	Greedy	SAT	IP	Greedy	SAT	IP	Greedy	SAT
Covered pairs	4382*	4380	unsat	19566*	19566*	unsat	60329*	60329*	unsat	476997*	337374	unsat
Covered pairs (%)	88.5	88.5	n/a	62.9	62.9	n/a	48.4	48.4	n/a	95.5	67.5	n/a
Run time(s)	24	0.2	4.9	1731	4.2	102	1388	20.3	336	2283	433	6828

Table 3.1: Performance comparison between the IP, greedy heuristic, and SAT methods when $maxNumSensors = 20$. Optimal solutions are denoted by *.

Whether this is due to the size of the problem or the characteristic of the particular instance is unknown.

As for the SAT method, since it cannot provide the coverage detail, one way of providing useful information is through determining whether the given number of sensors can cover all the nodes. zChaff was very good at determining the satisfiability for all the problem sizes. In Table 3.1, almost all of the run-time for SAT is due to the generation of the CNF formulas as zChaff proved satisfiability almost instantly. However, we have seen greater than 1 order of magnitude of speed-up when going from Python to C++ for the greedy heuristic, and we can expect similar kind of speed-up for SAT generation.

There is a significant difference in the run-time for the IP and greedy heuristic. For the first three problems, the greedy heuristic is faster than the IP method by around 2 orders of magnitude. The gap in the run-time decreased for the 1000 attacks case, where the greedy heuristic was only 5 times faster than the CPLEX. Further, to our surprise, the dramatic increase in run-time with the problem size in the greedy heuristic is not seen for the IP method. In fact, the run-time actually decreased going from 250 to 500 attacks.

At this point, the role that SAT for this application is to complement the solution of either the greedy heuristic. SAT can "quickly" verify whether the number of sensors used to cover all the attacks is optimal. Another way of using SAT is to find the optimal number of sensors required by repeatedly running the problem with different $maxNumSensors$. To this end, we solved the `Inversion250` instance using 188 sensors, which is the minimum number of sensors required to fully be able to distinguish all the attack combinations. It took 4204 seconds to generate the CNF formula and 4998 seconds to solve it using zChaff. On the other hand, CPLEX's result was a little confounding in that the number of attack combinations distinguished was *one less* (31124) than the total number of attack combinations of $\binom{250}{2} = 31125$ (the run-time was 9.3). At this point, we found no explanation for the disparity in the answer.

There are a couple of interesting observations with SAT: first the CNF generation time increased by about 40 times with the increase in $maxNumSensors$. This was expected due to the recursive style used for the encoding of the Constraint 3.1. Secondly, it actually took slightly longer to solve the problem than to generate the CNF formula with the increased $maxNumSensors$. This is in stark contrast with the previous results in Table 3.1, where the CNF generation time dominated the total generation+solving time. Intuitively, it does make sense, however, for the problem to be tougher in the region where it is close to satisfiability since the solver must explore deeper into the search space before concluding satisfiability. Thus, we expect the run-time with 188 sensors to be close to the upper bound for this instance. By the same argument, we can expect decreasing run-time as $maxNumSensors$ increases beyond 188. These issues of problem difficulty in SAT are discussed in detail in Chapter 5.

There is room for improvement for SAT especially in the way the problem is encoded into CNF. With the SAT encoding employed here, the number of clauses explodes with respect to the number of attacks and sensors. Essentially, every node (except for those near the boundary) requires mn lines of XOR's, where mn is the maximum number of sensors. Since every line containing XOR's must be further translated into CNF formula, the number of clauses quickly multiplies. There have been previous work done on generating compact CNF formulas using *Renaming* and *Skolemization* [48] that may significantly improve the time it takes to encode the CNF and further allow MAX-SAT to be a feasible approach to the problem.

3.4 Conclusions and Future Work

A novel way of approaching the Water Network Security problem was discussed: The binary IP formulation for the problem can be translated into a CNF formula, which can be solved using a state-of-the-art SAT solver. The results also showed that SAT may be used in conjunction with the greedy heuristic as a way to verify the optimal number of sensors required to cover all the attack pairs.

Comparison between the greedy heuristic and the IP method show that the greedy heuristic makes relatively few sacrifices in terms of approaching optimality to achieve significant speed-up and scalability compared to the IP method. At the same time, as problem size grew, the search cost for the greedy heuristic increased significantly while that for the IP method stayed near constant, which was surprising.

As shown in the results, a major portion of the run-time for solving this problem came from the generation of the CNF formula. There are more compact SAT-encoding methods that could facilitate faster generation of CNF formulas, as well as better SAT-solving performance, and ultimately allow SAT to be more useful in practice for these types of problems.

Chapter 4

Comparison of Constructive and Local Search Methods: Problem Domains and Search Cost Factors

Generally speaking, there are two main types of satisfiability (SAT) solvers: one is constructive search, also known as complete search, where every possible combination of variable assignment is explicitly or implicitly tried to determine whether the given SAT formula is satisfiable or not. The other type is stochastic local search. Typically, with a local search algorithm, one starts from a randomly generated truth assignment for all the variables. The algorithm then changes or "flips" the assignment of the variable that leads to the largest decrease of its cost function, where the cost function is usually defined as the number of unsatisfied clauses [56].

Due to the differences in the search techniques, the resulting behaviours as well as the strengths and weaknesses are quite different for constructive and local search methods. For example, constructive search methods can prove that a formula is unsatisfiable by exhausting all the combinations in the search space while local search methods cannot. In terms of performance, constructive search has been known to be more effective on highly structured problems with many dependent variables such as hardware and software verifications [63]. On the other hand, local search methods can often find satisfying assignments for extremely large CNF formulas that are far beyond the capability of current constructive search methods [53]. They tend to be highly effective on hard random k-SAT problems, logistics planning formulas, graph colouring, and circuit synthesis problems [63].

Recently, there has been some work on the underlying reason behind a local search method's performance on various problem domains. In particular, past studies have suggested the number of optimal solutions for a given formula and backbone size to be of significance in determining local search cost [8, 57]. Such factors, however, have not been studied to the same extent for constructive search algorithms.

The purpose of this chapter is two-fold: first, it attempts to identify the problem domains of each method's strengths and weaknesses experimentally. Secondly, it applies some of the factors that affect local search cost to a constructive search method, namely number of optimal solutions and backbone size, to see if the same factors influence constructive search cost as well.

4.1 Background Information

One SAT solver each from constructive and local search methods are used for this chapter: for the constructive search, *zChaff 2004.5.13* [47] developed by SAT Research Group of Princeton is used, and for the local search, *Walksat* [55] is used. The following is a brief description of the two algorithms.

4.1.1 zChaff Algorithm

As with most other complete SAT solvers, zChaff is based on the Davis-Putnam, Logemann, and Loveland (DPLL) [13, 12] backtrack search algorithm (refer to Chapter 2 for more details on DPLL). One of the two major improvements of zChaff from the basic DPLL algorithm comes from a more powerful *Boolean Constraint Propagator (BCP)* for unit propagations. Since major portion of constructive SAT solvers' run-time is spent in the BCP process, even a small increase in its efficiency has a noticeable impact on the end results. One way to reduce the number of clauses the BCP must check is to visit only those clauses that are possibly *unit* (have only one literal) for unit propagation. That is, there is no need to visit those clauses that have at least two literals present *after* the variable instantiation since the BCP will not be able to perform unit propagation. To achieve this, zChaff "watches" (any) two uninstantiated literals in every clause (unless it has less than two literals already). Clauses need to be visited only when one of the watched literals is deleted from the clause by a variable instantiation. When one of the literals is deleted, there are two possibilities:

1. There is another literal (other than the other watched literal), which can be assigned to be watched. There is no further unit propagation here.

2. The only uninstantiated literal left in the clause is the other watched literal, at which point, unit propagation can take place.

A visit to the clause does not guarantee implication (*i.e.* unit propagation), but the clause just needs to update the watched literals. Watched literals save many unnecessary visits to clauses without implications. Further, it is very cheap even during backtracking since there is no need to modify the watched literals [47].

The other improvement in zChaff over conventional DP solver is from a clever decision heuristic for choosing a variable to branch on. This heuristic, called Variable State Independent Decaying Sum (VSIDS), chooses the literal that appears most frequently. This is done by simply maintaining a counter for each literal of the frequency of its appearance. However, this counter is periodically divided by a constant, which makes the literals in clauses that are recently added to the set of *conflict clauses* weigh more than older literals. Conflict clauses are formed from the set of assignments that has no possible satisfying solution. Since difficult instances generate many conflict clauses, VSIDS heuristic ensures zChaff will concentrate on getting the "tough" literals correct.

4.1.2 Walksat Algorithm

Walksat, introduced by Selman and Kautz in 1994 [55], is still one of the best performing local search algorithms for SAT and serves as the core engine of state-of-the-art SAT solvers [44, 30].

The detailed description of Walksat is given in Section 2.3. The Walksat implementation by Tompkins called *UBCSAT* [59] is used.

4.2 Search Cost of Constructive and Local Search on Various Problem Domains

Both zChaff and Walksat are run on a diverse set of instances to measure their effectiveness in various problem domains. All instances for the testing are drawn either from Zhang's work [68] or the Satisfiability Library [33]. The test set has a good mix of practical problems as well as randomly generated ones including hardware verification, parity problems, quasigroup, graph colouring, random k-SAT, and planning.

4.2.1 Methodology

For Walksat, because the algorithm is highly stochastic, every instance was run 20 times independently with the maximum number of flips set to 10 million per run. zChaff, on the other hand, was run once per instance since the algorithm is largely deterministic. Run-time (the mean run-time for Walksat) is used as the metric that measures the search effort for both algorithms as any other measures such as the number of flips or decisions are algorithm specific. For zChaff, the maximum time allowed for search is set as the average time that it takes for Walksat to either find a satisfying solution or finish all 10 million flips (unless otherwise stated in the results).

4.2.2 Results

The results are separated into three tables: Table 4.1 and 4.2 feature mostly structured instances from Zhang's paper [68]. Table 4.1, which consists of instances from Table 3 in Zhang's paper, is considered "easier" of the two tables in the paper. Table 4.2 (Table 4 in Zhang's paper), the harder of the two, consists of instances that Walksat could not solve within the cutoff value of 10 million flips limit in any of the 20 runs. Finally, Table 4.3 consists of random k-SAT instances from the Satisfiability Library.

Table 4.1, 4.2, and 4.3 show quite a disparity in performance in the two search methods. Planning, parity learning problems, quasigroups, and hardware verification were dominated by zChaff while Walksat was superior in random k-SAT instances and graph colouring. For example, on hardware verification problems, zChaff was able to completely solve some instances in less than 10 seconds while Walksat left hundreds of clauses unsatisfied in longer time frame. On k-SAT instances, however, it took less than 10 seconds for Walksat to solve most of the problems while zChaff could not solve any of them in less than 100 seconds. While the domains of each algorithm's strength was expected from the previous studies, the degree of disparity in performance was surprising. Possible reasons for such results is discussed below.

					Walksat			zChaff
Domain	Instance	# vars	# clauses	clause/var	# solns fnd	# unsat clauses	Time	Time
Planning	bw_large.c	3016	50457	16.7	3	0.85	31.6	0.33
	bw_large.d	6325	131973	20.9	0	4.45	56.34	2.62
Parity learning problems	par8-1	350	1149	3.3	10	0.5	6.73	0
	par8-2	350	1157	3.3	7	0.95	7.07	0
	par8-3	350	1171	3.3	3	1.15	8.47	0.01
	par8-4	350	1155	3.3	2	1.1	8.93	0
	par8-5	350	1171	3.3	0	1.3	9.53	0
Quasigroup	qg1-08	512	148957	290.9	7	0.9	276.76	6.42
	qg2-08	512	148957	290.9	1	3.65	359.28	22.4
	qg3-08	512	10469	20.4	13	0.35	22.14	0.01
	qg6-09	729	21844	30.0	0	1.7	61.47	0
	qg7-09	729	22060	30.3	5	0.75	52.18	0.01
Graph colouring	g125.17	2125	66272	31.2	4	0.85	100.19	TO - 110
	g250.29	7250	454622	62.7	7	0.8	371.42	TO - 383

Table 4.1: An “easier” set of instances from Zhang’s paper [68]. For Walksat, it shows the number of times a satisfying solution was found out of 20 runs, the average number of unsatisfied clauses at the end of the run, and the average run-time. For zChaff, it shows the time taken to solve the problem unless it timed out (TO).

					Walksat		zChaff
Domain	Instance	# vars	# clauses	clause/var	# unsat clauses	Time	Time
Hardware verification	bmc-ibm-1	9685	55870	5.8	23.3	31.81	1.09
	bmc-ibm-2	3628	14468	4.0	5.15	24.65	0.01
	bmc-ibm-3	14930	72106	4.8	109.95	33.2	0.05
	bmc-ibm-4	28161	139716	5.0	106.55	38.28	1.47
	bmc-ibm-5	9396	41207	4.4	10.6	327.67	0.03
	bmc-ibm-6	51654	368367	7.1	314.9	75.63	2.29
	bmc-ibm-7	8710	39774	4.6	17.65	56.59	0.01
	bmc-galileo-8	58074	294821	5.1	75.3	586.94	1.26
	bmc-galileo-9	63624	326999	5.1	73.6	559.04	8.73
	bmc-ibm-10	61088	334861	5.5	380.45	69.8	10.5
	bmc-ibm-11	32109	150027	4.7	424.8	48.75	7.09
	bmc-ibm-12	39598	194778	4.9	531	60.39	26.53
	bmc-ibm-13	13215	65728	5.0	81.6	28.09	3.39
Parity learning problems	par-16-1-c	317	1264	4.0	6.4	11.55	0.74
	par-16-1	1015	3310	3.3	11.2	10.68	1.56
	par-16-2-c	349	1392	4.0	6.85	11.92	1.52
	par-16-2	1015	3374	3.3	11.4	11.03	2.1
	par-16-3-c	334	1332	4.0	6.8	11.87	0.26
	par-16-3	1015	3344	3.3	12.2	10.85	0.54
	par-16-4-c	324	1292	4.0	6.2	11.78	0.01
	par-16-4	1015	3324	3.3	12.15	10.76	0.47
	par-16-5-c	341	1360	4.0	7	11.87	1.08
	par-16-5	1015	3358	3.3	11	10.97	1.45
	par-32-1-c	1315	5254	4.0	23.6	14.67	TO - 30
	par-32-1	3176	10277	3.2	33.55	13.75	TO - 30
	par-32-2-c	1303	5206	4.0	22.1	14.53	TO - 30
	par-32-2	3176	10253	3.2	34.05	13.71	TO - 30
	par-32-3-c	1325	5294	4.0	23.1	14.64	TO - 30
	par-32-3	3176	10297	3.2	33.15	13.75	TO - 30
	par-32-4-c	1333	5326	4.0	23.65	14.72	TO - 30
	par-32-4	3176	10313	3.2	32.5	13.89	TO - 30
	par-32-5-c	1339	5350	4.0	23.1	14.67	TO - 30
	par-32-5	3176	10325	3.3	33.45	13.94	TO - 30
random 3-SAT	f2000	2000	8500	4.3	1.6	21.69	TO - 30

Table 4.2: A “harder” set of instances from Zhang’s paper [68] for which Walksat could not find the satisfying solution for. For Walksat, it shows the average number of unsatisfied clauses at the end of the run and the average run-time, out of 20 runs. For zChaff, it shows the time taken to solve the problem unless it timed out (TO).

				Walksat			zChaff
Instance	# vars	# clauses	clause/var	# solns fnd	# unsat clauses	Time	Time
gen-k10-r720-1719	38	27360	720.0	20	0	186	105.84
gen-k5-v131-1749	131	2816	21.5	20	0	11.8	TO - 100
gen-k5-v170-1758	170	3655	21.5	17	0.15	29.2	TO - 100
gen-k9-v46-1808	46	16422	357.0	20	0	31.3	TO - 100
gen-k9-v46-1810	46	16422	357.0	20	0	20.2	TO - 100
hgen6-n520-815	520	2184	4.2	4	1.4	15.4	TO - 100
hgen6-n520-816	520	2184	4.2	1	1.6	16.5	TO - 100
hgen6-n520-817	520	2184	4.2	4	0.95	15.3	TO - 100
hgen6-n650-822	650	2730	4.2	0	1.8	17.6	TO - 100
hgen7-n520-866	520	2298	4.4	10	0.5	11.6	TO - 100
hgen7-n520-867	520	2298	4.4	9	0.55	14.2	TO - 100
hgen7-n650-870	650	2872	4.4	18	0.1	6.97	TO - 100
hgen7-n650-871	650	2872	4.4	6	0.7	15.6	TO - 100
hgen7-n650-872	650	2872	4.4	1	0.95	17.2	TO - 100
uf600-r4.25-1116	600	2550	4.3	15	0.25	9.13	TO - 100
uf700-r4.25-1120	700	2975	4.3	20	0	0.87	TO - 100
uf700-r4.25-1121	700	2975	4.3	0	1.3	17.5	TO - 100
uf700-r4.25-1122	700	2975	4.3	17	0.15	7.83	TO - 100
uf700-r4.5-1135	700	3150	4.5	0	5	18.8	TO - 100
uf700-r4.5-1136	700	3150	4.5	0	5	18.9	TO - 100
uf700-r4.5-1137	700	3150	4.5	0	4	18.5	TO - 100

Table 4.3: Random k-SAT instances from the Satisfiability Library [33]. For Walksat, it shows the number of times a satisfying solution was found out of 20 runs, the average number of unsatisfied clauses at the end of the run, and the average run-time. For zChaff, it shows the time taken to solve the problem unless it timed out (TO).

4.2.3 Unit Propagation and Implication

One way to classify the problem domains is by the structuredness of formulas [63]. In the domains where zChaff was successful, there were far more clauses with just one or two literals, which is one of the characteristics of structured instances. The numerous unit clauses quickly allowed zChaff to eliminate clauses and variables, which generated additional unit clauses from vast number of clauses with only two literals initially. Therefore, zChaff could go deep into the search tree without having to make many heuristic-based decisions while eliminating many constraints. On the other hand, for the local search algorithm, falsifying a unit clause was not unlikely, as a false assignment just penalized the search by one unit (just like any other clauses).

For example, in the hardware verification domain, where zChaff was particularly strong, the average number of unit clauses was 192 with some instances having as many as 447, and all had minimum of 49 unit clauses. Considering that 80% of the clauses have two literals for this class of problems, the unit clauses will lead to large number of propagations for zChaff.

Similar patterns can be detected in parity learning, another domain of zChaff's strength, where 3.7% of the total clauses in the par8-x instances were unit. The distribution is shown in Table 4.4. Similarly, in par16-x instances, 1.6% of the total clauses were unit.

The domains of Walksat's strengths were in random k-SAT and graph colouring problems. In random k-SAT, all the clauses have k literals per clause. In graph colouring problems, there are no unit clauses and almost all ($>$99%) clauses have exactly two literals. In these instances, there were no "free" assignments that can be made through unit propagation for zChaff, but instead, many nodes must be assigned first according to some heuristic and backtracked when found to be dead-ends.

4.3 Effects of Number of Optimal Solutions and Backbone Size to Search Cost for Random k-SAT Instances

Past studies have shown that the number of optimal solutions (*i.e.* satisfying solutions for SAT) predicts the local search cost well for small-backbone 3-SAT instances [29, 57]. Also, Parkes [51] and Singer et al. [57] have suggested that the local search cost is positively correlated with the size of backbone. A backbone literal is one that must be true in all optimal solutions to a given problem instance [15]. The aim of the this section is to investigate if such correlations exist for constructive search algorithms, where these factors have not received much attention from researchers as cost determinants.

Number of literals	1	2	3
Distribution (%)	3.7	27.8	68.5

Table 4.4: Distribution of clause size for parity problems (par8-x).

4.3.1 Methods and Test Instances

In order to find the backbone for a given instance, all the optimal solutions must be enumerated (this will also give us the number of optimal solutions obviously). This is done by running zChaff such that each time a satisfying solution is found, the negation of the satisfying set of assignment is added as a clause, and zChaff is run until the instance is no longer satisfiable. As enumerating all the optimal solutions is very expensive computationally, this affected the type and size of instances we could use for this experiment.

All of the instances used for this section are random 3-SAT with 100 variables. One of the reasons for focusing on the random 3-SAT instances is due to their availability in various constrainedness as well as the wide range in the number of optimal solutions and backbone sizes for a given problem size (number of variables) and constrainedness c/v. Here, c represents the number of clauses and v represents the number of variables. The higher the c/v ratio, the more constrained the instance is.

Each instance has 400, 430, or 471 clauses to represent under-, critically-, and over-constrained regions, and they are all satisfiable. Again, $c/v = 4.3$ is the phase transition region. Critically-constrained instances are from the Satisfiability Library [33]. The other sets of instances are generated using a typical random k-SAT generation method: Variables were negated with 50% probability, and no duplicate variables (regardless of its polarity) were included for a given clause as was the case for duplicate clauses. The number of instances used is summarized in Table 4.5.

As seen in Figure 4.1, the over-constrained instances ($c/v = 4.71$) are dominated by high-backbone instances while for the under- and over-constrained instances, backbone size is a little more evenly distributed. For the under-constrained region, about half of the instances were omitted since generating all the satisfying solutions for them were extremely expensive computationally. Inclusion of these instances would have shifted the distribution curve left towards the smaller $|backbone|$ region. Here, $|backbone|$ represents the fraction of the variables that are backbone.

Again, zChaff was used to represent the constructive search algorithms and Walksat was used to represent local search algorithms. Because both algorithms solved the problem instances rather quickly, instead of time, the number of decisions and the number of flips were used as the measure of search cost (finding 1 satisfying solution) for zChaff and Walksat respectively. Since we are interested in comparing the performance of the two algorithms to themselves independent of each other, we can use separate metrics for each algorithm. An advantage of using these metrics is that the results are independent of the machines and the intra-algorithm comparison of results is more accurate. Figure 4.2 shows that the run-time is highly dependent on the number of decisions, justifying the use of the number of decisions

Number of clauses	400	430	471
Number of instances	37	102	72

Table 4.5: Number of instances used for each constrainedness (all instances have 100 variables).

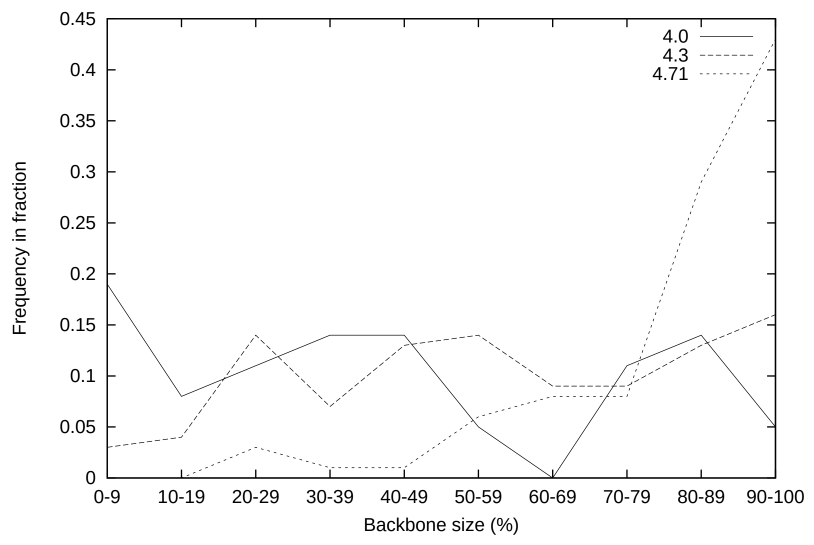


Figure 4.1: Distribution of problem instances used for this chapter according to their backbone sizes for various constrainedness ($c/v = 4.0, 4.3, 4.71$).

made as an accurate measure of search cost for zChaff. Similarly, the run-time of Walksat can be accurately determined by the number of flips needed to find a satisfying assignment.

Walksat was run with no time limit until a satisfying solution is found for an instance. Every instance was run twenty times with different random seeds, and the results were averaged over all the runs.

4.3.2 Number of Optimal Solutions

According to Singer et al. [57], small backbone instances ($|backbone| = 0.1$) had especially high $log - log$ correlation between the number of optimal solutions and local search cost with the r value at approximately -0.78. As the backbone size increased, the correlation dropped to -0.69 ($|backbone| = 0.5$) and -0.11 ($|backbone| = 0.9$). The drop in the correlation was not surprising as the large backbone size implied that the optimal solutions were tightly clustered and the local search's main difficulty was finding this cluster [57].

Hoos [29] observed a similar drop in the correlation with respect to the constrainedness for local search algorithms. The correlation in the under-constrained region ($c/v = 3.26$) was -0.90, whereas that in the over-constrained region ($c/v = 5.46$) was -0.51 for 50 variable instances. Since the under-constrained region has smaller backbone size on average and vice versa, the two studies are in agreement. The difference in the magnitude of drop is due to the less-than-perfect correlation between the c/v and $|backbone|$. Figure 4.1 illustrates this point as over-constrained $c/v = 4.71$ instances have instances with low to moderate $|backbone|$ and so on.

For this experiment, similar to Hoos' study, but with the addition of results for a constructive search algorithm, we show the correlation between the cost and the number of optimal solutions with respect to various levels of constrainedness.

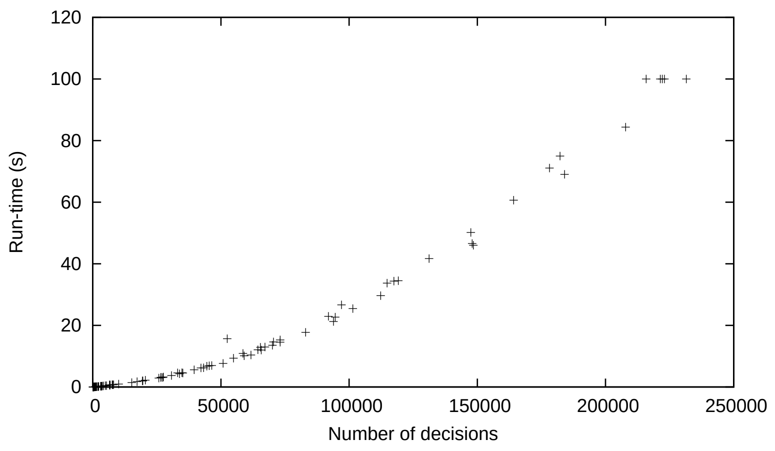


Figure 4.2: Scatter-plot of zChaff's run-time versus the number of decisions it makes before arriving at a satisfying solution for 3-SAT 250-variable instances.

Walksat The number of optimal solutions has a strong negative $log-log$ correlation with the number of flips (the search cost) of Walksat for both under- and critically-constrained instances (Figure 4.3 *left*, 4.4 *left*). As the constrainedness increases, the drop in the correlation can be seen from Figure 4.5 (*left*), which agrees with findings of both Hoos and Singer et al. Table 4.6 shows the numerical results from the correlation as well as the regression analysis. However, the slope a of the *least square line* monotonically increases ($|a|$ decreases) with constrainedness contrary to Hoos' results, which showed greatest a value for the critically-constrained region. The discrepancy may be from the fact that Hoos used GWSAT instead of Walksat. Singer et al.'s study supports the monotonically decreasing a with constrainedness since according to their study, with fewer optimal solutions available, the majority of the search cost is incurred from getting to the right region of the search space. Thus, the correlation as well as the a value should decrease with respect to c/v.

zChaff The correlation between the zChaff's search cost and the number of optimal solutions is significantly weaker than that for Walksat across all the constrainedness (see *right* side of Figures 4.3, 4.4, 4.5). The correlation coefficient and the parameters for least square line are

c/v	r	a	b
4.0	-0.83	-0.371	4.854
4.3	-0.75	-0.261	4.257
4.71	-0.47	-0.153	3.626

c/v	r	a	b
4.0	-0.49	-0.192	2.845
4.3	-0.29	-0.112	2.411
4.71	-0.06	-0.026	2.019

Table 4.6: Data from regression analysis of the correlation between number of optimal solutions and search cost for Walksat(*left*) and zChaff(*right*). r is the correlation coefficient, a and b are the parameters for least square line $ax + b$. All instances have 100 variables.

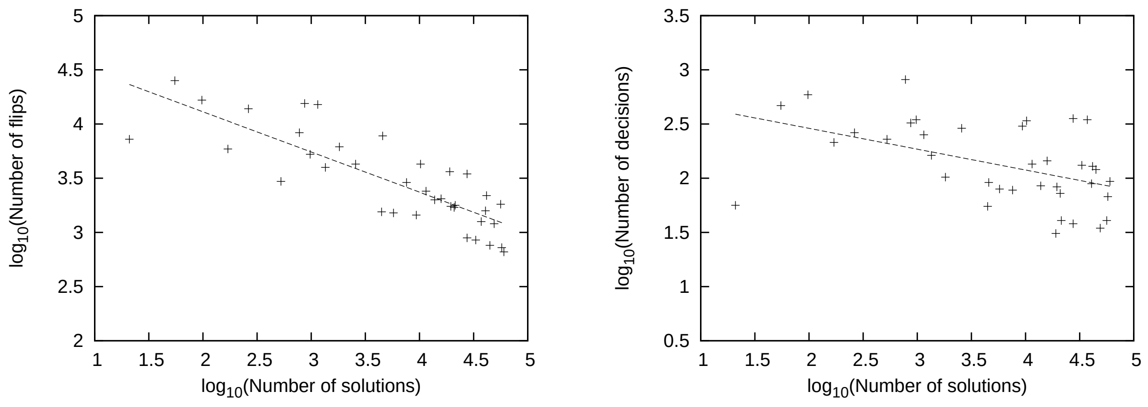


Figure 4.3: Scatter-plots of the search cost versus the number of optimal solutions for Walk-sat(*left*) and zChaff(*right*) for under-constrained instances (100 variables, 400 clauses).

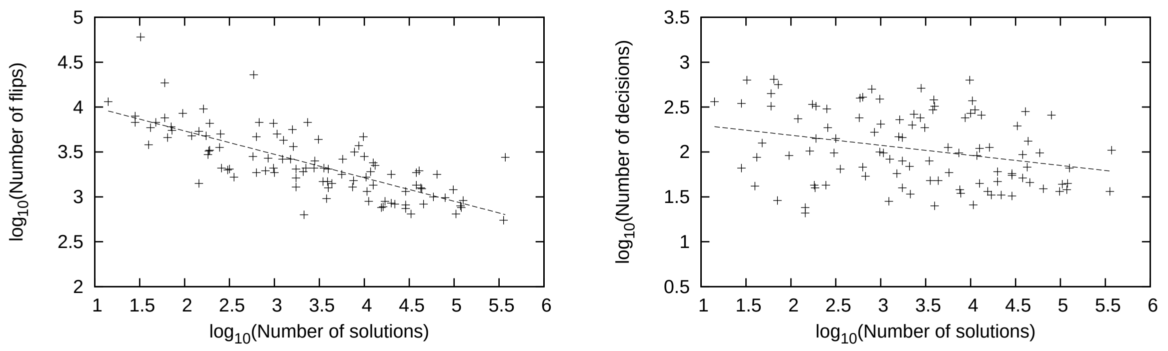


Figure 4.4: Scatter-plots of the search cost versus the number of optimal solutions for Walk-sat(*left*) and zChaff(*right*) for critically-constrained instances (100 variables, 430 clauses).

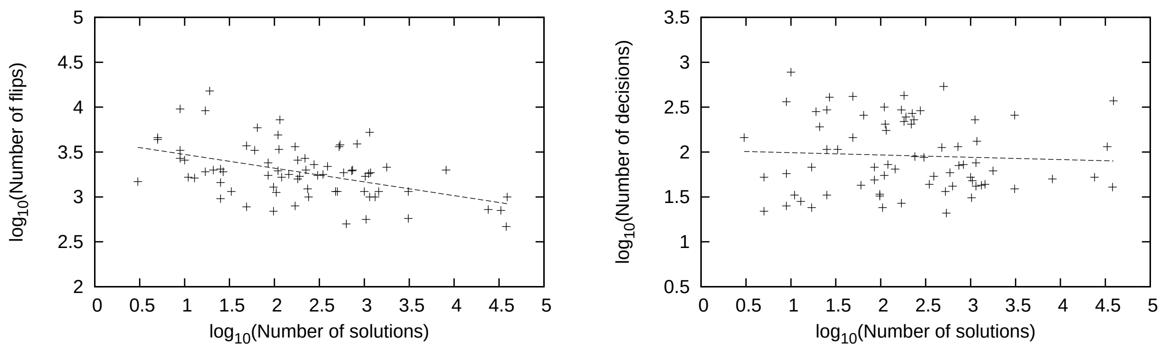


Figure 4.5: Scatter-plots of the search cost versus the number of optimal solutions for Walk-sat(*left*) and zChaff(*right*) for over-constrained instances (100 variables, 471 clauses).

summarized in Table 4.6 (*right*). The weaker correlation for zChaff (compared to Walksat) is due to zChaff's heavy reliance on propagation to assign values; the greater number of optimal solutions does not necessarily provide more opportunities for propagation. On the other hand, Walksat's highly stochastic nature, as well as the fact that it starts with a "complete" solution (all variables are assigned truth value) allows it to stumble upon a satisfying solution with higher probability when a greater number of optimal solutions are available throughout the search space.

zChaff shows a drop in both the correlation coefficient r and the regression slope a with increasing constrainedness similar to that seen in Walksat. The correlations at and beyond the critically-constrained region are fairly weak to non-existent. This can be attributed to the same reason as Walksat in that for over-constrained instances, the difficulty of search comes from finding the right cluster. Thus, if a cluster of an over-constrained instance has a few more satisfying solutions than a cluster of another over-constrained instance, the difference in the number of satisfying solutions will not greatly affect the search cost.

4.3.3 Backbone Size

Parkes [51] demonstrated that for under-constrained instances, only a small fraction of the variables appear in the backbone. However, as the c/v ratio approached the critically-constrained region, instances with large backbones (around 75-95% of the variables) rapidly emerged. Hence, it was suggested that the peak in average Walksat cost near the critical value may be due to the emergence of large-backbone instances at this point. Parkes further showed that the cost for the local search is strongly influenced by the size of the backbone. In this section, we verify Parkes' observations for local search as well as investigating the effects of backbone to constructive search cost.

Walksat In accordance with Parkes' claim, the log of the number of flips (search cost) for Walksat showed positive correlation with the backbone size for all constrainedness (see *left* in Figures 4.6, 4.7, 4.8). Whereas from Parkes' results it is difficult to understand how strong the correlation is, Table 4.7 (*left*) clearly shows that the correlation is much stronger for under-constrained instances than for over-constrained instances. It also shows that only when $|backbone|$ is small to moderate is $|backbone|$ a good search cost predictor. If $|backbone|$ is large enough, the cost is determined by the effort required to find the right cluster in the search space, at which point, the difference in $|backbone|$ is not a significant factor (similar argument as in the number of optimal solutions).

It is interesting to note that the magnitude of correlation $|r|$ between the local search cost and the $|backbone|$ is extremely similar to that between the local search cost and the number of optimal solutions ($|a|$ is significantly higher for the number of optimal solutions). Also, intuition tells us that higher $|backbone|$ will result in smaller number of satisfying solutions, and vice versa. This suggests possibly high negative correlation between the number of optimal solutions and the $|backbone|$, which is further investigated in Chapter 6.

zChaff The correlation for zChaff between the log of the number of decisions made and the backbone size was not as strong as that of Walksat as shown in Table 4.7 (*right*). Again,

c/v	r	a	b
4.0	0.85	0.012	2.964
4.3	0.72	0.010	2.779
4.71	0.47	0.008	2.586

c/v	r	a	b
4.0	0.630	0.008	1.793
4.3	0.390	0.006	1.676
4.71	0.110	0.003	1.740

Table 4.7: Data from regression analysis of the correlation between $|backbone|$ and search cost for Walksat(*left*) and zChaff(*right*). r is the correlation coefficient, a and b are the parameters for least square line $ax + b$. All instances have 100 variables.

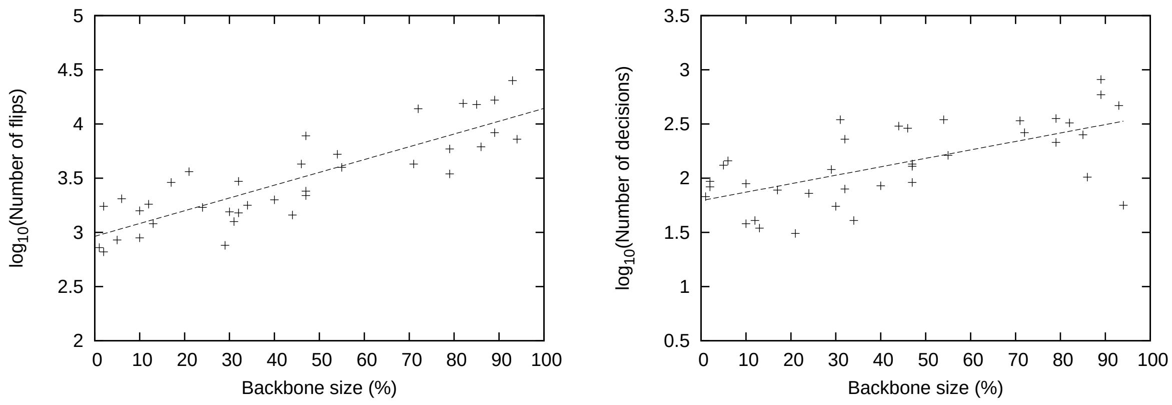


Figure 4.6: Scatter-plots of the search cost versus the backbone size for Walksat(*left*) and zChaff(*right*) for under-constrained instances (100 variables, 400 clauses).

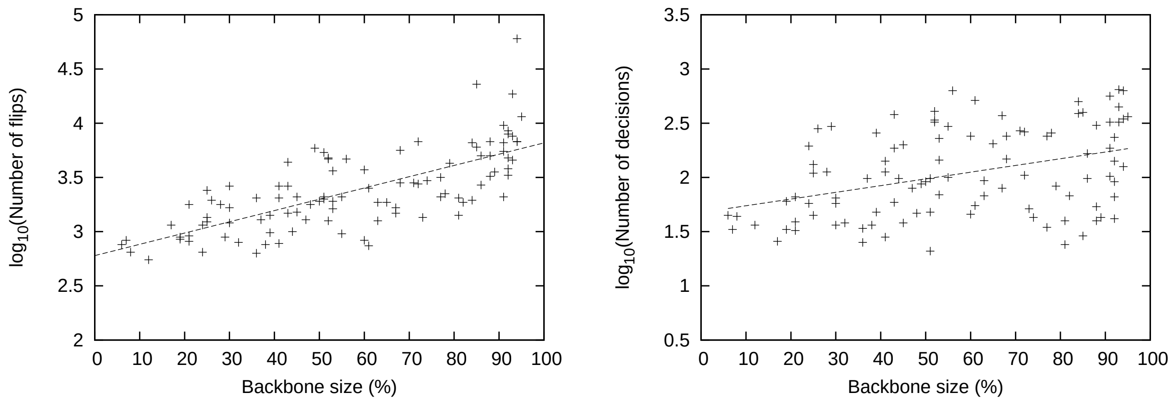


Figure 4.7: Scatter-plots of the search cost versus the backbone size for Walksat(*left*) and zChaff(*right*) for critically-constrained instances (100 variables, 430 clauses).

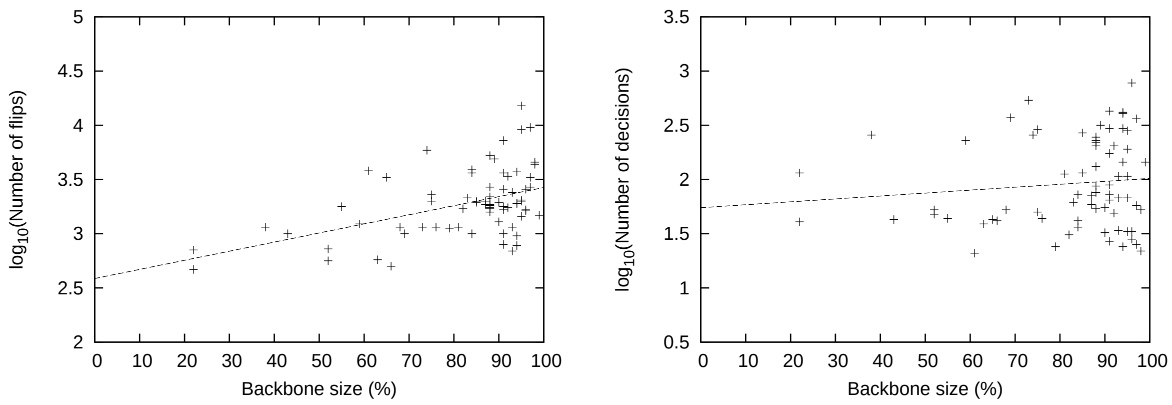


Figure 4.8: Scatter-plots of the search cost versus the backbone size for Walksat(*left*) and zChaff(*right*) for over-constrained instances (100 variables, 471 clauses).

we believe that the disparity in the correlation strength between Walksat and zChaff stems from the stochastic nature of Walksat (see Figure 4.7). Smaller $|backbone|$, which has satisfying solutions spread throughout the search space will allow Walksat to be in a promising region with higher probability while zChaff's propagation method cannot take advantage of the wide-spread satisfying solutions. Similar to Walksat, however, zChaff showed a monotonic decrease with respect to constrainedness for both r and a. Another possible reason for decreasing r with constrainedness is from the fact that there is much larger spread in $|backbone|$ for under- and critically-constrained instances (Figure 4.6, 4.7) than for over-constrained instances (Figure 4.8). For example, if there were more instances with smaller $|backbone|$ for the over-constrained case, we suspect the correlation between the search cost and the backbone size to be slightly higher.

4.4 Conclusions and Further Research

Our contribution from this chapter is two fold: first, we identified the strengths and weaknesses of both the constructive and local search algorithms in terms of problem domains. Previous to our study, there was little work done comparing the two different types of algorithms. Secondly, the two factors that affect the local search cost on 3-SAT problems, namely the number of optimal solutions and the backbone size, were applied to a constructive search algorithm to see if they have similar effects to its search cost.

1. zChaff and Walksat were selected to represent the constructive and local search method respectively. We showed that zChaff was far more effective than Walksat on hardware verification, planning, parity learning, and quasigroups problems. Walksat, on the other hand, was dominant on random k-SAT and graph colouring problems. Wei [63] suggested the structuredness of the problems in the domains to be the reason for zChaff's success. A similar observation was made in this study, where instances with many unit clauses were more advantageous to zChaff and those with no “free” assignments were more effectively handled by Walksat.

2. It has been known that the number of optimal solutions and the backbone size affect the local search cost for a given problem instance. We verified this by showing that random 3-SAT instances of various constrainedness have moderate to strong negative correlation between the Walksat search cost and the number of optimal solutions, and moderate positive correlation between the cost and the backbone size. In both cases, the over-constrained instances showed weaker correlation compared to the under- and critically-constrained instances. When the same search cost factors were applied to zChaff, weaker correlations than what we saw from Walksat were observed for both the number of optimal solutions and the $|backbone|$. We believe this is due to the fact that zChaff relies mostly on propagation, which does not heavily depend on the number of optimal solutions or $|backbone|$, for performance. Further, the fact that Walksat starts from a complete solution and moves among solutions makes it more dependent on these factors than zChaff. As with Walksat, however, similar weakening of the correlations was observed in zChaff with increasing constrainedness.

For further research in the performance of the constructive and local search, it may be interesting to look at different ways of characterizing the structuredness in problem instances. One of the cost determinants for zChaff could be measured by the number of implicants per variable assignment.[1] In other words, after every variable instantiation, the number of propagations made should affect the search cost. Another cost determinant could be the fraction of variable assignments that can be made before having to make a heuristic decision.

With respect to the study of the search cost factors for the two algorithms, the factors should be studied on structured instances as well. One of the challenges for this study will be from the difficulty in controlling the constant elements (*e.g.* problem size, class of problems) while manipulating variables such as the constrainedness and the number of optimal solutions.

[1]This suggestion is from Fahiem Bacchus.

Chapter 5

Problem Difficulty for Local Search Algorithms

As shown in the Chapter 4, the number of optimal solutions has a significant impact on the local search cost and to a lesser extent, on the constructive search cost as well. For the under-constrained region, the negative $log - log$ correlation between the number of solutions and the average search cost of Walksat is significant. The negative correlation follows from the fact that the greater number of solutions will yield higher probability that Walksat will "land" on a satisfying solution for a fixed number of possible states (*i.e.* variables). Thus, by extending this argument, we can expect the local search cost to continuously rise with respect to the constrainedness due to the falling number of satisfying solutions. However, even for satisfiable instances, as problems get more constrained past the critically-constrained region, problems tend to get a little easier to solve for local search algorithms [10]. Although in the previous chapter we observed a weakening of the negative correlation between the number of solutions and the local search cost in the over-constrained region, this weakening does not explain the drop in the search cost past the critically-constrained region.

For constructive search algorithms, such a drop in the search cost in the over-constrained region is intuitive: when a problem instance is over-constrained, problems actually get easier since constructive search algorithms can quickly fathom nodes without exploring too deep into unpromising regions. These algorithms find critically-constrained problems to be the toughest since they do not have a lot of solutions nor allow quick pruning of literal assignments.

For local search algorithms, which cannot prove satisfiability for a given instance, there is no notion of eliminating conflicting variable assignments or unpromising nodes. Thus over-constraining the problems should not necessarily make the problem easier. However, a similar (though not as significant in magnitude) drop in the search cost is observed for this type of search algorithm in the over-constrained region. The drop in local search cost in spite of the decreasing number of optimal solutions shows that there ought to be additional factors that make over-constrained problems easier for local search algorithms.

This chapter looks at why such a drop in search cost is observed beyond the critically-constrained region for local search algorithms. There have been a few studies on this matter. Notably, Singer et al. [57] and Yokoo [67] provide interesting insights and a good starting point for the chapter. Both papers presented plausible if not conclusive evidence for the observed phenomenon. The purpose of this chapter is to address any shortcomings in the two studies and validate their conjectures to form a better understanding for the local search cost peak found in the critically-constrained region.

In addition to satisfiability, we also look at the problem difficulty for local search in *job-shop scheduling problems (JSP)* and address similarities and differences between the two domains. Based on the work of Watson et al. [61], we apply the problem difficulty model for JSP to SAT.

5.1 Background Information

In this section, the actual local search cost around the critically-constrained region is presented to verify the easy-hard-easy pattern in the local search cost. Further, the works of Singer et al. and Yokoo are discussed.

5.1.1 Search Cost for Various Levels of Constrainedness

Phase transition region or *crossover point* is a region where the probability of solution (satisfiability) changes abruptly from near 0 to near 1. In this region, where the problem is critically-constrained, the computational difficulty for finding a satisfying solution for SAT is also at its greatest [7]. Crawford and Anton [10] empirically showed that the abrupt change in satisfiability and the peak in search cost indeed coincide at this phase transition point for constructive search algorithms.

Later, Clark et al. [8] showed that a similar easy-hard-easy pattern across the constrainedness is found for local search algorithms. Here, we experimentally verify that the local search cost peak actually occurs at the critically-constrained region.

Test Instances For all the experiments in this chapter, we use a randomly generated set of instances as shown in Table 5.1 and 5.2, unless stated otherwise. For the problem generation, variables were negated with 50% probability, and no variable (regardless of sign) appeared more than once in a clause. Also, no duplicate clauses were allowed.

Results All the instances were run 20 times with the maximum number of flips set to 100000. The mean search cost for every instance was again averaged according to the constrainedness. Figure 5.1 (*left*) clearly shows that the difficulties at local search is the greatest at $c/v = 4.3$ for satisfiable instances, similar to that reported in the previous studies [7, 10, 8]. To the right

c/v ratio	4.0	4.2	4.3	4.41	4.5	4.71
Number of instances	75	69	80	82	66	76

Table 5.1: Number of satisfiable instances used across various c/v ratio (random 3-SAT, 100 variables).

c/v ratio	4.2	4.3	4.41	4.5	4.7	5.0
Number of instances	62	77	75	67	45	81

Table 5.2: Number of unsatisfiable instances used across various c/v ratio (random 3-SAT, 100 variables).

of the $c/v = 4.3$ region, the search cost quickly plateaus at around 2500 flips. For the under-constrained region, however, the average cost continuously drops at a similar rate past the $c/v = 4.0$. At $c/v = 2.0$, the search cost was around 28.[1] This is not surprising considering the abundance of satisfying solutions and the loose constraints at $c/v = 2.0$. We could not test for instances with $c/v > 4.71$ due to the scarcity of satisfiable instances in that region.

As a reference, such easy-hard-easy pattern is not observed for unsatisfiable instances when using the local search algorithm (see Figure 5.1 *right*). Rather, there is a monotonic increase in the cost with the increasing constrainedness. Here, the search cost for unsatisfiable instances is defined as the number of flips it takes to get to an optimal solution, which is found *a priori* using *maxsat* [6].

5.1.2 Previous Studies on Additional Factors

Characteristics of local minima found during local search algorithms are critical to the algorithms' effectiveness in finding globally optimal solution. Local minima, in general, have been studied in great depth in a variety of literature. For example, along with Yokoo [67] and Singer et al.'s [57] work on local minima, Hoos [34] investigated the distribution and density of local minima for *travelling salesman problems*. In Hoos' work, to quantify the local search characteristics, *fitness-distance analysis* was used. Fitness-distance analysis aims to evaluate the nature of the relationship between the solution quality and the distance between solutions

[1]Not shown in Figure 5.1

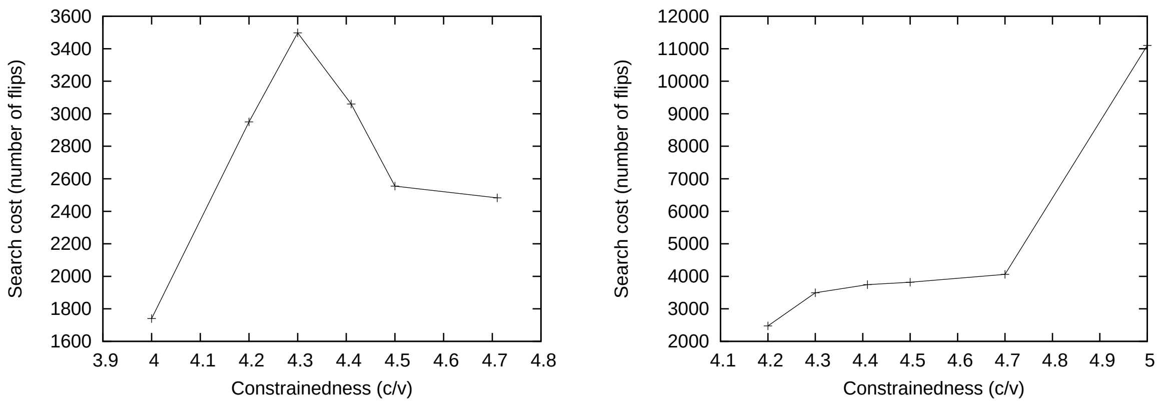


Figure 5.1: Average search cost for Walksat for satisfiable (*left*) and unsatisfiable (*right*) instances of various c/v ratios (100 variables).

within a given search landscape [34]. Also, Watson et al. [61] showed the importance of local minima in *job-shop scheduling problems*, and presented a variety of models as a function of local minima that influence the search cost for tabu search.

Yokoo's study [67] examined the change in the number of local minima across the constrainedness. The definition of *true* local minimum (optimum) is a set of assignments that has the fewest unsatisfied clauses among all the neighbouring states. Here, a neighbouring state is a set of assignments that can be reached with a flip of a variable assignment from the current set of assignments. The study observed that the number of local minima decreases monotonically with the increasing constrainedness (see Figure 5.2 *right*). The study claimed that the fewer the local minima, the easier the instance is for local search, because there are fewer sub-optimal solutions that will distract the search, misleading it possibly away from an optimal solution. According to Yokoo, this factor, along with the number of optimal solutions, form the two factors that account for the peak in the local search cost at the critical region. Where the instance is critically constrained, the combination of low number of solutions while having enough local minima to distract the search algorithm, makes it hard to reach optimality.

However, Yokoo's study did not show why the cost-peak should necessarily occur at the critically-constrained region. The rate of decrease in the number of local minima is dramatically reduced well before the critical region of $c/v = 4.67$ [67] (c/v ratio moves slightly as a function of the number of variables [10]). Also, the reduction in the number of available solutions is very smooth and lacks any notable features across the critical region. Another weakness in the study's argument is that the intuition that the number of local minima should affect the local search cost was never verified experimentally or analytically.

Singer et al. [57] agreed that the decreasing number of solutions explains the increasing search cost towards the critically-constrained region. However, they identified the competing factor that makes the search easier past the critical region to be the *extensiveness* of the area where the local minima reside in. Extensiveness of local minima refers to how spread out they are in the search space. They claimed that as the problem gets more constrained, local minima encountered during the search are clustered closer to the optimal solutions. On the other hand, when a problem is under-constrained, local minima tend to be spread out throughout the search space, yielding attractive, low-cost solutions far away from the global minima. This requires the search to wander through a greater portion of the search space before arriving at an optimal solution. The reason that the under-constrained instances are easy to solve despite the extensiveness of local minima is due to the abundance of optimal solutions throughout the search space.

For their experiment, Singer et al. defined a local optimum as a set of assignment with *five* unsatisfied clauses. They then determined the *proximity* between a local optimum to and its closest global optimum by simply taking the Hamming distance between the two solutions. The results from Singer et al. in Figure 5.3 clearly shows the decrease in the proximity of local minima to their closest global minima with the increasing constrainedness across all the backbone sizes. Thus, they claimed that the decrease of search cost observed in the over-constrained region is due to this extensiveness of local minima.

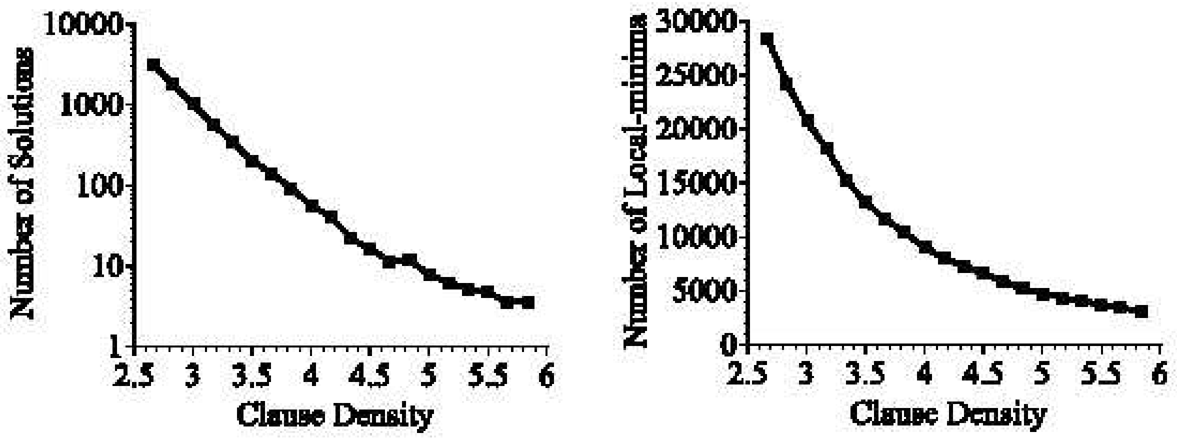


Figure 5.2: Average number of optimal solutions (*left*) and average number of local optima (*right*) for 3-SAT problems (20 variables) from Yokoo [67].

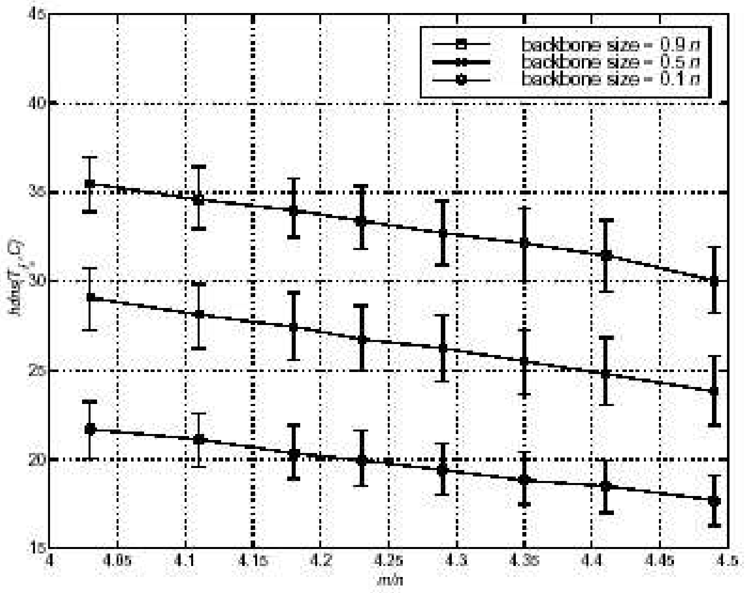


Figure 5.3: Hamming distance to nearest optimal solution (hdns) from local optima according to pre-set backbone sizes from Singer et al. [57].

5.2 Number of Local Minima

Yokoo [67] formed an argument around the number of local minima as the factor that contributes to the decreasing local search cost past the phase transition point. However, the actual relationship between the number of local minima and the local search cost was not shown in the study. A moderate to strong negative correlation is expected between the two variables in order to support Yokoo's study. Here, we use a set of satisfiable, 20 variable instances across various c/v ratio (see Table 5.3). The chosen size for problem instances is due to computational difficulty in enumerating all the local minima. Here, a local minimum is defined as a solution state that has less than or equal number of unsatisfied clauses than its neighbours.[2] In case of a plateau, all solutions that have the fewest unsatisfied clauses are counted as local minima.

However, as shown in Figure 5.4, the $log - log$ correlation between the number of local minima and the search cost of Walksat is not as strong as we expected with $r = -0.44$ and the slope of the least squares regression fit line at $a = -0.775$. The moderate r value shows that the number of local minima indeed is a contributing factor to that makes local search cost drop past the transition phase but likely not the only factor.

5.3 Extensiveness of Local Minima

The purpose here is to empirically verify Singer et al.'s [57] claim that the local minima extensiveness is indeed greater in the under- and critically-constrained region than in the over-constrained region. One of the things we did differently from Singer et al.'s study is to aggregate the results for all the backbone sizes. This way, we are able to better isolate the contributing factors that affect for various constrainedness since we are interested in the search cost for *all* the backbone sizes. Also, we use a different definition of local minimum from Singer et al.'s study; we found that Singer et al.'s local minimum definition to be too arbitrary, and thus, it is newly defined in the following section.

Along with the differences above, we also use *fitness-distance analysis (FDA)* in order to measure the spread of local minima in the search space. As mentioned earlier, FDA is a well-known technique that has been used in other problem domains to assess the relationship between local minima and global minima. In the SAT domain, fitness refers to the number of unsatisfied clauses, and the distance is the Hamming distance between an assignment to its closest global optimum. For Walksat, the evaluation function is simply the number of unsatisfied clauses since it determines which of the unsatisfied clauses to "fix". *Fitness-distance correlation* simply measures the correlation between the quality of local minima and their dis-

[2]Those solution states that are one flip away from the current state.

c/v ratio	3.0	3.5	4.0	4.3	4.5	4.7	4.85	5.0	5.2	5.5
Number of instances	100	95	86	60	53	87	58	87	63	48

Table 5.3: Number of satisfiable instances used across various c/v ratio (20 variables).

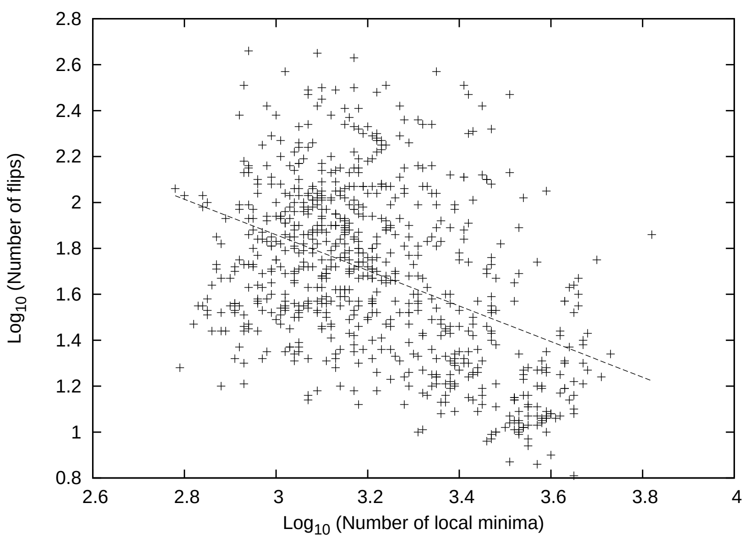


Figure 5.4: Search cost of Walksat versus the number of local minima for 3-SAT, satisfiable, 20 variable instances of various constrainedness.

tance to the nearest globally optimal solutions [34].

Zhang [68] performed a similar analysis on Walksat by plotting the quality of local minima against their distance to nearest global minima on a three-dimensional plot using the frequency of the FDA-value pairs as the third axis. Here, a FDA-value pair for a local minimum is characterized by two values: the distance to nearest optimum and the number of unsatisfied clauses. The major difference between Zhang's and our method is that whereas Zhang used aggregated data over all the instances for a given c/v ratio, we examined instances on individual basis as typically done in FDA. We believe that our instance-based results give better idea as to the spread of local minima as opposed to the "average" results for the entire instances of a c/v ratio. To our knowledge, FDA and FDC have never been applied in the SAT domain.

Following from Singer et al.'s conjecture that the extensiveness of the local optima in the over-constrained region is less than under- and critically-constrained region, we should see that local optima in the over-constrained region have shorter distance to their nearest global optimum than those found in the under- and critically-constrained region. Thus, we should see higher FDC for over-constrained region than under- and critically-constrained region.

5.3.1 Methods and Test Instances

Again, Walksat is used to represent local search algorithms. The same set of 3-SAT instances in Table 5.1 and 5.2 were used.

Local minima used for this study are defined as those solutions that have fewer unsatisfied clauses than the solutions visited just before and after the current solution. They may be non-strict local minima, in which case, only one of the local minima is used as the "representative" from that region in search space. They are collected by running Walksat 10 times per instance until each run finds optimal solution for satisfiable instances. For unsatisfiable instances, the maximum number of flips was set to 100,000.

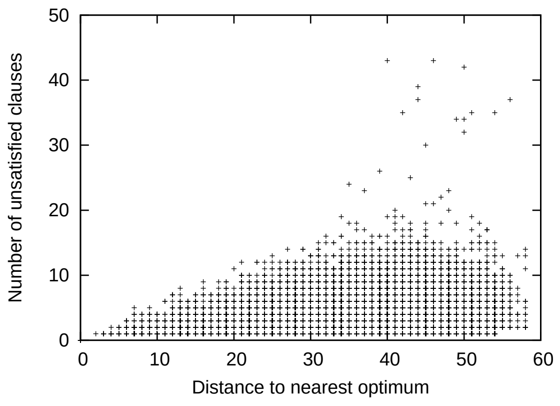
Enumeration of all the optimal solutions for satisfiable instances are done by running *zChaff* [47] solver, and for unsatisfiable instances, *maxsat* [6] solver was used.

5.3.2 Fitness-Distance Analysis Results

Fitness-distance plots for individual satisfiable instances for various constrainedness are given in Figure 5.5 and 5.6. Again, they do not necessarily represent the "average" case for each respective clause size (if such exists) but rather portray fairly "typical" fitness-distance plots from our experience. What is surprising from the results is how spread-out local minima are throughout the search space for *all* of the constrainedness levels. There are numerous local minima that have only one unsatisfied clause and yet are half-way across the search space from the closest global optimum (*i.e.* Hamming distance of 50 for 100-variable instances).

The prevalence of high-quality local minima far away from the global optima for the entire range of constrainedness is contrary to our expectation. Preliminary results from the 3-dimensional plot with the third axis being the frequency of the FDA-value pairs, further show that the local search algorithm *consistently* visits these local minima that are considerably distant from the global minima (see Figure 5.7 and Figure 5.8). In fact, as Figure 5.8 shows, a majority of the local minima visited by the search have Hamming distance of 20 or more to their closest global optimum. This is because, to Walksat, for a given number of unsatisfied clauses, a local minimum that is distant from the global minimum is just as good as the one that is close to the optimum. The general shape of the fitness-distance plots is quite different from that from that of travelling salesman problems, where the gradient towards the bottom left of the plot is much more clearly defined along the diagonal of the plot area [34].

Fitness Distance Correlation Fitness-distance correlation (FDC) shows that there is a slight increase in the correlation coefficient r with respect to the increasing constrainedness (see Table 5.4). This result supports Singer et al.'s conjecture that the proximity between local minima and their closest global minimal reduces with greater c/v ratio, since the reduction in the number of high quality local minima in the region distant from the global optimum will result in a more



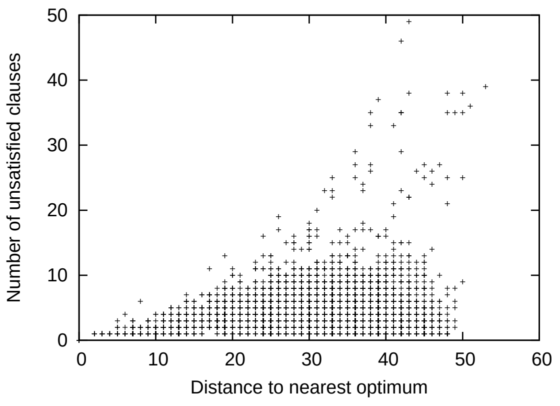
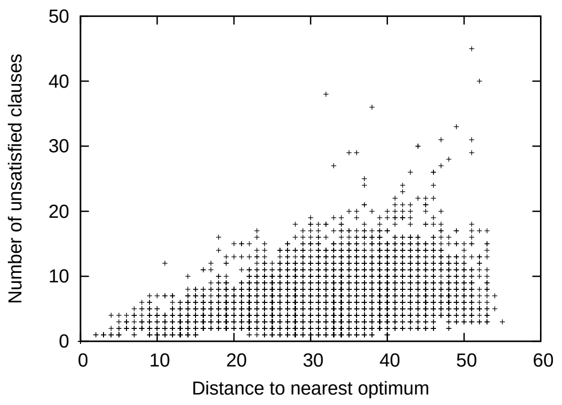


Figure 5.5: Fitness-distance plots for an under-constrained $c/v = 4.0$ (*left*) and a critically-constrained $c/v = 4.3$ (*right*) instance. Both are satisfiable and have 100 variables.



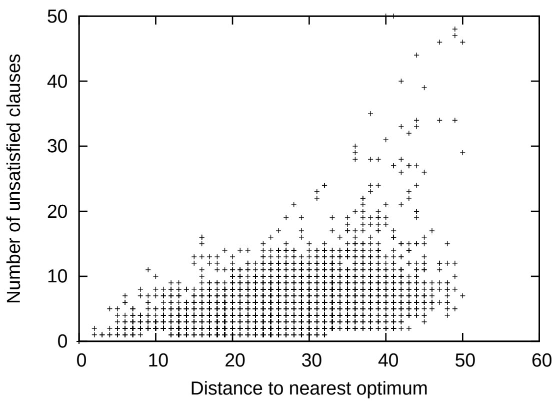


Figure 5.6: Fitness-distance plots for over-constrained instances. $c/v = 4.5$ (*left*) and $c/v = 4.71$ (*right*). Both are satisfiable and have 100 variables.

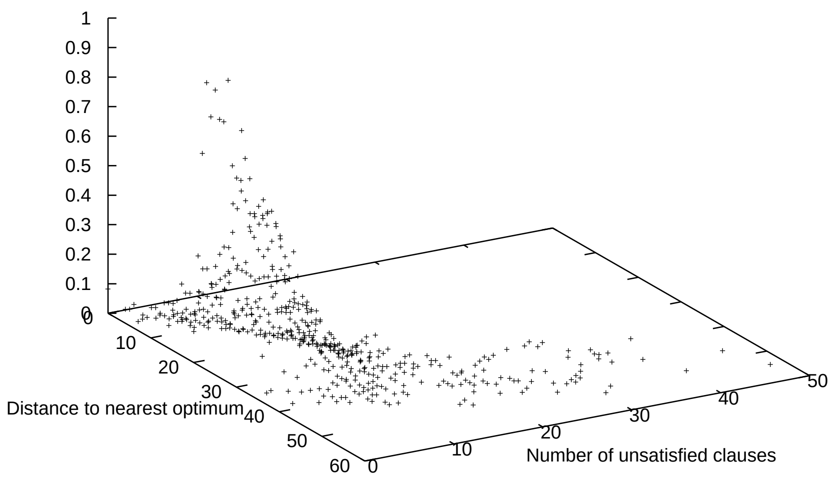
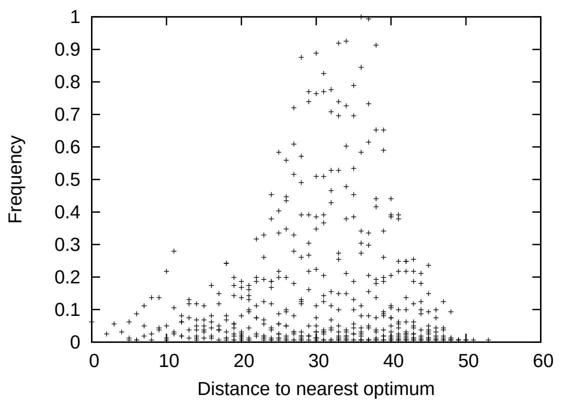


Figure 5.7: A 3-dimensional fitness-distance plot for a critically-constrained ($c/v = 4.3$), satisfiable, 100-variable instance with z-axis being frequency of FDA-value pairs. Identical instance as Figure 5.5 (*right*).



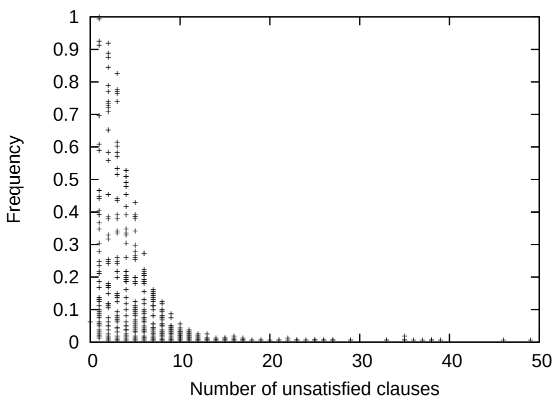


Figure 5.8: Orthogonal projections of the 3-dimensional plot in Figure 5.7. *Left* shows the normalized frequency of FDA-value pairs against distance to nearest optimum. *Right* shows the frequency of FDA-value pairs against number of unsatisfied clauses.

"streamlined" set of points along the diagonal of the plot area, thus increasing FDC. However, the increase in FDC with c/v is not as strong as we expected.

We also looked at FDC across various c/v ratios for unsatisfiable instances as a reference. Here, the r value actually drops with the increasing constrainedness, which suggests greater spread of high-quality local optima for over-constrained instances. However, given the fact that the search cost does not decline past the phase transition region (Figure 5.1 *left*), the decreasing FDC is not very surprising.

FDC and Search Cost Based on the fitness-distance analysis, we conjecture that the search cost and FDC are negatively correlated. For high FDC instances, the search will not get trapped in local optima far away from global optima, thus advancing from low-quality and distant region to high-quality, lesser-distant (from the nearest optimum) region more quickly. On the other hand, harder instances, where search algorithm spends more time in high-quality, more-distant region, will reduce the correlation between quality and number of clauses.

In Figure 5.9, satisfiable instances in the under- (*left*) critically-constrained (*right*) region are plotted based on the average local search cost for 10 runs and its FDC value. As seen from the figure, there is indeed a strong negative correlation between the logarithm of the search cost and the FDC value, ranging from -0.55 to -0.78. Figures 5.10 shows the similar kind of correlation exist in the over-constrained region with $r = -0.59$ (*left*) and $r = -0.81$ (*right*). The differences in the strength of correlation do not seem to have a particular relationship with

c/v ratio	4.0	4.2	4.3	4.41	4.5	4.71
Average FDC r	0.40	0.42	0.43	0.46	0.44	0.50

Table 5.4: Average fitness-distance correlation value for satisfiable instances of different constrainedness (100 variables).

c/v ratio	4.2	4.3	4.41	4.5	4.71	5.0	6.0
Average FDC r	0.62	0.60	0.60	0.56	0.55	0.52	0.50

Table 5.5: Average fitness-distance correlation value for unsatisfiable instances of different constrainedness (100 variables).

respect to the constrainedness.

5.3.3 Average Distance of Local Minima from Global Minimum

One major difference among instances of various constrainedness (shown in Figure 5.5 and Figure 5.6) is the extensiveness of high-quality local minima. For the instances shown, the spread of high-quality local minima decreases with increasing constrainedness. For example, in the under-constrained region (Figure 5.5 *left*), there are local minima with only *one* unsatisfied clause that are 55 Hamming distance away from the closest global optimum. In the over-constrained region (Figure 5.6 *right*), however, the maximum distance for local minima with one unsatisfied clause is 33.

This is more clearly displayed in Figure 5.11, which shows the average Hamming distance of local minima visited by Walksat for various constrainedness (the same set of instances as the fitness-distance analysis is used). Here, for a given number of unsatisfied clauses, local minima from instances with lower c/v ratio is much further away from their closest global minima than those from the higher c/v ratio. This results in Walksat spending much of its time far away from the global optimum for critically-constrained instances and makes them more difficult than over-constrained instances despite having more optimal solutions.

Algorithm Independence of Results It is conceivable that the results seen in Figure 5.11 are due to the intrinsic characteristics of Walksat. From the way Walksat chooses one clause

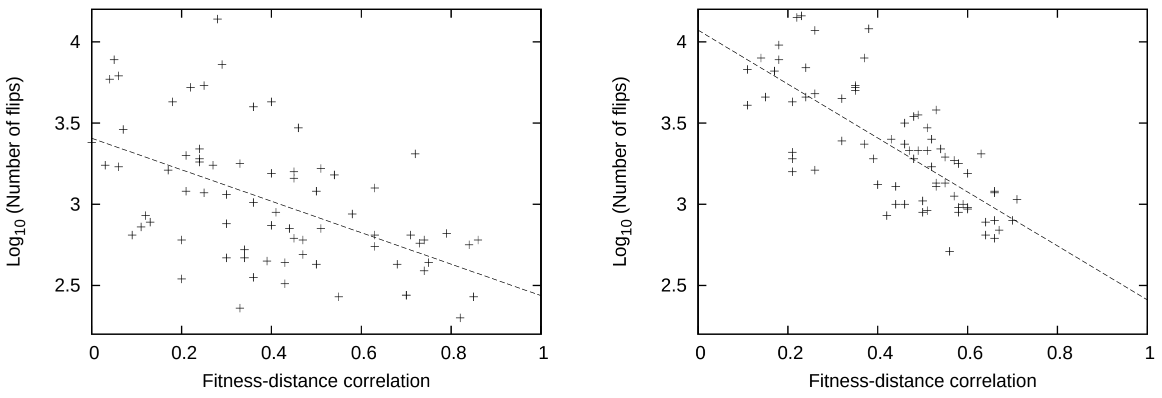


Figure 5.9: Correlation between the search cost (in log scale) and FDC value for satisfiable, under-constrained (*left*, $c/v = 4.0$) critically-constrained (*right*, $c/v = 4.3$) instances.

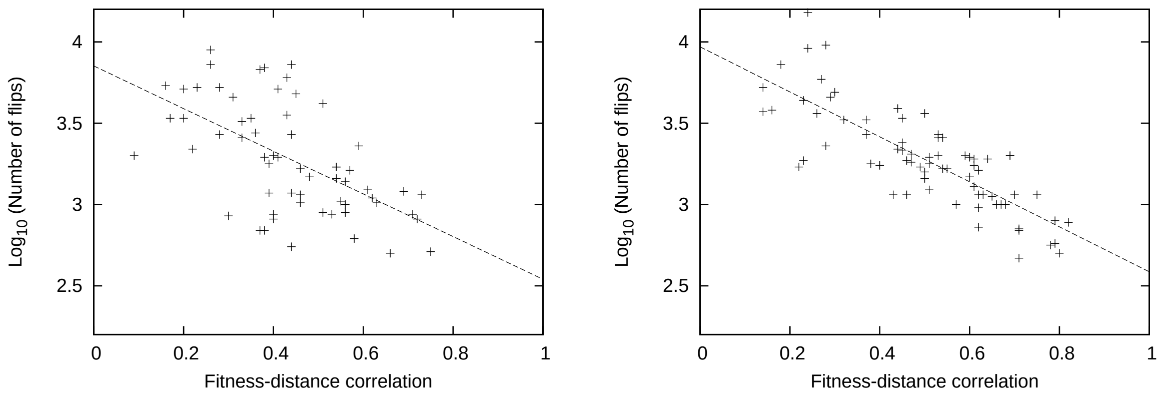


Figure 5.10: Correlation between the search cost (in log scale) and FDC value for satisfiable, over-constrained instances (***left***, $c/v = 4.5$; ***right***, $c/v = 4.71$).

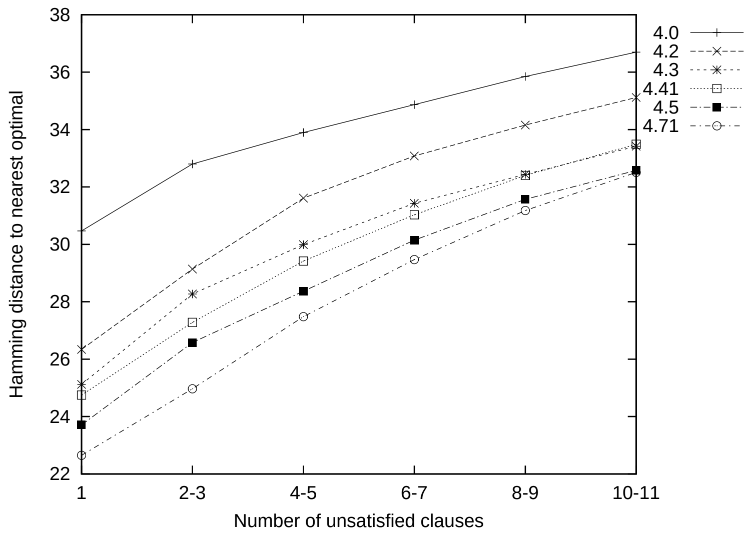


Figure 5.11: Distance to nearest optimum versus number of unsatisfied clauses for local minima visited by Walksat.

out of all the unsatisfied clauses and the way it determines the variable to flip, there is bias built into Walksat (as with all other search algorithms) that affects the type of local minima it visits. Also, it is inherently very stochastic, more so than other SAT solvers [55], due to the frequency at which it makes random movement even when hill-climbing (improving) moves are available.

In order to provide evidence that Figure 5.11 is the property of the instances themselves and not algorithm-dependent, GSAT was applied to the same set of instances. Local minima sampled by GSAT are considered *true* local minima because GSAT *always* chooses to make hill-climbing moves until it cannot find a better solution from the current set of neighbouring solutions. For this experiment, we used GSAT implementation in UBCSAT [59] with 10 independent runs on each instance, and each run was restarted from a random spot each time it was stuck at a local minimum, (at which point, the state was recorded as a local minimum) until it found a satisfying solution.

The resulting plot of mean distance-to-nearest-optimum versus the number of unsatisfied clauses for varying c/v ratio is shown in Figure 5.12. As with the Walksat case, in general, under-constrained instances tend to have high-quality local minima that are further away from the global minimum compared to over-constrained instances. However, the difference in the average Hamming distance between the various constrainedness for a given number of unsatisfied clauses is not as dramatic for GSAT as it was for Walksat. There is also a notable exception from the usual pattern; the local minima with 1 unsatisfied clause for $c/v = 4.71$ is further away from the global minimum than those for $c/v = 4.5$. At this point, we suspect that this is from a random error, but this has not been verified. The convergence of the plots at lower-quality local minima is similar to what was observed in Walksat results although the convergence is much faster for GSAT.

In general, we can conclude that regardless of the search algorithm used, local minima found in under-constrained region tend to be further away from the global optima, contributing to decreasing search cost beyond the critically-constrained region.

5.3.4 Summary

In Section 5.3, we have looked at the extensiveness of local minima from a different perspective compared to the previous study (Singer et al. [57]): we demonstrated using fitness-distance analysis that the local minima visited by Walksat are spread out throughout the search space for the various c/v ratios we have tested. High-quality local minima that are very distant from their nearest optimal solutions are common in all c/v ratios and appear frequently. The fitness-distance correlation indeed increased with c/v though not to the extent we expected.

When the average distance to nearest optimum was plotted for local minima of varying qualities for different c/v ratios (Figure 5.11), it was clear that for local minima of given quality, those from under-constrained region are further away from their nearest optima than those from over-constrained region. We believe that this difference in extensiveness of local minima is a major contributing factor for the decreasing local search cost in the over-constrained region.

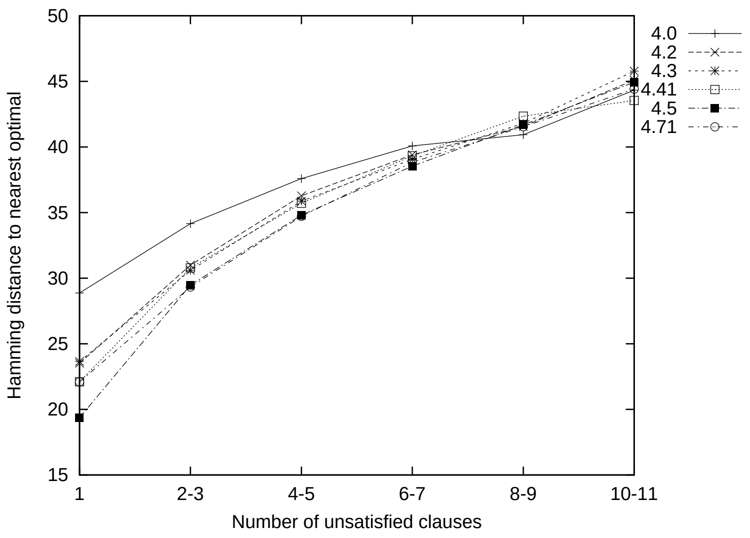


Figure 5.12: Distance to nearest optimum versus number of unsatisfied clauses for local minima visited by GSAT.

5.4 Problem Difficulty in Job-Shop Scheduling Problems vs SAT

Many parallels can be drawn across various problem domains in the field of optimization. For example, search space consisting of high-quality local optima (minima) away from global optima is prevalent in both *job-shop scheduling problems* (JSP) and satisfiability. Also, many *constraint satisfaction problems* show the phase-transition property, where the constrainedness of an instance plays an important role in determining the satisfiability and search cost [7]. We also pointed out that the high fitness-distance correlation is an important characteristic of *travelling salesman problems*. In this section, we examine the problem difficulty for local search algorithms in job-shop scheduling problems and how it compares with that of SAT.

Watson et al. [61] presented a series of analyses on job-shop scheduling problems and offer insight on the local search cost factors. They considered a $n \times m$ job-shop scheduling problem, where n jobs must be processed exactly once on each of m machines. Each job is routed through each of the m machines in some pre-defined order (see Section 6.1 for a more detailed description of JSP). Among a few models Watson et al. proposed for factors determining the search cost for JSP, the $d_{lop-opt}$ model was one of the most effective models. The $d_{lopt-opt}$ model, which was motivated from Singer et al.'s [57] study, demonstrated that instances with higher average distance between local optima ($lopt$) and their nearest global optima (opt) are harder to solve than those with smaller distance. This makes sense because if the local minima visited by the algorithm are closer to the global minima, then the search is more likely to be able to move quickly from a local minimum to its nearest global minimum.

For both 6×4 and 6×6 cases, the correlation between the search cost and the average distance to the nearest optimal solutions for local minima was very strong with r^2 values at 0.774 and 0.682 respectively [61]. However, Watson et al. noticed that the residuals were much greater for very high-cost instances. Further, Watson et al. pointed to Singer et al.'s [57] results where they showed higher residuals in very high-backbone instances for their $d_{lopt-opt}$ model. Following these two results, Watson et al. conjectured that the $d_{lopt-opt}$ model will be less accurate (thus higher residuals) for very high-cost SAT instances than for the entire range of search cost.

To test this hypothesis, 365 satisfiable instances of 3-SAT with 100 variables were studied. The instances were fairly evenly divided among 420, 430, 441, 450, and 471 clauses. For each instance, all the optimal solutions were enumerated. Then, using Walksat, each instance was solved 10 times while storing all the local optima visited during the runs. The same definition of local minimum as Section 5.3 is used. For each instance, the average distance of local minima from their closest global minima is plotted against the average search cost. As shown in Figure 5.13 (*left*), across the entire cost range, the correlation between the cost and distance to the nearest optimum is strong with the coefficient $r = 0.73$. Unlike the JSP results, however, it is hard to tell from the figure whether the residuals are actually higher in the very high-cost region.

The instances are then divided into two sets depending on their average search cost. If the average search cost (measured by number of flips) is greater than 5000, the instance was considered high cost, and otherwise, it is considered low cost. The correlation coefficient and the residual standard error for the two sets as well as for the entire set is shown in Table 5.6. Contrary to the Watson et al.'s conjecture, the high cost instances do not show any higher residual error than the low cost instances. In fact, the high cost residual error is slightly lower. Also, the correlation coefficient declines for both sets when the original set is divided into high- and low-cost sets. This is more likely from the fact that the range of the search cost has been reduced, which should make the correlation drop.

In order to further test the hypothesis, an additional set of high-cost instances were considered: 192 instances (100 variables and 430 clauses) with search cost greater than 5000. The resulting residual standard error is 0.47, which is similar to the results seen in Table 5.6. The plot for this set of instances can be seen in Figure 5.13 (*right*). The result is further supported when considering only the instances with 100 variables and 430 clauses from the first set (with all the cost range considered). In this case, the residual error is 0.64 (see Table 5.6).

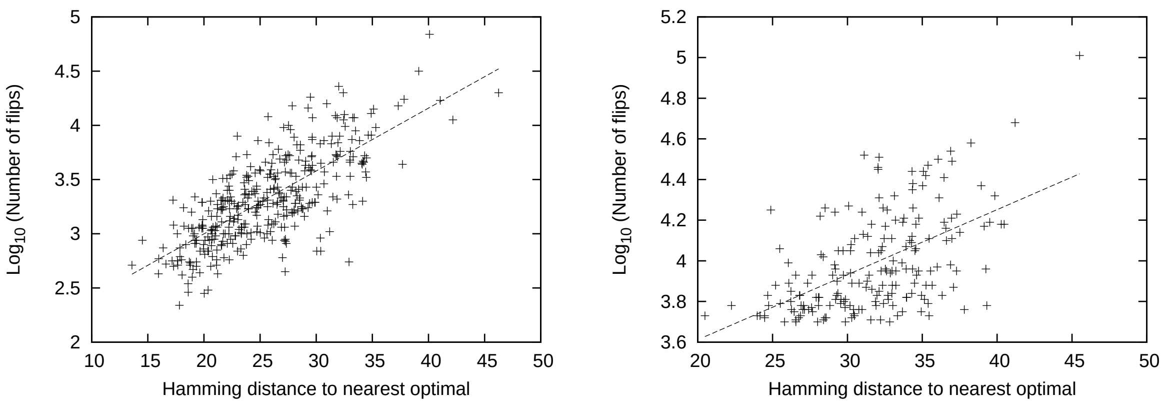


Figure 5.13: Average search cost versus $d_{lopt-opt}$ for all instances (*left*) and for high-cost ($>$ 5000 flips) instances (*right*).

	correlation r	residual error
All cost, All c/v	0.73	0.64
High cost, All c/v	0.52	0.44
Low cost, All c/v	0.60	0.53
All cost, Only $c/v = 4.3$	0.69	0.64

Table 5.6: The correlation coefficient r and residual error for 3-SAT instances with 100 variables and various c/v ratios. The instances were divided into high (> 5000) and low (≤ 5000) cost instances.

Based on our results, Watson et al.'s results of inaccuracy of the $d_{lopt-opt}$ model for very high-cost instances in JSP do not seem to be applicable for SAT domain. Also, Singer et al.'s results do not seem to either support or contradict Watson et al.'s conjecture; despite the fact that the correlation coefficient r decreases with increasing $|backbone|$ for a given c/v ratio, which supports the conjecture, the r value hardly changes at all across c/v ratio for a given $|backbone|$, which does not support the conjecture.

One major limitation for our results arises from the relative ease with which Walksat was able to solve these 100 variable instances. Even with some of the hardest instances from the entire set, Walksat was able to solve it within a few seconds. It is plausible that this conjecture hold true for much more difficult instances. However, this could not be tested due to the extreme computational cost of enumerating all the solutions and finding distance-to-nearest-optimum for more difficult instances.

5.5 Conclusions and Future Work

This chapter looked at problem difficulty for local search algorithms in SAT. Unlike constructive search algorithms, which can prove a given problem instance is unsatisfiable, local search algorithm cannot take advantage of reduced search space and quick pruning of variables that are present in over-constrained instances. Thus, the decline in the local search cost with increasing constrainedness past the phase transition point is surprising given that the number of solutions continue to decrease.

Yokoo [67] claimed that it is the decreasing number of local minima that reduces the local search cost past the phase transition region. However, we showed that when the average search cost is plotted as a function of the number of local minima for 20 variable instances, the correlation was not as strong as we expected. Thus, we concluded that the number of local minima is one of the contributing factors but not the sole factor that leads to the reduction in local search cost past the phase transition point.

Singer et al. [57] claimed that the competing factor that makes SAT instances easier with increasing constrainedness is the extensiveness of local minima in the under-constrained region. We validated the claim by performing fitness-distance analysis on instances of various constrainedness and observing that the fitness-distance correlation (FDC) increased with respect to the constrainedness. Higher FDC suggests that the spread of local minima is less extensive, which makes the problem easier to solve since the local search algorithm will not be distracted towards high-quality local minima that are far away from global minima. We verified this by showing the strong negative correlation between the local search cost and the FDC for all c/v ratios. In the end, we believe it is the combination of the two factors, one being the decreasing number local minima and the other being the decreasing extensiveness of local minima, that make the over-constrained problems easier than the critically-constrained problems.

For future studies on this topic, given that the number of local minima decreases and the average distance between high-quality local minima and global minima decreases with increasing constrainedness, one can directly verify this result by studying the local minima that disappear from adding clauses to a given instance. We conjecture that the local minima that are far away from the global minima will be the first to disappear with added clauses.

In addition to the analysis of the problem difficulty for a SAT local search algorithm, we also compared the problem difficulty for SAT and JSP. In particular, Watson et al. [61] noted a deterioration of the $d_{lopt-opt}$ model for high-cost JSP instances and conjectured that a similar pattern will exist for high-cost SAT instances. Our results failed to support this hypothesis as the residuals in high-cost instances were actually lower than that for the overall results. However, this may be from the relatively easy nature of the problem instances used for the experiment. It would be interesting to see how the $d_{lopt-opt}$ model holds for much larger, difficult instances.

One area for future work is examining the problem difficulty for structured instances. Watson et al. suggest a possible deterioration of the $d_{lopt-opt}$ model for the structured instances in SAT. Along with the investigation of the $d_{lopt-opt}$ model, fitness-distance analysis should be performed on structured instances to see if similar extensiveness of local minima is found for structured instances. One of the challenges of studying the structured instances is the generation of all the solutions and having enough instances to make the results stable. This may be addressed by taking advantage of *morphing* random instances to introduce structuredness to the problem [19].

One of the ramifications of this study is the understanding of *backbone-guided search* mechanism and the reason behind its success. Backbone-guided search algorithms rely heavily on local optima to estimate the assignments to backbone variables and to eventually find an optimal solution. The fitness-distance analysis from this chapter raises some interesting questions: if a large subset of local optima is far away from the global optima as it has been throughout the fitness-distance analysis, how do backbone-guided search algorithms perform at the level that they claim to? Could there be some other benefits from keeping a pool of solutions other than for estimating backbone values? As the idea of backbone-guided search is getting more popular [17, 42], it is imperative to investigate the claims of backbone-guided search algorithms and gain better understanding of the mechanism. We turn to these questions in the next chapter.

Chapter 6

Understanding Backbones and Backbone-Guided Search

A *backbone* variable is one that has the same value in all optimal solutions to a given problem [15]. Since its introduction by Parkes et al. [51], the idea of backbone has received a lot of attention from researchers. Along with its implications in phase transition and problem difficulty for SAT [51, 57], the concept of backbone has also been applied to various SAT solvers to improve their performance. Zhang [68] built a mechanism for estimating the backbone prior to solving a problem and integrated it with a local search algorithm (Walksat). It has also been applied with success to a constructive search algorithm in Dubois and Dequen's work [17]. Further, in addition to satisfiability, backbone has been studied in other combinatorial problems including job-shop scheduling and travelling salesman problems [61, 42].

Even with the active research on backbones, it is still an area that is largely untapped and speculative. For example, Watson et al. [61] noted that the correlation between the number of optimal solutions and the backbone size is extremely high for job-shop scheduling problems (JSP), and suggested that the two factors may be redundant. Is this an intrinsic property of backbone, or does it only apply to JSP? Does backbone really provide no more information than knowing the number of optimal solutions? Things are even cloudier when it comes to the applications of backbone. Do the Backbone-Guided search algorithms really take advantage of the backbone information given that we do not know how accurate the backbone estimates are?

The purpose of this chapter is to answer the questions surrounding backbones and contribute to our understanding of the nature of backbones and their usage in applications.

6.1 Job-Shop Scheduling Problems

Here, we introduce job-shop scheduling problems (JSP) in detail since we will re-visit Watson et al.'s [61] conjectures regarding backbones in JSP. Watson et al. consider the well-known $n \times m$ static job-shop scheduling problem (JSP), in which n jobs must be processed exactly once on each of m machines. Each job i $(1 \leq i \leq n)$ is routed through each of the m machines in some pre-defined order π_i, where $\pi_i(j)$ denotes the j^{th} machine $(1 \leq j \leq m)$ in the routing order. The processing of a job on a machine is called an *operation*, and the processing of job

i on machine $\pi_i(j)$ is denoted by o_{ij}. An operation o_{ij} cannot begin processing until o_{ij-1} has completed processing, and pre-emption and concurrency are not allowed.

A solution s to an instance specifies a processing order for all of the jobs on each machine, and implicitly specifies an earliest start time $est(x)$ and earliest completion time $ect(x)$ for each operation x [61]. The objective is to minimize the *makespan* $C_{max}(s)$, where,

$$C_{max}(s) = max(ect(o_{1m}), ect(o_{2m}), ..., ect(o_{nm})) \tag{6.1}$$

An important piece of information in any solution to a JSP is its *critical path* as it defines the makespan for the solution. A critical path of a solution s consists of a sequence of operations $o_{1k_1}, o_{2k_2}, ..., o_{lk_l}$, such that $est(o_{1k_1}) = 0$, $ect(o_{lk_l}) = C_{max}(s)$, and $est(o_{ik_i}) = ect(o_{i-1k_{i-1}})$ for $1 \leq i \leq l$ [61]. Local search algorithms for JSP typically focus on rearranging the sequence of operations on the critical path in order to find a better makespan. Figure 6.1 illustrates the critical path for a 10×10 JSP example.

Backbones for Job-Shop Scheduling Problems Similar to SAT, a notion of *backbone* exists for JSP. A common way to represent a JSP is through *disjunctive graph*, where $\binom{n}{2}$ binary variables are defined for each of the m machines. The variables represent the precedence relationship between pairs of jobs on the same machine. Consequently, a backbone of a JSP can be defined as the set of binary variables that have the same value in all optimal solutions. Watson et al. [61] further defined $|backbone|$ as the number of such variables normalized by $m\binom{n}{2}$.

For the 6×6 case considered by Watson et al., the order of the 6 jobs can be represented by $15 = \binom{6}{2}$ binary variables by establishing precedence relationship for each pair of jobs for a resource. For all 6 resources, 90 binary variables are generated. The definition of the binary variables is as follows for n jobs and m resources case ($n \times m$):

> Let x_{ijk} represent a binary variable that describes the precedence relationship between job i and job j on machine k.
> If $x_{ijk} = 1$, job i comes before job j on machine k.
> If $x_{ijk} = 0$, job j comes before job i on machine k.
> where,
> $i = \{0, 1, .., n-2\}$
> $j = \{i+1, i+2, .., n-1\}$
> $k = \{0, 1, .., m-1\}$

6.2 High correlation between $|backbone|$ and number of optimal solutions

"Exceptional similarity" was reported between the r values of the $|backbone|$ and $|optsols|$ models in Watson et al. [61], where $|optsols|$ represents the number of optimal solutions. It was found that the correlation between $|backbone|^2$ and $log_{10}|optsols|$ for 6×4 and 6×6 problems were -0.9337 and -0.9103 respectively. The paper concluded that for job-shop

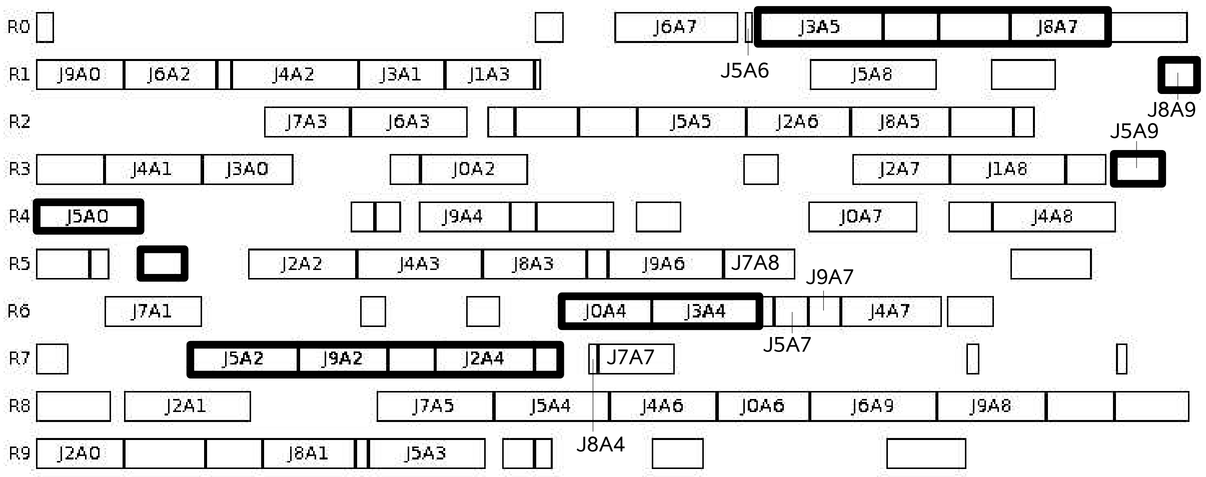


Figure 6.1: A 10×10 JSP example with the critical path highlighted by the thick borders. $JiAp$ represents p^{th} operation of job i.

scheduling problems, the number of optimal solutions and $|backbone|$ are redundant factors in their correlation to search cost. Watson et al. then conjectured that such a strong correlation between the two factors may be present in SAT.

We apply a similar methodology used in Watson et al.'s work to investigate this conjecture. By enumerating all the optimal solutions to a set of SAT instances, we can find the $|backbone|$ and the number of optimal solutions, and compute their correlation. Satisfiable, random 3-SAT instances with 100 variables and various number of clauses are analyzed. The test instances are the same set of instances used in Chapter 5. The variables for these instances were negated with 50% probability, and no duplicate clauses were allowed. Again, zChaff [47] was used to enumerate all the optimal solutions and determine the $|backbone|$ for each instance. Unsatisfiable instances were not considered. The ensemble of instances used for the experiment is summarized in Table 6.1.

One of the limitations here is the different number of instances used for different c/v ratio. We believe that we have a sufficient number of instances for $c/v > 4.0$. The reason for smaller number of instances used for $c/v = 4.0$ is because we eliminated those instances that we could not completely enumerate all the solutions due to the computational expense.

Figure 6.2 (*left*) shows the scatter graph when $log_{10}|optsols|$ is plotted against $|backbone|^2$. As can be seen, $|backbone|^2$ is strongly negatively correlated to the number of solutions. It

c/v ratio	4.0	4.2	4.3	4.41	4.5	4.71
Number of instances	37	69	80	82	66	76

Table 6.1: Number of SAT instances used for various constrainedness (all satisfiable, 100 variables).

is interesting that for instances of very high $|backbone|^2$, there is almost a perfect correlation between the number of optimal solutions and $|backbone|^2$. The correlation coefficient is -0.81, which is high but less than that of the JSP.

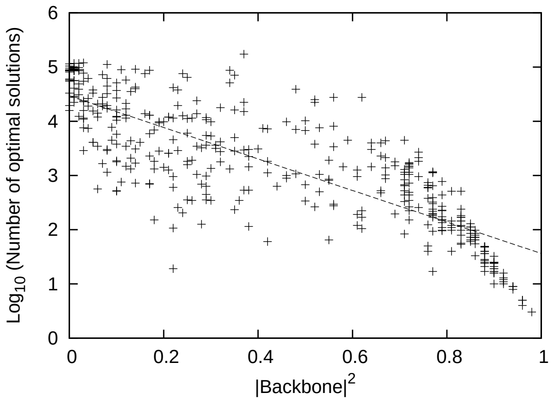
In order to show the negative correlation between the number of optimal solutions and $|backbone|^2$, we took a subset of 6×6 instances (605 instances) from Watson et al. and plotted the two factors as shown in Figure 6.2 (*right*). The particular subset here has a slightly lower correlation value of $r = -0.87$ between the two factors than the entire set used by Watson et al. Here, the data used are directly from Watson et al.'s work, and we did not re-run their experiment.

One noticeable difference between SAT (Figure 6.2 *left*) and JSP (Figure 6.2 *right*) is the distribution of the instances in terms of $|backbone|^2$: JSP has greater number of high backbone instances than SAT. This suggests that these JSP instances are more highly constrained. Also, even though the correlation values for these particular sets of instances are fairly similar, the residual standard error s_{res} of SAT is much larger than that of JSP at 0.647 and 0.533 respectively. Here, $s_{res} = \sqrt{\frac{\sum(Y-Y_{est})^2}{n-2}}$, where Y is the observed value, Y_{est} is the predicted values from the regression line, and n is the number of data points used.

The higher degree of correlation in JSP as well as smaller residual error indicates that given the number of optimal solutions of an instance, the backbone size of JSP can be predicted with higher accuracy than in SAT. A possible reason for the higher correlation between the number of optimal solutions and $|backbone|^2$ in JSP is that a solution in JSP is dominated by the operation sequence on the critical path. Presumably, backbone variables will tend to be between a pair of activities on the critical path in most optimal solutions. Thus we should be able to swap those activities that are not on the critical path (likely non-backbone) and come up with another optimal solution. If not, the new solution should still be fairly close to another optimal solution, not counting the one we just used to derive the new solution. Using Figure 6.1 as an example, a pair of operations that are not on the critical path such as $J5A7$ (7^{th} operation of job 5) and $J9A7$ can be swapped without affecting the overall makespan. The same thing can be said about $J8A4$ and $J7A7$. On the other hand, for SAT, non-backbone variables may or may not be as loosely constrained depending on individual instances and inter-dependence among clauses, thereby creating higher *variation* in the number of optimal solutions.

We note that for JSP (and SAT) we are not claiming that JSP solution can be generated by identifying the backbone and simply counting the combinations of the non-backbone variables. The combinations of the non-backbone variables rather provide a loose upper bound for the number of optimal solutions. What we are claiming here is that the effects of changing the values of non-backbone variables are more predictable because of the regularity of JSP.

Another way to look at this is from the scope of non-backbone variable assignments. In SAT, a change in a non-backbone variable assignment is potentially more "global", meaning that it could affect other parts of the solution (including backbone variables) since the changed variable is linked to other variables through the clauses which the variable is part of. In JSP, however, change in a non-backbone variable assignment (swap of operation sequence for a particular machine) is more likely to be "local" in that it is less likely to affect the rest of the solution. Thus, in JSP, non-backbone variables can change their assignments much more freely, producing a consistent number of solutions. Since we suspect that the *degree of implications*, which is a measure of a variable's global effect, for non-backbone variables in SAT is greater



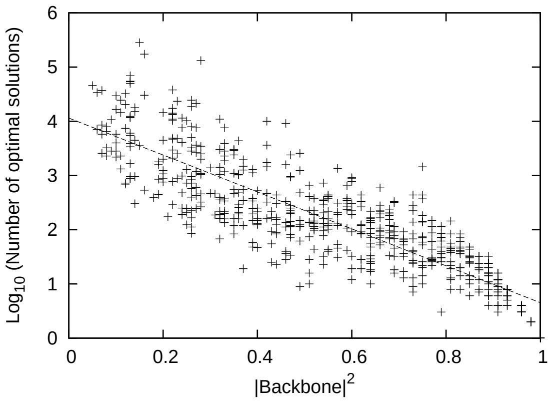


Figure 6.2: Scatter plots of the number of optimal solutions versus the $|backbone|^2$ for SAT (*left*) and JSP (*right*). Satisfiable instances of 100 variables and various constrainedness (see Table 6.1) for SAT and 6×6 instances for JSP.

than those in JSP, the number of non-backbone variables (hence, the number of backbone variables) in SAT is not as good of a predictor for number of optimal solutions as it is in JSP.

6.2.1 Degree of Implication

Following the observations above, we conjecture that the greater correlation between the number of solutions and $|backbone|^2$ for JSP is due to the *smaller* degree of implication for its non-backbone variables than SAT. The higher the degree of implication, the greater the impact a non-backbone variable has globally in the entire solution space, which results in a more unpredictable number of solutions for a given backbone (or non-backbone) size.

Measuring Degree of Implication We need a way to measure the degree of implication for both SAT and JSP. One way is to calculate the *modified distance-to-nearest-optimum*. Distance-to-nearest-optimum (dno) measures the Hamming distance between a solution and its nearest optimal solution (as seen in Section 5.3). *Modified* dno ($mdno$) specifies that the local minimum's nearest optimum must come from the subset of optimal solutions with a non-backbone variable of interest set to a particular value. For example, if we want to calculate the degree of implication for a non-backbone variable x flipped to $true$, then, we calculate $mdno$ based on the subset of optimal solutions with variable $x = true$.

For a given instance, we have a set of optimal solutions, and a set of backbone and non-backbone variables can be subsequently identified. If we pick a non-backbone variable from an optimal solution and flip its value, we will get a new solution. The Hamming distance from this new solution to its nearest optimal solution with the non-backbone variable set to this new value (*i.e.* $mdno$) will represent the impact of a change in the non-backbone assignment. The actual algorithm to calculate $mdno$ for a given instance is shown in Figure 6.3.

Methods For SAT, the same ensemble of 100-variable instances used in Section 6.2 was used for this experiment (see Table 6.1 for details).

```
$total\_mdno := 0$
$hasBeenSeen := \{\}$
**for** each optimal solution $s \in S$, where $S$ is a set of all optimal solutions
    **for** each non-backbone variable $nbv \in s$
        $s' := s$ with $nbv$'s value flipped
        **if** $s' \notin hasBeenSeen$ container
            Compute $mdno$ among $S$ that have the same truth value for $nbv$
            $total\_mdno += mdno$
            $hasBeenSeen := hasBeenSeen \cup s'$
$avg\_mdno = \frac{total\_mdno}{sizeof(hasBeenSeen)}$
```

Figure 6.3: Algorithm for finding the average Modified Distance-to-Nearest-Optimum ($mdno$) for a problem instance.

For JSP, 593 6×6 instances from Watson et al.'s collection were used. These are the same subset of the instances used for Figure 6.2 (*right*) except that 12 instances were omitted because each had only one optimal solution. In such cases, $mdno$ cannot exist by definition since there are no non-backbone variables. The number of variables in the JSP instances is 90, which is similar in size as the SAT problems.

For each problem instance (both SAT and JSP), all solutions were enumerated, which in turn revealed $|backbone|$. Then using the algorithm in Figure 6.3, average $mdno$ can be found for each instance. The distribution of the instances with respect to $|backbone|$ can be seen in Table 6.2.

Results and Discussion Since the problem size for SAT and JSP is different, we normalized $mdno$ with respect to the number of variables. The plot of average $mdno$ with respect to $|backbone|$ is as shown in Figure 6.4. The vertical bars for each data point indicate the standard deviation.

$\|backbone\|$	0-.09	.10-.19	.20-.29	.30-.39	.40-.49	.50-.69	.60-.69	.70-.79	.80-.89	.90-1
# SAT instances	21	32	29	33	37	44	23	30	81	80
# JSP instances	0	0	13	37	40	84	76	82	118	143

Table 6.2: Number of instances used for various constrainedness. Satisfiable instances with 100 variables for SAT. 6×6 instances for JSP (90 variables).

As expected, $mdno$ for SAT is significantly greater on average than for JSP for backbone size less than 0.7. This indicates that the impact of a change of non-backbone variable in SAT is much more global than it is for JSP, thus making it more difficult to predict the number of solutions given a backbone size. These results directly support our hypothesis.

Interestingly, at $|backbone| > 0.7$, JSP displays greater $mdno$ than SAT, although not by a significant margin. Although this seems contradictory to our hypothesis, a closer look at Figure 6.2 reveals that SAT should indeed have a small $mdno$ at high $|backbone|$. In this region, the correlation between the $|backbone|^2$ and the number of satisfying solutions seems to be comparable to that of JSP. It is the small to moderate $|backbone|^2$ region that SAT seems to have much weaker correlation than JSP.

6.3 Backbone as Search Guidance

So far, we have focused on the characteristics of backbone, how it can affect the search cost and its relationship with other attributes of SAT. In this section, we look at how backbone information can be used in SAT solvers to guide the search for a solution. In particular, *Backbone-Guided (BG) local search algorithm* by Zhang [68] is analyzed. The purpose of this study is to examine the actual mechanism of backbone guidance and verify whether the performance observed is due to the guidance from the backbone estimates. It is not clear whether the success (or the lack thereof on certain problem instances) really results from taking advantage of the backbones or from other benefits of maintaining a pool of local minima.

6.3.1 Backbone-Guided Local Search Algorithm by Zhang

Zhang's [68] Backbone-Guided local search algorithm is built around the notion that more constrained variables must be assigned to the correct value first. In the case of backbone variables, they must be set to the correct value or otherwise the instance cannot be satisfied. Thus, correctly setting these variables should get priority from the algorithm. However, the backbones for a given instance cannot be found without enumerating all the optimal solutions. Therefore, what Zhang does instead is to *estimate* the backbones using *local minima* of the given instance. He claims that Walksat, the underlying algorithm used, is quite adept at finding high-quality local minima.

The Backbone-Guided (BG) algorithm has two major phases in its operation: the *initialization phase* consists of *pseudo backbone frequency* computation, where the backbones are estimated using local minima generated by Walksat. This is done simply by running Walksat multiple times on a problem instance for a fixed number of flips and collecting the best local minimum reached in each run. Then, from the set of local minima, pseudo backbone frequency for each literal can be computed. Specifically, pseudo backbone frequency $p(l_j)$ for a literal l_j is defined as follows:

$$p(l_j) = \frac{\sum_{\forall s_i \in S, l_j \in s_i} 1}{|S|}$$

where S is the set of local minima, i is the i^{th} local minimum, and j is the j^{th} variable. Also, by definition, $p(l_j) = 1 - p(\neg l_j)$, where $\neg l_j$ is the negation of l_j.

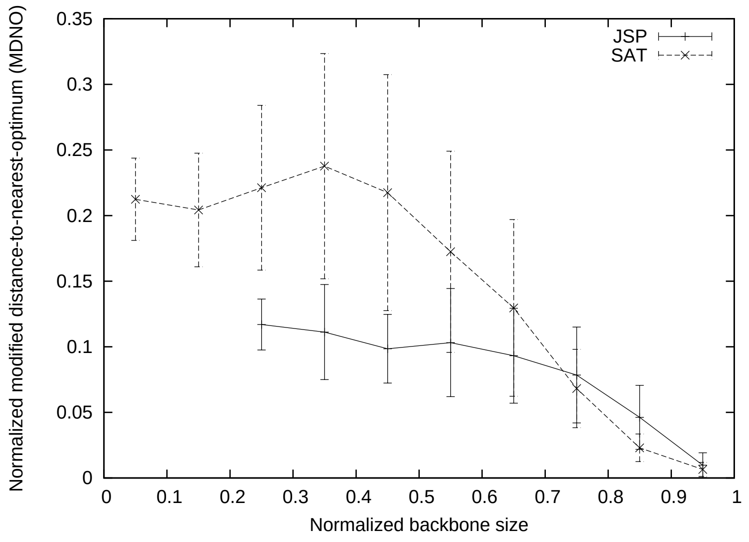


Figure 6.4: *Modified* distance-to-nearest ($mdno$) plots for satisfiable 3-SAT (100 variables, various number of clauses) and JSP (6×6). Error bars represent the variance in $mdno$ from instance to instance for the particular backbone size.

Zhang presented another way of calculating $p(l_j)$, in which, the quality of each local minimum (*i.e.* number of unsatisfied clauses) is taken into consideration since not all local minima are of equal quality. However, for our set of instances, the first method performed slightly better than the second method. Thus, only the first method is considered.

The second or *main phase* is where BG uses the pseudo backbone information to steer Walksat towards "fixing" tightly constrained variables (literals with high pseudo backbone frequencies) first. Zhang suggested many heuristics to bias Walksat's choices. There are four main ways BG algorithm can bias Walksat: *BG-ClausePick* biases the way Walksat chooses a clause to fix while *BG-NoisePick* and *BG-GreedyPick* bias the way Walksat chooses a variable to fix once a clause is selected. *BG-InitPick* influences the way the initial assignment is made. To understand how the four heuristics are integrated with Walksat, recall Walksat's architecture from Section 2.3.2. Walksat employs two stages for selecting a variable to flip:

1. A clause from the list of all unsatisfied clauses is picked at random; this is where the BG-ClausePick takes place (if used).

2.
   - From the selected clause, if there exists a variable with zero $break$ count[1], flip that variable.
   - If no such variable exists, with probability p, randomly pick any one of the literals from the selected clause; this is where the BG-NoisePick takes place. With probability $1 - p$, select the literal with the least $break$ count; this is where the BG-GreedyPick takes place (in case of a tie).

[1]$Break$ count is the number of clauses made unsatisfied from flipping a variable's value.

Depending on which of the four heuristics are selected, the heuristics will influence various portions of Walksat's search. Further, the backbone estimates can be used to bias the initial assignments when the algorithm re-starts. A more detailed description of the heuristics is as follows:

- *BG-InitPick*: generate an initial assignment based on the pseudo backbone frequencies. Specifically, a variable is assigned a particular value with a probability proportional to the corresponding literal's pseudo backbone frequency.

- *BG-ClausePick*: probabilistically pick a clause among all unsatisfied clauses based on their constrainedness, which corresponds to the sum of the literals' pseudo backbone frequencies in the clause. BG algorithm selects a clause C among all unsatisfied clauses in K with probability $p_C = q_C/Q$. Here, the q_C is the sum of the pseudo backbone frequencies of all the literals in clause $C \in K$, and $Q = \sum_{C \in K} q_C$ is a normalizing factor.

- *BG-NoisePick*: probabilistically pick a literal from selected clause based on its pseudo backbone frequency. The algorithm chooses literal l_j to flip with probability $\frac{(1-p(l_j))}{\sum_{i=1}^{w}(1-p(l_i))}$ given that the selected clause has w literals.

- *BG-GreedyPick*: probabilistically pick a literal from selected clause among all the literals that have the smallest $break$ count (*i.e.* a tie situation) based on its pseudo backbone frequency. As in BG-NoisePick, literal l_j is chosen with probability $\frac{(1-p(l_j))}{\sum_{i=1}^{w}(1-p(l_i))}$.

Any combination of the heuristics can be used for the main phase, where the Walksat with the biased moves will be run multiple times. The local minima found during the main phase can also be combined with the existing pool of local minima to update the pseudo backbone frequencies. BG Walksat achieves this by simply adding the new local minima to the existing pool and re-calculating all the pseudo backbone frequencies periodically.

When Zhang tested BG Walksat on random 3-SAT instances with c/v =4.3, 6.0, and 8.0, using BG-NoisePick on its own yielded the best results by improving the base Walksat for all the c/v ratios. Some of the other strategies actually worsened the Walksat's performance or improved it only for certain c/v ratios [68].

6.3.2 Perfect Backbone Experiment

Intuitively, if the performance of BG Walksat depends on the guidance from the pseudo backbones, the accuracy of pseudo backbone frequencies generated in the initial phase of BG Walksat algorithm must be one of the determining factors that affects the overall search cost in finding a satisfying solution. For example, perfect information about the true backbone of an instance should quickly guide Walksat to a satisfying solution since Walksat will be forced to make *biased* choices, correctly fixing backbone variables that do not match the given information. On the other hand, poor backbone information can have a worsening effect on Walksat since it will tend to mislead the algorithm *away* from a satisfying solution.

In this experiment, Zhang's algorithm will be given the *actual backbone information* (to a certain extent) from which it can bias the clause and variable selection. The aim here is to reveal whether the BG algorithm is actually exploiting the backbones of instances. More specifically, pseudo backbone frequencies for backbone variables are based on *prior knowledge* (pk) as follows:

$$p(l_j) = |b_j - (1 - pk)|$$

where, $0 \leq pk \leq 1$, and b_j is the correct backbone value found *a posteriori* for variable j. Naturally, the higher the prior knowledge, the closer the pseudo backbone estimate is to the real backbone value. For example, if a backbone variable has a prior backbone knowledge of $pk = 0.7$, $p(l_j)$ will be set to 0.7 if $b_j = true$ and 0.3 if $b_j = false$. Thus, the closer pk is to 1, the more prior knowledge is provided.

Further, in order to simulate different levels of randomness in the prior knowledge, instead of using a fixed value of pk, an upper and a lower bound are specified such that pk is selected uniformly between the two bounds for each variable. For example, with the bounds set to [1, 1], BG algorithm has a "perfect" information while [0, 1] bounds will have pk set anywhere between the 0 and 1 with uniform probability, representing complete randomness. Although all the backbone variables in a problem share the same upper and lower bounds, each backbone variable has its own pk.

Methods A mix of satisfiable and unsatisfiable 3-SAT instances were used. They all have 100 variables, and the number of clauses range from 430 to 600. The number of instances used is summarized in Table 6.3. On every instance, Walksat and variations of Backbone-Guided Walksat are run 100 times. Each run continues until either an optimal solution has been found or a maximum number of flips has been reached. For unsatisfiable instances, an optimal solution has the fewest unsatisfied clauses.[2] For Walksat, UBCSAT implementation with the noise value of 0.5 was used.

For the BG Walksat initialization phase, 15 local minima were collected from 15 independent Walksat runs of 300 steps. Based on the pool of local minima, pseudo backbone frequency for each non-backbone variable is calculated by the method described in Section 6.3.1. For backbone variables, their pseudo backbone frequencies are replaced with the prior backbone knowledge pk. This initial phase is then followed by the main phase with 4 runs of 25000 steps, for maximum of 104500 steps. *BG-NoisePick* was used as the heuristic that biases Walksat. In particular, this heuristic biases Walksat only when it chooses a variable to flip randomly. Out of all the heuristics suggested by Zhang, this is the best performing heuristic for these instances. Further, when BG-GreedyPick was used for the same set of instances, the effect of prior backbone knowledge was very similar.[3]

Results and Discussion Two key observations can be made from the results (see Figure 6.5 to 6.7). First, for satisfiable instances with $c/v = 4.3$ and $c/v = 4.71$ (Figure 6.5) BG Walksat does not perform much better than the [0, 1] case, where the prior knowledge can be anywhere from 0 to 1. This brings up the question of the accuracy of the pseudo backbone frequencies used in regular BG Walksat. Specifically, for $c/v = 4.3$ and $c/v = 4.71$ cases, random

[2]Optimal (fewest) number of unsatisfied clauses were found using *maxsat*[6].

[3]The results are not shown here.

c/v ratio	4.3	4.71	5.0	6.0
Satisfiable	44	76	-	-
Unsatisfiable	48	-	100	58

Table 6.3: Number of instances used for various constrainedness (all 100 variables).

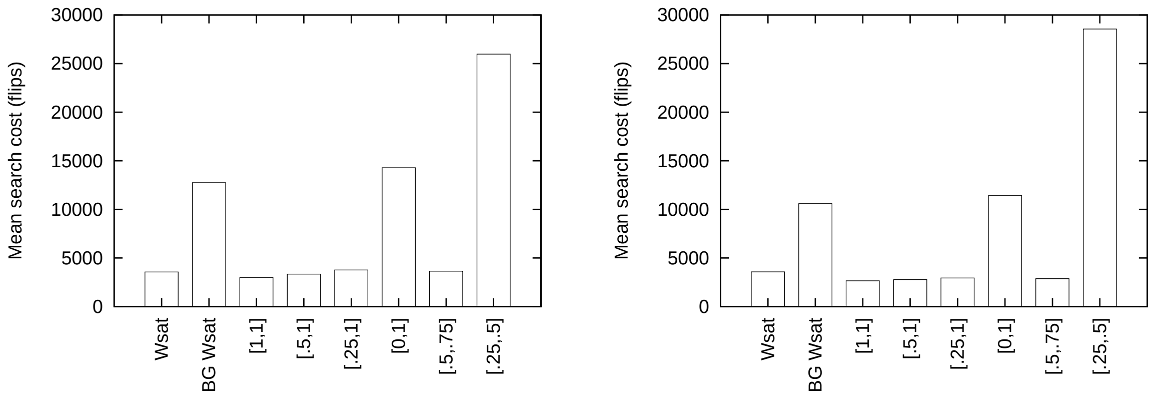


Figure 6.5: The mean search cost for various amounts of prior backbone knowledge for BG Walksat for satisfiable, 100 variable instances. 430 clauses (*left*) and 471 clauses (*right*). The numbers inside square brackets indicate lower and upper bound of the amount prior backbone knowledge.

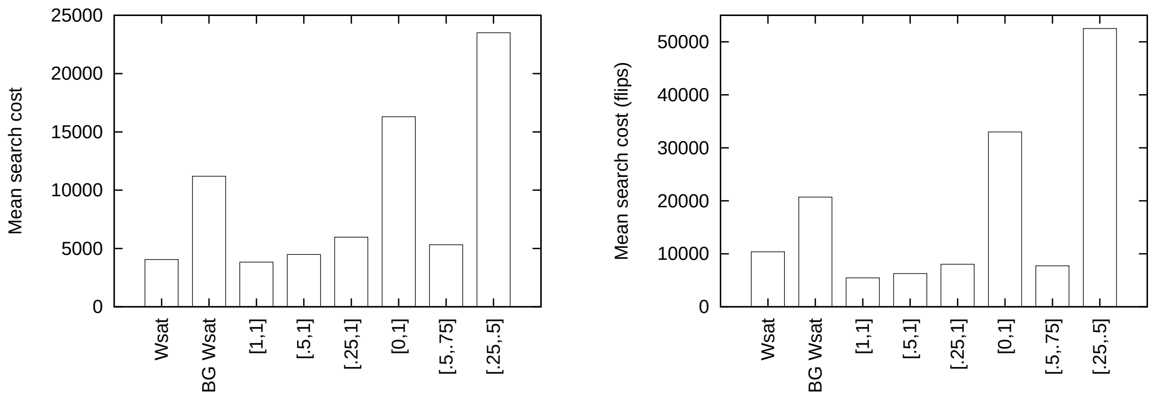


Figure 6.6: The mean search cost for various amounts of prior backbone knowledge for BG Walksat for unsatisfiable, 100 variable instances. 430 clauses (*left*) and 500 clauses (*right*). [pk lower bound, pk upper bound].

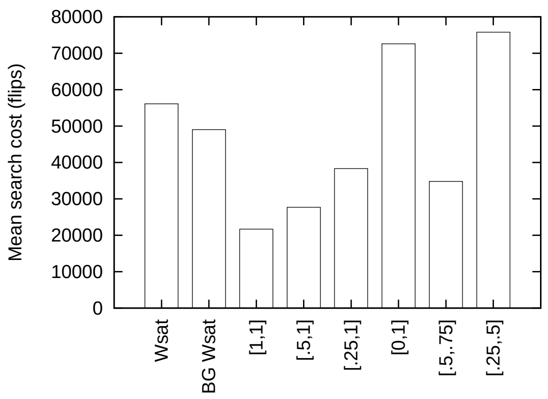


Figure 6.7: Mean search cost for various amounts of prior backbone knowledge for BG Walksat (unsatisfiable, 100 variables, 600 clauses). [pk lower bound, pk upper bound].

assignment of pseudo backbone frequencies is as good as the estimation method used in BG algorithm. The accuracy issue with the pseudo backbones is further discussed in Section 6.3.3. Secondly, as expected, prior knowledge of backbone variables affect the results significantly. When BG Walksat algorithm is given high prior backbone knowledge, it performs much better than when given low prior knowledge.

Surprisingly, the effect of lesser prior knowledge is not very high until the information really starts to degrade. For example, for all the satisfiable cases, perfect backbone information ([1, 1]) case has similar mean search cost as the [0.25, 1] case, where pseudo backbone frequencies can be as low as only 25% correct. However, the search cost drastically increases when the backbone information is further lowered to [0, 1]. Similarly, the difference in search cost is very dramatic for [0.5, 0.75] and [0.25, 0.5] case; the search cost for [0.5, 0.75] is very good (except for the case of 600 clauses, where it shows a slight drop in performance) while that for [0.25, 0.5] was exceptionally poor. These results indicate that the performance of BG Walksat is not appreciably affected by the range of lower bound as long as the average pk (average of upper and lower bounds) is kept above certain level. Once the pk reaches below some threshold, it seems like the performance degrades significantly.

Similar to the results shown in Zhang's paper [68], BG Walksat outperforms Walksat only in the most constrained region $c/v = 6.0$. In all other regions, BG actually hurts the performance of Walksat. More discussion of the BG's performance with respect to c/v ratio is given in Section 6.3.3.

Perfect Backbone Experiment with Fixed Prior Knowledge To investigate further into this idea of threshold value, fixed prior knowledge values are used this time instead of variable value between upper and lower bound. Figure 6.8 (*left*) shows that the search cost remains steady for $pk > 0.5$. But below this point, the search cost dramatically increases. In fact, the search cost for $pk < 0.5$ would have been larger if the maximum number of flips was greater than 104500 since BG Walksat could not find a satisfying solution in some cases.[4] In

[4]Naturally, the number of satisfying runs decreased with lower prior backbone knowledge.

retrospect, this result is intuitive since $pk > 0.5$ would lead the BG Walksat towards the correct value, where as $pk < 0.5$ would mislead the algorithm.

What is unexpected, however, is the fact that there is such a sharp transition in the search cost at around $pk = 0.5$ and that the performance of BG Walksat does not depend very much on the prior knowledge as long as pk is above 0.5. A similar observation of flat search cost with respect to prior knowledge can be made for $pk < 0.3$. This observation is valid for larger backbone instances and unsatisfiable instances as both Figure 6.8 (*right*) and 6.9 show similar "transition zone" around $pk = 0.5$. These results show that BG Walksat is quite robust with respect to the pseudo backbone estimates; as long as the estimation of the pseudo backbone frequencies are reasonable (*i.e.* $pk > 0.5$), it will perform better than Walksat, especially for the over-constrained instances (see Figure 6.6 *right* and 6.7).

6.3.3 Accuracy of Pseudo Backbones

As suggested in the previous section, the estimation of the pseudo backbone variables are critical to the performance of BG Walksat. More specifically, BG Walksat needs the backbone estimation to be $pk > 0.5$ such that the algorithm is guided towards the "right direction". According to Figure 6.5, BG Walksat does not perform much better than the [0, 1] case for satisfiable instances (in fact, BG Walksat is worse). However, for unsatisfiable instances, especially with high c/v ratio, BG Walksat's effectiveness increases, outperforming Walksat for $c/v = 6.0$ [68] (see Figure 6.6 and 6.7).

Given what we know about the BG Walksat's performance with respect to the prior backbone knowledge, by examining the pseudo backbones, we can find out whether it is the backbones that makes BG Walksat efficient in over-constrained problems or some other benefits from maintaining the pool of *elite solutions* during the search. More specifically, through this experiment, we want to find out how close pseudo backbone frequencies are compared to the actual backbone values by measuring the discrepancy between them for each backbone variable.

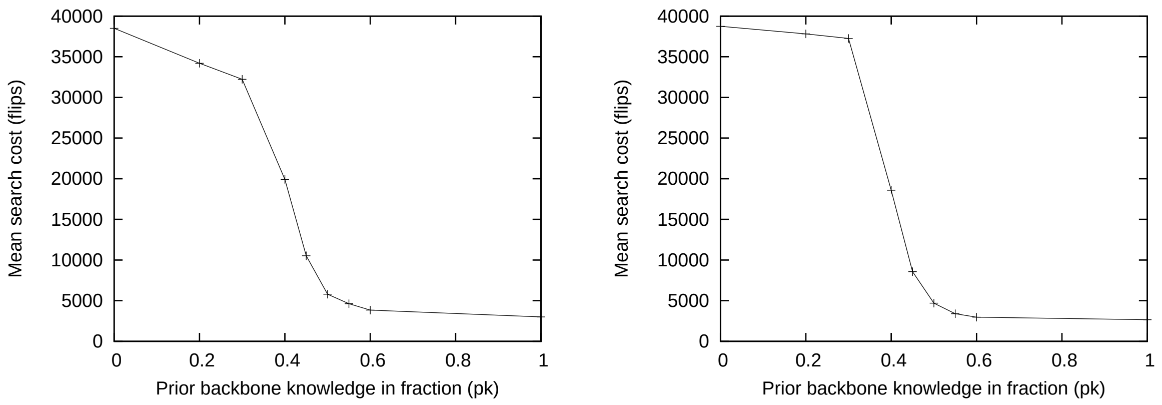


Figure 6.8: Mean search cost for varying prior backbone knowledge for BG Walksat for satisfiable, 100 variables instances. 430 clauses (*left*) and 471 clauses (*right*).

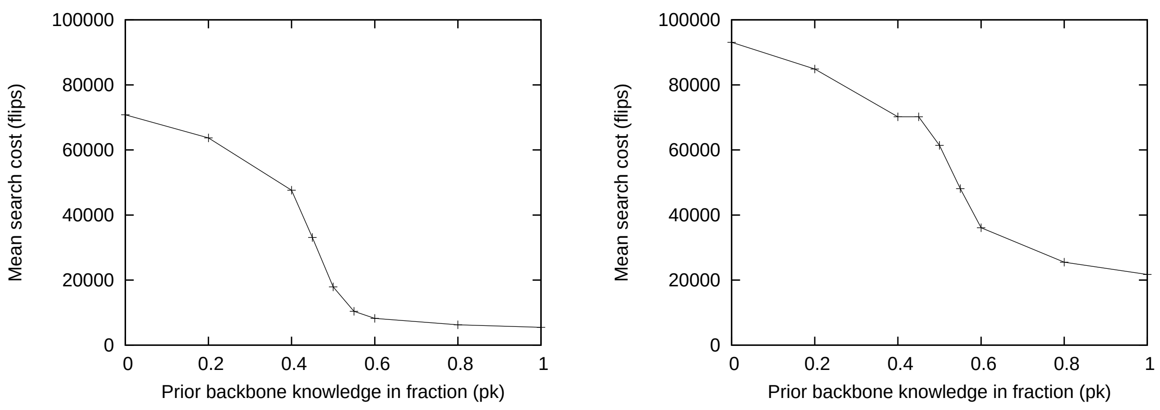


Figure 6.9: Mean search cost for varying prior backbone knowledge for BG Walksat for unsatisfiable, 100 variable instances. 500 clauses (*left*) and 600 clauses (*right*).

Methods Two sets of data were collected for this experiment: one is the actual backbone variables and their values, and the other is the pseudo backbone estimates using the method described in Section 6.3.1. To get the actual backbone information, we enumerated *all* the satisfying solutions using zChaff [47] for each instance. As for finding the pseudo backbones, for each instance, Walksat was run 150 times independently with each run having a maximum of 500 flips. From each run, the assignment with the lowest number of unsatisfied clauses is chosen,[5] and from this set of local minima, ***pseudo backbone frequencies*** $p(l_j)$ for each literal l_j are calculated.

As shown in the Perfect Backbone experiment in Section 6.3.2 (and also mentioned in Zhang [68]), BG Walksat performs well only when the problems are over-constrained. Thus, the instances tested here are concentrated in the over-constrained region. However, under- and critically-constrained instances are considered as well. Because not many satisfiable instances exist in the highly over-constrained problems, only the unsatisfiable instances (MAX-SAT) are considered. As usual, the number of variables is 100. The number of instances used for each c/v ratio is summarized in Table 6.4.

Once the two sets of data, namely the pseudo backbone estimates and the actual backbone results are aggregated, the two sets can be compared on a variable-by-variable basis for each instance to see how accurate the estimates of backbone variables are using Zhang's pseudo backbone method.

For each backbone variable $x_j \in B$, where B is the set of backbone variables in an instance,

[5]If a satisfying set of assignments is found in the run, it is discarded from the set of local minima.

c/v ratio	4.2	4.3	4.7	5.0	6.0
Number of instances	66	81	95	100	58

Table 6.4: Number of instances used for various constrainedness (all unsatisfiable, 100 variables).

the discrepancy d_j is the difference between the actual backbone value $b_j \in \{0,1\}$ and the pseudo backbone frequency $p(l_j)$[6]: $d_j = b_j - p(l_j)$, where $d_j = [0,1]$.

Naturally, the smaller the value of d_j, the more accurate the pseudo backbone estimate is for that variable, and vice versa. It should be noted that d_j corresponds to $1 - pk$, since for the prior knowledge pk, higher is better while the opposite is true for d_j. We expect to see better pseudo backbone estimates for highly constrained problems as BG Walksat performs significantly better in this region than low constrained problems [68].

Results and Discussion For each c/v ratio, the distribution of the backbone variables according to their discrepancy values d_j is plotted in Figure 6.10. For each c/v ratio, each backbone variable is separated into 5 different "bins" depending on its d_j value. For instance, $0 \leq d_j \leq 0.4$ is a very good estimation of the backbone value while $d_j > 0.7$ is a very poor one. The distribution of backbone variables is normalized by the $|backbone|$. As can be seen from the figure, most of the backbone estimates (over 70% of $|backbone|$) are fairly close to the actual backbone value $d_j < 0.4$. As seen from the Perfect Backbone experiment, prior knowledge $pk > 0.5$ will help BG search, and that is the case for $d_j < 0.5$.

The percentage of those backbone variables having discrepancy value greater than 0.5, thereby misleading the BG Walksat, is 15%, 14%, 15%, 18%, and 14% respectively for c/v ratio of 4.2, 4.3, 4.7, 5.0, and 6.0. In general, Figure 6.10 shows that for given a backbone variable, the probability of it having a "good" ($d_j < 0.5$) pseudo backbone estimate is fairly similar for various c/v ratio and is not necessarily dependent on the constrainedness. This is somewhat unexpected given how much the performance of BG Walksat varies with the constrainedness as well as the results from the extensiveness of local minima (see Section 5.3).

One reason for BG Walksat's superior performance in the over-constrained problems may simply come from the fact that the instances in this area have larger $|backbone|$ than those in lesser constrained region (see Table 6.5). Since most of the pseudo backbone variables are "good" estimates ($d_j < 0.5$), the greater $|backbone|$, the more information BG Walksat will have when deciding which variables to flip. Figure 6.11 illustrates this point. Instead of normalizing the discrepancy value by the $|backbone|$ as in Figure 6.10, it is normalized by the total number of variables, in order to show the distribution of the discrepancy in pseudo backbone information with respect to the entire set of variables.

As expected, the fraction of backbone variables having "good" estimates increases monotonically with respect to the c/v ratio. Similar monotonicity is observed for cases of poor estimates ($d_j > 0.5$) except for the slight drop in the discrepancy for the $c/v = 6.0$ case.

In general, the performance of BG Walksat is better in over-constrained region, *not* because

[6]$p(\neg l_j) = 1 - p(l_j)$.

c/v ratio	4.2	4.3	4.7	5.0	6.0
$\|backbone\|$	0.230	0.281	0.450	0.547	0.564

Table 6.5: Mean $|backbone|$ each number of clauses.

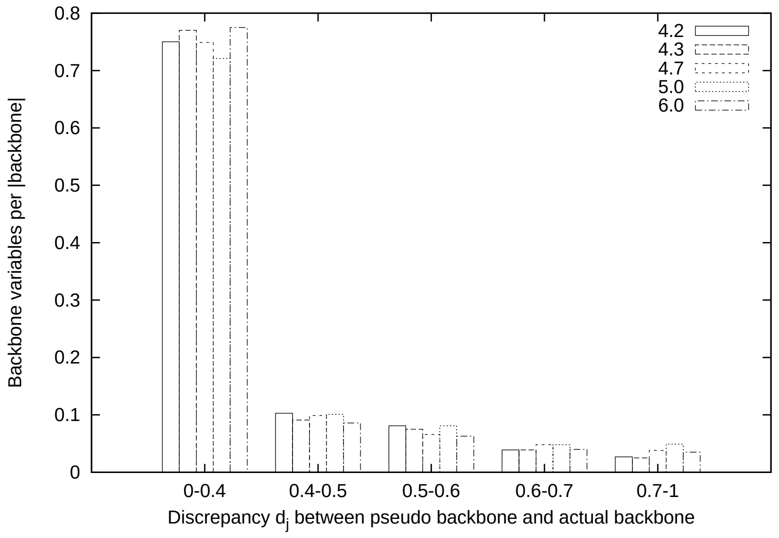


Figure 6.10: The distribution of the backbone variables according to their discrepancy d_j between the pseudo backbone estimates and the actual backbone values for various c/v ratio, when normalized by $|backbone|$.

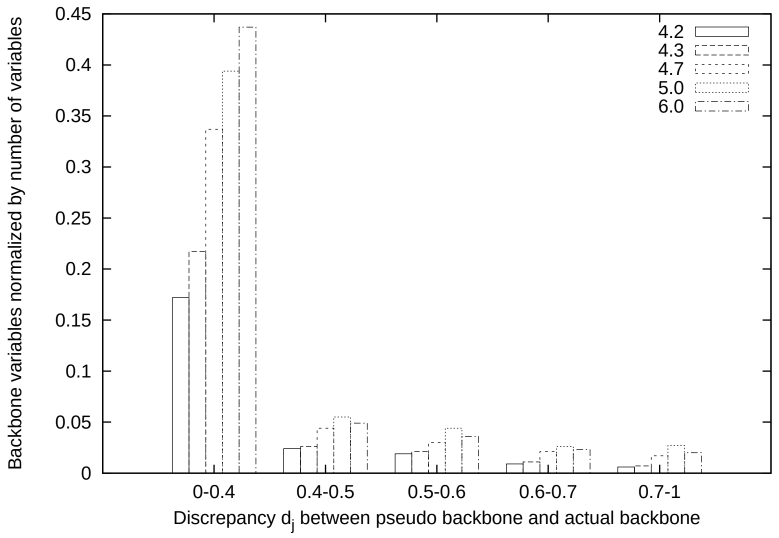


Figure 6.11: The distribution of the backbone variables according to their discrepancy d_j between the pseudo backbone estimates and the actual backbone values for various c/v ratio, when normalized by the number of variables.

it estimates the backbones better during the preliminary phase, but rather, due to the larger $|backbone|$ that takes advantage of "good" pseudo backbone estimates.

6.4 Conclusions and Future Work

In this chapter, the notion of backbone variables in problem instances was discussed as well as the general characteristics of backbones across different problem domains. In both the job-shop scheduling problems (JSP) and satisfiability (SAT) problems, the backbone size for an instance had a high correlation with the number of optimal solutions. Especially for JSP, Watson et al. [61] reported the negative correlation between $|backbone|^2$ and the number of optimal solutions to be exceptionally high at $|r| > 0.91$, and its magnitude was noticeably higher than the magnitude of the correlation coefficient we obtained for SAT at $|r| = 0.81$. We believe that the difference between the two r values in JSP and SAT is due to the smaller *degree of implication* for JSP than SAT. For JSP, from an optimal solution, if a non-backbone variable changes its value, it is still fairly close to the next closest optimal solution (not including the original solution), thus displaying smaller degree of implication. This allows JSP to produce a more consistent number of solutions per flip in non-backbone variable. On the other hand, for SAT instances, a change in the non-backbone assignment affects much greater search space due to its global influence and thus, the size of non-backbone (*i.e.* size of backbone) is not as good of a predictor for the number of optimal solutions as it is for JSP.

We also investigated how backbones work as a guiding tool for a search algorithm. There are numerous algorithms that claim to take advantage of backbones to guide the search to an optimal solution. Among these, Backbone-Guided local search (BG Walksat) by Zhang [68] was tested to see whether the algorithm really takes advantage of the backbone information. By supplying the BG Walksat with pre-determined quality of backbone information, we were able to establish that the information quality must be better than a certain threshold value to really take advantage of the given information. This threshold for the random 3-SAT instances is around 50%, that is, the backbone information must at least point the algorithm to the right "direction" when deciding between $true$ and $false$ for the variable assignment. When the information is not far from the actual backbone value, it can drastically reduce the performance of BG Walksat.

We also noted from the Pseudo Backbone experiment that the accuracy of pseudo backbone frequencies is generally very good ($d_j < 0.4$) for over 75% of the backbone variables. To our surprise, the accuracy of pseudo backbone frequencies did not show noticeable dependence on the constrainedness. However, as observed in our experiment as well as through Zhang's documented results [68], BG Walksat clearly performed much better in the over-constrained region than under- and critically-constrained region. We provided evidence that the reason for this is not because BG Walksat is necessarily better in estimating the pseudo backbone values for over-constrained problems, but rather, due to larger $|backbone|$ that allows the algorithm to take advantage of the "good" information found in the preliminary run.

Overall, both the Perfect Backbone experiment and the Pseudo Backbone experiment showed the importance of the set of local minima and backbone estimation to the performance of the Backbone-Guided search algorithm. It was to our surprise that the backbone estimation for a given variable only needs to be fairly good but not great (discrepancy with the actual backbone value less than 40%) in order to take advantage of the pseudo backbone information. These results suggest not only the usefulness of keeping a pool of local minima but also the robustness of the mechanism. It would be interesting to see how this holds for other problem domains or different algorithms such as genetic algorithms [26], ant colony optimization algorithms [34], and multi-point constructive search [3] that use the idea of maintaining a pool of solutions as a metaheuristic.

One of the factors that should be considered in the future work for the Perfect Backbone experiment is the role that the non-backbone variables play. Although non-backbone variables are not as constrained as backbone variables, in practice, they could be very highly constrained and affect the search cost appreciably. For example, if there is a variable that has an actual backbone value of 0.02 (*i.e.* $false$ in 98% of optimal solutions), setting the variable to $false$ will make the rest of the search significantly easier than if it were set to $true$. With most of the current research focused on backbone variables, this could reveal the importance of non-backbone variables in algorithm designs and search cost.

Also, most of the experiments done in this chapter can be repeated for constructive search algorithms that use backbone guidance. It would be interesting to see how accurate the backbone estimation needs to be in order to have an effective Backbone-Guided constructive search given its depth-first nature.

Chapter 7

Applying a Metaheuristic to Local Search Algorithms for SAT

In the previous chapter, we investigated the use of estimates of the backbone as a local search guidance tool. Backbone-Guided (BG) local search algorithm by Zhang [68] first collects a pool of local minima in its initiation phase using Walksat. Then from the pool of local minima, the algorithm estimates the pseudo backbone frequencies and uses that information to bias Walksat to “fix” the over-constrained variables first. One of the most important concepts in the BG local search is the use of the pool of local minima or *elite solutions*. The creation of this *elite set*, as well as its maintenance, update, and the utilization the available information in the elite set are critical to the performance of the BG algorithm. An elite set provides a mechanism for an algorithm to learn from the previous runs and enables *intensification* around “good” solutions from the past as well as the *diversification* away from the current search space. Also, elite solutions can serve as the basis from which new solutions can be created.

In general, there are two ways to make use of the elite solutions. One is an *aggregate* method, where all or a subset of the elite solutions are combined or aggregated such that the characteristics of the entire subset are captured as a whole. BG Walksat falls into this category as does Ant Colony Optimization [16]. Typically, the “average” or the most frequently appearing characteristics of the elite set provide some sort of information for the algorithm or a place in search space where the algorithm can start a new search.

The other way to utilize the elite solutions is by a *non-aggregate* or *instance-based* method, where individual solutions from the elite pool rather than the average population data form the basis for (a) new solution(s) or a starting point. Genetic algorithms are a well-known approach that uses this non-aggregate method. Here, an individual instance (likely a local minimum) of the elite pool can be used as the starting point of a new search effort, or a pair of local minima can be combined in various ways to extract features of the two solutions.

One of the disadvantages of the aggregate method is that it encourages the search algorithm to focus on the search space that is preferred by the elite solutions. While this is beneficial if the elite solutions are pointing to the right direction, if not, the search will likely be stuck in an unpromising region. And provided the evidence in Section 5.3 that a major portion of high-quality local minima is indeed very far from an optimal solution, the non-aggregate method may be better suited for SAT local search algorithms than the aggregate method. We believe that the non-aggregate method will allow the underlying search algorithm to explore a greater portion of search space than the aggregate method, thus making it less sensitive to the accuracy of elite solutions. Thus, we conjecture that a non-aggregate method will be especially beneficial in the under- and critically-constrained region, where the extensiveness of local minima is greater than in the over-constrained region.

In this chapter, we introduce a way to integrate a non-aggregate method to local search algorithms for SAT. In particular, *path relinking* introduced by Glover [23] is integrated with Walksat [55] and Novelty+ [44, 30], and their performance is compared to the underlying algorithms. The implementation of path relinking with Walksat/Novelty+ is motivated by Nowicki and Smutnicki's work with i-TSAB [49] algorithm. i-TSAB, based on a tabu search, is one of the best performing local search algorithms for job-shop scheduling problems, and path relinking is used as a metaheuristic [62]. To our knowledge, such a non-aggregate method has never been applied to local search algorithms for SAT.

7.1 Background Information

The way path relinking is used in Nowicki and Smutnicki's i-TSAB is briefly discussed here.

7.1.1 Path Relinking for Tabu Search

One of the best performing local search solvers for job-shop scheduling problems (JSP) is Nowicki and Smutnicki's [49] i-TSAB algorithm [62]. Based on a particular implementation of tabu search, i-TSAB employs sophisticated *move operators* as well as a mechanism for diversification through path relinking. Due to the highly "tuned" nature of the i-TSAB algorithm, Watson et al. [62] implemented a simpler version of i-TSAB called i-STS (Simplified Tabu Search), which facilitates understanding of the effects of diversification with marginal decrease in performance.

Here, we are interested in the way i-STS achieves diversification through path relinking. As mentioned in Section 2.4.1.2, path relinking generates combinations of a set of elite solutions by exploiting paths between and beyond these solutions [23]. New solutions along the paths can be a part of another set of reference solutions depending on specific designs. This is the basic approach that i-STS takes: using a bi-level search procedure, it uses its underlying algorithm, STS, to construct a set of elite solutions E in the preliminary phase. In the main phase, path relinking is used to combine individual solutions from E to form a new solution, where the underlying algorithm can start the search again.

The details of the algorithm for i-STS's path relinking step is similar to our implementation of WsatPR shown in Section 7.2 as we inherited most of the design from i-STS.

7.1.2 Walksat and Novelty+ Algorithms

As before, the UBCSAT version 1.0.0 by Tompkins [59] is used as the underlying Walksat and Novelty+ algorithms. Detailed descriptions of Walksat and Novelty+ are provided in Section 2.3.

In terms of the computational effort, path relinking requires little overhead. Thus, the run-time for the algorithms with path relinking is nearly identical to that of the underlying algorithms given the number of total flips is the same.

7.2 Architecture of Walksat with Path Relinking (WsatPR)

In this section, the architecture of the new algorithm, Walksat with path relinking (*WsatPR*) is discussed. Novelty+ with path relinking is constructed in the same way, and thus, it will not be individually described.

There are two phases to WsatPR: in the preliminary phase of the algorithm, a set of *elite solutions* (local minima) are found using regular Walksat and stored in the *elite pool*, much like Zhang's Backbone-Guided local search. There are a few differences in terms of the termination criteria, but for the most part, the initial phase is similar for the two algorithms. In the main phase, path relinking with Walksat or pure Walksat is performed using the elite solution(s) as a starting point. If a satisfying assignment is not found by the end of the run, the best solution found so far can replace either one of the two starting elite solutions. The main phase is repeated until a termination criterion (often the maximum cutoff) is met.

7.2.1 Elite Phase: Collecting Initial Elite Solutions

As in the standard Walksat procedure, we start with a random solution. If not all the clauses are satisfied, Walksat is run until it reaches a preset number of flips without finding a new best assignment (*i.e.* stagnation threshold st_1). The best assignment is inserted into the list of elite solutions (assignments) E. This procedure is repeated until N solutions are inserted into E.

We believe that setting a stagnation threshold allows for a more robust design of the algorithm than a maximum number of flips for terminating runs since a stagnation threshold will compensate for the difficulty of a given instance to a certain degree.

7.2.2 Main Phase: Path Relinking and Walksat

Once all the elite solutions are generated, path relinking and Walksat can be applied to these assignments. With probability $1-prp$, we perform the regular Walksat on a randomly selected solution from E, and with probability prp, we perform path relinking between two randomly chosen solutions from E, which is then immediately followed by a regular Walksat run. The main phase repeatedly performs the two procedures until a satisfying assignment is found or a maximum number of flips for a given instance (maximum cutoff) has been reached. The detailed description of the two procedures is as follows:

Regular Walksat Picked with probability $1 - prp$. Walksat is run on a randomly chosen assignment e_i from E until it reaches a stagnation threshold st_2 (or until a satisfying solution is found). At the end of the run, if the best solution found from this run e^* is better (fewer unsatisfied clauses) than e_i, e^* replaces e_i.

Path Relinking with Walksat Picked with probability prp. Two solutions, e_m and e_n, are chosen randomly from E. Then let X_d be a set of variables that have different values in e_m and e_n. Starting from the better one of the two solutions, flip a variable from X_d that will result in the greatest net gain in terms of the number of satisfied clauses

($make - break$[1] count). Repeatedly flip subsequent variables in X_d with the next best $make{-}break$ count until the new solution is half-way between the two original solutions e_m and e_n. That is, if the two solutions were k flips apart in Hamming distance in the beginning, from X_d, flip $k/2$ variables. Then from the new solution e_p, Walksat can be applied until it reaches a stagnation threshold st_2 (or until a satisfying solution is found), resulting in e^*. It should be noted that e^* can only come from those solutions visited ***after*** the path relinking stage. If e^* is better than the $random(e_m, e_n)$, which returns e_m or e_n with a uniform probability, e^* replaces that assignment in E.

The idea here is to find the right mix of diversification and intensification of the solutions in the elite set E. With path relinking, diversification is achieved by combining two solutions and guiding the algorithm across the search space between the two disparate regions. Intensification is provided by simply letting Walksat re-visit the past solution and start from that point. However, as will be discussed in Section 7.3, path relinking can actually hurt the diversity of an elite pool, and the terms "diversification" and "intensification" may be misnomers [60].

Figure 7.1 and 7.2 show the architecture of Walksat with path relinking.

7.3 Effects of Parameters

Contrary to the regular Walksat algorithm, which requires only one parameter, WsatPR requires additional parameters for its added features: namely, the size of the elite pool $|E|$, stagnation thresholds st_1 and st_2, and the path relinking probability prp. The following set of experiments show how we arrived at parameter settings that we used for Section 7.4. These experiments further show insights and characteristics from which we glean knowledge of the new algorithm. The Walksat noise parameter p, which determines the probability with which Walksat will flip a random variable instead of making a greedy choice, is set to 0.5 for *all* the runs.

7.3.1 Stagnation Threshold for Walksat

In the WsatPR algorithm, Walksat plays an instrumental part in generating the elite pool and ultimately driving WsatPR towards an optimal solution. Thus, before looking at stagnation thresholds for WsatPR, we investigate the effect of placing a stagnation threshold for Walksat. Typically, Walksat is run without any stagnation threshold due to its ability to easily escape local minima. However, since we impose stagnation thresholds for Walksat within WsatPR, we ran the regular Walksat with a threshold to learn its behaviour with respect to the threshold. When Walksat is run with a stagnation threshold, it restarts with a random solution whenever it fails to find the new best solution within the threshold. This is repeated until the total number of flips reaches the maximum cutoff value.

We used 100 satisfiable, uniform 3-SAT instances with 250 variables and 1065 clauses ($c/v = 4.26$) from the Satisfiability Library [28]. Walksat was run 10 times per instance for each stagnation threshold, and the maximum cutoff value for each run was set to 1 million

[1]$make$ is the number of clauses made satisfied from flipping a variable's value and $break$ is the number of clauses made unsatisfied from the flip.

```
# Elite phase
while (! $E$ full)
    while (! reached stagnation threshold $st_1$)
        Flip a variable according to Walksat heuristic
        if satisfying solution found return ‘satisfiable’
    end
    $E := E \cup e^*$ # where $e^*$ is best solution found in most recent run
end
# Main phase
while (! reached maximum number of flips)
    if ( probability($prp$) )
        # Path relinking
        Randomly select two solutions $e_m, e_n \in E$
        $e_p$ := pathRelinking$(e_m, e_n)$
        while (! reached stagnation threshold $st_2$)
            Starting from $e_p$, flip a variable according to Walksat heuristic
            if satisfying solution found return ‘satisfiable’
        end
        if $e^*$ is better than $e_j := random(e_m, e_n)$
            $E := E \setminus e_j \cup e^*$
    else
        # Regular Walksat
        while (! reached stagnation threshold $st_2$)
            Starting from $e_i \in E$, flip a variable according to Walksat heuristic
            if satisfying solution found return ‘satisfiable’
        end
        if $e^*$ is better than $e_i$
            $E := E \setminus e_i \cup e^*$
    end
end
```

Figure 7.1: The architecture of Walksat with path relinking (WsatPR). See Figure 7.2 for the *pathRelinking* procedure.

```
**pathRelinking** $(e_m, e_n)$
**Input:** two solutions $e_m, e_n$
**Output:** a new solution $e_p$
$X_d :=$ a set of variables with different values in $e_m$ and $e_n$
$k := |X_d|$
$e_p :=$ better of $e_m$ and $e_n$
**for** (from $i := 1$; until $i = k/2$; increment $i$ by 1)
    $x_d :=$ variable from $X_d$ with the largest $make - break$
    $e_p := e_p$ with $x_d$ flipped
    $X_d := X_d \setminus x_d$
**end**
**return** $e_p$
```

Figure 7.2: Path relinking procedure.

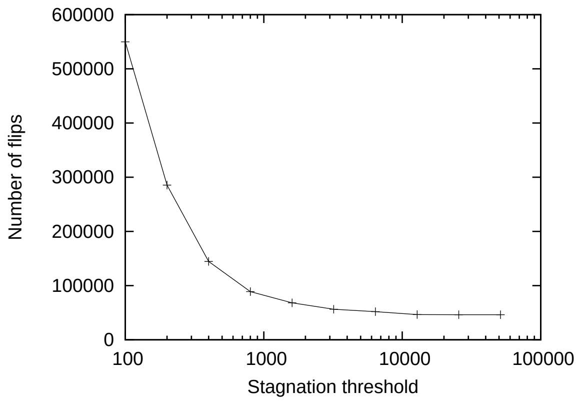


Figure 7.3: Mean search cost for varying stagnation threshold for Walksat.

flips. For each stagnation threshold value, the search cost was averaged across all the runs and all the instances. Figure 7.3 shows the total number of flips required to reach a satisfying solution when the stagnation threshold is varied for Walksat. For the lower-end of the stagnation threshold range, the number of flips required to find a satisfying solution goes down sharply with respect to the stagnation threshold. However, a plateau is reached when the stagnation threshold is at approximately 10000. The plateau verifies Walksat's ability to escape local minima quickly and shows that too frequent restarts can hurt its performance.

7.3.2 Parameters for WsatPR

Here, we show WsatPR's performance with respect to its stagnation thresholds. Again, the duration of each Walksat's run within WsatPR is determined by the stagnation threshold st_1 and st_2 respectively for the preliminary and main phase.

The 21 instances used for the next set of experiments for WsatPR parameter setting is from SAT-03 competition [28]. These are the same set of instances used for Table 7.3 (on page 91). The number of variables vary from 38 to 700 and the c/v ratios range from 4.2 to 720, and all instances are satisfiable. Each instance was run 5 times independently with the maximum number of flips set to 1 billion. Every instance here is considered very difficult; Walksat could not find a satisfying solution for a few instances within the cutoff limit. The number of runs per instance is set to a relatively small number due to the computational cost.

The goal here is not necessarily to find exact parameter settings to be used for Section 7.4, where we compare WsatPR's performance with other algorithms. Rather, the aim is to get a general idea as to how WsatPR reacts to various changes in the settings.

Preliminary Phase Stagnation Threshold At the elite solution collection stage, it is critical to let Walksat run long enough so that it has time to incrementally improve upon the current best solution for a given run and arrive at a high-quality local minimum. At the same time, if Walksat runs too long without improving, it will leave us with less time at the main phase. The key is in striking the right balance between the quality of the elite solutions and the effort required to achieve such quality.

For this experiment, we set all other parameters other than st_1 to fixed values. The elite pool size $|E| = 8$ comes from the i-TSAB [49], and st_2 =10 million and $prp = 0.5$ are determined from our experience.

As Figure 7.4 (*left*) shows, neither the search cost nor the success rate (percentage of runs finding a satisfying solution) fluctuates very much with respect to the threshold (with the exception of the spike in the search cost at $st_1 = 400$). We believe that this is from the combination of the two effects: first, Walksat was able to find high-quality elite solutions with relative ease. Second, even when Walksat was not able to find high-quality solutions initially, with the updates of elite pool in the main phase, the overall quality improved quickly.

Both the success rate and the cost steadily deteriorates (*i.e.* cost getting larger) with $st_1 > 2500$. This is due to the fact that the quality of elite solutions did not improve very much after this point for most instances, and the increased time spent in the preliminary run ended up reducing the time remaining for the main phase.

Even though $st_1 = 1600$ performed the best on average, we conservatively choose to use

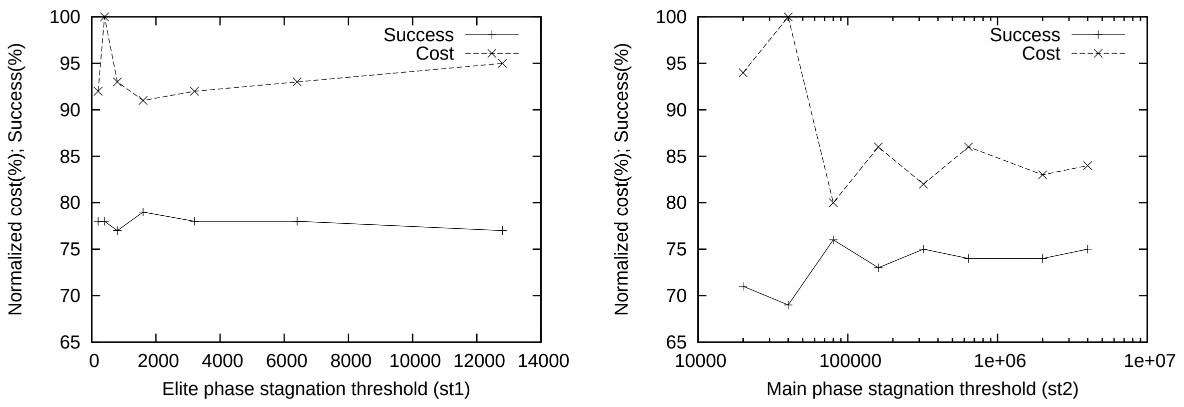


Figure 7.4: Normalized search cost and success rate (%) for varying elite phase stagnation threshold st_1 (*left*); for varying main phase stagnation threshold st_2 (*right*).

$st_2 = 10000$ from here on, because there are a few instances that are exceptionally difficult and benefit from increased time spent in the elite phase.

Main Phase Stagnation Threshold The main stagnation threshold is similar to the elite phase threshold in that it needs to recognize when the search is not improving and in such case, allow the underlying algorithm to restart from a different area in search space. At the same time, the algorithm must be able freely explore the search space without interruption.

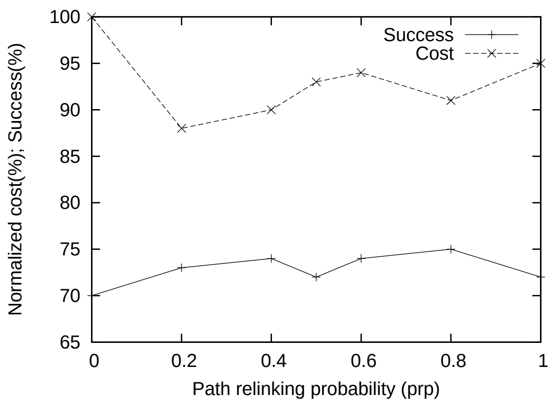
Figure 7.4 (*right*) shows that the search cost decreases sharply around $st_2 = 80000$ and remains relatively constant for $st_2 > 80000$. The same type of pattern can be seen for the success rate. To our surprise, both figures in Figure 7.4 show a search cost peak with small threshold values $st_1 = 400$ and $st_2 = 40000$. At this point, it is not clear as to why this is the case.

From this point and on, we set $st_2 = 70000$. The reason for choosing a relatively small st_2 is to maximize the effect of path relinking without sacrificing Walksat's ability to explore.

Path Relinking Probability With Path Relinking probability prp, the algorithm performs path relinking on two randomly selected elite solutions, and with probability $1 - prp$, it performs Walksat on an elite solution.

With the other parameters fixed to $|E| = 8$, $st_1 = 10000$, and $st_2 = 70000$, the path relinking probability prp was varied from 0 to 1. As Figure 7.5 (*left*) shows, the average search cost was the smallest with $prp = 0.2$. On the other hand, the success rate was the best when $prp = 0.8$. Overall, the WsatPR performed quite consistently when $0.2 \leq prp \leq 0.8$. It is interesting to note that all ranges with $prp > 0$ performed better than $prp = 0$, which shows that path relinking indeed improves Walksat's performance on average. However, it is important to note that $prp = 0$ does not exactly represent the pure Walksat case since pure Walksat does not have to spend any time collecting elite solutions. We choose to use $prp = 0.5$ for the rest of the experiment from here on.

Elite Pool Size Elite pool size $|E|$ determines how many elite solutions will be collected and maintained for WsatPR. The greater the size, the more diverse the set of elite solutions will



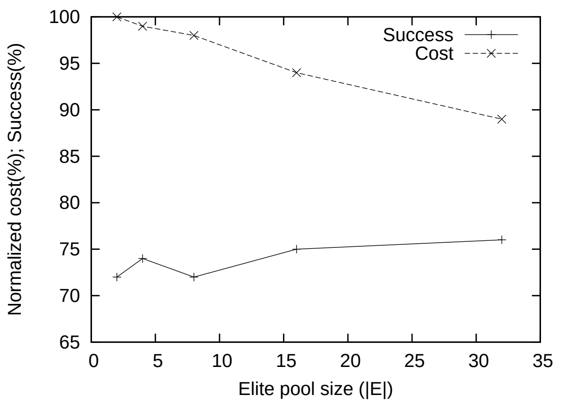


Figure 7.5: Normalized search cost and success rate (%) for varying path relinking probability prp (*left*); for varying elite pool size $|E|$ (*right*).

be, thus allowing for larger area of search space to be explored. On the other hand, with a smaller elite pool size, the quality of the elite pool will improve more quickly during the main phase since fewer solutions need to be updated. As $|E|$ gets even smaller, the path relinking algorithm will behave more like regular Walksat with restarts because there is very little path relinking to be done among a small number of elite solutions.

Here, we used $st_1 = 10000$, $st_2 = 70000$, and $prp = 0.5$, and varied $|E|$ from 2 to 32. As Figure 7.5 (*right*) shows, both the success rate and the search cost improves monotonically with respect to the pool size (with the exception of the slight decrease in success rate at $|E| = 8$). We believe the poor performance of WsatPR with small $|E|$ is due to the degradation of diversity in the elite pool, which is a result of two effects that feed off of each other: first, especially with more difficult problems, excessive amount of path relinking between a small group of solutions quickly degrades the diversity of the pool because the way path relinking keeps the common traits of the starting solutions and flips only those variables with different values. Secondly, once all the solutions in the elite pool reach a certain quality, the updates of the pool seldom happen, because the best solution found during the run e^* replaces a starting solution only if it is better. Since the diversity of the pool has degraded, the elite pool updates are less frequent; since there are fewer updates, the elite pool gets more stale and the diversity does not improve.

We conjecture $|E|$ to be a function of min(maximum cutoff for instance, average search cost for instance). The larger the cutoff value (or the more difficult the problem), the larger $|E|$ should be in order to allow greater diversity in the pool while allowing for frequent path relinking. With smaller min(cutoff, search cost), large $|E|$ hurts the performance, because it requires greater portion of the time to be spent on collecting the elite solutions, and also, the improvement in the overall quality of the elite pool is slower. Thus for Section 7.4, where the maximum cutoff value was set to 10 million (instead of 1 billion), we found that a relatively small $|E| = 8$ works well for most of the instances (also, i-STS used $|E| = 8$). When we tested $|E| \gg 8$ with the maximum cutoff set to 10 million on a few instances, there was a clear degradation of performance compared to $|E| = 8$.

Additional Factors In addition to the parameters above, there are a few other adjustable factors for WsatPR. One of the more influential factors for the pool diversity is the replacement strategy for e_m and e_n by the best solution found during the main phase (e^*). In the current implementation, e^* must be better (fewer unsatisfied clauses) than $random(e_m, e_n)$ (see Figure 7.2). While this protects the elite pool from accepting a low-quality solution, it is susceptible to stagnation in the elite pool from infrequent updates. However, on average, we found that having this quality barrier helpful. One way to increase the frequency of pool updates is by lowering the quality barrier such that e^* needs only be better than the worst solution in E. Also, another way could be accepting e^* with some probability. We have not attempted these two variations for WsatPR.

We found that using the *adaptive noise strategy* by Hoos [32] improves the performance of WsatPR on average. Here, the probability p, with which Walksat makes a random flip instead of using its greedy heuristic, is dynamically varied throughout the search such that the algorithm is more deterministic in the beginning where the improvements in solutions are significant and frequent. When the algorithm struggles to improve upon the best known solution so far, it becomes more stochastic in order to become more mobile and be able to cover greater area in the search space.

7.4 Results and Discussion

In this section, we verify the validity of the conjecture made earlier in the chapter and compare the performance of path relinking algorithms to their underlying algorithms.

7.4.1 Conjecture Verification

One of the motivations for implementing path relinking on the existing local search algorithms for SAT is to take advantage of its perceived ability to explore a wide range of search space. Whereas the aggregate method is susceptible to being trapped in one part of search space due to its dependence on elite solutions, the non-aggregate method should allow greater mobility of the search algorithm. This follows from the fact that the latter method exploits the diversity in the traits of individual instances rather than the average trait of the elite pool. Thus, we conjecture that in the under- and critically-constrained region, where the extensiveness of local minima is significant, the non-aggregate method will outperform the aggregate method and vice versa for the over-constrained region.

Methods In order to verify our conjecture, we compared the performance of an aggregate method and a non-aggregate method against their underlying algorithm for varying c/v ratio. Zhang's BG Walksat represents the aggregate method, while WsatPR represents the non-aggregate method. Walksat is the underlying algorithm for both methods.

For all three algorithms, adaptive noise was used, which is denoted by a "*A" suffix on the algorithm's name. The cutoff value for the maximum number of flips was set to 1 million. WsatPR*A was run with $|E| = 8$, $st_1 = 3000$, $st_2 = 21000$, and $prp = 0.5$. BG Walksat*A was run with the same parameters as described in Zhang's paper [68], namely $|E| = 30$, 10000 flips in the preliminary phase, and 100000 flips in the main phase with 7 re-starts (which adds up to 1 million flips). Here, BG Walksat*A was run only with the BG-NoisePick as the addition of both BG-GreedyPick and BG-InitPick worsened BG Walksat*A's performance for $c/v < 8.0$. Finally, Walksat*A was run without any re-starts.

The test instances are random 3-SAT with 1000 variables. For each number of clauses 4000, 4200, 4300, 5000, 6000, and 8000, 100 instances were used, which may be satisfiable or unsatisfiable. The phase transition region for 1000 variables seemed to be between $c/v = 4.2$ and $c/v = 4.3$. Each algorithm was run 20 times on each instance, and the average best number of unsatisfied clauses was reported for each c/v ratio.

Cost Difference Figure 7.6 shows the difference in the average solution quality (measured by the best number of unsatisfied clauses) of the two metaheuristics when compared to the underlying algorithm Walksat*A. The results do not support our conjecture that the non-aggregate method (WsatPR*A) will outperform the aggregate method in the under- and critically-constrained region and vice-versa for the over-constrained region. While WsatPR*A is indeed better than BG Walksat*A in the under- and critically-constrained region, the performance gap actually increases significantly as c/v increases. This is surprising given what we were led to believe about to the extensiveness of local minima and the characteristics of the aggregate and the non-aggregate methods.

In terms of the BG Walksat*A's inability to improve Walksat*A's performance to any appreciable degree, with more "tuning" of parameters, we may able to improve the performance of BG Walksat*A. Zhang's results on 2000-variable instances showed BG Walksat*A had about 30 fewer unsatisfied clauses than Walksat*A at $c/v = 8.0$. However, this is still inferior to the cost difference of 43 by WsatPR*A, which is achieved on smaller, 1000-variable instances.

7.4.2 Performance Comparison

Instances from various problem domains are used to test and compare the performance of the base algorithms (namely Walksat and Novelty+) to their variants including path relinking and adaptive noise strategy. Algorithms with path relinking are denoted by a "*PR" suffix on the algorithm's name and those with adaptive noise are denoted by "*A". Also, the results from the Backbone-Guided Walksat (BG) by Zhang [68] are included. With the results from Section 7.3 in mind, as well as from our past experience, $|E| = 8$, $st_1 = 10000$, $st_2 = 70000$, and $prp = 0.5$ are used for all the instances tested in this section. The cutoff for the maximum number of flips for any given instance is set to 10 million.

The instances used in this section are identical to the ones used in Section 4.2. As before, the results are broken down into three different tables: Table 7.1 corresponds to the "easier" set of instances from Zhang's work (Table 3 in Zhang [68]). Table 7.2 corresponds to the "harder" set of instances from Zhang's work (Table 4 in Zhang), where the regular Walksat could not solve any of the instances within 20 tries. Table 7.3 is a set of random k-SAT instances from the SAT-03 competition [28]. Readers should note that only Table 7.1 shows the results for the base algorithms since the algorithms with adaptive noise strategy outperformed the base algorithms on most of the instances.

For the instances in Table 7.1, which shows the number of satisfying solutions found out of 20 trials, path relinking clearly improved the performance of Walksat for both the fixed noise value and the adaptive noise strategy case. For both cases, the average number of satisfied

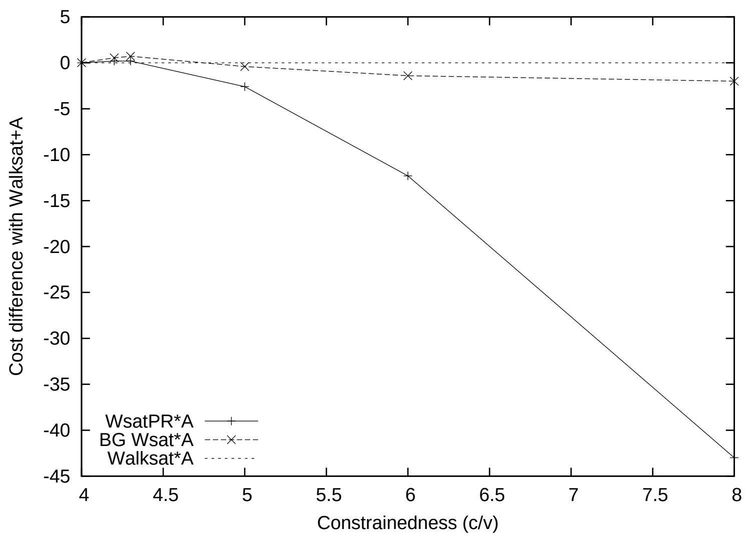


Figure 7.6: Difference in the average solution quality of WsatPR*A and BG Walksat*A compared to Walksat*A. The solution quality is measured by the number of unsatisfied clauses.

			Walksat				Novelty+				BG
Instance	$\#vars$	c/v	Base	Base+PR	Base+A	Base+PR+A	Base	Base+PR	Base+A	Base+PR+A	Base+A
bw_large.c	3016	16.7	0	0	3	8*	0	1*	**12**	9	0
bw_large.d	6325	20.9	0	0	0	10*	0	0	2	**14***	0
par8-1	350	3.3	5	7*	10	11*	**20**	**20**	**20**	**20**	19
par8-2	350	3.3	4	1	7	10*	**20**	**20**	**20**	**20**	18
par8-3	350	3.3	1	2*	3	11*	**20**	**20**	**20**	**20**	19
par8-4	350	3.3	0	1*	2	5*	**20**	**20**	**20**	**20**	13
par8-5	350	3.3	0	2*	0	2*	**20**	**20**	**20**	**20**	9
qg1-08	512	290.9	0	0	7	**13***	0	0	11	12*	12
qg2-08	512	290.9	0	0	1	**6***	0	0	1	3*	**6**
qg3-08	512	20.4	6	16*	13	13	**20**	**20**	17	19*	18
qg6-09	729	30.0	0	0	0	0	**2**	1	0	1*	0
qg7-09	729	30.3	0	0	5	**12***	3	1	7	10*	8
g125.17	2125	31.2	0	0	4	7*	4	12*	**20**	**20**	2
g250.29	7250	62.7	0	0	7	7	1	4*	**20**	0	4
Average			1.14	2.07*	4.43	8.21*	9.29	9.93*	**13.57**	13.43	9.14
Number of best results			0	0	0	3	7	6	8	7	1

Table 7.1: An "easier" set of instances from Zhang's paper [68]. It shows the number of times a satisfying solution was found out of 20 runs. +PR refers to path relinking and +A refers to adaptive noise strategy added to the base algorithm. The best result for each instance is in **bold**. The results that are improved by path relinking are denoted by *.

			Walksat		Novelty+		BG Wsat
Instance	$\#vars$	c/v	Base+A	Base+PR+A	Base+A	Base+PR+A	Base+A
bmc-ibm-1	9685	5.8	23.3	8.4*	20.6	23.7	**4.7**
bmc-ibm-2	3628	4.0	5.2	**1.0***	**1.0**	**1.0**	1.1
bmc-ibm-3	14930	4.8	110.0	102.3*	82.0	125.0	**30.5**
bmc-ibm-4	28161	5.0	106.6	135.7	118.7	260.6	**50.6**
bmc-ibm-5	9396	4.4	10.6	**1.0***	11.7	23.0	1.4
bmc-ibm-6	51654	7.1	314.9	504.9	190.8	602.0	**135.3**
bmc-ibm-7	8710	4.6	17.7	**4.8***	20.9	37.2	7.9
bmc-galileo-8	58074	5.1	75.3	20.4*	663.3	1136.0	**15.8**
bmc-galileo-9	63624	5.1	73.6	24.7*	777.2	1313.0	**17.8**
bmc-ibm-10	61088	5.5	380.5	522.2	233.6	603.4	**166.0**
bmc-ibm-11	32109	4.7	424.8	505.6	**240.2**	496.2	370.3
bmc-ibm-12	39598	4.9	531.0	598.6	**295.1**	561.4	460.7
bmc-ibm-13	13215	5.0	81.6	88.8	69.2	98.6	**3.2**
f2000	2000	4.3	1.6	0.9*	**0.4**	1.1	1.9
par-16-1-c	317	4.0	6.4	4.8*	**0.9**	1.1	5.6
par-16-1	1015	3.3	11.2	9.6*	2.4	**2.1***	11.2
par-16-2-c	349	4.0	6.9	5.3*	2.3	**1.8***	6.9
par-16-2	1015	3.3	11.4	9.9*	2.3	**2.0***	11.8
par-16-3-c	334	4.0	6.8	5.1*	**1.8**	**1.8**	6.3
par-16-3	1015	3.3	12.2	9.7*	2.9	**2.3***	11.2
par-16-4-c	324	4.0	6.2	4.8*	**1.6**	1.7	6.0
par-16-4	1015	3.3	12.2	9.9*	2.7	**2.1***	11.2
par-16-5-c	341	4.0	7.0	5.3*	**2.1**	2.2	6.5
par-16-5	1015	3.3	11.0	10.4*	2.5	**2.2***	11.3
par-32-1-c	1315	4.0	23.6	20.1*	13.2	**10.7***	22.5
par-32-1	3176	3.2	33.6	31.6*	12.3	**10.1***	30.9
par-32-2-c	1303	4.0	22.1	20.6*	11.4	**9.8***	22.6
par-32-2	3176	3.2	34.1	31.3*	12.0	**9.4***	31.5
par-32-3-c	1325	4.0	23.1	20.3*	12.1	**10.0***	23.1
par-32-3	3176	3.2	33.2	30.2*	12.1	**9.6***	30.5
par-32-4-c	1333	4.0	23.7	20.7*	12.4	**10.4***	23.0
par-32-4	3176	3.2	32.5	31.1*	12.9	**9.7***	31.0
par-32-5-c	1339	4.0	23.1	20.5*	12.7	**10.6***	23.6
par-32-5	3176	3.3	33.5	31.6*	12.7	**9.9***	31.3
Average			74.4	83.9	84.4	158.9	**47.8**
Number of best results			0	3	8	18	8

Table 7.2: A “harder” set of instances from Zhang’s paper [68]. It shows the mean number of unsatisfied clauses found out of 20 runs. +PR refers to path relinking and +A refers to adaptive noise strategy added to the base algorithm. The best result for each instance is in **bold**. The results that are improved by path relinking are denoted by *.

			Walksat		Novelty+		BG Wsat
Instance	$\#vars$	c/v	Base+A	Base+PR+A	Base+A	Base+PR+A	Base+A
gen-k10-r720-1719	38	720.0	**20**	**20**	**20**	**20**	19
gen-k5-v131-1749	131	21.5	**20**	**20**	**20**	**20**	18
gen-k5-v170-1758	170	21.5	17	**20***	**20**	**20**	18
gen-k9-v46-1808	46	357.0	**20**	**20**	**20**	**20**	**20**
gen-k9-v46-1810	46	357.0	**20**	**20**	**20**	**20**	**20**
hgen6-n520-815	520	4.2	4	**5***	4	**5***	1
hgen6-n520-816	520	4.2	**1**	0	**1**	0	**1**
hgen6-n520-817	520	4.2	4	12*	15	**19***	2
hgen6-n650-822	650	4.2	0	0	0	0	0
hgen7-n520-866	520	4.4	10	4	**14**	10	3
hgen7-n520-867	520	4.4	**9**	5	5	6*	6
hgen7-n650-870	650	4.4	18	11	**20**	18	7
hgen7-n650-871	650	4.4	6	5	7	**10***	1
hgen7-n650-872	650	4.4	**1**	0	0	**1***	0
uf600-r4.25-1116	600	4.3	15	**16***	15	14	6
uf700-r4.25-1120	700	4.3	**20**	**20**	**20**	**20**	**20**
uf700-r4.25-1121	700	4.3	0	0	0	0	0
uf700-r4.25-1122	700	4.3	17	**19***	15	14	8
uf700-r4.5-1135	700	4.5	0	0	0	0	0
uf700-r4.5-1136	700	4.5	0	0	0	0	0
uf700-r4.5-1137	700	4.5	0	0	0	0	0
Average			9.62	9.38	10.29	**10.33**	7.14
Number of best results			7	9	9	10	4

Table 7.3: Random k-SAT instances from SATLIB [28]. It shows the number of times a satisfying solution was found out of 20 runs. +PR refers to path relinking and +A refers to adaptive noise strategy added to the base algorithm. The best result for each instance is in **bold**. The results that are improved by path relinking are denoted by *.

answers found doubled by employing path relinking. For Novelty+, the improvement is not as dramatic. In fact, for Novelty+*A, use of path relinking resulted in slightly lower average number of satisfied answers. Comparing by instances, while Walksat*A could not solve a logistics instance `bw_large.d`, adding path relinking enabled it to find a satisfying solution 10 out of 20 attempts. A similar remarkable improvement is seen for Novelty+*A increasing the number of satisfying solutions found from 2 to 14.

However, for a graph colouring problem `g250.29`, path relinking was detrimental to Novelty+*A's performance. Novelty+*A found a satisfying solution in all 20 attempts whereas Novelty+*PR*A could not find one at all for any of its 20 attempts. This is surprising in that path relinking improved the performance for Novelty+ algorithm for both `g125.17` and `g250.29`. We believe that this is due to a stagnation threshold that is too small for the instance. As will be further noticed in Table 7.2 and 7.3, there are instances that are extremely large and hard for Walksat that there is a significant period of status quo between improvements in the best number of unsatisfied clauses. For these instances, path relinking actually interrupts the Walksat's run, preventing the potential improvements. However, for majority of the instances in Table 7.1, which both Walksat(*A) and Novelty+(*A) reached a plateau in the solution's quality, path relinking was able to guide the base algorithm to a different search space, making a positive difference in performance.

Comparing WsatPR*A and Zhang's BG Wsat*A, both performed on a comparative level for the most part. While BG Wsat*A dominated the parity problems, for the logistics problems (`bw_large`), which could not be solved using BG Wsat*A, WsatPR*A solved almost half of the time.

For Table 7.2, none of the instances (with exception to `f2000` and `par-16-1-c`) could be solved optimally, and thus, the average (best) number of unsatisfied clauses is shown for each instance. The results vary significantly from instance to instance. For example, for some of the `bmc` (hardware verification) instances, path relinking improved Walksat*A by a substantial degree. Also, for `f2000` (random 3-SAT) and `par` (parity) instances, path relinking consistently improved Walksat*A's performance. 28 out of 34 instances found in Table 7.2 were better solved with WsatPR*A than Walksat*A. However, there were a few `bmc` instances that path relinking hurt Walksat*A's performance by a significant margin that the average number of unsatisfied clauses for all instances in Table 7.2 was actually higher with path relinking.

For Novelty+*A, the improvement from adding path relinking is modest while the same instances for which the performance degraded by path relinking for Walksat*A was similarly degraded for Novelty+*A. These instances tend to be extremely large in size, and thus it takes time for the base algorithms to improve upon the best solution so far. We believe that with larger stagnation thresholds for both the preliminary and main phases, the algorithms with path relinking will be more competitive.

For most of the hardware verification problems (`bmc`), BG Wsat*A significantly outperformed all other algorithms. However, WsatPR*A was slightly better on the parity problems than BG Wsat*A. This agrees with the results in Section 4.2.2 and Zhang [68] that BG Wsat performs well on over-constrained instances.

In Table 7.3, the integration of path relinking did not affect the underlying algorithms' performance very much. Only noticeable differences are for `hgen6-n520-817`, where the path relinking improved Walksat*A by 200% and for `hgen7-n520-866/7`, where it degraded the performance by 100%. For Novelty*A's case, the effects were very little as well resulting in a very slight improvement on average.

Both Walksat*A and Novelty+*A, as well as their *PR variants (almost completely) dominated Zhang's BG Wsat*A, suggesting that the backbone-guidance hurt the performance of the underlying algorithm for this set of instances. This is not surprising given the past struggles of BG Wsat on critically-constrained instances. Further, it is interesting to note that BG Wsat*A is good for problems that are very effectively solved with zChaff [47]. For example, the hardware verification problems, which BG Wsat*A excels at (in relative terms against Walksat/Novelty and their variants), zChaff outperforms BG Wsat*A by a wide margin (see Section 4.2.2 for results on zChaff).

7.5 Conclusions and Future Work

The motivation for integrating path relinking, a metaheuristic, with the existing SAT local search algorithms is two fold: first, Backbone-Guided local search by Zhang [68] provided an example of how maintaining a pool of elite solutions can be beneficial to local search algorithms. It provides a mechanism for learning from the previous (preliminary) runs and biasing the underlying algorithm based on the aggregated information. With path relinking, instead of aggregating the results, we randomly select and combine a pair of solutions from the elite set. This way, the search algorithm is not confined to the search space that is directed by the average traits of the elite solutions, thus possibly allowing greater search space to be explored.

The second motivation is from the successful integration of path relinking in local search algorithms in other problem domains. i-TSAB [49], the state-of-the-art solver for job-shop scheduling problems, uses path relinking as an diversification mechanism [62]. We believe that a similar kind of benefits of path relinking can be exploited for the SAT domain.

Following the design of i-STS [61], which is a simplification of i-TSAB, Walksat and Novelty+ were retrofitted with path relinking. Experimental results with various parameter settings revealed many insights on the issues of elite pool maintenance, pool diversity, and the effects of path relinking in general. Stagnation thresholds need to find a right balance between letting the underlying algorithm explore on its own and taking advantage of the elite pool. Further, the size of the elite pool is critical in maintaining the pool diversity especially for extremely difficult problems. Path relinking, known as a diversification mechanism, can actually hurt the diversity of the pool by making the two elite solutions that are being path-relinked more alike. The effect of diversification and long-term memory in general has been examined in the JSP context [60], and further research is necessary for SAT.

Our conjecture that the non-aggregate method will outperform the aggregate method in the under- and critically-constrained region and vice versa in the over-constrained region was not supported by the results on random 3-SAT instances of various c/v ratio. The conjecture followed from our study in Section 5.3, where the extensiveness of local minima was greater in the under- and critically-constrained region than in the over-constrained region. However, WsatPR*A, the non-aggregate method, outperformed BG Walksat*A throughout the c/v ratio, with the performance gap actually increasing with the c/v ratio. While the results suggest that the mechanisms of path relinking is still not well understood, they do show promising results in this direction of research.

Overall, we observed mixed results of WsatPR when tested on a wide range of problem domains. For logistics and quasigroup problems, path relinking provided significant improvement for the base algorithms. Also, for parity problems, consistent improvement was observed by using path relinking. On the contrary, for the instances that were especially difficult for Walksat and Novelty+ such as hardware verification problems, path relinking worsened the underlying algorithms' performances. We believe this is due to the fact that the instances in this class of problems are extremely large. More refined stagnation thresholds and elite pool size, such as making them dependent on the size of the problem, c/v ratio, and/or given maximum cutoff value, should be able to improve the performance of path relinking.

Another possible way to improve its performance is by adopting a hybrid approach; with a very diverse elite pool, an instance-based method such as path relinking can be used to take advantage of its tendency to visit the unexplored search space between elite solutions. With a less diverse elite pool, an aggregate method such as BG can be used, which does not deteriorate the pool diversity.

Chapter 8

Conclusions and Future Work

In this final chapter, we re-visit the major contributions from the thesis and suggest future areas of research stemming from this work. We then end with concluding remarks.

8.1 Contributions

We recount the contributions from this thesis. Most of the major contributions are direct consequence of extending the body of work for understanding local search algorithms for SAT and surrounding issues including their problem difficulty, performance on various problem domains, and the idea and applications of backbones.

8.1.1 Fitness-Distance Analysis on SAT

The easy-hard-easy pattern observed for satisfiable instances when using local search algorithms was surprising to many researchers. Since the discovery of such pattern, much effort was put into understanding the behaviour of local search algorithms. In the thesis, we tested the idea of extensiveness of local minima for various constrainedness (measured by c/v ratio) using fitness-distance analysis. To our surprise, fitness-distance correlation did not fluctuate appreciably throughout the constrainedness though a slight increase was observed. However, fitness-distance plots on individual instances confirmed what Singer et al. [57] has conjectured about the extensiveness of local minima: in the under-constrained instances, there are numerous high-quality local minima that are extremely distant from their closest optimal solutions, while in the over-constrained instances, high-quality local minima tended to be closer to optimal solutions. When the average distance to optimal solutions was plotted against the quality of local minima for various constrainedness, it confirmed the visual evidence from the fitness-distance plots. This factor along with the number local minima contributes to the decreasing local search cost in the over-constrained region in spite of the small number of optimal solutions.

8.1.2 Backbones as Search Guidance

The concept of backbones has been successfully applied to existing algorithms as a heuristic. Backbone-Guided local search (BG Walksat) by Zhang [68] in particular is known to be effective on over-constrained, random 3-SAT instances as well as structured problems. Before this thesis, however, there was not much research done as to why BG Walksat performs well particularly on over-constrained problems. We revealed through the Perfect Backbone Experiment that BG Walksat benefits from the backbone information as long as the information is relatively close to the true backbone. On average, if the backbone information was pointing to the right polarity between the two literals, BG Walksat was better than regular Walksat. We believe that this robustness is what makes BG Walksat effective.

In terms of the actual quality of the backbone estimates for various constrainedness, over 70% of the backbone estimates were sufficiently close to the true backbone values. We were surprised to find that the estimates in over-constrained instances were not any more accurate than the estimates in under- and critically-constrained instances. We concluded that the discrepancy in BG Walksat's performance for different levels of constrainedness is simply from the fact that over-constrained instances have larger backbones; since the backbone estimates need only be somewhat accurate, instances with larger backbones provide greater backbone information to the algorithm.

8.1.3 Degree of Implication

In evaluating the correlation between the backbone size and the number of optimal solutions, we observed that the correlation was higher for job-shop scheduling problems (JSP) than SAT. We conjectured that the result is from the greater influence the non-backbone variables in SAT have on the global solution space compared to JSP. The greater the influence of non-backbone variables, the more difficult it will be to predict the number of optimal solutions given the backbone size. In order to test this hypothesis, we designed a way to measure this influence, a measure called *degree of implication*, for non-backbone variables. This measurement clearly showed that the non-backbone variables in SAT indeed have greater degree of implication than those in JSP, thus leading to the weaker correlation between the backbone size and the number of optimal solutions for SAT.

8.1.4 Long-term Memory

Path relinking was integrated to Walksat [55] and Novelty+ [44, 30] as a form of long-term memory. To our knowledge, such instance-based use of long-term memory has never been implemented on local search SAT solvers. It showed strong results especially for logistics planning, quasigroup, and parity problems. Compared to the aggregate-method used in BG Walksat [68], our instance-based method was much stronger throughout the various levels of constrainedness on 3-SAT instances.

In addition, through empirical testing, characteristics of path relinking, elite pool maintenance, and diversification were revealed. Specifically, path relinking, known as a diversification mechanism, can actually hurt the diversity of an elite pool when the pool size is sufficiently small. This is due to the fact that path relinking rewards those variables that are the same between two elite solutions while setting the variables with opposing values to the same values.

8.1.5 Other Contributions

Another contribution comes from the application of satisfiability to a real-world problem. Water Network Security problem, which is often solved using Integer Programming or Greedy set-cover heuristic, was encoded as a satisfiability problem. We showed that with further research, satisfiability could be useful as a verification tool for this application.

We also compared the performance of a constructive search algorithm and a local search algorithm. Through empirical testing, we validated the previous claims that constructive search algorithms are superior in structured instances while local search algorithms outperform in random problems. The discrepancy in search effort for the two types of algorithms on various domains was dramatic. Further, we showed that the factors that affect local search cost affect constructive search algorithms to a much lesser extent. These factors include the number of optimal solutions and the backbone size of an instance.

8.2 Future Work

As a consequence of this thesis, many interesting questions arose for further research. Here are some possible avenues for future work.

8.2.1 Structured Instances

Most of the empirical work done in this thesis is based on random 3-SAT instances. The use of random 3-SAT instances facilitated the manipulation of the properties of the problems such as their size and constrainedness. However, it would be critical to apply the same set of experiments on structured instances if this work were to have an impact on real-world problem solving. From our limited experience, the characteristics of structured instances significantly differ from the random instances, and it would not be surprising to see a different set of results. Provided that we can generate a sufficient number of instances of desired properties, future study of structured instances should include the following:

Problem difficulty for structured instances From what we have seen in the past, *a priori* factors that affect search cost for random instances such as the problem size and constrainedness were not as good of a predictor for structured instances. Also, it would be interesting to see how much the *a posteriori* factors affect structured instances. These include the number of optimal solutions, number of local minima, and the extensiveness of local minima. There have been previous work for job-shop scheduling problems, where Watson et al. [61] noticed that with structured instances, the accuracy of $d_{lopt-opt}$ model[1] deteriorated considerably. They conjectured that a similar type of decline in the model's accuracy for SAT.

[1]$d_{lopt-opt}$ model describes the search cost versus the average distance between the local minima and their closest optimal solutions.

Degree of structuredness In addition to the comparisons between structured and random instances, the degree of structuredness should influence problem difficulty and accuracies of various models. One way to vary the degree of structuredness is by *morphing* a given problem [19]. Morphing can introduce structure or randomness into a wide variety of problems.

Performance testing In most of the literature including this thesis, most of the performance testing of algorithms were based on random instances. In general, more work should be done with structured instances as they have more direct implications to SAT applications than random instances.

8.2.2 Constructive Search

A comprehensive study on constructive search algorithms should complement this thesis very well. In Chapter 4, we presented some preliminary work on the factors that affect constructive search cost by applying the same factors that affect local search cost, namely the number of optimal solutions and the backbone size, to zChaff [47]. Although the concept of being trapped in local minima does not exist for constructive search methods, the number of local minima could still affect the search cost since the abundance of high-quality local minima will still attract the solver deeper into the search tree. One of the possible factors for constructive search cost is the number of implicants per variable assignment.[2] Because constructive search algorithms are driven by unit propagations, this factor could be critical to the difficulty of a given instance.

Although our focus of backbones in Chapter 6 was in the context of local search algorithms, the idea of applying backbones and backbone estimates does exist for constructive search algorithms as well. In Dubois and Dequen's work [17], the backbone was estimated using partial assignments and used as a branching heuristic (see Section 2.2.1). It would be interesting to see how accurate their method of backbone estimation is through the Pseudo Backbone Experiment (Section 6.3.3). Furthermore, the Perfect Backbone Experiment (Section 6.3.2) can also be applied here to see if the constructive search based backbone algorithm is just as robust (to the quality of backbone estimates) as the local search based ones.

Finally, with respect to the use of metaheuristics, we can adopt some long-term memory strategy for constructive search algorithms. In fact, such method has already been successfully applied on a constructive search algorithm in Constraint Satisfaction Problem domain. Multi-Point Constructive Search [3] uses a set of elite solutions to heuristically guide constructive search through periodically restarting the search from an elite solution. We believe that constructive search algorithms in SAT domain can easily adopt such technique to improve its scalability and speed.

8.2.3 Backbones

In addition to the study of backbones on constructive search algorithms, there are many extensions to the Perfect Backbone and Pseudo Backbone Experiments. For instance, while the thesis concentrated on the actual backbones, there are non-backbone variables that are highly

[2]This suggestion is from Fahiem Bacchus.

constrained as well. These variables should have a noticeable impact especially in the under- and critically-constrained region, where there is a large number of non-backbone variables. For the Perfect Backbone Experiment, we expect to see a larger cost difference between good and poor backbone estimates. If non-backbone variables are included for the Pseudo Backbone Experiment, the accuracy of the backbone estimates for over-constrained instances should be better than those that are less tightly constrained following from the results of local minima extensiveness (Section 5.3).

8.2.4 Long-term Memory

The integration of path relinking on Walksat (WsatPR) and Novelty+ (Novelty+PR) in Chapter 7 showed us that this could be fertile area of research. In more general terms, we have shown the potential of the use of long-term memory for local search algorithms as did Zhang [68]. Here are some more specific ideas on this topic:

Development of SAT solvers with long-term memory The performance we have achieved through WsatPR serves as a motivation for building better, more sophisticated algorithms using long-term memory. This could involve a form of long-term memory other than path relinking or improving the current implementation based on the existing architecture of WsatPR. One area of immediate interest from the thesis is incorporating *a priori* information of a given instance (*i.e.* number of variables, constrainedness, etc.) to allow better parameter settings such as the elite pool size and stagnation thresholds.

Intensification and diversification The existing work claims that the benefits of the use of long-term memory come from the diversification achieved through maintaining a pool of elite solutions and using appropriate metaheuristics to explore disparate search space [23, 22, 62]. Path relinking is thus known as a diversification mechanism while the underlying algorithms are associated with intensification. Though this logic has an intuitive appeal, there is not much literature that critically examines this idea for SAT (there has been work in the job-shop scheduling context [60]). From our experience, in many cases, path relinking hurt the diversity of elite pool. By the same argument, Walksat is able to quickly move away from starting solutions (at least for smaller instances) using its superior mobility. This suggests the reversal of roles of intensification and diversification. It is plausible that the two observations we made were algorithm/domain-specific. Nonetheless, more research in this area is needed to advance our knowledge of metaheuristics and for better algorithm designs.

Importance of diversity Aside from the fact that we do not have a clear understanding of what constitutes intensification and diversification, the importance of pool diversity is an area that is not well studied. Researchers believe (as we do from our preliminary results) that the pool diversity is important to the underlying algorithm, but there is little empirical or analytical evidence to support this idea.

8.3 Conclusions

The central thesis of this dissertation is that an empirical analysis leads to a deeper understanding of local search methods and problem difficulty in satisfiability (SAT) and that the understanding can form a foundation for better algorithm designs. The thesis filled the gaps in understanding problem difficulty for local search algorithms by accounting for the dip in the search cost past the phase transition region. It also examined the idea of backbone and backbone-guided local search algorithms. In doing so, we came up with a new way of measuring the global effect of non-backbone variables, as well as explaining the type of performance seen in a backbone-guided local search algorithm. Finally, motivated by its successful applications on non-SAT local search algorithms as well as by our problem difficulty results, we integrated a long-term memory strategy with local search algorithms for SAT, opening a new avenue of research in the area.

Bibliography

[1] Yael Abarbanel-Vinov, Neta Aizenbud-Reshef, Ilan Beer, Cindy Eisner, Daniel Geist, Tamir Heyman, Iris Reuveni, Eran Rippel, Irit Shitsevalov, Yaron Wolfsthal, and Tali Yatzkar-Haham. On the effective deployment of functional formal verification. *Formal Methods in System Design*, 19(1):35–44, 2001.

[2] Carlos Ansótegui, Alvaro del Val, Iván Dotú, Cèsar Fernàndez, and Felip Manyà. Modeling choices in quasigroup completion: SAT vs. CSP. In *Proceedings of the 19th National Conference on Artificial Intelligence*, pages 137–142, 2004.

[3] J.C. Beck. Multi-point constructive search. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, pages 737–741, 2005.

[4] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. *Lecture Notes in Computer Science*, 1579:193–207, 1999.

[5] Armin Biere and Wolfgang Kunz. SAT and ATPG: Boolean engines for formal hardware verification. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, pages 782–785, New York, NY, USA, 2002. ACM Press.

[6] B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2:299–306, 1997.

[7] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 331–337, Sidney, Australia, 1991.

[8] DA Clark, J Frank, IP Gent, E MacIntyre, N Tomov, and T Walsh. Local search and the number of solutions. In *Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming*, pages 119–133. Springer, 1996.

[9] J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 1092–1097, Menlo Park, California, 1994. AAAI Press.

[10] James M. Crawford and L. D. Auton. Experimental results on the crossover point in satisfiability problems. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 21–27, Menlo Park, California, 1993. AAAI Press.

[11] James M. Crawford and Larry D. Auton. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, 81(1-2):31–57, 1996.

[12] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communication of the Association for Computing Machinery*, 5(7):394–397, 1962.

[13] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7(3):201–215, 1960.

[14] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[15] Heidi E. Dixon, Matthew L. Ginsberg, and Andrew J. Parkes. Generalizing Boolean satisfiability I: Background and survey of existing work. *Journal of Artificial Intelligence Research.*, 21:193–243, 2004.

[16] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.

[17] Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 248–253, Seattle, WA, USA, 2001.

[18] Jeremy Frank. Learning short-term weights for GSAT. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 384–391, Nagoya, Japan, 1997.

[19] Ian P. Gent, Holger H. Hoos, Patrick Prosser, and Toby Walsh. Morphing: Combining structure and randomness. In *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 654–660, Orlando, Florida, 1999.

[20] Ian P. Gent and Toby Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 28–33, Washington DC, USA, 1993.

[21] Fred Glover. Tabu search: A tutorial. *Interfaces*, 20(1):74–94, 1990.

[22] Fred Glover. Scatter search and path relinking. *New Ideas in Optimization*, pages 297–316, 1999.

[23] Fred Glover and Fred Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[24] E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat solver. In *Proceedings of Design Automation and Test in Europe (DATE)*, pages 142–149, 2002.

[25] Edward A. Hirsch and Arist Kojevnikov. Unitwalk: A new SAT solver that uses local search guided by unit clause elimination. *Annals of Mathematics and Artificial Intelligence*, 2002.

[26] John H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA, 1992.

[27] John N. Hooker and V. Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, 15(3):359–383, 1995.

[28] Holger H. Hoos. The satisfiability library. page http://www.cs.ubc.ca/ hoos/SATLIB/.

[29] Holger H. Hoos. *Stochastic Local Search - Methods, Models, Applications*. PhD thesis, Technische Universitat Darmstadt, Germany, 1998.

[30] Holger H. Hoos. On the run-time behaviour of stochastic local search algorithms for sat. In *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 661–666. AAAI Press, 1999.

[31] Holger H. Hoos. SAT-encodings, search space structure, and local search performance. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 296–303, Stockholm, Sweden, 1999.

[32] Holger H. Hoos. An adaptive noise mechanism for walksat. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 655–660. AAAI Press, 2002.

[33] Holger H. Hoos and Thomas Stützle. SATLIB: An Online Resource for Research on SAT. In *SAT 2000*, pages 283–292.

[34] Holger H. Hoos and Thomas Stutzle. *Stochastic Local Search: Foundations and Applications*. Elsevier Science Publishers Ltd., Essex, UK, 2004.

[35] Alan J. Hu. Formal hardware verification with BDDs: An introduction. In *Pacific Rim Conference on Communications, Computers, and Signal Processing (PACRIM)*, pages 677–682. IEEE, 1997.

[36] Frank Hutter, Dave A. D. Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 233–248, London, UK, 2002. Springer-Verlag.

[37] Robert G. Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[38] Valnir Ferreira Jr. and John Thornton. Longer-term memory in clause weighting local search for SAT. In *Proceedings of the Australian Conference on Artificial Intelligence*, pages 730–741, 2004.

[39] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201, Menlo Park, California, 1996. AAAI Press.

[40] Henry A. Kautz, Yongshao Ruan, Dimitris Achlioptas, Carla P. Gomes, Bart Selman, and Mark E. Stickel. Balance and filtering in structured satisfiable problems. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 351–358, Seattle, WA, USA, 2001.

[41] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363, 1992.

[42] Philip Kilby, John K. Slaney, and Toby Walsh. The backbone of the travelling salesperson. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 175–180, Edinburgh, Scotland, 2005.

[43] Olivier C. Martin, Rémi Monasson, and Riccardo Zecchina. Statistical mechanics methods and phase transitions in optimization problems. *Theoretical Computer Science.*, 265(1-2):3–67, 2001.

[44] David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 321–326, Providence, Rhode Island, 1997.

[45] David G. Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions for SAT problems. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 459–465, Menlo Park, California, 1992. AAAI Press.

[46] Paul Morris. The breakout method for escaping from local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 40–45, 1993.

[47] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

[48] Andreas Nonnengart, Georg Rock, and Christoph Weidenbach. On generating small clause normal forms. *Lecture Notes in Artificial Intelligence*, 1421:397–411, 1998.

[49] Eugeniusz Nowicki and Czeslaw Smutnicki. New algorithms for the job shop problem. Technical report, Institute of Engineering Cybernetics, Wroclaw University of Technology, Poland, 2003.

[50] Eugene Nudelman, Kevin Leyton-Brown, Alex Devkar, Yoav Shoham, and Holger Hoos. SATzilla: An algorithm portfolio for SAT. In *7th International Conference on Theory and Applications of Satisfiability Testing, SAT 2004 Competition: Solver Descriptions*, pages 13–14, 2004.

[51] Andrew J. Parkes. Clustering at the Phase Transition. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 340–345, Providence, RI, 1997.

[52] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[53] Dale Schuurmans and Finnegan Southey. Local search characteristics of incomplete sat procedures. *Artificial Intelligence.*, 132(2):121–150, 2001.

[54] Bart Selman and Henry A. Kautz. An empirical study of greedy local search for satisfiability testing. In *Proceedings of the 11th National Conference on Artificial Intelligence*, Washington DC, 1993.

[55] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 337–343, Seattle, 1994.

[56] Bart Selman, Hector J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, 1992. AAAI Press.

[57] Josh Singer, Ian P. Gent, and Alan Smaill. Backbone fragility and the local search cost peak. *Journal Artificial Intelligence Research*, 12:235–270, 2000.

[58] Stephen Smith and C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 139–144, 1993.

[59] Dave A. D. Tompkins and Holger H. Hoos. Warped landscapes and random acts of SAT solving. In *Proceedings of the 8th International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, 2004.

[60] Jean-Paul Watson. On metaheuristic “failure modes”: A case study in tabu search for job-shop scheduling. In *Proceedings of the 6th Metaheuristics International Conference*, 2005.

[61] Jean-Paul Watson, J. Christopher Beck, Adele E. Howe, and L. Darrell Whitley. Problem difficulty for tabu search in job-shop scheduling. *Artificial Intelligence.*, 143(2):189–217, 2003.

[62] Jean-Paul Watson, Adele E. Howe, and L. Darrell Whitley. Deconstructing Nowicki and Smutnicki’s -TSAB Tabu search algorithm for the job-shop scheduling problem. *Computers and Operations Research*, 33:2623–2644, 2006.

[63] W. Wei and B. Selman. Accelerating random walks. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 216–232. Springer, 2002.

[64] Colin P. Williams and Tad Hogg. Exploiting the deep structure of constraint problems. *Artificial Intelligence*, 70:73–117, 1994.

[65] Zhe Wu and Benjamin W. Wah. An efficient global-search strategy in discrete lagrangian methods for solving hard satisfiability problems. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence*, pages 310–315. AAAI Press / The MIT Press, 2000.

[66] Zhao Xing and Weixiong Zhang. Maxsolver: An efficient exact algorithm for (weighted) maximum satisfiability. *Artificial Intelligence.*, 164(1-2):47–80, 2005.

[67] Makoto Yokoo. Why adding more constraints makes a problem easier for hill-climbing algorithms: Analyzing landscapes of CSPs. In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming*, pages 356–370, 1997.

[68] Weixiong Zhang. Configuration landscape analysis and backbone guided local search: Part I: Satisfiability and maximum satisfiability. *Artificial Intelligence.*, 158(1):1–26, 2004.

[69] Weixiong Zhang and Joseph C. Pemberton. Epsilon-transformation: Exploiting phase transitions to solve combinatorial optimization problems initial results. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 895–900, 1994.