

RETROSPECTIVE ANALYSIS: REFINEMENTS OF  
LOCAL SEARCH FOR SATISFIABILITY TESTING

by

JOACHIM PAUL WALSER

A THESIS

Presented to the Department of Computer  
and Information Science  
and the Graduate School of the University of Oregon  
in partial fulfillment of the requirements  
for the degree of  
Master of Science


August 1995

"Retrospective Analysis: Refinements of Local Search for Satisfiability Testing," a thesis prepared by Joachim Paul Walser in partial fulfillment of the requirements for the Master of Science degree in the Department of Computer and Information Science. This thesis has been approved and accepted by:

  
\_\_\_\_\_  
Dr. James M. Crawford, Advisor (Computational Intelligence Research Lab)

Aug. 18, 1995  
Date

Accepted by:

  
\_\_\_\_\_  
Vice Provost and Dean of the Graduate School

© 1995 Joachim Paul Walser

An Abstract of the Thesis of  
Joachim Paul Walser for the degree of Master of Science  
in the Department of Computer and Information Science  
to be taken August 1995  
Title: RETROSPECTIVE ANALYSIS: REFINEMENTS OF  
LOCAL SEARCH FOR SATISFIABILITY TESTING

Approved: \_\_\_\_\_

  
Dr. James M. Crawford (CIRL)

Local Search routines typically depend on parameters that control the search, such as how long to search before restarting. Optimizing these parameters improves performance and is important for a fair comparison of differing approaches. However, careful optimization is computationally expensive and has been undoable for larger problem sizes.

Here, a probabilistic method, retrospective parameter optimization, is presented. Retrospective analysis allows certain parameters to be tuned using previously collected runtime-data. The method is applied to optimizing mean performance of WSAT on Random 3SAT and scheduling problems by tuning the Max-Flips parameter. Evidence is provided that the optimal value of Max-Flips scales quadrat-

ically for Random 3SAT. Further, we show that parallelizing WSAT leads to almost linear speedup on Random 3SAT for a moderate number of processors.

Finally, retrospective analysis is used to test refinements of WSAT, including an implicit propagation mechanism which improves performance on Sadeh's scheduling problems by exploiting their structure.

## CURRICULUM VITA

NAME OF THE AUTHOR: Joachim Paul Walser

PLACE OF BIRTH: Tübingen, Germany

DATE OF BIRTH: October 31, 1968

### GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon  
Universität Stuttgart

### DEGREES AWARDED:

Master of Science in Computer and Information Science, 1995,  
University of Oregon  
Bachelor of Science (Vordiplom) in Computer Science, 1992,  
Universität Stuttgart, Germany

### AREAS OF SPECIAL INTEREST:

Artificial Intelligence, Constraint Satisfaction

### PROFESSIONAL EXPERIENCE:

Teaching Assistant, Department of Computer Science,  
Universität Stuttgart, 1992-93

### AWARDS AND HONORS:

Scholarship from the Exchange Program of the Universität Stuttgart

## ACKNOWLEDGEMENTS

I would like to thank my advisor, Jimi Crawford, for starting me on Local Search and for giving me substantial and continuous feedback during the course of this thesis. I am grateful to Andrew Parkes for many invaluable discussions and for continuously supplying me with SAT problem instances.

Thanks to Andrew Baker for the use of his implementation of WSAT and to Bart Selman for pointing out related work on Las Vegas algorithms. Thanks also to all members of the Computational Intelligence Research Lab for many helpful suggestions in and outside the inspiring weekly Friday Meeting; especially Ari Jónsson, Bart Massey, David Etherington, Joe Pemberton, Matt Ginsberg, and Tania Bedrax-Weiss. Thanks also to Gene Luks and Art Farley (at the Computer Science Department) for useful comments on this thesis, and to Betty Lockwood for being an important help with the requirements of the Master's program.

I would also like to acknowledge the Department of Computer and Information Science for the approximately 1.72 trillion 'variable flips' on the parallel Silicon Graphics machines. I owe many thanks to the exchange program of the University of Stuttgart and the OSSHE for the scholarship that brought me to Oregon and supported my stay.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION . . . . .	1
II. EVALUATING LOCAL SEARCH . . . . .	8
Propositional Satisfiability Testing . . . . .	8
Local Search . . . . .	11
WSAT . . . . .	13
Measurements and Statistics . . . . .	17
Towards an Experimentation Environment . . . . .	20
Related Work on Las Vegas Algorithms . . . . .	21
III. RETROSPECTIVE OPTIMIZATION OF MAXFLIPS . . . . .	22
Analysis . . . . .	23
Results on Random 3SAT . . . . .	30
Results on Sadeh's Scheduling Problems . . . . .	37
Strategies with Non-Constant Maxflips . . . . .	41
Summary and Future Work . . . . .	42
IV. PARALLELIZATION OF LOCAL SEARCH . . . . .	43
Uniform Repeating Strategies . . . . .	44
Retrospective Analysis for Parallelization . . . . .	44
Results . . . . .	52
Summary and Future Work . . . . .	55



V. REFINEMENTS OF WSAT . . . . . 56

    Exploiting Structure in Sadeh's Problems . . . . . 56

    Selman's WALKSAT . . . . . 66

    Summary and Future Work . . . . . 69

APPENDIX

    THE GSAT FAMILY . . . . . 70

BIBLIOGRAPHY . . . . . 76

## LIST OF TABLES

Table		Page
1.	Summary of Applied Statistical Formulae . . . . .	20
2.	Successful and Unsuccessful Sample Tries at Maxflips = 8000 . . . . .	23
3.	WSAT's Performance as the Problem Size is Varied . . . . .	31
4.	Parameters of Least Squares Fit to Maxflips* Scaling . . . . .	33
5.	WSAT's Performance as the Constrainedness is Varied . . . . .	35
6.	Non-Optimized/Optimized Runtimes of WSAT on Scheduling Problems.	38
7.	Comparison of Various Procedures on Sadeh's Scheduling Problems . .	41
8.	Results for HWSAT on Scheduling Problems . . . . .	63
9.	Summary of Results on Scheduling Problems . . . . .	64
10.	Results of HWSAT on Random 3SAT . . . . .	64
11.	Results of Various Refinements of WSAT . . . . .	67
12.	Behavior of Selman's WALKSAT for Finite and Infinite Maxflips . . . . .	68

## LIST OF FIGURES

Figure	Page
1. Conceptual Model of Local Search for SAT . . . . .	12
2. The WSAT Procedure . . . . .	14
3. Success-Distribution for WSAT . . . . .	17
4. Success-Distribution for WSAT, Cutoff at Maxflips = 10K . . . . .	18
5. Retrospective Analysis for Maxflips, Accuracy Results . . . . .	29
6. Scaling of WSAT on Random 3SAT . . . . .	32
7. Scaling of Optimal Maxflips . . . . .	34
8. WSAT's Performance as the Constrainedness is Varied . . . . .	35
9. Behavior of the Standard Deviation as Maxflips is Varied . . . . .	36
10. Varying Maxflips on Sadeh Classes with One Bottleneck . . . . .	40
11. Varying Maxflips on Sadeh Classes with Two Bottlenecks . . . . .	40
12. Uniform Repeating Strategy . . . . .	44
13. The Parallel WSAT Procedure (PWSAT) . . . . .	45
14. Expected Speedup due to Parallelization. Range 1–100 Processors . . .	53
15. Expected Speedup due to Parallelization. Range 1–1000 Processors . .	54
16. Expected Speedup due to Parallelization for Various Values of Maxflips	54
17. WSAT's Behavior on Binary Chains in Scheduling Problems . . . . .	62
18. HWSATB's Behavior on Binary Chains in Scheduling Problems . . . . .	65
19. Selman's Implementation of WALKSAT . . . . .	67
20. Performance of WSAT and WALKSAT as Maxflips is Varied . . . . .	68
21. A Generic Procedure for the GSAT Family . . . . .	71

## CHAPTER I

### INTRODUCTION

CONSTRAINT SATISFACTION has come to be viewed as a core problem in many applications of artificial intelligence; for example planning, vision, and natural language processing. Given a set of *variables*, the goal is to assign values to all of them, subject to a set of *constraints* that restrict certain combinations of values the variables can take. Constraint satisfaction is essentially a search problem in a combinatorial search space.

Many problems can naturally be viewed as a constraint satisfaction problem (CSP) by formalizing the constraints inherent in the problem. For example, consider the map coloring problem, in which a number of countries on a map are to be colored, subject to the restriction that no adjacent countries have the same color. Here, the countries are the variables that can take colors as values; the constraints state that no adjacent countries may be colored equal. In this particular example, all variables can assume values from a common *domain* – the domain of colors.

An earlier, but closely related problem is the propositional satisfiability problem, SAT (Cook, 1971; Garey and Johnson, 1979). SAT can be viewed as a CSP in which each variable can assume the values *true* or *false*. SAT plays a central role in the study of computational complexity and is an important representation for a variety of applied problems.

Unless  $P$  equals  $NP$ , in general solving constraint satisfaction problems requires combinatorial search; the search space consists of the combinations of all

possible assignments to variables. In a CSP, a domain is associated with each variable, therefore the search space has the size of the product of all the domain-sizes. This is what makes the CSP hard.

Many systematic search strategies have been proposed that exploit specific features of CSPs (Tsang, 1993; Baker, 1995, for overviews). Most of the work has assumed a backtracking approach in which a partial assignment of the variables is incrementally extended.

Recently, procedures have been presented for both CSP and SAT that take a different approach, and that form the family of *Local Search*. In the CSP domain, a heuristic repair method, *min-conflicts*, was introduced by (Minton et al., 1990). For SAT, various procedures were presented by (Selman et al., 1992; Gu, 1992; Gent and Walsh, 1993a), the best known being GSAT (Selman et al., 1992).

Local Search routines have proven to be the most successful approaches to date on a variety of realistic and randomly generated satisfiability and constraint-satisfaction problems. For example, they can solve hard randomly generated satisfiability problems that are almost an order of magnitude larger than those solved by conventional systematic algorithms based on the Davis Putnam procedure (Davis and Putnam, 1960; Davis et al., 1962).

Local Search routines are called “local” since they perform hill climbing on a *local gradient* rather than exploring the search space systematically. Following a local gradient of a heuristic is advantageous if the heuristic is correct with a certain chance. Most Local Search routines have no knowledge about which parts of the search space have been examined; Local Search is therefore referred to as *non-systematic*.

Most *systematic* search methods start with a partial variable assignment that is consistent with the posted constraints and try to extend it to a total assignment. In contrast, Local Search starts from an inconsistent total assignment and successively repairs constraint violations. This process of local repair follows a heuristic which determines the order of the constraints to be repaired and variables to be changed. It proceeds by identifying a variable that is in conflict with some constraints and changing its value to make it consistent (Minton et al., 1992).

It has been argued (Minton et al., 1992) that one reason the repair-based approach leads to more efficient procedures is because the total assignment provides more information to the heuristic, and higher informedness of a heuristic improves the quality of its decisions (*informedness hypothesis*, (Minton et al., 1992)). In addition to that, it has been argued (Ginsberg and McAllester, 1994) that non-systematic routines can be more effective because they can follow a local gradient while systematic procedures explore the search space in a fixed order that is not related to the local gradient.

The price for the efficiency of Local Search is *incompleteness*. Because of their non-systematicity, all Local Search procedures are semi-decision procedures. In the SAT domain this means that Local Search procedures cannot prove a formula to be unsatisfiable.

By their nature, systematicity and heuristics that follow local gradients appear to be in conflict with each other, but both are clearly desirable (Ginsberg and McAllester, 1994). Efforts to combine the advantages of both strategies have been made very recently in algorithms like *partial-order dynamic backtracking* (Ginsberg and McAllester, 1994), or *limited discrepancy search* (Harvey and Ginsberg, 1995).

This thesis focuses on Local Search for satisfiability testing, however, parts of the work carry directly over into the CSP domain. Unless  $P$  equals  $NP$ , any SAT testing procedure takes exponential runtime in worst case. However, from a practical point of view, it is quite possible that most instances that occur in real applications are easy and can be solved quickly. Since different procedures can have quite different performance profiles, one needs a more refined analysis than just worst case analysis. At the time being, *empirical evaluation* is often the only possible runtime analysis to obtain average results.

A recently proposed efficient Local Search procedure, WSAT, will be examined throughout this thesis (Selman et al., 1994). WSAT performs greedy hill climbing on the number of satisfied clauses in the input formula. Additionally, it uses *noise* to escape local maxima, i. e. it randomly perturbs certain variables.

To evaluate runtime of satisfiability procedures, randomized classes of problems have been examined, including the class Random 3SAT (Mitchell et al., 1992). The study of Random 3SAT shows that a set of moderately sized problems exist that are very hard to solve in practice. Random 3SAT is therefore an effective tool to evaluate SAT-testing procedures and will be the main source of SAT-problems here.

Unfortunately, average runtime evaluation of a Local Search procedure is computationally very expensive for two reasons. First, most Local Search routines are *Las Vegas Algorithms*. A Las Vegas algorithm is a randomized algorithm that always produces the correct answer when it stops but whose running time is a random variable (Brassard and Bratley, 1988). Local Search typically has large runtime variation, so a large number of runs have to be performed on many problem instances to obtain accurate estimates of the mean runtime.

Second, Local Search typically depends on parameters which control the search strategy, such as how long to examine a region of the search space until restarting with a different initial assignment. This *cutoff parameter* is called Maxflips in procedures related to GSAT. Tuning Maxflips is important since it can lead to a strong performance improvement. Also, a *fair empirical comparison* between different procedures can only be done provided all procedures operate in their optimal parameter region.

The main focus of this thesis is on a new practical method for parameter optimization by performing only one set of experiments at fixed parameter settings and from these figures estimating the mean runtime at other parameter settings probabilistically. The approach, which will be called *retrospective parameter optimization*, allows simultaneously for (i) an optimization of certain Local Search parameters and (ii) accurately estimating peak-performance (with respect to those parameters). Its practical application is simple and it can be applied to optimize various statistical measures.

Recent systematic experiments on small problem sizes have suggested (Gent and Walsh, 1993a) that the optimal value of Maxflips on Random 3SAT scales quadratically with the problem size. Having introduced retrospective parameter optimization, it will be applied to provide further evidence that Maxflips indeed scales quadratically, even for much larger problems. Additionally, results on the scaling behavior of WSAT on Random 3SAT will be given.

An interesting property of Local Search is that it can be parallelized very easily. Therefore, the question arises what speedup can be expected from parallelization. For the class of Las Vegas algorithms, it was shown (Luby et al., 1993) that superlinear speedup cannot be expected. This thesis investigates the



quantitative speedup of Local Search from parallelization, again using a retrospective analysis. It will be shown that parallelization is efficient for WSAT on Random 3SAT, in the sense that it leads to almost linear speedup provided that the number of processors is moderate.

A further refinement is reported within this thesis that tackles a weakness of Local Search that has recently been addressed by (Crawford, 1995b): Local Search suffers from a *lack of forward propagation*. Most modern depth-first search based methods such as TABLEAU (Crawford and Auton, 1993) use mechanisms that propagate variable values through the input formula to determine values of other variables. One feature of realistic problems is that they often contain structures for which propagation is useful. In (Crawford, 1995b), Local Search was upgraded by unit-propagation (USAT); the mechanism achieves a significant speedup with respect to the average number of variable flips on certain problem classes. However, for implementational reasons, flips become much more expensive. Here, inspired by USAT, an implicit propagation strategy is investigated that propagates values through binary clauses only and that conforms with the spirit of Local Search. It arose from an examination of Sadeh's scheduling problems and exploits a certain structural feature. A performance improvement on Sadeh's problems is reported empirically.

Finally, it will be shown that the details of Local Search algorithms are critical for optimal performance. In the publication of WSAT (Selman et al., 1994), parts of the variable selection strategy were left open. A straightforward implementation of the published version of WSAT will be compared with the implementation that is now publicly available from Bart Selman. It will be shown that although the difference between the strategies is subtle, it leads to a significant difference in

performance and in the behavior of the algorithm as `Maxflips` is varied.

The remaining chapters are organized as follows. Chapter II introduces the framework for an empirical evaluation of Local Search; Random 3SAT and WSAT are defined formally. Chapter III presents the retrospective analysis technique for optimizing the cutoff parameter `Maxflips` and reports results on Random 3SAT and Sadeh's scheduling problems. Chapter IV introduces and applies a retrospective approach for investigating parallelization of Local Search. Finally, chapter V presents an implicit propagation mechanism for WSAT, and reports on refinements of WSAT's heuristic. The appendix provides a collection of the various flavors of Local Search routines related to GSAT that were proposed in the literature.

## CHAPTER II

### EVALUATING LOCAL SEARCH

The main focus of this thesis is on practical methods that reduce the computational cost of empirical evaluation of Local Search procedures. This chapter formally presents the domain of satisfiability testing, introduces the relevant Local Search procedures and the problem class Random 3SAT on which most subsequent experimentation is based. Additionally, relevant statistical issues are discussed including the dual source of runtime variation. The role of the experimentation environment is briefly discussed.

#### Propositional Satisfiability Testing

The propositional satisfiability problem (SAT) has been the backbone of the theory of NP-completeness since (Cook, 1971). It is of great importance in artificial intelligence because many practical problems can be represented in terms of SAT after a conversion to boolean form (booleanization).

#### Propositional Satisfiability

The propositional satisfiability problem is the following (Garey and Johnson, 1979). Let  $U = \{u_1, u_2, \dots, u_n\}$  be a set of *boolean* variables (also *propositions*), that can assume the values *true* or *false*. A *truth assignment*  $A$  for  $U$  is a function  $A : U \rightarrow \{\text{true}, \text{false}\}$ . If  $A$  assigns values to all the variables, it is a *total assignment*. A *literal* is either a variable  $u$  or its negation  $\bar{u}$ . We say  $\bar{u}$  is a *negation* of

$u$ , and vice versa. A clause is a set of variables that is interpreted as a *disjunction*. A *formula* or *problem instance* is a set of clauses that is interpreted as *conjunctive normal form* (CNF). Let  $\mathcal{F}$  be a formula over  $U$  (i. e.  $\mathcal{F}$  contains only variables from  $U$ ), and  $C$  be a clause in  $\mathcal{F}$ . A truth assignment  $A$  for  $U$  *satisfies*  $C$  if and only if at least one literal  $u \in C$  is *true* under  $A$ . An assignment  $A$  for  $U$  satisfies  $\mathcal{F}$  if and only if it satisfies every clause in  $\mathcal{F}$ . A satisfying assignment  $A$  is also called a *model* for  $\mathcal{F}$ .

The *satisfiability problem* (SAT) is:

Instance: A set  $U$  of variables and a formula  $\mathcal{F}$  (in CNF) over  $U$ .

Question: Is there a truth assignment for  $U$  that satisfies  $\mathcal{F}$ ?

### Satisfiability Testing

Clearly, it can be determined whether a satisfying assignment exists by trying all possible assignments. However, if the set  $U$  contains  $n$  variables then there are  $2^n$  assignments.

Therefore, other strategies have to be considered, which as mentioned in the introduction, typically either (i) start by assigning values to some variables and forward propagate these to find forced values of other variables; or (ii) follow the local gradient of a *heuristic distance measure* to a solution, hoping that the gradient guides the search to a solution.

Strategies in the first class include depth first search strategies such as the Davis-Putnam procedure (Davis and Putnam, 1960; Davis et al., 1962) and its recent derivatives such as TABLEAU (Crawford and Auton, 1993), or non-systematic versions of it such as ISAMP (Langley, 1992).

Strategies in the second class are mainly Local Search procedures related to GSAT (Selman et al., 1992) or combinations of local and systematic strategies such as *partial-order dynamic backtracking* (Ginsberg and McAllester, 1994), or *limited discrepancy search* (Harvey and Ginsberg, 1995).

Independent of the particular strategy, unless  $P$  equals  $NP$ , its worst case behavior will be exponential. However, in practical applications it is quite possible that most problems are solved quite easily and there are only a few hard outliers. Average runtime behavior of many SAT testing procedures is beyond the scope of today's theoretical analysis, therefore empirical evaluation is important.

#### The Random $k$ -SAT model

To evaluate runtime of satisfiability procedures, a class of randomized benchmark problems, Random 3SAT, has been examined (Mitchell et al., 1992; Mitchell, 1993). Finding a suitably hard set of problems is a difficult problem in itself (Cheeseman et al., 1991; Mitchell et al., 1992; Crawford and Auton, 1993) because both real and handcrafted problems tend to contain particular kinds of structure, and there is a potential danger of overfitting the strategies to these structures.

The study of Random 3SAT shows that a set of moderately sized problems exists that are very hard to solve in practice. Random 3SAT is therefore an effective tool to evaluate SAT-testing procedures and will be the main source of SAT-problems here. Additionally, Sadeh's scheduling problems (Sadeh, 1992) will be examined.

Random  $k$ -SAT uses the fixed clause length model in which all clauses have the same length. Problems in random  $k$ -SAT with  $n$  variables and  $l$  clauses are generated as follows: a random subset of size  $k$  of the  $n$  variables is selected for each clause, and each variable is made positive or negative with probability  $1/2$ .

## Crossover

For Random 3SAT there is a sharp phase transition from satisfiable to unsatisfiable when the value of  $l$  is approximately  $4.3n$  (Mitchell et al., 1992; Crawford and Auton, 1993). At lower  $l$ , most problems generated are *under-constrained* and are thus satisfiable, at higher  $l$ , most problems are *over-constrained* and are thus unsatisfiable. The phase transition where 50% of the instances are satisfiable is referred to as the *crossover point*. These problems are typically more difficult to solve and are thus generally considered to be a good source of hard SAT problems.

Recent experiments indicate that the most computationally difficult problems for the Davis-Putnam Procedure tend to be found near the crossover point (Mitchell et al., 1992; Cheeseman et al., 1991; Crawford and Auton, 1993). It is subject of current research where the hardest 3SAT problems are for Local Search. Experimental results on page 34 vaguely suggest that WSAT encounters a hardness peak in the *crossover region*.

Unless otherwise noted, all experiments have been conducted at the crossover point, and for all streams of input formulae, it has been verified that approximately 50% are unsatisfiable.

## Local Search

Several Local Search based procedures for finding solutions to SAT instances have been presented recently by (Selman et al., 1992; Gu, 1992; Gent and Walsh, 1993a; Selman et al., 1994).

To introduce the general strategy of Local Search, figure 1 explains the conceptual model of Local Search for SAT-testing. In the next section, WSAT will be explained and in the appendix, a summary of various flavors of Local Search

algorithms will be given (GSAT, GSAT+walk, GenSAT, etc.).

```

1 proc Local-Search-SAT
2   Input a SAT formula  $\mathcal{F}$ 
3   Output a satisfying total assignment of  $\mathcal{F}$ , if found
4    $\equiv$ 
5   initialize: drop a particle  $A$  randomly in the state space
6                 (set  $A$  to a random truth assignment)
7   while  $A$  does not satisfy  $\mathcal{F}$  do
8     if a stopping criterion has been reached, return No
9     if a restart criterion has been reached, goto initialize
10    i) move  $A$  to the “best” local neighbor position, or
11    ii) move  $A$  to a random local neighbor position
12  end
13  return  $A$ 
14 end

```

Figure 1: Conceptual model of Local Search for SAT.

Certain randomized optimization algorithms can be viewed as mechanisms for moving a *particle* around in a *state space* (Aldous and Vazirani, 1994). In the case of SAT-testing, the state space consists of the set of truth assignments, and the particle holds a truth assignment.

In figure 1, typically the local “neighbors” of  $A$  are those truth assignments that differ from  $A$  in only one variable. “Best” is measured according to a heuristic estimation of the distance to a solution (the *local gradient*). Typically, “best” means the largest number of satisfied clauses or some closely related measure.<sup>1</sup> Stopping and restart criterion are typically some number of moves in the state space. The decision whether to do a *greedy* move (according to the local gradient) or a *random* move is typically made according to some probability distribution.

---

<sup>1</sup>If  $\mathcal{F}$  is a formula that is not in conjunctive normal form (CNF), a linear technique exists to compute the “score” without constructing the CNF conversion itself (Sebastiani, 1994).

The above general procedure is *sound*, i. e. it will only find a model when given a satisfiable formula. But as stated, it is *incomplete*, i. e. it can never be certain that an input formula is unsatisfiable, and is therefore a *semi-decision procedure*. No matter when it is cut off, a certain chance remains that the input formula is satisfiable. For this reason, Local Search is only interesting if there is a benefit (generally efficiency) at the cost of the incompleteness.

Local Search is efficient. While current implementations of conventional procedures for SAT testing, such as TABLEAU (Crawford and Auton, 1993), are able to solve hard Random 3SAT formulas up to 400 variables, recent Local Search procedures can solve up to 2000 variable problems. It has been shown that flavors of GSAT also perform well on encodings on graph-coloring problems, N-queens, and boolean induction (Selman and Kautz, 1993a).

### WSAT

In the experiments for this thesis, a recent local search procedure, WSAT, has been examined. WSAT (“walk” satisfiability) was introduced by (Selman et al., 1994), and is a successor of “greedy” GSAT that did not perform random moves in the state space. This section describes WSAT.

The basic WSAT procedure, which is shown in figure 2, performs hill-climbing on the number of satisfied clauses in a truth assignment, starting from a total random assignment, and then repeatedly changing (*flipping*) the assignment of a variable that leads to the largest increase in the number of satisfied clauses. The number of satisfied clauses will also be referred to as the *score*.

WSAT is a combination of two strategies. The greedy strategy chooses the “best” variable in a clause according to which variable leads to the highest increase



```

1 proc WSAT
2   Input a set of clauses  $\alpha$ , Maxtries, and Maxflips
3   Output a satisfying total assignment of  $\alpha$ , if found
4    $\equiv$ 
5   for  $i := 1$  to Maxtries do
6      $A :=$  random truth assignment
7     for  $j := 1$  to Maxflips do
8       if  $A$  satisfies  $\alpha$  return  $A$ 
9        $C :=$  random unsatisfied clause
10      with probability  $p$ :  $P :=$  random variable in  $C$ 
11      probability  $1 - p$ :  $P :=$  "best" variable in  $C$ 
12       $A := A$  with  $P$  flipped
13    end
14  end
15  return  $No$ 
16 end

```

Figure 2: The WSAT Procedure.

of the score, if it is flipped.<sup>2</sup> The walk strategy aims at escaping local maxima in the search space by occasionally flipping random variables in unsatisfied clauses. These random flips allow for downhill moves and help the algorithm to reduce the number of restarts. If the greedy part offers a choice between several "best" variables, a random decision is made. To appropriately describe the experiments, we will use the following terms.

*Instance* An *instance* of a problem class is any well formed SAT formula that is tested for satisfiability. Because WSAT is incomplete and a central goal within this framework is to give performance estimates, all experiments have been performed exclusively on *satisfiable instances*. Therefore, unless otherwise noted, "instance" will refer to a satisfiable SAT formula.

---

<sup>2</sup>However, a slightly different implementation of WSAT exists (by Bart Selman) which uses the number of clauses that get broken if a variable is flipped as the the gradient. See section V.2 for a discussion.

*Instance collection* An *instance collection* is any collection of instances from a specific problem class. Throughout this thesis, instance collections of Random 3SAT formulas consist of the satisfiable instances of a stream of randomly generated problems. All instances in this stream were tested for satisfiability using TABLEAU, a complete method, and only the satisfiable ones were considered for experimentation. No experiments larger than 400 variables were conducted for Random 3SAT, the reason being that at present, this appears to be the limit of the problem size for which unsatisfiability can be checked in reasonable time.

*Try* In WSAT, a number of flips are made before the procedure is restarted with a new initial assignment in some other region of the search space. One such sequence of at most Maxflips flips will be referred to as a *try*. A try can either be successful or fail.

*Run* A *run* is a sequence of tries in which the last and only the last try is successful. Here, most experiments were performed fixing the number of runs rather than tries, see section III.1 for a discussion.

#### WSAT's Parameters

WSAT has three parameters, Maxflips, Maxtries, and  $p$  which influence the search. The influence of Maxtries is simple as increasing it simply increases the chance of finding a satisfying assignment.

The role of  $p$  is to control how much noise should pervade the search. (Selman et al., 1994) report that  $p$  has been found to be optimal between 0.5 and 0.6 (including strategies that vary  $p$ , such as simulated annealing, see appendix, p. 74).

The role of Maxflips is more complicated than that of Maxtries. Varying the

cutoff parameter *Maxflips* usually influences the runtime of Local Search, and although increasing *Maxflips* increases the probability of success in a single try, it may decrease the probability of success in a given time-bound.

(Gent and Walsh, 1993a) shed light on his effect by investigating different phases of local search. They identify two different phases: First, *hill-climbing* in which the score is increased by each flip, followed by *plateau-search* in which the vast majority of the flips are 'sideways', i. e. they do not increase the score.

#### Success-distribution and Maxflips

Figure 3 plots the percentage of successful tries against the total number of flips made by WSAT. The graph is based on a collection of 5000 tries on a specific problem instance (with 200 variables and 854 clauses); each try was solved after a certain number of flips. The curve plots the percentage of finished tries after a given number of flips.

To repeat an observation by (Gent and Walsh, 1993b), the following behavior is repeated in each try: during the initial hill climbing phase, almost no problems are solved. After this phase, the gradient changes, there is now a significant chance of finding a satisfying assignment. Finally, the gradient declines again noticeably. Throughout the text, we will refer to this cumulative distribution as the *success-distribution* and make use of it to illustrate the behavior of Local Search procedures.

Figure 4 illustrates the effect of *Maxflips*. The graph shows the behavior for the same instance (as in figure 3) with a setting of  $\text{Maxflips} = 10K$ .<sup>3</sup> The cutoff due to *Maxflips* can have the effect of a faster increase in the percentage of satisfiable instances. If *Maxflips* is chosen too large, much time is spent in the late plateau phase

---

<sup>3</sup>The same characteristic behavior can be observed if the average over a collection of instances is plotted, see (Gent and Walsh, 1993b).

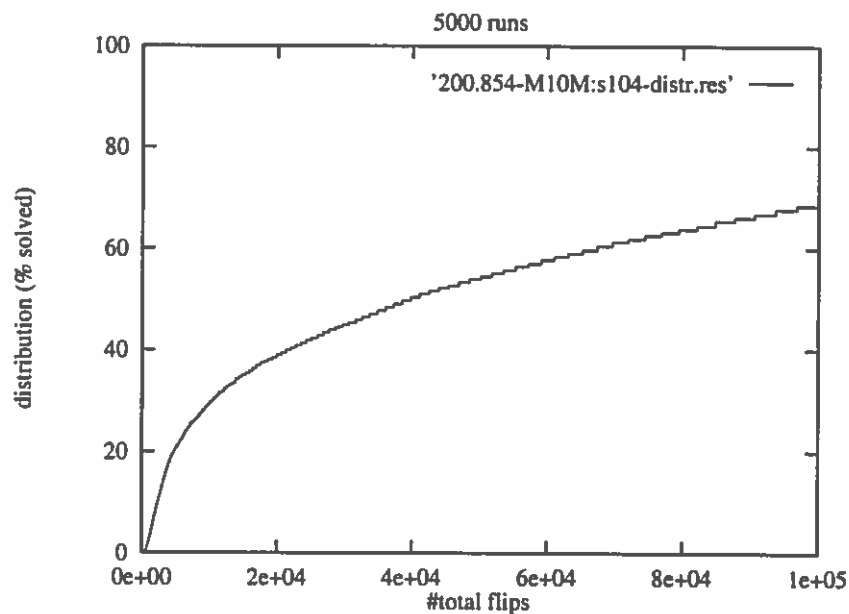


Figure 3: Success-distribution for WSAT. Single instance with 200 variables and 854 clauses.

in which not much progress is to be expected. Choosing it too small, on the other hand, is non-optimal since it cuts off right in the phase in which the most progress is being made (or worse, in a phase where almost no progress is being made).<sup>4</sup>

Since the optimal setting of Maxflips varies largely between different instances, the value that results in optimal mean performance for a variety of problems has to be determined experimentally on a large collection of instances.

### Measurements and Statistics

The attempt to optimize the runtime of a randomized procedure such as WSAT on a given problem class (such as Random 3SAT) raises the question of which measure to optimize. *Mean, median, maximum, percentiles, tail probability, or standard*

<sup>4</sup>Note that the optimal value for Maxflips for this instance would be 4000. Thus, Maxflips = 10000 cuts off too late and is non-optimal.

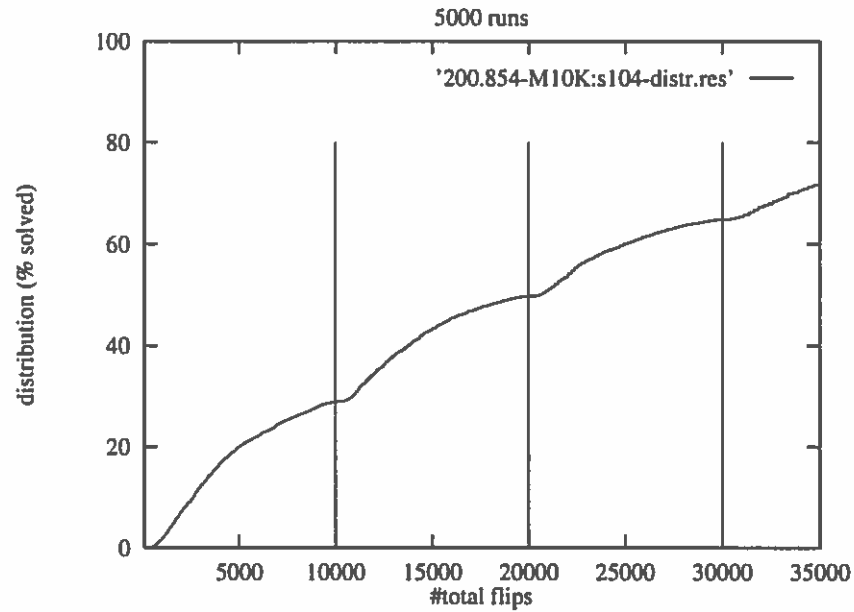


Figure 4: Success-distribution for WSAT, cutoff at Maxflips = 10K. Single instance with 200 variables and 854 clauses.

*deviation* are all reasonable candidates. All these measures map the distribution of the runtimes to a single number and might therefore not be sufficient to answer specific questions.

Nevertheless, the hope is to be able to draw conclusions from a simple measure, which can function to optimize and compare procedures. This measure should reflect certain aspects of the distribution, such as the runtime on the bulk of instances as well as on hard outliers. If the measures are in a conflict with each other, the question of which quantity to optimize depends to large extend on the application in which the procedure is embedded.

The remaining chapters will focus on optimizing the *mean runtime* of local search since the mean is an overall measure that reflects both the bulk of the problem instances as well as hard outliers—provided that there be no instances that take infinitely long to solve. Although the mean will be optimized here, the proposed

approach can easily be applied to other statistical measures as well.

An alternative question to ask besides for *optimal average runtime* is to ask for maximum *safety*—in the sense of ‘How likely will a solution be found within a certain time bound?’ This amounts to minimizing *tail probabilities* which will not be addressed in this thesis.<sup>5</sup>

Interestingly, as it turns out, minimizing the mean runtime (by optimizing WSAT’s Maxflips parameter on Random 3SAT) minimizes the standard deviation at the same time, an effect that will also be discussed on page 35. Because outliers influence the standard deviation, minimizing the standard deviation is likely to improve the behavior on outliers.

#### Dual Source of Variation

A difficult problem for runtime evaluation of Local Search is the large runtime variation of these procedures. This problem is difficult to handle since any accurate mean estimate has to consider many runs on a large number problem instances. Essentially, the runtime variation comes from two different sources.

1. Variation of the runtime between different instances of a given problem class, which will be referred to as *intra-class variation*.
2. Variation of the runtime within the set of tries on a single instance, which will be referred to as *intra-instance variation*.

It is important to bear this distinction in mind since these two types of variation also impose two different measures for it, namely *intra-instance variance* and *intra-class variance*.

---

<sup>5</sup>The question of reducing tail probabilities is discussed by (Alt et al., 1991).

Table 1: Summary of applied statistical formulae.

Variance:	$\sigma^2 = \sum (x^2 \cdot P(x)) - \sum (x \cdot P(x))^2$
Standard deviation:	$\sigma = \sqrt{\sigma^2}$
Population mean:	$\bar{x} = \sum x f / n$ (frequency $f$ )
Standard error of mean:	$\sigma_{\bar{x}} = \sigma / \sqrt{n}$
Confidence intervals:	$\bar{x} \pm t(df, \alpha/2) \cdot s / \sqrt{n}$ with Student's $t$ -statistic sample standard deviation $s$ 95% confidence: $\alpha = 0.05$ degrees of freedom $df = n - 1$

Standard deviation, standard error, and confidence intervals are defined in terms of the variance. Because there are two different kinds of variance, all these measures can be computed on the basis of intra-instance or intra-class variance, and it will be reported which source of variation a measure is based on. Table 1 gives the statistic formulae that will be used in the analysis.

### Towards an Experimentation Environment

Since experimentation is crucial in the domain of Local Search, a lot of the work to evaluate the routines needs to be dedicated to systematically running experiments and to statistically evaluate and compare the results.

To alleviate the burden of experimentation, a substantial effort for this research has been devoted to setting up an experimentation environment. This environment comprises of two parts; *controlling* and *evaluating* experiments.

Comprehensive tables containing average runtime figures suggest that measuring the procedures' performance involves mainly starting out jobs and collecting the results. However, substantial effort was necessary to come closer to a state

where experimentation could be conducted in this fashion. The main purpose of the experimentation environment is to perform large numbers of runs on instance collections and evaluate the results as automatically as possible. The environment involves a wide variety of scripts to setting off long term jobs, controlling processes, collecting results, automatically suspending jobs for overload protection, combining results, retrospectively analyzing results and plotting graphs, plotting distributions, and so on. All scripts of the experimentation environment have been implemented in PERL, a C-like language that offers expressive power and support of system commands, and allows for easy prototyping.

#### Related Work on Las Vegas Algorithms

The class of *Las Vegas Algorithms* has been analyzed in a theoretical framework in (Luby et al., 1993; Luby and Ertel, 1993). A Las Vegas algorithm is a randomized algorithm that always produces the correct answer when it stops but whose running time is a random variable. Luby *et al.* considered various sequential and parallel strategies for Las Vegas Algorithms that optimize the mean runtime over the case in which the algorithm is run without a cutoff. Clearly, WSAT is a Las Vegas algorithm provided a *specific* problem instance is given. However, given a *collection* of problem instances, the results do not directly apply anymore. This is due to the fact that in this case there exists no single random variable but a collection of random variables for the individual instances.



## CHAPTER III

## RETROSPECTIVE OPTIMIZATION OF MAXFLIPS

The runtime of Local Search varies largely—as described earlier. First, between different SAT problem instances (intra-class variation) and second between different runs on the same instance (intra-instance variation).<sup>1</sup> Therefore, determining the mean runtime behavior can be an extremely time consuming enterprise, since many experiments have to be performed on a large number of problem instances.<sup>2</sup> Therefore, optimizing parameters in a given *range* seems to be almost undoable for interesting problem sizes.

This chapter will present a practical method for accurately estimating the runtime of Local Search routines such as WSAT as the *cutoff* parameter Maxflips is varied. Prior to the estimation, WSAT is run with fixed parameter settings on a number of instances to produce a set of *sample tries*. Based on the runtimes (in flips) of these samples, varying Maxflips can be simulated quickly and accurately without additional experiments. Because the analysis proceeds in retrospect, it will be referred to it as *retrospective analysis*.

This chapter is organized as follows. The first section introduces a probabilistic approach for estimating the runtime of WSAT as Maxflips is varied, based on a

---

<sup>1</sup>Intra-instance variation occurs since most Local Search routines are randomized. Note that randomization does not appear to be important for Random 3SAT, (Gent and Walsh, 1993a).

<sup>2</sup>For example, an experiment to obtain stable results for WSAT's mean runtime on Random 3SAT at 200 vars, 854 clauses takes 16 hours of runtime on a Silicon Graphics Power Challenge. We measured 1000 instances, each 50 runs at optimal Maxflips = 8000, at a performance rate of 70Kflips/s.

sample of actual tries. Thereafter, an experimental trick will be described that helps improve the accuracy of the estimation, and accuracy results will be given empirically. Thereafter, we will present runtime figures of WSAT on Random 3SAT, providing further evidence that the optimal value for Maxflips scales quadratically for Random 3SAT, even at large problem sizes. Additionally, scaling of WSAT will be reported, and the hardness profile of Random 3SAT will be examined as the constrainedness of the problems is varied; experiments suggest that WSAT peaks in the crossover region. The behavior of the standard deviation will be discussed briefly, and we will report results of the retrospective analysis of scheduling problems. This chapter concludes by discussing strategies with non-constant Maxflips.

### Analysis

The idea behind the retrospective approach is the following. By running WSAT at a fixed value of Maxflips on one problem instance, we obtain a set of sample tries, each either successful or unsuccessful. Table 2 shows an example. Given these

Table 2: Successful and unsuccessful sample tries at Maxflips = 8000.

1. sat	1042
2. sat	3367
3. sat	483
4. unsat	8000

samples, we essentially have data that approximates the distribution of “percentage of tries successfully finished after  $x$  flips” against the number of flips  $x$ , for all values of Maxflips  $\leq 8000$ . That is, we have a discrete approximation of the success-distribution.

Given this approximation, we can make predictions about what would happen

if the value of Maxflips were set to say, 3000. In the example given in table 2, one successful try (2) would be converted into an unsuccessful one, thereby reducing the chance for finding a solution within Maxflips. On the other hand, we would save 5000 flips on try 4 which was ‘unsat’ anyway. This is the tradeoff to be optimized.

In the following this idea will be formalized, the goal being to compute the expected runtime of a Local Search procedure as Maxflips is varied. The time is measured according to the total number of flips to find a solution, given a specific value for Maxflips,  $m$ . The estimation is based on a set of sample tries,  $S_0$ , for a *single* problem instance. First, some definitions:

Definitions:

- Consider a collection of sample tries  $S_0 = \{x_1, \dots, x_n\}$  on which the analysis is based. Each  $x_i$  reflects the number of flips in try  $i$  until a solution was found. For reasons of clarity, we assume that all  $x_i$  be distinct, thus  $S_0$  is a proper set (it will be obvious how to handle the case in which  $S_0$  is a multi-set, i. e. duplicates will be allowed).

Let  $S$  be a discrete random variable which can assume any of the  $x_i \in S_0$  as values.  $S$  is assumed to be uniformly distributed:<sup>3</sup>

$$P(S = x_i) = 1/n$$

- Let  $m$  denote the given value for the parameter Maxflips.

---

<sup>3</sup>In the multi-set case,  $P(S = x_i)$  can be estimated according to the relative frequency of  $x_i$  in the sample tries.

- Let  $S_0^m$  denote the set of all successful tries in  $S_0$  which took at most  $m$  flips:

$$S_0^m := \{x_i \in S_0 \mid x_i \leq m\}$$

and let  $S^m$  be the respective random variable.

- The probability that a random try with  $\text{Maxflips} = m$  is successful (i. e. that it takes at most  $m$  flips) can be estimated by

$$P(S \leq m) = \frac{|S_0^m|}{n} =: p_m.$$

It will turn out to be useful to also define  $q_m := P(S > m) = 1 - p_m$ .

- We can directly estimate the expected number of flips in the case where a solution was found within at most  $m$  flips. This is simply the arithmetic mean of  $S_0^m$ :

$$E[S^m] = \overline{S_0^m}$$

The expected number of flips for any  $m$  can now be computed using a simple probabilistic argument: With probability  $p_m$ , the solution will be found on the first try. In this case,  $\overline{S_0^m}$  flips are expected. With probability  $p_m q_m$ , the first try will fail, but the second will succeed. Then, it takes  $\overline{S_0^m} + m$  flips, and so on.

The expected number of flips for a *single problem instance*, given  $m$  as the parameter for  $\text{Maxflips}$ , can then be expressed as

$$\begin{aligned} E_m &= p_m \overline{S_0^m} + p_m q_m (m + \overline{S_0^m}) + p_m q_m^2 (2m + \overline{S_0^m}) + \dots \\ &= \sum_{i=0}^{\infty} q_m^i \cdot p_m \cdot (i m + \overline{S_0^m}) \end{aligned}$$

$$\begin{aligned}
&= p_m \left( m \cdot \sum_{i=0}^{\infty} i q_m^i + \overline{S_0^m} \cdot \sum_{i=0}^{\infty} q_m^i \right) \\
&= p_m \left( m \cdot \frac{q_m}{(1 - q_m)^2} + \overline{S_0^m} \cdot \frac{1}{1 - q_m} \right) \\
&= m \cdot \frac{1 - p_m}{p_m} + \overline{S_0^m} \quad \square \tag{III.1}
\end{aligned}$$

Estimating  $p_m$  and  $\overline{S_0^m}$  is now straightforward using a given set of sample tries, and allowing duplicates within  $S_0$  just involves proper counting.

### Instance Collections

To compute the expected number of flips for a *collection* of instances, we can simply compute the arithmetic mean of all the instances: If  $E_{j,m}$  is the above expectation for instance  $j$  at  $\text{Maxflips} = m$ , the expectation for a collection of instances  $C = \{1, \dots, z\}$  is simply

$$E_{C,m} = \sum_{1 \leq j \leq z} E_{j,m} / z.$$

This is naturally correct provided that the results for the individual instances are correct.<sup>4</sup> The value of  $\text{Maxflips}$  which yields the lowest average expected number of flips for a collection  $C$  of instances will be referred to as  $\text{Maxflips}^* = \min_{m>0} E_{C,m}$ .

---

<sup>4</sup>The alternative strategy of directly estimating  $E_m$  for the entire instance collection (by estimating  $p_m$ ) is difficult since at any given  $m$ , it would be necessary that all instances have the same number of successful runs.

## Practical Application

To improve the accuracy of the retrospective analysis without wasting cycles, this section describes how to conduct the experiments in practice.

### Finite Maxflips

First, for an implementation of the experiments it is obviously not a good choice to collect the samples at infinite Maxflips, since the average runtime is generally much higher at large Maxflips. Additionally, since we are primarily interested in WSAT's behavior around the optimal value (around  $n^2$ ), collecting data at large Maxflips will not be of use. Estimating the  $p_m$  for equation III.1 remains straightforward in the case where the sample tries are collected at finite Maxflips (the reader might reconsider table 2). However, the Maxflips value in the retrospective analysis can of course not be larger than the value chosen in collecting the sample runs.

### Fixing Runs rather than Tries

For collecting the sample tries, fixing the number of *runs* per instance rather than the number of *tries* can improve accuracy. The reason is that  $p_m$  is estimated as

$$p_m = \frac{\text{successful tries with at most } m \text{ flips}}{\text{total tries}}.$$

Thus fixing the number of successful tries (i. e. runs) can reduce the error on hard instances (for which only a small percentage of tries succeed) and saves runtime on the easy instances (since here, the number of runs is approximately equal to the number of tries).

For example, suppose that we were to fix the number of *tries* to 100, and  $p_m$  would really be 0.01 for a given  $m$ . In this case it would be pretty likely that 100

arbitrary samples would include (i) none or (ii) exactly two successful tries. The results would be

- (i) No prediction is possible, since the instance appears to be unsolvable, or,
- (ii)  $E_m = 49m + \overline{S_0^m}$  instead of  $99m + \overline{S_0^m}$ , thus a large underestimation. At the same time,  $\overline{S_0^m}$  would be based on only two results—thus a potentially large error on a hard and thus important instance.

#### Accuracy of Retrospective Analysis

Of course, the accuracy of the estimation depends crucially on how many instances and runs the analysis is based on. The error will not be estimated theoretically, since this would involve statistical reasoning involving the success-distribution, instead we will investigate the accuracy empirically.

Experimentally, it turned out that for a single instance (collected at  $\text{Maxflips} = 2n^2$ ) the estimation has to be based on over 1000 sample runs to yield accurate results even for small values of  $\text{Maxflips}$  (where not many runs might succeed). This is due to the fact that for hard instances, relatively few tries succeed at a small number of flips.

For *collections* of 1000 instances, however, a much smaller number of runs (100–200 runs per instance) turned out to lead to accurate results, presumably because the noise for the individual instances cancels out. Figure 5 shows the accuracy of the retrospective analysis for a collection of 1000 instances.

#### Interpretation

The curve in figure 5 shows that 200 runs were a reasonable approximation to find  $\text{Maxflips}^*$  for the instance collection of 1000 instances (of 200 variables, 854

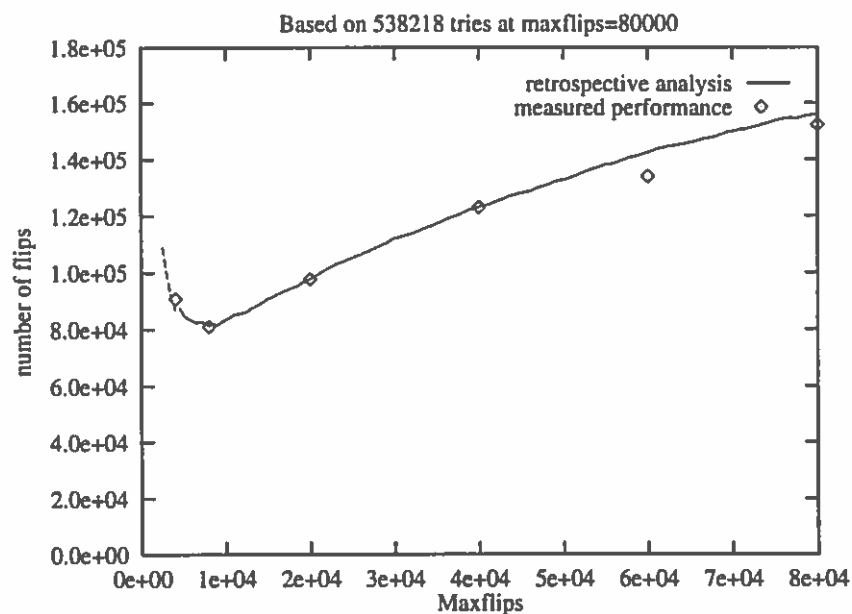


Figure 5: Retrospective analysis for Maxflips, accuracy results. Plotting prediction and actual results for 200 variables, 854 clauses. The prediction is based on 1000 instances, 200 runs per instance. The diamonds are experimental results for the same 1000 instances, 100 runs each (except 80K with 200 separate runs).

clauses). The optimal value of Maxflips is  $\text{Maxflips}^* = 8000$  flips; the minimum is not very sharp. In all subsequent tables 5% intervals will be included that give the range in which  $E_{C,m}$  differs from the optimal value by less than 5%.

An important lesson from the figure is that Maxflips better not be chosen too small since the curve to left of optimal increases very steeply. Thus, it is generally much safer to choose Maxflips too large rather than too small. We observed similarly shaped curves for other problem sizes (between 100 and 400 variables) of Random 3SAT, and for scheduling problems (see page 37).

The solid curve is cut off at the left (4500) for the following reason. Since the sample runs were done at a large value of Maxflips, it happens that although 200 successful tries exist at this value, there are problem instances for which no try



has been successful with less than 4500 flips. Conservatively, the adopted strategy was not to include estimates in this case. The problem can be overcome by either reducing the Maxflips value when collecting the sample runs (thereby forcing more successful tries to occur at smaller  $m$ 's), or to simply collect more runs. The dashed line left of 4500 is a result of collecting more runs (a total of 400 runs per instance).

It could be argued that a collection of 1000 instances is not large enough to estimate Maxflips\* accurately for a given problem size.<sup>5</sup> However, we observed that different instance collections (for the same problem size) yielded values of Maxflips\* that were close together—even when the collections were smaller and  $E_{C,m}$  varied by a factor of 2. Additional evidence for the accuracy of Maxflips\* is that its scaling behavior seems to include a low amount of noise (see figure 7).

### Results on Random 3SAT

In the following, results for a variety of problem sizes for the class Random 3SAT at the crossover point will be summarized. The scaling of WSAT's runtime at the crossover point will be reported in the range between 50 and 400 variables (in steps of 50), on the basis of 1000 instances each. Additionally, the scaling of Maxflips\* with the problem size will be reported, and evidence is provided that it scales quadratically, even for large problem sizes. Also, we will report on series of experiments that varies the clause/variable ratio to study the effect of the constrainedness on WSAT's performance.

---

<sup>5</sup>An argument for this would be that the confidence intervals based on the intra-class variance are indeed very large here.

### Varying the Problem Size

Table 3 gives a summary of the behavior of WSAT as the problem size is varied (for all problem sizes, the experiments were performed at the crossover point). All instance collections used in these experiments were taken from a stream of randomly generated problems. All instances in this stream were tested for satisfiability using a complete method, TABLEAU, and only the satisfiable ones were chosen.

Table 3: WSAT's performance as the problem size is varied. All results at crossover (50% instances unsatisfiable). Given are, in order, the size of the problem class (in variables and clauses), the interval around Maxflips\* in which the runtime varied by less than 5%, Maxflips\*, the optimal expected number of flips, and the confidence interval of the estimation (based on the intra-class variance). The last column contains a coefficient  $e$  that is discussed in section III.2.

Variables	Clauses	5% int. (K)	Maxflips*	Opt.exp.flips	95%-conf.	$e$
25	113	[ .05, .1 ]	70	161	7	0.11
50	218	[ .2, .6 ]	300	868	66	0.12
100	430	[ 1, 3 ]	1,500	6,689	684	0.15
150	641	[ 3, 8 ]	4,600	22,676	2,400	0.20
200	854	[ 5, 11 ]	8,000	83,771	22,312	0.20
250	1066	[ 8, 17 ]	11,000	202,599	80,055	0.18
†300	1279	[ 13, 42 ]	20,000	315,724	62,443	0.22
†350	1491	[ 20, 46 ]	27,000	681,830	193,498	0.22
†400	1704	[ 25, 60 ]	38,000	1,114,301	193,462	0.24

For all problem sizes, experiments were conducted on 1000 instances but with varying precision, mainly for the reason that experimentation at  $n \geq 300$  is computationally expensive. The reason why the experiments stop at 400 variables is that at the time being, no complete method can check much larger problems for unsatisfiability in reasonable time.

Results in the table marked with † are to a small degree less precise than the

other figures since either less runs were performed or a large Maxflips was chosen.<sup>6</sup> However, these variations only affect the per-instance accuracy, and do not appear to have a strong effect on the overall accuracy. The more important limit for accuracy appears to be the number of 1000 instances. Therefore, the confidence intervals are based on the *intra-class variance* and actually reflect the main source of variation. The results of WSAT's scaling are plotted in figure 6.

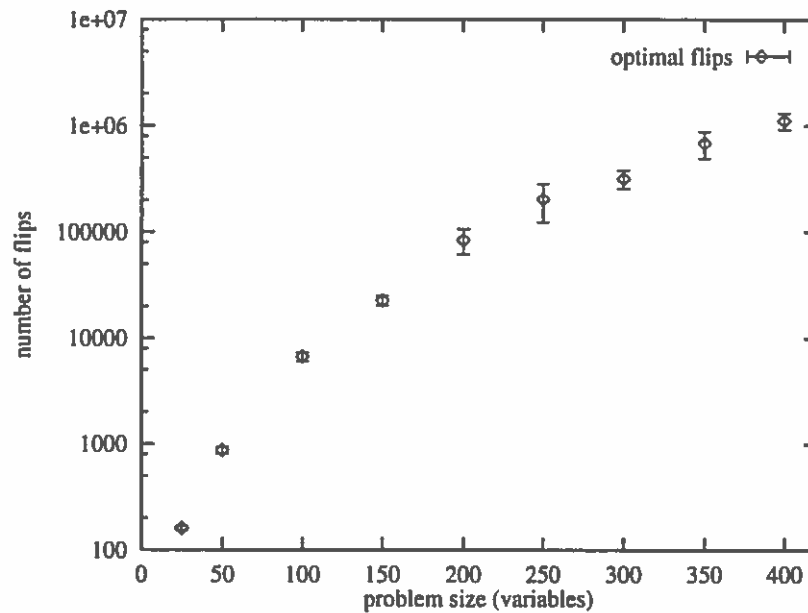


Figure 6: Scaling of WSAT on Random 3SAT.

#### Scaling of Maxflips\*

Recent systematic experiments have suggested (Gent and Walsh, 1993a) that the optimal value of Maxflips scales quadratically with the problem size. However,

<sup>6</sup>Experimentation was started at  $\text{Maxflips} = 2n^2$  to get estimates about the performance variation in an interesting Maxflips range. The intermediate points at  $n = 150, 250, 350$  were performed later at  $\text{Maxflips} = 0.5n^2$ , and since a smaller Maxflips leads to improved accuracy, the number of runs could be reduced in these experiments. As a result, at  $n = 100, 200$  results are based on 400 runs, at  $n = 300, 400$ , results are based on 200 runs, all at  $\text{Maxflips} = 2n^2$ . The next sequence of experiments for  $n = 25, 50, 150, 250, 350$  was performed at  $\text{Maxflips} = 0.5n^2$  with a reduced number of 200 runs; except at  $n = 350$  with 100 runs.

Table 4: Parameters of least squares fit to Maxflips\* scaling.

	value	68.3% conf. interval
$c_1$	0.029	$\pm 0.013$
$c_2$	0.351	$\pm 0.077$

these experiments had only been done systematically for  $n \leq 70$ .

From table 3, we can read the scaling of Maxflips\*. Since Maxflips\* scales almost quadratically, a coefficient  $e$  is computed according to

$$\text{Maxflips}^* = e \cdot n^2,$$

where  $e$  appears to increase slightly with the problem size. Only at  $n = 250$ ,  $e$  is not monotonically increasing; please note that the confidence interval is unusually large at this point also. To obtain the values for Maxflips\*, the minimum of the estimation curve (in steps of 500 flips) was used, as reported in table 3. In figure 7, the scaling of Maxflips\* is plotted, including a least squares fit according to the function

$$om(x) = c_1 \cdot x^{c_2} \cdot x^2.$$

The parameter values ( $c_1$  and  $c_2$ ) are reported in in table 4 as optimized by a nonlinear least squares fit according to the Marquardt-Levenberg algorithm.<sup>7</sup>

---

<sup>7</sup>The fit was done using the Marquardt-Levenberg algorithm as employed by GNUMFIT, an enhancement of GNUPLOT. It has been implemented by Carsten Grammes at the University of Saarbrücken, Germany, 1993.

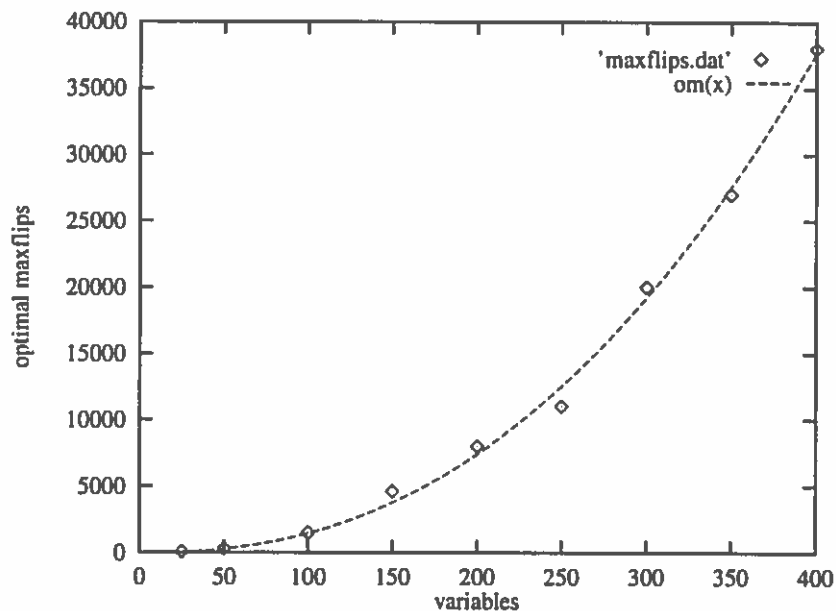


Figure 7: Scaling of optimal Maxflips.

#### Varying the Constrainedness

Table 5 gives results how the performance of WSAT varies with the ratio of clauses/variable. The columns are organized as before in table 3. Again, the results are estimated using retrospective analysis based on each 1000 instances and 200 runs. Figure 8 plots these results in terms of hardness against constrainedness.

Although the confidence intervals are large, it seems likely that the peak of the hardness curve occurs somewhere in the the crossover region. The behavior of Maxflips\* as the clause/variable ratio is varied is not yet clear from these figures. Roughly, Maxflips\* appears to be small in the strongly under-constrained and probably also in the over-constrained region. More careful experimentation is needed, though.

Table 5: WSAT's performance as the constrainedness is varied.

% Unsat	Clauses	5% (K)	Maxflips*	Opt.exp.flips	95%-conf.	$e$
4%	810	[3, 9]	4,500	5,517	29	0.11
9%	820	[4,18]	7,000	7,550	40	0.17
20%	830	[5,11]	6,500	62,563	13565	0.16
30%	840	[7,15]	10,500	60,803	10804	0.26
38%	845	[7,12]	9,000	63,657	8432	0.22
50%	854	[5,11]	8,000	83,771	22312	0.20
60%	860	[5,12]	8,000	74,165	19718	0.20
70%	870	[6,14]	9,000	72,311	10427	0.22
80%	880	[5,11]	7,000	70,658	12360	0.17

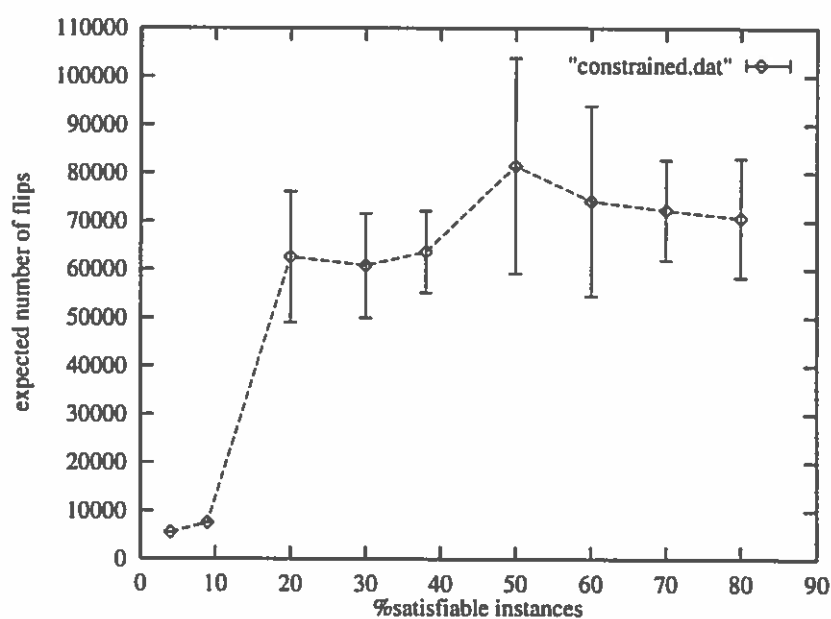


Figure 8: WSAT's performance as the constrainedness is varied.

#### Behavior of the Standard Deviation

In the introductory section on statistics (page 17), it was briefly pointed out that optimal mean does not necessarily mean optimal 'safety' in the sense that at Maxflips\*, outliers might have a stronger influence than at some other Maxflips value. An interesting observation throughout the experiments is that the standard

deviation is minimized *closely* to the optimal mean. Figure 9 plots the retrospective analysis of both the mean and the standard deviation (intra-class) for 200 variables and 854 clauses. Since the standard deviation is influenced by outliers, this effect is very welcome.

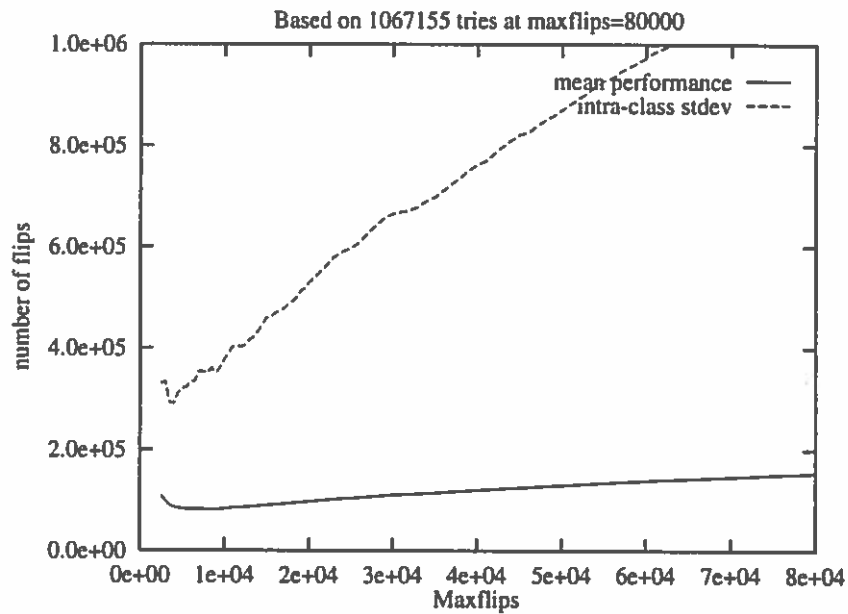


Figure 9: Behavior of the standard deviation as Maxflips is varied. 200 variables, 854 clauses, based 1000 instances each 400 runs. Both mean and standard deviation are shown.

The remaining sections of this chapter report on the results of retrospective analysis as applied to experimentation with WSAT on Random 3SAT and Sadeh's scheduling problems.

### Results on Sadeh's Scheduling Problems

Crawford and Baker report a series of experiments with the goal to determine to which extent Local Search is applicable to scheduling problems (Crawford and Baker, 1994). Runtimes of WSAT and another search procedure, ISAMP, a variant of TABLEAU were compared.

As pointed out earlier, for a fair comparison of WSAT with another algorithm, one might want to compare peak performance. At least in cases where it turns out that Maxflips<sup>\*</sup> can be fixed for a given problem class, the optimized performance is of interest since it can subsequently be used in solving instances in the future. This section reports results for optimized performance of WSAT (with respect to Maxflips) on Sadeh's scheduling problems, again using retrospective parameter optimization, and compares these with results reported in (Crawford and Baker, 1994).

Sadeh's scheduling problems are a test suite of machine shop scheduling problems that are intended to represent a range of the types of problems encountered in the field. To be solved with Local Search as satisfiability problems, (Crawford and Baker, 1994) used a SAT encoding that is due to (Smith and Cheng, 1993). They used the sixty scheduling problems produced by Sadeh (Sadeh, 1992). These problems contain ready times and deadline that were generated randomly using several distributions. The distributions were predefined by two parameters: First, degree of constraint: (w) wide, (n) narrow, and (t) tight, and second the number of bottlenecks: (1) one or (2) two.<sup>8</sup>

Table 6 compares optimized results in K-flips for WSAT with the results that have been reported in (Crawford and Baker, 1994) (column CB94).

---

<sup>8</sup>An additional linear time simplification was performed on the propositionally encoded scheduling problems that consisted of running unit-propagation to completion on the initial theory, and



Table 6: Non-optimized and optimized runtimes of WSAT on Sadeh's scheduling problems. All results are reported in K-flips. The differences for the classes marked with † are explained in a section below.

Class	CB94 (K)	5% range (K)	Maxflips*(K)	Opt.Flips (K)	Stddev
w/1	†390	[ 30, 65 ]	40	33	12
w/2	290	[ 250, 1000 ]	500	204	79
n/1	310	[ 80, 125 ]	105	119	179
n/2	550	[ 700, 1000 ]	950	540	633
t/1	1,100	[ 100, 140 ]	125	634	1,357
t/2	†2,900	[ 3125, 4000 ]	3200	12,718	36,291

#### Interpretation

Obviously, problem classes with two bottlenecks are more difficult and result in higher runtimes of WSAT. It is also noticeable that Maxflips\* is consistently small for the problem classes that have *one* bottleneck and larger for the classes with *two* bottlenecks. For all one-bottleneck classes, a speedup of about a factor of two could be achieved using the optimization. For the two-bottleneck classes, it made less than a 5% difference to fix Maxflips to the upper interval limit (1000 K-flips for 'w' and 'n', 4000 K-flips for 't').

In the propositional encoding, all problems contain between 5000 and 6000 variables. Clearly, although all problems have about the same number of variables, Maxflips\* varies largely. This shows that Maxflips\* depends to a large degree on the particular problem rather than just its number of variables. Noticeable is that the value of Maxflips\* for a 5000 variable Random 3SAT problem would be 5,500 K-flips, and that all classes of Sadeh's problems require a smaller Maxflips.

Figures 10 and 11 each summarize the effect of varying Maxflipson expected performance (for problem classes having 1 or 2 bottlenecks respectively).

---

deleting clauses that were subsumed by a unit literal.

### Notes on Differences for w/1 and t/2

*w/1*: In a direct comparison between the results on the class *w/1*, using the same implementation of WSAT and the same problem instances, a mean of 89K-flips was obtained as opposed to 390K-flips as reported. So far, we could not find an explanation for this improvement.

*t/2*: The result in column CB94 is smaller because (Crawford and Baker, 1994) introduced a maximum time bound beyond which runtime results did not influence the mean.<sup>9</sup>

### Conclusion

Comparing the optimized behavior with the reported results for WSAT and ISAMP, we obtain the runtimes in table 7. We conclude that Maxflips optimization improved on the problems with one bottleneck but to a lesser extent for the two bottleneck problems, since the reported results were already almost optimal. Although WSAT looks somewhat better now, ISAMP is still better on 'n' and 't' problems.

---

<sup>9</sup>The set divides up in one hard instance and 9 easy instances. The mean of the 9 easy instances was approximately 1.3mio. The results in (Crawford and Baker, 1994) were computed with Maxflips = 4mio and a maximum of 10 tries. What exceeded this limit did not influence the mean. Around 75% of the runs on the hard instance exceed this limit. Therefore, the *expected* percentage would be 93% compared to 97%, as reported in their paper. With some luck (on only ten performed experiments per instance quite possible), 97% might have happened. The mean of the successful runs with at most 40mio flips (the time out) is 18mio. Therefore, in the 97% case, the expected mean would be:  $(1.3M \cdot 90 + 18M \cdot 7)/97 = 2.5M$  which is indeed close to the reported value.

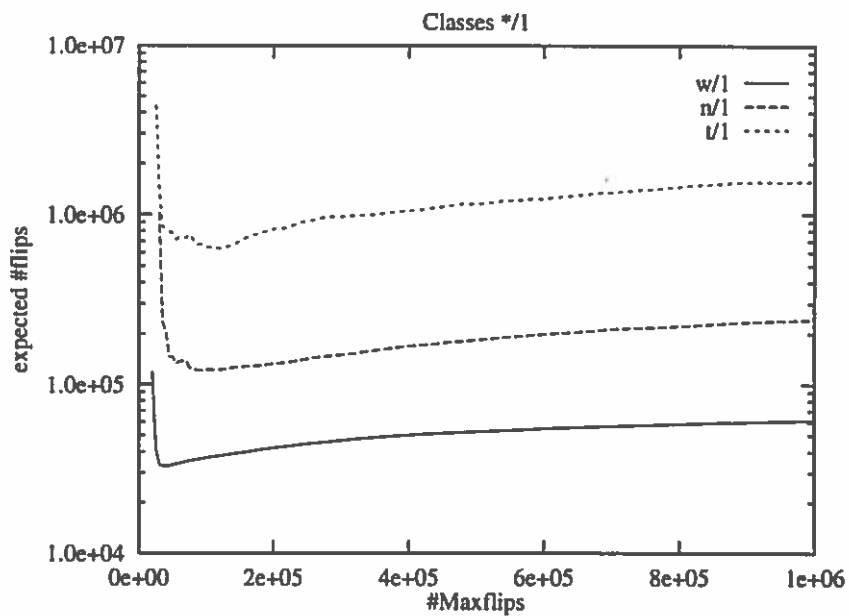


Figure 10: Varying Maxflips on Sadeh classes with one bottleneck.

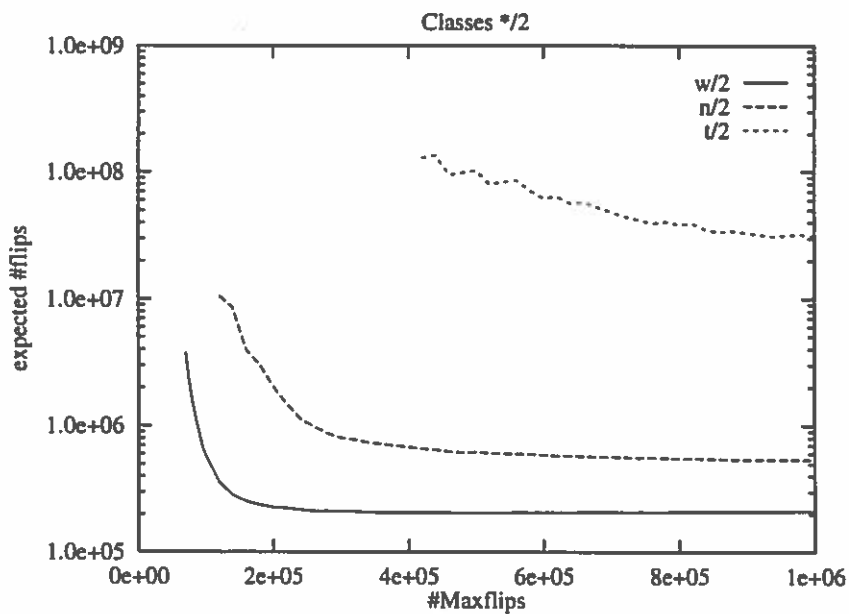


Figure 11: Varying Maxflips on Sadeh classes with two bottlenecks

Table 7: Comparison of various procedures on Sadeh's scheduling problems. Maxflips optimized and non-optimized. All times in seconds, at a fliprate of approximately 14 K-flips/s.

Class	WSAT, CB94	WSAT Maxflips*	ISAMP
w/1	(27)	2	7
w/2	23	15	10
n/1	23	8	8
n/2	43	38	15
t/1	77	45	16
t/2	(211)	908	43

#### Strategies with Non-Constant Maxflips

The previous section showed how to optimize the parameter Maxflips empirically. The obvious next question is how a strategy would do that does not fix Maxflips, but instead uses a non-constant Maxflips-sequence. Let

$$\mathcal{M} = (m_0, m_1, m_2, \dots)$$

be a Maxflips-sequence for which WSAT is run with Maxflips =  $m_0$  in the first try, then independently with Maxflips =  $m_1$  in the second try, and so on. Having knowledge about the success-distributions for all problem instances of a specific collection, what is the optimal strategy of choosing the cutoffs  $m_0, m_1, m_2 \dots$ ?

For the class of Las Vegas algorithms, this question has been examined by (Luby et al., 1993). For a single problem instance with full knowledge about the distribution of the algorithm, they show that fixing the cutoff at  $m_0 = m_1 = \dots = m^*$  is always optimal. However, if a *collection* of instances is considered, these results do not directly apply anymore.

Using a generalized retrospective analysis for a Maxflips-sequence, we evaluated

various sequences empirically: The expected number of flips for a given Maxflips-sequence  $\mathcal{M}$  can be estimated as

$$E_{\mathcal{M}} = \sum_{i=0}^{\infty} ((S_0^{m_i} + \sum_{j=0}^{i-1} m_j) \cdot p_{m_i} \cdot \prod_{j=0}^{i-1} (1 - p_{m_j}))$$

In all our experiments (on different instance collections of Random 3SAT), we could not find a sequence that improved over a constant optimal Maxflips.

### Summary and Future Work

This chapter has introduced retrospective analysis as a tool for the optimization of statistical measures of the performance of Local Search algorithms. The technique has been applied to optimizing mean performance of WSAT on Random 3SAT, and scaling figures have been reported. Further evidence has been provided that optimal Maxflips scales quadratically with the problem size. Additionally, we have investigated the possible speedup due to Maxflips optimization for the various classes of Sadeh's scheduling problems.

Future work related to this chapter could be the following. Reasonable curve fits should be investigated that approximate WSAT's scaling on Random 3SAT. Additionally, a possible criticism with respect to the accuracy of the scaling results has been (Crawford, 1995a; Parkes, 1995) that 1000 problem instances might be too few as a basis for scaling results since outliers might appear that are extremely hard. A larger number of problem instances could also be considered to improve the confidence intervals of WSAT's performance profile as the constrainedness is varied. An open question is whether there is a general relation between the constrainedness of a problem and the optimal cutoff time. An interesting theoretical question imposed by figure 9 is for the relation between the optimal mean and the standard deviation.

## CHAPTER IV

## PARALLELIZATION OF LOCAL SEARCH

The previous chapter examined the question of finding the optimal cutoff point (Maxflips\*) for a sequential Local Search procedure. In this chapter, parallelization of Local Search is considered. As in the previous chapter, a retrospective analysis is employed here to examine the speedup from parallelization.

This chapter is concerned with examining how *efficient* parallelization of Local Search is, i.e. what the expected speedup is from employing  $k$  processors. We investigate a parallel strategy that was analyzed for the class of Las Vegas algorithms by (Luby et al., 1993).<sup>1</sup>

This parallel strategy, *uniform repeating strategy*, will be investigated empirically regarding WSAT on the problem space Random 3SAT. It will be shown that parallelization of WSAT according to this strategy leads to an almost linear speedup for a moderate number of processors. Furthermore, it will be shown that as the problem size gets larger, the almost linear speedup from adding processors persists longer.

---

<sup>1</sup>For Las Vegas algorithms, (Luby and Ertel, 1993) have shown that even with full knowledge about the distribution, superlinear speedup cannot be expected from parallelization, i.e. using  $k$  processors, the speedup can at most be  $k$ . For any Las Vegas algorithm, (Luby and Ertel, 1993) have shown that the uniform repeating strategy is within a constant factor of *optimal*, provided that the distribution is known.

### Uniform Repeating Strategies

In the uniform repeating strategy, all processors use the same fixed cutoff  $\text{Maxflips} = m$ , as shown in figure 12.

Proc#	Time →			
1	$m$	$m$	$m$	...
2	$m$	$m$	$m$	...
3	$m$	$m$	$m$	...
⋮		⋮		

Figure 12: Uniform repeating strategy.

In the following, a version of WSAT is presented that encodes the uniform repeating strategy. A retrospective analysis will be developed that simulates a repeating uniform strategy of  $k$  processors, all using  $\text{Maxflips} = m$ . As before, the retrospective approach uses a collection of runtime samples (that approximate the success-distribution) as the basis for estimating the effect of varying the parameters.

In the next section, the analysis will be applied to analyze the speedup of WSAT due to parallelization and at various values of  $\text{Maxflips}$ . In figure 13, a parallelized version of WSAT, PWSAT, is given.

### Retrospective Analysis for Parallelization

PWSAT has three explicit parameters:  $\text{Maxtries}$ ,  $\text{Maxflips}$  and  $\text{Numthreads}$ . The following analysis calculates the expected runtime (in number of flips) as these explicit parameters are varied. Since PWSAT is a semi-decision procedure, the performance analysis will be limited to the case of satisfiable input formulas. Therefore,  $\text{Maxtries}$  is set to  $\infty$ . Additionally, what has been a *try* in WSAT now becomes a

```

1 proc PWSAT
2   Input a set of clauses  $\alpha$ , Maxtries, Maxflips, and Numthreads
3   Output a satisfying total assignment of  $\alpha$ , if found
4    $\equiv$ 
5   for  $i := 1$  to Maxtries do
6     par  $k \in \{1, \dots, \text{Numthreads}\}$  do
7        $A_k :=$  random truth assignment
8     end
9     for  $j := 1$  to Maxflips do
10      par  $k \in \{1, \dots, \text{Numthreads}\}$  do
11        if  $A_k$  satisfies  $\alpha$  return  $A_k$ 
12         $C_k :=$  random clause, unsatisfied by  $A_k$ 
13        with probability  $p$  :  $P_k :=$  random variable in  $C_k$ 
14          probability  $1 - p$  :  $P_k :=$  best variable in  $C_k$ 
15         $A_k := A_k$  with  $P_k$  flipped
16      end
17    end
18    return  $No$ 
19  end
20  return  $No$ 
21 end
22 end

```

Figure 13: The parallel WSAT procedure(PWSAT).

*parallel try* of the  $k$  processors in PWSAT. Since the analysis is primarily interested in the expected clock time of PWSAT,  $k$  parallel flips will be counted as one flip.

The formalization of the analysis divides up into two parts. The first part assumes that PWSAT does not perform restarts, i. e.  $\text{Maxflips} = \infty$ . The second part will additionally take Maxflips into consideration and give estimates of PWSAT's performance as both parameters, Maxflips and Numthreads, are varied.



### Infinite Maxflips

An expected runtime estimate for PWSAT can be given using a set  $S_0$  of sample tries as before. In the following, the expected runtime (number of flips) will be estimated for PWSAT with Numthreads =  $k$ .

#### Definitions:

- As before, let  $S_0$  be a set of  $n$  sample tries. This time, however, we require  $S_0$  be sorted: let  $s_i$  be the  $i$ -th smallest sample in  $S_0$ .

$$S_0 = \{s_1, s_2, \dots, s_n \mid s_1 < s_2 < \dots < s_n\}$$

This section will first give results provided that no two samples have the same value; i. e. we require  $S_0$  be strictly ordered. In the next section, it will be shown that this result remains correct in the case where  $S_0$  contains multiple instances of the same element, i. e.  $S_0$  will become a multi-set.

- Let  $S$  be a random variable that can assume any value in  $S_0$  with equal probabilities. We can estimate the probability for an arbitrary choice of  $S_0$  being greater than a specific  $s_i$  (for any  $i$ ) by

$$P(S \geq s_i) = \frac{n - i + 1}{n}.$$

- Let  $T_k(S_0)$  denote a (fixed) random choice of  $k$  elements from  $S_0$  (allowing repetitions):

$$T_k(S_0) \in \underbrace{S_0 \times S_0 \dots \times S_0}_k$$

Thus we can treat  $k$  processors like  $k$  random choices from the sample set  $S_0$ , or like one random choice of  $T_k(S_0)$  from the permutations of  $S_0$ .<sup>1</sup>

Provided that a parallel try is *successful* and  $k$  processors run in parallel, the number of flips it takes is the smallest number of flips for any of the  $k$  processors, since the first successful processor will terminate the search.

The expected number of flips it takes for a successful parallel try can be denoted by a new random variable

$$\mathcal{T}_k := \min T_k(S_0).$$

To derive the probabilities for the random variable  $\mathcal{T}_k$ , we need to estimate the probability for any  $s_i \in S_0$  being the minimal element in a random choice  $T_k(S_0)$ :

$$\begin{aligned} P(s_i = \min T_k(S_0)) &= P(\forall x \in T_k(S_0) : x \geq s_i \wedge \neg \forall x \in T_k(S_0) : x > s_i) \\ &= \left(\frac{n-i+1}{n}\right)^k - \left(\frac{n-i}{n}\right)^k \end{aligned}$$

With the above estimate of the probabilities, we can give the expected number of total flips if  $k$  processors are running in parallel. The expected minimal number of flips when running  $k$  processors is

$$\begin{aligned} E[\mathcal{T}_k] &= \sum_{i=1}^n s_i \cdot P(s_i = \min T_k(S_0)) \\ &= \sum_{i=1}^n s_i \left[ \left(\frac{n-i+1}{n}\right)^k - \left(\frac{n-i}{n}\right)^k \right]. \end{aligned} \quad (\text{IV.2})$$

---

<sup>1</sup>Although  $T_k(S_0)$  is a tuple, we will use the  $\in$ -relation on it with the obvious semantics.

This equation gives the expectation as the weighted sum of all the sorted sample tries. Please note that we have ignored the Maxflips parameter up to this point.

### Correctness for multisets

As mentioned above, it is not obvious that above equation also holds in the case where  $S$  is a *multi-set*. Hence, we have to prove that the calculation of  $E[T_k]$  remains correct in this case. The order is loosened and the set is allowed to contain elements more than once,

$$S_0 = \{s_1, s_2, \dots, s_n \mid s_1 \leq s_2 \leq \dots \leq s_n\}.$$

As above, the random variable  $S$  can assume any value from  $S_0$ . However, the probabilities  $P(S \geq s_i)$  and  $P(S > s_i)$  can no longer be computed as above, since the duplicates have to be accounted for by appropriate counting.

Nevertheless, it will next be shown that equation (IV.2) remains correct in this case.

**Fact 1** *For every selection of samples*

$$s_1 \leq \dots \leq s_{p-1} < s_p = s_{p+1} = \dots = s_i = \dots = s_{p+q}$$

(and either  $p+q = n$  or  $s_{p+q} < s_{p+q+1}$ ), equation (IV.2) gives the correct expectation, since independent of the  $s_1, \dots, s_{p-1}$ , the following equivalence holds:

$$P(s_i = \min T_k(S_0)) = \sum_{j=p}^{p+q} \left[ \left( \frac{n-j+1}{n} \right)^k - \left( \frac{n-j}{n} \right)^k \right] \quad (\text{IV.3})$$

Proof: The above probabilities  $P(S \geq s_i)$  and  $P(S > s_i)$  change to the following:

$$\begin{aligned} P(S \geq s_i) &= \frac{n - p + 1}{n} \\ P(S > s_i) &= \frac{n - p - q}{n} \end{aligned}$$

Using these probabilities, we can recompute

$$\begin{aligned} P(s_i = \min T_k(S_0)) &= P(\forall x \in T_k(S_0) : x \geq s_i \wedge \neg \forall x \in T_k(S_0) : x > s_i) \\ &= \left(\frac{n - p + 1}{n}\right)^k - \left(\frac{n - p - q}{n}\right)^k \end{aligned} \quad (\text{IV.4})$$

Now, the equivalence (IV.3) follows directly, since its right hand sum is equivalent to (IV.4) because of a telescopic sum that collapses. Hence, we have proven that we can still use (IV.2) to estimate the multiset case.  $\square$

### Varying Maxflips and Numthreads Simultaneously

We will continue the estimation by setting Maxflips to  $m < \infty$ . Thus, in PWSAT all processors will be restarted after each has made  $m$  flips. In each *parallel try*, we encounter the following two possible cases:

- (i) None of the  $k$  processors finishes (using less than  $m$  flips). We call this the *loose* condition and formalize it as  $L := \forall x \in T_k(S_0) : x > m$ .

$$P(L) = \left(\frac{n - |S_0^m|}{n}\right)^k =: p_{loose}$$

- (ii) At least one processor finishes, using at most  $m$  flips. We call this the *win* condition  $W := \neg \forall x \in T_k(S_0) : x > m$ .

$$P(W) = 1 - p_{\text{lose}} =: p_{\text{win}}$$

We can now derive the respective two conditional expectations as follows.

- (i) If none of the  $k$  processors finishes with less than  $m$  flips, it simply takes  $m$  flips for one parallel try, and then a restart takes place.

$$E[T_k|L] = m$$

- (ii) If at least one processor finishes with at most  $m$  flips, we have to modify  $E[T_k]$  from the previous section such that it reflects the win condition  $W$ .

The conditional probability  $P(A|B)$  is defined as  $P(A|B) = P(A \cap B)/P(B)$ .

We are interested in

$$P(s_i = \min T_k(S_0) \mid \exists x \in T_k(S_0) : x \leq m). \quad (\text{IV.5})$$

In our case we know that for all  $s_i \in S_m$ ,  $A$  implies  $B$ , since if  $s_i \leq m$  is minimal in  $T_k(S_0)$ , it is certain that at least one  $x \leq m$  exists in  $T_k(S_0)$ . For this reason, (IV.5) is equivalent to:

$$\frac{P(s_i = \min T_k(S_0))}{P(\exists x \in T_k(S_0) : x \leq m)} = \frac{P(s_i = \min T_k(S_0))}{p_{\text{win}}}$$

Additionally, we must correct the upper bound of the sum: if at least one processor finishes with at most  $m$  flips, it is not possible that any  $s_i > m$

is the smallest element in  $T_k(S_0)$ . Only the first  $|S_0^m|$  elements influence the expectation:

$$E\{T_k|W\} = \frac{1}{p_{win}} \cdot \sum_{j=1}^{|S_0^m|} s_j \left[ \left( \frac{n-j+1}{n} \right)^k - \left( \frac{n-j}{n} \right)^k \right] \quad (\text{IV.6})$$

We can now combine the two conditional expectations, which finally leads to the expected number of flips for any given combination of  $m$  Maxflips and  $k$  processors, for a single problem instance.

$$\begin{aligned} E_{m,k} &= P(W) E\{T_k|W\} + P(L) P(W) (E\{T_k|L\} + E\{T_k|W\}) + \dots \\ &= p_{win} \sum_{i=0}^{\infty} p_{loose}^i \cdot (i m + E\{T_k|W\}) = \dots \\ &= m \cdot \frac{1 - p_{win}}{p_{win}} + E\{T_k|W\} \end{aligned} \quad (\text{IV.7})$$

with  $E\{T_k|W\}$  as in (IV.6) and using the same argument to reduce the sum as for equation (III.1).  $\square$

As before, we estimate the mean for a collection of instances by computing the mean of the all instances in the collection.

For a practical estimation of PWSAT, the same counting scheme as in the previous chapter (page 27) has been applied. Please note that, as in the previous chapter, no estimates can be given for any Maxflips-value that is larger than the one that has been used to collect the sample tries.

## Results

We are now in a position to give results using retrospective analysis based on collected runtime samples. First, figure 13 empirically demonstrates the accuracy of the retrospective analysis for an interval between 1 and 100 processors. The estimation curve for  $n = 200$  variables is based on a collection of 1000 instances at crossover and 400 runs on each instance. The diamonds in the figure reflect the mean of each 100 actual runs of PWSAT (on the same 1000 instances).<sup>2</sup>

Figures 13 and 14 plot the expected speedup factor due to parallelization in PWSAT on Random 3SAT. The interpretation of the curves is as the follows: A factor of 20 at a particular  $k$  means that running PWSAT with the (sequentially optimal)  $m = \text{Maxflips}^*$  and  $k$  processors is expected to run in 1/20th of the time of sequential WSAT. Thus, the straight line given by the function  $f(x) = x$  would mark a speedup that is exactly linear in the number of processors.

## Interpretation

The first observation of the results is that PWSAT is indeed efficient: An almost linear relative speedup arises from parallelization on Random 3SAT, provided that the number of processors is moderate. Further, both figures 13 and 14 suggest that the larger the problem size, the longer the speedup from parallelization can be expected (as more processors are added).

In addition, figure 13 suggests that the speedup converges as the number of processors is getting fairly large. This result is consistent with the results in (Luby and Ertel, 1993).<sup>3</sup>

---

<sup>2</sup>PWSAT was implemented sequentially in lock-step mode.

<sup>3</sup>However, no attempt has yet been made to carefully examine the distribution of WSAT on

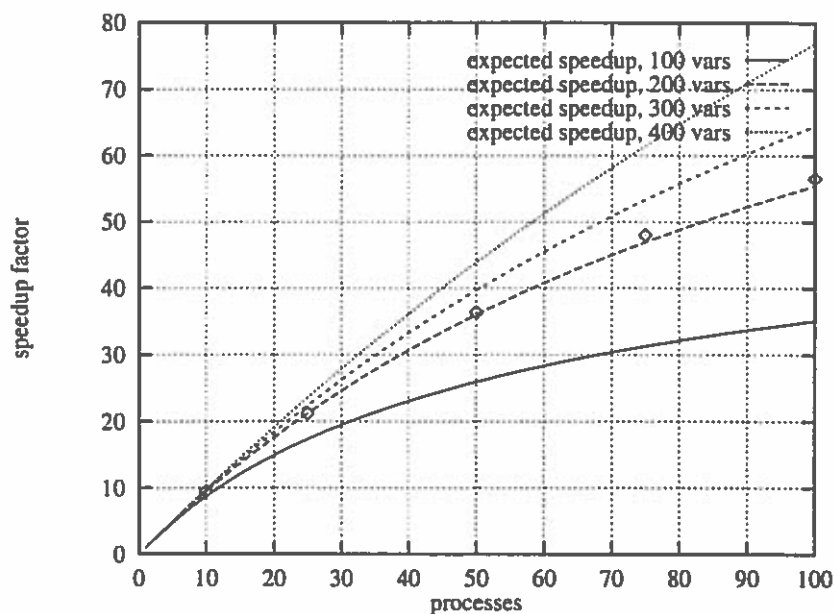


Figure 13: Expected speedup due to parallelization for  $n = 100, 200, 300, 400$  variables in the range of 1–100 processors.

### Parallelization with Suboptimal Maxflips

Figure 15 plots the speedup from parallelization, considering several values for Maxflips, 8K, 30K and 80K. All speedups are relative to the *sequential* case with optimal Maxflips ( $\text{Maxflips}^* = 8\text{K}$ ). From this figure (and similar ones for other problem sizes), it appears that on Random 3SAT, the optimal sequential value  $\text{Maxflips}^*$  is optimal also for the parallel strategy, independent of the number of processors.<sup>4</sup> This suggests a two step optimization: First, Maxflips can be optimized in the sequential case, and second parallelization can be considered.

---

Random 3SAT, whether it meets the criterion given in (Luby and Ertel, 1993) under which an almost linear speedup is to be expected.

<sup>4</sup>The graph does not contain a result for  $m < \text{Maxflips}^*$  because it would become less clear. However, at smaller Maxflips there is also less speedup from parallelization.



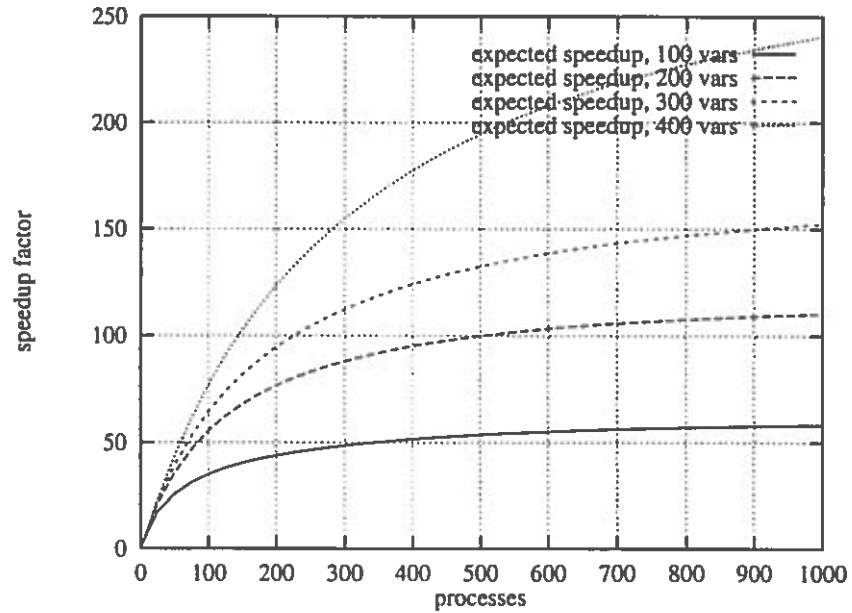


Figure 15: Expected speedup due to parallelization for  $n = 100, 200, 300, 400$  variables in the range of 1–1000 processors.

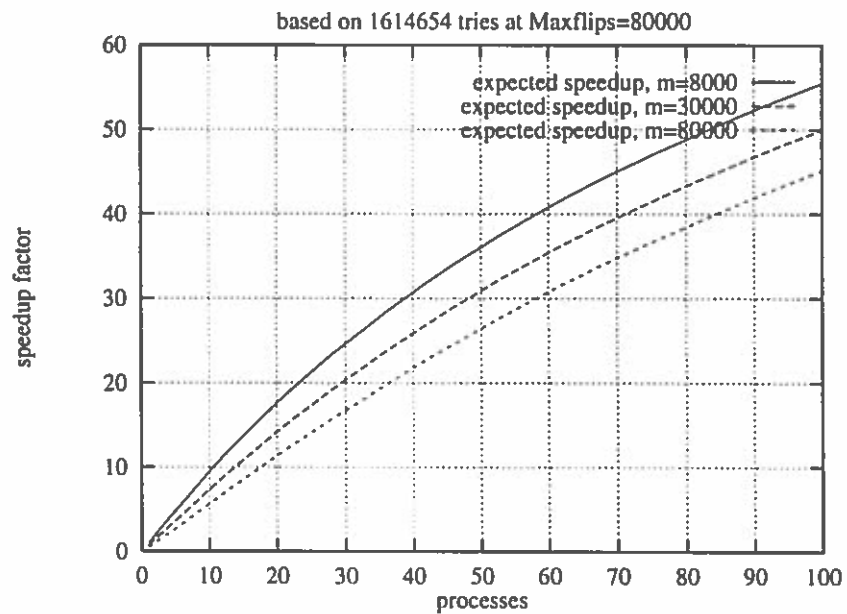


Figure 16: Expected speedup due to parallelization for various values of Maxflips. 200 variables, 854 clauses. All speedups are relative to the sequential performance at Maxflips\*.

### Summary and Future Work

This chapter has examined the speedup from parallelization of Local Search. A retrospective analysis has been introduced for simulating parallel behavior from collected runtime data, reducing the computational cost for experimentation. It has been shown that parallelization of Local Search is efficient, and accurate estimates of the quantitative speedup of WSAT for Random 3SAT have been given. Parallelization of Local Search is of practical interest since the routines can be parallelized easily. If parallelization turns out to be as efficient for realistic problem classes, it might be an attractive alternative to refinements of the sequential procedures in practical applications.

Future work could include an investigation of the speedup from parallelization of Local Search procedures on other problem classes, and the behavior of tail probabilities could be examined. Additionally, intelligent parallel strategies along the lines of (Aldous and Vazirani, 1994) could be considered in which the set of processes is viewed as a population and is globally controlled.

## CHAPTER V

### REFINEMENTS OF WSAT

This chapter reports on refinements of WSAT's strategy. The first refinement tackles a weaknesses of Local Search that has recently been addressed by (Crawford, 1995b): Local Search suffers from a *lack of forward propagation*. The first section incorporates a simple implicit mechanism for forward propagation into WSAT that propagates values through binary clauses.

Additionally, a subtle difference between two implementations of WSAT will be discussed, and it will be shown that subtle details of WSAT's strategy matter for optimal performance.

#### Exploiting Structure in Sadeh's Problems

In chapter on page 37, experimental results of WSAT on Sadeh's scheduling problems have been reported. Taking a closer look at the booleanization of these problems due to (Smith and Cheng, 1993), one can frequently see long *chains* of binary implications. By a *chain of binary implications*, or *chain*, we mean a sequence of implications with the semantics

$$l_1 \rightarrow l_2 \wedge l_2 \rightarrow l_3 \wedge \dots \wedge l_{n-1} \rightarrow l_n$$

In this chain of literals, setting one literal  $l_i$  to *true* forces all subsequent literals to the right of  $l_i$  to be *true*. Setting a literal to *false* forces all literals to the left to be

*false*. This implies that there must be a frontier in each chain, to the left of which all literals are *false* and to the right of which all literals are *true*. Syntactically, these chains appear in the Sadeh problems as a collection of binary clauses:

$$(\overline{x_1}, x_2) (\overline{x_2}, x_3) (\overline{x_3}, x_4) \cdots (\overline{x_{n-1}}, x_n)$$

Each chain typically occurs with 100–130 different variables in binary clauses. In total, there are typically about 50 chains in each instance of Sadeh's problems.

#### Origin of Binary Implication Chains

It is important to note that most variables within chains appear in other clauses of the theory as well, thus they cannot be reduced in a simple preprocessing step. The chains appear to be a result of the encoding of coherence conditions. For instance, in the booleanization there exist variables  $sa_{i,t}$  that express that an operation  $i$  starts at time  $t$  or later, and  $eb_{i,t}$  meaning operation  $i$  ends by time  $t$ . If operation  $i$  requires processing time  $p_i$  (given as part of the problem), one type of coherence condition has to ensure that if  $i$  starts at or after time  $t$ , it cannot end before time  $t + p_i$ . Therefore, the following condition is stated over all relevant operations  $i$  and all relevant times  $t$ :

$$sa_{i,t} \rightarrow \neg eb_{i,t+p_i-1}$$

Coherence conditions like this one seem to be the origin of the binary implication chains. Because they express coherence conditions, the chains should be kept consistent with the variables that express the decisions being made, e. g. variables that tell in which order to schedule two particular operations.

With some understanding of the nature of randomized Local Search, it is clear that these structures are not easily dealt with by WSAT, since the local gradient (the number of unsatisfied clauses) treats all variables equally. Figure 16 shows WSAT's behavior on a specific chain; one might describe it as *non-resolute* for the nature of the implication chains is mostly ignored.

#### Motivation—USAT

The need for a general propagation mechanism in Local Search was recently addressed by (Crawford, 1995b) and lead to the procedure USAT. USAT does randomized Local Search with incorporated *unit propagation*.

USAT has the notion of *control variables*; a set of variables that trigger changes in other variables. Each control variable, if modified, propagates changes to a number of *dependent variables* in other clauses. Intuitively, control variables should be those variables that reflect actual decisions being made (e.g. the order in which to schedule two operations), whereas dependent variables merely account for consistency (e.g. coherence conditions). Unfortunately, on the level of the boolean encoding of a problem, it is not clear which variables are control variables. Practically, any variable can be treated as control variable, however, some variables are better candidates than others because they have more dependent variables and thus allow for more unit propagations.

USAT has been shown (Crawford, 1995b) to exploit a similar structure of a specific problem class, *loopy 3SAT*. On this class, the speedup in the number of flips reaches more than four orders of magnitude (compared to WSAT). However, the associated improvement in time is only about one order of magnitude on this class.

An open research question at present is how USAT can be implemented effi-

ciently. An important concern in the implementation of Local Search procedures is the efficiency of a single flip—and WSAT's flip rate is with 14K-flips/s on a Sparc10<sup>1</sup> strikingly high. With complex enhancements it happens that, although the number of flips can be reduced, the overall time may increase due to the cost that is added to each individual flip.

### Implicit Propagation

The strategy that USAT takes is an *explicit* combination of hillclimbing and unit propagation. In contrast, the approach taken here is an *implicit* propagation by changing the heuristic strategy. Note that the implicit propagation that will be proposed here is not as powerful as USAT since propagation is only done through binary clauses.

#### Implicit Unit Propagation through Binary Clauses

The main objective of the propagation is to keep binary implication chains consistent with a set of control variables—the set of variables that occur in non-binary clauses. One way to achieve this goal is by controlling WSAT's variable flips explicitly with a mechanism that keeps track of the state of each chain. However, an implementation can get complicated since variables can occur in multiple chains and the order in which propagations are to be done has to be decided. Additionally, for an explicit mechanism the chains have to be isolated in a preprocessing step.

The following implicit strategy, which was suggested by Andrew Parkes in a discussion about explicit control, is a combination of two existing techniques: A *history mechanism* and a *binary preference* strategy. Both strategies are known

---

<sup>1</sup>83K-flips/s on a Silicon Graphics Power Challenge.

for SAT testing, however, only the combination achieves the goal of keeping binary chains consistent.

The overall goal is to (i) define a frontier in each chain and (ii) be *resolute* when moving the frontier, in the sense that once a chain becomes inconsistent (because a control variable has been flipped), reestablishing its consistency should occur without changing the propagation direction.

1. *Binary preference strategy*: Achieving (i) is straightforward. Preferably satisfying binary clauses (before trying to satisfy larger clauses) will achieve the goal of keeping the chains consistent with the rest of the theory. Of course, in the process of repairing binary clauses, it may happen that other clauses (binaries and non-binaries) break.
2. *History mechanism*: To achieve (ii), a history mechanism that was introduced as HSAT by (Gent and Walsh, 1993a) is applied. Given the choice of two “best” variables, the one is flipped that hasn’t been flipped the longest time ago. This encourages that the propagation direction within a chain not be changed.<sup>2</sup> HSAT is an enhancement of GSAT. Given the choice of two variables to flip, HSAT chooses to flip the variable that has been flipped least recently. This same principle was incorporated into WSAT here.

### HWSATB

This combination leads to HWSATB, which will be described in the following in two separate parts.

---

<sup>2</sup>Using a tabu list in WSAT has anecdotally been reported by Bart Selman to strongly improve WSAT’s performance on loopy 3SAT, a problem class that has a similar structure. How the behavior of a tabu list differs from a history mechanism is an open question.

- (i) *Binary preference.* In the implementation chosen for HWSATB, separate lists are kept for unsatisfied *binary clauses* and unsatisfied *non-binary clauses*. With a probability  $q$ , an unsatisfied binary clause is picked, and with probability  $1-q$ , an unsatisfied non-binary clause is picked. A variable from the picked clause is then flipped.
- (ii) *History mechanism.* The history mechanism is the adaptation of HSAT's strategy to WSAT (with the exception that it is not deterministic like HSAT).

```

1 proc HWSAT-variable-selection
2   Input a set of clauses  $\alpha$ 
3   Output a variable to flip
4    $C :=$  random unsatisfied clause
5   with probability  $p$  :
6      $P :=$  least recently flipped variable in  $C$ 
7     probability  $1 - p$  :
8        $P :=$  the variable in  $C$  that
9         (i) increases the score the most, and
10        (ii) among these has been flipped least recently
11   end
12   return  $P$ 
13 end

```

### Behavior on chains

Figures 16 and 17 show how WSAT and HWSATB flip variables in binary chains. We plot variable assignments for all variables in a particular chain as variables are flipped. The vertical axis reflects time, the  $\Delta t$  column shows how many flips occurred since the last change of any variable in the chain. This particular chain occurred in the hardest class  $t/2$  of the Sadeh problems. One can observe that WSAT is indeed non-resolute and more often produces holes within the chain, which are eliminated again in the next step. Thus, although it seems to achieve consistency in the chain before it performs many flips in other clauses, it is non-resolute.





Sadeh's problems (table 6), we notice that the exact value of Maxflips is much less important using HWSATB, since the 5% intervals get significantly larger. Additionally, we observe a speedup of HWSAT over WSAT which is best on class  $n/2$  with about a factor of 7. The performance (in seconds) is reported in table 10.<sup>3</sup>The maximal speedup of HWSATB over WSAT is about a factor of 18 on class  $t/1$ . Unfortunately, on the hard class  $t/2$ , the speedup is only about a factor of 2. An open question at present is why HWSATB does not improve as well on class  $t/2$  as it does on the other classes.

To conclude, we note that implicit propagation was indeed effective to exploit a structural feature of scheduling problems. We have shown that exploitation of subtle structures in the problem encoding can be beneficial.

Table 9: Results for HWSAT on scheduling problems

Class	5% range (K)	Maxflips*(K)	Opt.exp.flips (K)	Stddev. (K)
w/1	[ 100, $\infty$ ]	>100	19	5
w/2	[ 100, $\infty$ ]	>100	27	8
n/1	[ 200, $\infty$ ]	>200	52	68
n/2	[ 150, $\infty$ ]	>150	71	74
t/1	[ 250, 850 ]	400	215	381
t/2	[ 450, 600 ]	450	3,088	7,584

### HWSAT on Random 3SAT

To investigate HWSAT further, table 11 contains results for HWSAT on Random 3SAT, based on the same collection of problem instances. HWSAT is almost an

---

<sup>3</sup>The results were obtained at optimal Maxflips with the following flip rates on a Silicon Graphics Power Challenge: WSAT: 83K-flips/s, HWSAT: 70K-flips/s, HWSATB: 64K-flips/s. The HWSATB rate is a lower limit, measured on a theory without binary clauses. Times do not include loading time which for these problems were often longer than the runtimes.

Table 10: Summary of results on scheduling problems.

WSATVersion	w/1	w/2	n/1	n/2	t/1	t/2
K-flips						
WSAT	33	204	119	540	634	12,718
HWSAT	19	27	52	71	215	3,088
HWSATB, $q = 1$	11	15	17	52	34	5,685
Seconds						
WSAT	0.4	2.5	1.4	6.5	7.6	153.2
HWSAT	0.3	0.4	0.7	1.0	3.1	44.1
HWSATB, $q = 1$	0.2	0.2	0.3	0.8	0.5	84.4

Table 11: Results of HWSAT on Random 3SAT. Results from retrospective analysis, 200 variables, 854 clauses

Version	5% int. (K)	Maxflips*	Opt.exp.flips	95%-conf.
WSAT	[ 5, 11 ]	8,000	83,771	22,312
HWSAT		$\geq 80,000$	551,106	161,621

order of magnitude slower than WSAT. This was not expected since (i) HWSAT performed well on Sadeh's problem, and (ii) it has been shown (Gent and Walsh, 1993a) that on Random 3SAT, the history mechanism of HSAT improves over GSAT's performance. This shows, once again, that combining techniques which individually improve performance does not necessarily lead to better algorithms.



### Selman's WALKSAT

In this section, we show that the details of Local Search algorithms are critical for optimal performance. In the publication of WSAT (Selman et al., 1994), parts of the variable selection strategy were left open. In the following, we compare the variable selection strategy of a straightforward implementation of the published version of WSAT (by Andrew Baker) with the implementation that is now publicly available from Bart Selman. We will show that although the difference between the strategies is subtle, it leads to a significant difference in performance and in the behavior of the algorithm as the cutoff parameter Maxflips is varied. To avoid the naming conflict, Selman's version will be referred to as WALKSAT and Baker's version as WSAT.

WALKSAT is different from WSAT in two respects. (i) hillclimbing is done on the number of clauses that *break* if a variable is flipped, rather than on the heuristic score of the *total* number of satisfied clauses. And (ii) no random move is done if an uphill move is possible, i. e. if a clause can be fixed without breaking any other clause, it is fixed. The WALKSAT variable selection is shown figure 18.

### Results

WSAT and WALKSAT were compared on a set of 1000 Random 3SAT instances at 200 variables, 854 clauses, as before using retrospective analysis. The results in table 12 indicate better performance of WALKSAT than WSAT at optimal Maxflips.<sup>4</sup>

Additionally, WALKSAT shows a wider 5% range in table 12, which suggests

---

<sup>4</sup>The results are based on our implementation of Selman's WALKSAT strategy, which has a reduced fliprate of 74K-flips/s, compared to 83K-flips/s of Baker's WSAT. This is due to the fact that the number of breaking clauses have to be computed even if a random move will be made. Avoiding unnecessary counters could probably smooth out this effect to a certain degree.

```

1 proc WALKSAT-variable-selection
2   Input a set of clauses  $\alpha$ 
3   Output a variable to flip
4    $\equiv$ 
5    $C :=$  a random unsatisfied clause
6    $P :=$  a variables in  $C$ , such that its flip “breaks”
7     the least number of clauses
8    $u :=$  number of clauses that “break” if  $P$  is flipped
9   if  $u > 0$  then
10    with probability  $p$  : return random variable in  $C$ 
11  end
12  return variable  $P$ 
13 end

```

Figure 18: Selman’s implementation of WALKSAT

Table 12: Results of various refinements of WSAT. Results from retrospective analysis, 200 variables, 854 clauses

Version	5% int. (K)	Maxflips*	Opt.exp.flips	95%-conf.
Baker’s WSAT	[ 5, 11 ]	8,000	83,771	22,312
Selman’s WALKSAT	[ 5, 25 ]	10,000	41,844	9,682
HWSAT		$\geq 80,000$	551,106	161,621

that the influence of Maxflips is not as strong on WALKSAT as it is on WSAT. This phenomenon becomes even more apparent from figure 19 which shows the expected number of flips as Maxflips is varied.

While WSAT’s performance degrades noticeably as Maxflips is increased, WALKSAT’s performance remains almost constant. This is an interesting property, because the difficulty of choosing the right Maxflips value is avoided—especially for problem classes for which the value of Maxflips\* is unknown this would be advantageous. More experimentation revealed that the effect continues so that WALKSAT’s mean performance is still close to optimal on Random 3SAT *without any restarts*. How-

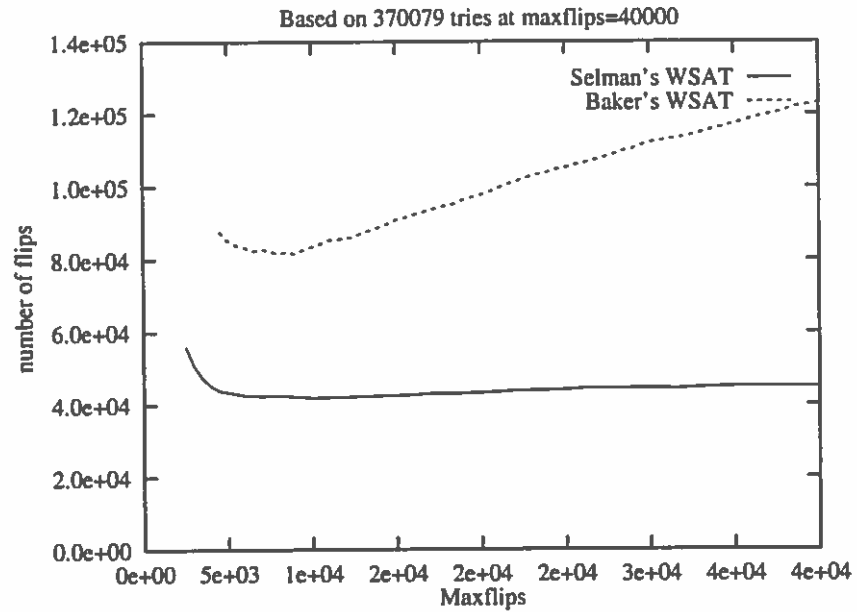


Figure 19: Performance of WSAT and WALKSAT as Maxflips is varied.

ever, reducing the number of restarts has a drawback, since as further investigation reveals, the standard deviation increases as Maxflips goes to infinity. Table 13 shows those results.

Table 13: Behavior of Selman's WALKSAT for finite and infinite Maxflips.

Maxflips	Mean	Standard deviation
Maxflips* $\approx 10,000$	41,844	156,214
40,000	44,958	199,941
$\infty$	48,333	248,453

### Summary and Future Work

This chapter has introduced an implicit propagation mechanism for WSAT. We have shown that this propagation mechanism exploits certain features of the structure present in Sadeh's scheduling problems. Additionally, a comparison between two different versions of WSAT has shown that small differences in the strategy of a Local Search procedure may cause effects on performance that matter.

Future work related to this chapter could include experiments on various combinations of the strategies described in this chapter. For example, applying HSAT to Sadeh's scheduling problems, modifying WALKSAT to HWALKSAT, and experiments with various values of  $q$  in HWSAT to determine its optimal value. An interesting open question is how much further the notion of implicit propagation can be pushed in Local Search for satisfiability testing—to exploit structure present in other realistic problems.



## APPENDIX

### THE GSAT FAMILY

This appendix summarizes various flavors of GSAT that have been proposed in the literature. The summary focuses on variable selection strategies in Local Search for satisfiability testing; it is not meant to be a complete enumeration of Local Search algorithms.

Several related strategies that will not be discussed here include refinements of GSAT that are orthogonal to variable selection (Selman and Kautz, 1993a; Selman et al., 1994), and recent search strategies that combine locality with systematicity such as *partial-order dynamic backtracking* (Ginsberg and McAllester, 1994), and *limited discrepancy search* (Harvey and Ginsberg, 1995)

This appendix takes a generalizing approach similar to GenSAT (Gent and Walsh, 1993a). However, some of the described procedures are not instances of GenSAT. Therefore, a somewhat different generic version of Local Search for satisfiability testing is presented in figure 20 which uses two subroutine calls for the variable selection.

Variable selection is presented as a two step process in which one function (*propose*) proposes a set of variables as candidates for a flip, and a second function (*select*) narrows this set down. If *select* still ties multiple variables, a random selection takes place automatically (unless otherwise noted). In all discussed GSAT variants, *initial( $\alpha$ )* produces a random initial assignment.

```

1 proc Generic-SAT
2   Input a set of clauses  $\alpha$ , Maxtries, and Maxflips
3   Output a satisfying total assignment of  $\alpha$ , if found
4    $\equiv$ 
5   for  $i := 1$  to Maxtries do
6      $A := \text{initial}(\alpha)$ 
7     for  $j := 1$  to Maxflips do
8       if  $A$  satisfies  $\alpha$  return  $A$ 
9        $\mathcal{P} := \text{propose}(\alpha, A)$ 
10       $P := \text{select}(\mathcal{P}, \alpha, A)$ 
11       $A := A$  with  $P$  flipped
12    end
13  end
14  return No
15 end

```

Figure 20: A generic procedure for the GSAT family.

### Different Variable Selection Strategies

All following procedures for variable selection have as input a set of clauses  $\alpha$  and a total assignment  $A$  and return a variable which is to be flipped next. A heuristic ‘score’ is used to measure the distance to a solution, according to which the variable selection takes place.

Let  $\alpha$  be a set of clauses and let  $A$  be the current assignment. Let  $P$  be a propositional variable. In the following,  $f_P$  will denote the number clauses in  $\alpha$  that *get fixed* (become satisfiable) if  $P$  is flipped in  $A$ , and  $b_P$  will denote the number of clauses that *break* (become unsatisfiable) if  $P$  is flipped in  $A$ . Thus,  $f_P - b_P$  is the difference in the total number of satisfied clauses in case that  $P$  is flipped next.

GSAT

(Selman et al., 1992)

Basic greedy Local Search for satisfiability testing, GSAT.

```

1 proc GSAT-propose( $\alpha, A$ )
2   return all variables in unsatisfied clauses of  $\alpha$ 
3 end
4 proc GSAT-select( $\alpha, A, \mathcal{P}$ )
5   return variable  $P \in \mathcal{P}$  with maximal  $f_P - b_P$ 
6 end

```

GSAT+walk

(Selman and Kautz, 1993a)

GSAT with *mixed random walk* adds noise to a greedy algorithm.

```

1 proc GSAT+walk-propose  $\equiv$  GSAT-propose
2 proc GSAT+walk-select( $\alpha, A, \mathcal{P}$ )
3   with probability  $p$ :  $P :=$  random variable in  $\mathcal{P}$ 
4     probability  $1 - p$ :  $P :=$  variable in  $\mathcal{P}$  with maximal  $f_P - b_P$ 
5   return  $P$ 
6 end

```

HSAT

(Gent and Walsh, 1993a)

HSAT uses historical information to choose a variable. Additionally, HSAT is deterministic, since if two variables have never been flipped in the current try, an arbitrary (but fixed) ordering is used to choose between them. That is, ties from *select* are never broken at random.

```

1 proc HSAT-propose  $\equiv$  GSAT-propose
2 proc HSAT-select( $\alpha, A, \mathcal{P}$ )
3    $\mathcal{P}' :=$  all variables  $P \in \mathcal{P}$  with maximal  $f_P - b_P$ 
4   return  $P \in \mathcal{P}'$  which has been flipped
5     least recently in the current try
6 end

```

WSAT

(Selman et al., 1994)

WSAT employs a two step random mechanism that flavors variables that appear in many unsatisfied clauses. For WSAT, two different implementations exist, one by Bart Selman at Bell Labs, and the other by Andrew Baker at CIRL. To avoid

the naming conflict, Selman's version will be referred to as WALKSAT and Baker's version as WSAT. Both use the following procedure for proposing variables.

```

1 proc WSAT-propose( $\alpha, A$ )
2    $C :=$  random clause in  $\alpha$  that is unsatisfied by  $A$ 
3   return all variables that occur in  $C$ 
4 end

```

The different implementations of *select* proceed as follows.

- Baker's WSAT

```

1 proc WSAT-select  $\equiv$  GSAT+walk-select

```

- Selman's WALKSAT

```

1 proc WALKSAT-select( $\alpha, A, \mathcal{P}$ )
2    $u := \min_{P \in \mathcal{P}} b_P$ 
3   if  $u > 0$  then
4     with probability  $p$  : return random variable in  $\mathcal{P}$ 
5   end
6   return variable  $P \in \mathcal{P}$  with minimal  $b_P$ 
7 end

```

## HWSAT

This thesis, page 61

HWSAT uses historic information in WSAT. However, it is not deterministic like HSAT, i. e. ties from *select* are broken at random.

```

1 proc HWSAT-propose  $\equiv$  WSAT-propose
2 proc GSAT+walk-select( $\alpha, A, \mathcal{P}$ )
3   with probability  $p$  :  $\mathcal{P}' := \mathcal{P}$ 
4     probability  $1 - p$  :  $\mathcal{P}' :=$  variables in  $\mathcal{P}$  with max  $f_P - b_P$ 
5   return  $P \in \mathcal{P}'$  which has been flipped
6     least recently in the current try
7 end

```

HWSATB

This thesis, page 61

HWSATB is like HWSAT, except that binary clauses are enforced to be satisfied first.

PWSAT

This thesis, page 45

A parallelized version of WSAT according to a “uniform repeating strategy” that has been described by (Luby and Ertel, 1993).

Simulated Annealing

(Kirkpatrick et al., 1983; Johnson et al., 1991)

Simulated annealing is a procedure closely related to the GSAT family. It uses a noise model based on statistical mechanics.

```

1 proc simulated-annealing
2   Input a set of clauses  $\alpha$ 
3   Output a satisfying truth assignment of  $\alpha$ , if found
4    $\equiv$ 
5    $A :=$  a randomly generated truth assignment;
6   while true do
7     if  $A$  satisfies  $\alpha$  then return  $A$  fi
8      $P :=$  one randomly selected propositional variable
9      $\delta :=$  change in the number of satisfied clauses when  $P$  is flipped
10    if  $\delta \leq 0$  (a downhill or sideways move)
11      then flip  $P$  in  $A$ 
12      else flip  $P$  in  $A$  with probability  $e^{-\delta/T}$ 
13    fi
14     $T :=$  Cooling-Schedule( $T$ )
15  end
16 end
17 end

```

GenSAT

(Gent and Walsh, 1993a)

Additional versions of GSAT are reported in (Gent and Walsh, 1993a) as in-

stances of a class which is called GenSAT. GenSAT additionally comprises of a deterministic version (DSAT), timid (anti-greedy) selection (TSAT), cautious selection (CSAT), indifferent selection (ISAT), plus historic (IHSAT), maximizing the variability of the initial assignment (VSAT), plus deterministic selection (VDSAT), sideway moves preferred (SSAT), opportunistic initial assignment (OSAT), initial truth assignments in “numerical” order (NSAT), and fixed initial truth assignment (FSAT).

## BIBLIOGRAPHY

- Aldous and Vazirani (1994). "go with the winners" algorithms. In *IEEE Symposium on Foundations of Computer Science (FOCS)*.
- Alt, H., Guibas, L., Mehlhorn, K., Karp, R., and Wigderson, A. (1991). A method for obtaining randomized algorithms with small tail probabilities. Technical Report TR-91-057, International Computer Science Institute, Berkeley, CA.
- Baker, A. (1995). *Intelligent Backtracking on Constraint Satisfaction Problems*. PhD thesis, University of Oregon.
- Brassard, G. and Bratley, P. (1988). *Algorithmics: Theory and Practice*. Prentice-Hall.
- Cheeseman, P., Kanefsky, B., and Taylor, W. M. (1991). Where the really hard problems are. *IJCAI-91*, 1.
- Cook, S. (1971). The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158.
- Crawford, J. M. (1995a). Personal communication.
- Crawford, J. M. (1995b). Propagating local search.
- Crawford, J. M. and Auton, L. (1993). Experimental results on the crossover point in satisfiability problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 21–27. AAAI Press/The MIT Press.
- Crawford, J. M. and Baker, A. B. (1994). Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1092–1097.
- Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Journal of the ACM*, 5:394–397.
- Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215.
- Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Company.
- Gent, I. and Walsh, T. (1993a). An empirical analysis of search in gsat. *Journal of Artificial Intelligence Research*, 1:47–59.
- Gent, I. and Walsh, T. (1993b). Towards an understanding of hill-climbing procedures for sat. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 28–33. AAAI Press/The MIT Press.

- Ginsberg, M. (1993). *Essentials of Artificial Intelligence*. Morgan Kaufman, San Mateo.
- Ginsberg, M. and McAllester, D. (1994). GSAT and dynamic backtracking. In Borning, A., editor, *PPCP'94: Second Workshop on Principles and Practice of Constraint Programming*, Seattle.
- Gu, J. (1992). Efficient local search for very large-scale satisfiability problems. *SIGART Bulletin*, 3(1):8-12.
- Harvey, W. D. and Ginsberg, M. L. (1995). Limited discrepancy search. In *Proceedings of IJCAI-95*.
- Johnson, D. S., Aragon, C. A., McGeoch, L. A., and Schevon, C. (1991). Optimization by simulated annealing: An experimental evaluation; Part II, Graph coloring and number partitioning. *Operations Research*, 39:378-406.
- Johnson, R. (1992). *Elementary Statistics*. Duxbury Press, 6th ed. edition.
- Kautz, H. and Selman, B. (1992). Planning as satisfiability. In *Proceedings ECAI-92*. Vienna.
- Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, Number 4598, 13 May 1983, 220, 4598:671-680.
- Langley, P. (1992). Systematic and nonsystematic search strategies. In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, pages 145-152. Morgan Kaufmann.
- Luby, M. and Ertel, W. (1993). Optimal parallelization of las vegas algorithms. Technical Report TR-93-041, International Computer Science Institute, Berkeley, CA. In *Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science, 1994*.
- Luby, M., Sinclair, A., and Zuckerman, D. (1993). Optimal speedup of las vegas algorithms. Technical Report TR-93-010, International Computer Science Institute, Berkeley, CA. In *Proceedings of the Second Israeli Symposium on Theory of Computing and Systems, 1993*.
- Minton, S., Johnston, M. D., Philips, A. B., and Laird, P. (1990). Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. *Artificial Intelligence*, 58:161-205.
- Minton, S., Johnston, M. D., Philips, A. B., and Laird, P. (1992). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161-205.



- Mitchell, D., Selman, B., and Levesque, H. (1992). Hard and easy distributions of sat problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 459–465. AAAI Press/The MIT Press.
- Mitchell, D. G. (1993). An empirical study of random sat. Master's thesis, Simon Fraser University.
- Parkes, A. J. (1995). Personal communication.
- Sadeh, N. (1992). Look-ahead techniques for micro-opportunistic job shop scheduling. Technical Report CMU-CS-91-102, School of Computer Science, Carnegie Mellon University.
- Sebastiani, R. (1994). Applying gsat to non-clausal formulas. *JAIR-94*, 1:309–314.
- Selman, B. and Kautz, H. A. (1993a). Domain-independent extensions to gsat: Solving large structured satisfiability problems. In *Proceedings of IJCAI-93*.
- Selman, B. and Kautz, H. A. (1993b). An empirical study of greedy local search for satisfiability testing. In *Proceedings of IJCAI-93*.
- Selman, B., Kautz, H. A., and Cohen, B. (1994). Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 337–343.
- Selman, B., Levesque, H., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI92), San Jose, CA*, pages 440–446.
- Smith, S. F. and Cheng, C. C. (1993). Slack based heuristics for constraint satisfaction scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 139–144.
- Tsang, E. (1993). *Foundations of Constraint Satisfaction*. Academic Press, London.

## INDEX

- HWSAT, 61
- HWSATB, 61
- WALKSAT, 67
- Maxflips\*, 26
- GSAT, 71
- GSAT+walk, 72
- HSAT, 72
- WSAT, 13, 72
- HWSATB, 74
- HWSAT, 73
- PWSAT, 44, 74
- Maxflips sequence, 41
- Binary preference, 60
- boolean, 8
- break, 66
- chain, 56
- chain of binary implications, 56
- chains, 56
- conjunctive normal form, 9
- control variables, 58
- crossover point, 11
- dependent variables, 58
- disjunction, 9
- flipping, 13
- formula, 9
- GenSAT, 74
- greedy, 12
- heuristic distance measure, 9
- History mechanism, 60
- incomplete, 13
- Instance , 14
- Instance collection , 15
- intra-class variation, 19
- intra-instance variation, 19
- Las Vegas Algorithms, 41
- literal, 8
- local gradient, 12
- loopy 3SAT, 58
- mean runtime, 18
- min-conflicts, 2
- model, 9
- negation, 8

non-resolute, 58

over-constrained, 11

particle, 12

problem instance, 9

propositions, 8

retrospective analysis, 22

run, 15

safety, 19

sample tries, 22

satisfiability problem, 9

satisfies, 9

score, 13

semi-decision procedure, 13

Simulated Annealing, 74

sound, 13

state space, 12

tail probabilities, 19

total assignment, 8

truth assignment  $A$ , 8

try, 15

under-constrained, 11

uniform repeating strategy, 44

unit propagation, 58