Solving Hard Satisfiability Problems: A Unified Algorithm Based On Discrete Lagrange Multipliers*

Zhe Wu and Benjamin W. Wah
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, IL 61801, USA
E-mail: {zhewu, wah}@manip.crhc.uiuc.edu
URL: http://www.manip.crhc.uiuc.edu

Abstract

In this paper, we present improved strategies in DLM-99-SAT to escape from traps and to select proper parameter sets to use when applied to solve some difficult but satisfiable SAT problems. One of the main issues in DLM-99-SAT is that it has a large number of tune-able parameters, making it difficult to determine the best parameters to use when given a new instance. To this end, we propose a set of performance metrics and a set of rules using these metrics that determine at run time the best parameters to use for a specific instance. Our experimental results show that these rules are able to select the most suitable parameter set for each instance with very little overhead. Finally, we verify the performance of DLM-99-SAT by solving some benchmarks in SATLIB and some of the most difficult but satisfiable DIMACS SAT problems, including par32-1-c *to* par32-4-c *and* hanoi4-simple.

1 Introduction

Satisfiability (SAT) is the most fundamental discrete, constrained decision problem among all NP-complete problems. A SAT problem is defined as follows. Given n clauses C_1, C_2, $\cdots$, C_n on m Boolean variables $x = (x_1, \cdots, x_m)$, $x_j \in \{0, 1\}$, and a Boolean formula in conjunctive normal form (CNF):

$$C_1 \wedge C_2 \wedge \cdots \wedge C_n, \tag{1}$$

find a truth assignment to x that satisfies (1), where a truth assignment is a combination of variable assignments that make (1) true.

Previous methods for solving SAT can be classified into two categories. Complete methods can prove satisfiability as well as unsatisfiability. Examples include resolution, constraint satisfaction and backtracking [1]. These methods are computationally expensive and are not suitable for solving large instances. Incomplete methods [3, 9, 8], in contrast, aim to find assignments if a formula is satisfiable but may handle unsatisfiable formulae heuristically. These methods generally perturb an assignment iteratively until a satisfiable assignment is found. They can solve larger instances but may have difficulty in solving hard-to-satisfy instances.

Following our previous successful approach [10, 12], we formulate in this paper a SAT problem as a discrete, constrained optimization problem as follows:

$$\begin{aligned} \min_{x \in \{0,1\}^m} \quad & N(x) = \sum_{i=1}^{n} U_i(x) \\ \text{subject to} \quad & U_i(x) = 0 \quad \forall i \in \{1, 2, \ldots, n\}, \end{aligned} \tag{2}$$

where $U_i(x)$ is a binary expression equal to zero when the i^{th} clause is satisfied and to one otherwise, and $N(x)$ is the number of unsatisfied clauses. Note that, when all constraints are satisfied, the objective function is automatically at its global minimum.

This paper aims to develop a unified approach in solving difficult but satisfiable problems in the DIMACS archive [2] using our previously developed Discrete Lagrange-Multiplier Method (DLM) [12, 11]. It addresses the issue on the many parameters used in DLM that are difficult to tune, and that each instance

*Research supported by National Science Foundation Grant NSF MIP 96-32316.
Proceedings of 11th IEEE International Conference on Tools with Artificial Intelligence, November 1999.

may need to be solved using a specific combination of parameter values [12]. It is generally hard to propose one method with a common set of parameter values that can solve all the benchmark problems because these problems were developed from applications of very different nature. To date, we have identified four performance metrics, five sets of parameter values, and the associated rules to determine a specific set of parameters to use after a small amount of evaluations of an instance. The significance of the result lies in the design of a single unified DLM procedure with a small number of parameter sets that can reproduce all previous results efficiently.

This paper is organized in five sections. Section 2 summarizes our previous work [10, 13, 12] on discrete Lagrange multipliers to solve (2). Section 3 presents an efficient implementation of DLM, identifies traps that limit a trajectory, and proposes a strategy to handle traps. Finally, Section 4 shows a set of rules to determine when to apply a specific combination of parameter values to solve a given SAT instance.

2 Discrete Lagrangian Formulations

We first define a *continuous constrained optimization problem* as follows:

$$\begin{array}{lll} \min_{x \in E^m} & f(x) & \\ \text{subject to} & h(x) = 0 & x = (x_1, x_2, \ldots, x_m) \end{array} \quad (3)$$

where $f(x)$ is an objective function and $h(x) = [h_1(x), \ldots, h_m(x)]^T$ is a set of m equality constraints. Further, $f(x)$ and $h(x)$, as well as their derivatives, are continuous functions.

According to classical optimization theory [4], all constrained local minima that are regular points satisfy the following first-order necessary conditions:

$$\nabla_x L_c(x, \lambda) = 0, \quad \nabla_\lambda L_c(x, \lambda) = 0 \quad (4)$$

These conditions are necessary to guarantee the (local) optimality of the solution to (3).[1]

For discrete optimization problems, all the variables x_i $(i = 1, 2, \cdots, n)$ take discrete values (e.g., integers). Little work has been done in applying Lagrangian methods to solve discrete constrained combinatorial optimization problems. The difficulty in traditional Lagrangian methods lies in the lack of a differentiable continuous space to find an equilibrium point.

Based on the discrete Lagrangian formulation proposed by Shang and Wah [10], we define the *discrete Lagrangian function* using Lagrange multipliers λ [11]:

$$L_d(x, \lambda, \mu) = f(x) + \lambda^T H(h(x)) \quad (5)$$

where λ can be continuous and H is a continuous transformation function that satisfies $H(x) = 0 \Leftrightarrow x = 0$. In this paper, H is chosen to be the identity function because equality constraints for SAT problems are always non-negative. More discussions about the properties of H can be found in the reference [11].

The discrete Lagrangian function in (5) cannot be used to derive first-order conditions similar to (4) in continuous space because there are no gradients or differentiation in discrete space. Without these concepts, none of the calculus in continuous space is applicable in discrete space.

An understanding of gradients in continuous space shows that they define directions in a small neighborhood in which function values decrease. To this end, we define in discrete space a *direction of maximum potential drop* (*DMPD*) [13, 11] for $L_d(x, \lambda)$ at point x for fixed λ as a vector[2] that points from x to $y \in \mathcal{N}(x)$, a neighbor of x with the minimum L_d:

$$\Delta_x L_d(x, \lambda) = \vec{\nu}_x = y \ominus x = (y_1 - x_1, \ldots, y_n - x_n) \quad (6)$$

where

$$y \in \mathcal{N}(x) \cup \{x\} \text{ and } L_d(y, \lambda) = \min_{\substack{x' \in \mathcal{N}(x) \\ \cup \{x\}}} L_d(x', \lambda). \quad (7)$$

Here, $\ominus$ is the vector-subtraction operator for changing x in discrete space to one of its "user-defined" neighborhood points $\mathcal{N}(x)$. $\mathcal{N}(x)$ must be defined in such a way that it is possible to reach any point in the search space from another point by traversing a sequence of neighborhood points. Intuitively, $\vec{\nu}_x$ is a vector pointing from x to y, the point with the minimum L_d among all neighboring points of x, including x itself. That is, if x itself has the minimum L_d, then $\vec{\nu}_x = \vec{0}$.

Based on DMPD, we define *saddle points* [10, 13, 11] in discrete space similar to those in continuous space [4]. A point (x^*, λ^*) is a saddle point when:

$$L_d(x^*, \lambda) \le L_d(x^*, \lambda^*) \le L_d(x, \lambda^*), \quad (8)$$

for all $x \in \mathcal{N}(x^*)$ and all possible λ. Note that the first inequality in (8) only holds when all the constraints are satisfied, which implies that it must be true for all λ. Starting from (8), we have proved stronger *first-order necessary and sufficient conditions* in discrete space that are satisfied by all saddle points [13, 11]:

$$\Delta_x L_d(x, \lambda) = 0, \quad h(x) = 0. \quad (9)$$

[1]There are second-order conditions to guarantee that the extremum found is a local minimum [4].

[2]To simplify our symbols, we represent points in the x space without the explicit vector notation.

Note that the first condition defines the DMPD of L_d in discrete space of x for fixed λ. Readers may refer to the correctness proofs in [11, 13].

The first-order necessary and sufficient conditions in (9) lead to the following discrete-space first-order search method that seeks discrete saddle points.

General Discrete First-Order Search Method

$$\begin{aligned} x(k+1) &= x(k) \oplus \Delta_x L_d(x(k), \lambda(k)) & (10) \\ \lambda(k+1) &= \lambda(k) + c_1 H(h(x(k))) & (11) \end{aligned}$$

where $\oplus$ represents vector addition ($x \oplus y = (x_1 + y_1, \ldots, x_n + y_n)$), and c_1 and c_2 are positive real numbers controlling how fast the Lagrange multipliers change.

It is easy to see that the necessary condition for (10) to converge is when $h(x) = 0$, implying that x is a feasible solution to the original problem. If any of the constraints is not satisfied, then λ on the unsatisfied constraints will continue to evolve. Note that, as in continuous Lagrangian methods, the first-order conditions are only satisfied at saddle points, but do not imply that the time to find a saddle point is finite, even if one exists.

3 DLM-99-SAT

Our solution of a SAT instance as a discrete Lagrangian search is based on a solid mathematical foundation [10, 13, 11]. Our theory also explains why other weight-update heuristics [3, 5] work in practice.

3.1 Description of Algorithm

The Lagrangian function for (2) is as follows:

$$L_d(x, \lambda) = N(x) + \sum_{i=1}^{n} \lambda_i U_i(x) \qquad (12)$$

A simple implementation of (10) and (11), like DLM-98-BASIC-SAT [12], can solve most problems in the DIMACS SAT archive. However it takes a very long time to solve a few difficult benchmark problems, like *f2000* and *par16-1-c* to *par16-5-c*, and cannot solve *hanoi4*, *hanoi4-simple*, *hanoi5*, *par16-1* to *par16-5*, and all *par32-* problems. From the output profiles of applying DLM-98-BASIC-SAT [12] to solve hard SAT problems, we have observed frequent "ping-pong" effects in such a way that some clauses are frequently flipped back and forth from being satisfied to being unsatisfied. For example, a clause was initially unsatisfied but became satisfied after a few flips due to increased penalty for that clause. It then became unsatisfied again after a few more flips due to increased penalties for other unsatisfied clauses. These "ping-pong" effects are tremendously inefficient because they trap the trajectory in one or several unsatisfiable assignments. Consequently, the search is confined to a small region in the search space. To quantify the above observation, we introduce next a new concept called traps.

A *trap* [12] is a combination of x and λ such that a point in it has one or more unsatisfied clauses, and any change to a single variable in x will cause the associated L_d to increase. Hence, a trap is a local minimum of L_d with respect to x. Note that a satisfiable assignment is not a trap because all its clauses are satisfied, even though its L_d may increase when x is perturbed.

In order to find feasible solutions efficiently in solving an optimization problem, we like a trajectory to never visit the same point twice because it is purely a waste of time. This is, however, difficult to achieve in practice because it is impractical to keep track of the history of an entire trajectory. Alternatively, we can try not to repeat visiting the same trap many times. The design of such a strategy will depend on how we escape from traps.

We have considered three ways to bring a trajectory out of traps [12]. Our strategy here places more penalty on clauses inside a trap by updating their λ in order to help the trajectory escape from the trap. Hence, unsatisfied clauses that are trapped more often will have very large Lagrange multipliers (penalties), making them less likely to be unsatisfied in the future. This strategy, therefore, implicitly reduces the probability of visiting that same trap again in the future.

Figure 1 outlines the new DLM (a newer version than that in [12]) for solving SAT. Its fundamental difference to previous versions (like DLM-98-SAT) lies in the identification of traps and the use of trap-escaping strategies. It defines a weight for each Lagrange multiplier and increases the weights of all unsatisfied clauses every time the search reaches a trap. When an undesirable out-of-balance situation happens in which some clauses are trapped much more than average, we increase the Lagrange multipliers of clauses with the largest weight in order to force them into satisfaction and pull the trajectory out of that trap. In our implementation, when the ratio of the largest weight to the average is larger than a predefined threshold, we increase the Lagrange multipliers of clauses with the largest weight in order to force them into satisfaction. If these increments are large enough, the corresponding unsatisfied clauses are not likely to be unsatisfied again in future, thereby resolving the out-of-balance situation. Also, the chance for the search to hit that

```
procedure DLM-99-SAT
/* An implementation of the discrete first-order
   method for solving SAT problems */
1.  Reduce original SAT problem;
2.  Generate a random starting point using a fixed seed;
3.  Initialize $\lambda_i \longleftarrow 0$ and $t_i \longleftarrow 0$;
4.  while solution not found and time not used up do
5.      Pick $x_j \notin$ TabuList that reduces $L_d$ the most;
6.      If search is in a trap then
7.          For all unsatisfied clauses $u$, $t_u \longleftarrow t_u + \delta_w$ end_if
8.      Maintain TabuList;
9.      Flip $x_j$;
10.     if $\#_{UpHillMoves} + \#_{FlatMoves} > \theta_1$ then
11.         $\lambda_i \longleftarrow \lambda_i + \delta_o$;
12.         if $\#_{Adjust} \% \theta_2 = 0$ then
13.             call DECREASE-LAMBDA end_if;
14.         call SPECIAL-INCREASE;
15      end_if
16. end_while
end
procedure SPECIAL-INCREASE
/* Special increases of $\lambda$ on certain clauses
   when their weights are out of balance */
17. Pick a set of clauses $S$;
18. if $\frac{\max_{i \in S} t_i}{\sum_{i \in S} t_i / n} \geq \theta_3$ then
19.     For clause $i$ in $S$ with the largest $t_i$, $\lambda_i \longleftarrow \lambda_i + \delta_s$;
20. end_if
end
procedure DECREASE-LAMBDA
/* Two alternatives to decrease Lagrange multipliers */
21. Alternative-1: $\lambda_i \longleftarrow \lambda_i - \delta_d$ for all clauses
22. Alternative-2: $\lambda_i \longleftarrow \lambda_i / \delta_d$ for all clauses when
        the average $\lambda$ of all clauses is larger than $\theta_4$
end
```

Figure 1. Pseudo code of *DLM-99-SAT.*

Table 1. Comparison of performance of DLM-99-SAT for solving some hard SAT problems and the *g*-class problems that were not solve well before [10]. (All our experiments were run on a Pentinum II 400-MHz computer with Solaris. WalkSAT/GSAT experiments were run on an SGI Challenge with MIPS processor, model unknown. "NA" in the last two columns stands for "not available.")

Problem ID	Success Ratio	CPU Sec.	Num. of Flips	WalkSAT/GSAT	
				SR	Sec.
par16-1	10/10	116.3	$1.6 \cdot 10^7$	NA	NA
par16-2	10/10	115.3	$1.7 \cdot 10^7$	NA	NA
par16-3	10/10	151.4	$2.2 \cdot 10^7$	NA	NA
par16-4	10/10	65.7	$9.2 \cdot 10^6$	NA	NA
par16-5	10/10	215.1	$3.0 \cdot 10^7$	NA	NA
f600	10/10	0.8	39935	NA	35*
f1000	10/10	4.4	217062	NA	1095*
f2000	10/10	19.5	655100	NA	3255*
hanoi4	10/10	17765	$5.4 \cdot 10^8$	NA	NA
hanoi4_s	10/10	17152	$9.2 \cdot 10^8$	NA	NA
par32-1-c	1/20	10388	$8.9 \cdot 10^8$	NA	NA
par32-2-c	1/20	123603	$9.2 \cdot 10^9$	NA	NA
par32-3-c	1/20	186274	$1.4 \cdot 10^{10}$	NA	NA
par32-4-c	1/20	139715	$1.1 \cdot 10^{10}$	NA	NA
g125-17	10/10	174.4	754560	7/10**	264**
g125-18	10/10	4.8	6819	10/10**	1.9**
g250-15	10/10	15.5	2426	10/10**	4.41**
g250-29	10/10	399.3	354453	9/10**	1219**

*: Results from [8] are for similar but not the same problems in the DIMACS archive
**: Results are from [7]

trap again is much lower.

Readers may refer to [12] for a detailed description of each step. We like to point out two alternative ways to decrease Lagrange multipliers in Procedure *DECREASE-LAMBDA* that are different from the previous version of the algorithm [12]. The first alternative reduces Lagrange multipliers by a common integer value δ_d (set to be 1 in our experiments), whereas the second alternative first calculates the average of all Lagrange multipliers and decreases them by a common factor δ_d (= 2) if the average is larger than θ_4 (set to be 4.0 in our experiments). We use the second alternative to reduce Lagrange multipliers when solving *hanoi4* and *hanoi4-simple* because it finds satisfiable assignments from ten randomly generated starting points in time shorter by an order of magnitude when compared to the strategy in [12].

3.2 Experimental Results of DLM-99-SAT

We have applied DLM-99-SAT to solve some hard but satisfiable SAT problems in the DIMACS archive. Table 1 shows that DLM-99-SAT can now solve quickly *f2000*, *par16-1-c* to *par16-5-c*, *par16-1* to *par16-5* and *hanoi4* with 100% success. These problems have not been solved well by any single local search method in the literature. Moreover, DLM-99-SAT can now solve *hanoi4-simple* with 100% success, and *par32-1-c* to *par32-4-c*, although not with high success ratios. These problems cannot be solved by any other local search method today. For other simpler problems in the DIMACS archive, DLM-99-SAT has similar performance as the best existing method. Due to space limitation, we do not present the details of these experiments here. For comparison, Table 1 also lists the experimental results on WalkSAT and GSAT.

So far, DLM-99-SAT is not able to find solutions to eight satisfiable problems in the DIMACS archive (*hanoi5*, *par32-5-c* and *par32-1* to *par32-5*). However,

Table 2. Parameter sets and their values. (The meaning of the parameters can be found in the algorithm in Figure 1. θ_4 is only relevant in P_5.)

Para. Set	δ_w	δ_o	δ_s	δ_d	θ_1	θ_2	θ_3	θ_4	*tabu_len*
P_1	1	1	0	1	100	128	1	-	1
P_2	1	1	1	1	50	12	2	-	100
P_3	1	1	1	1	36	7	1	-	8
P_4	1	1	1	1	16	46	3	-	10
P_5	1	1	1	-	16	5	3	4.0	10

we have found assignments to *hanoi5* and *par32-5-c* with only one unsatisfied clause.

4 Automatic Selection of Parameters

In the previous section, we have shown experimental results on applying DLM-99-SAT to solve some hard DIMACS SAT benchmark problems. The major difficulty in using the algorithm lies in the large number of instance-specific parameters that are difficult to tune. To address this issue, we present in this section a strategy to automatically choose a suitable set of parameters when given a SAT instance.

After classifying the different parameters used to solve the satisfiable DIMACS SAT benchmarks, we have identified five sets of parameters that are suitable for solving, respectively, the *f*-, *g*-, *par*-, *hanoi*- and all other problems. To simplify the notations, we denote P_1 to be the set of parameters that can be used to solve all the easy problems (*ii*-, *aim*-, *par8*-, *jnh*- classes and so on); P_2 for the *f*-class problems; P_3 for the *g*-class problems; P_4 for the *par*-class (not including the *par8*-problems); and P_5 for the *hanoi* problems. Table 2 lists the values of the parameters.

There are two approaches to consolidate the different parameters: a) dynamically adapt them to a problem instance, or b) pick from a set of predefined parameters. Obviously, the first approach is more flexible and more general. However, to apply the first approach, we need to be able to characterize a SAT instance, classify the various heuristics and strategies according to these inherent properties, and adjust the parameters in order to adapt to the characteristics of each instance. To date, there is no systematic approach to carry out these steps.

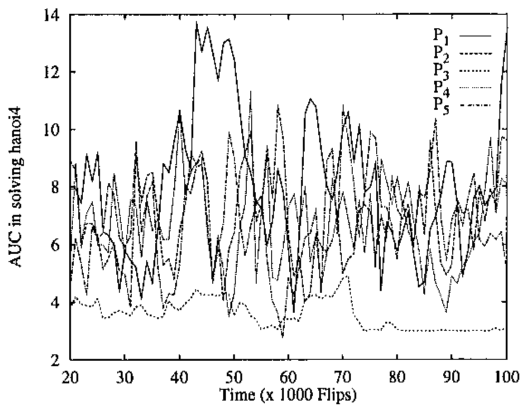
4.1 Procedure *ASPS*

In this paper, we choose the second approach because we already have five different sets of parameters from our experiments (Table 2), and we know that each set is good for solving a group of SAT instances. Our goal is to choose one of the five parameter sets that can be used to solve a SAT instance in a relatively short amount of time. A straightforward way is to run DLM-99-SAT for a few iterations using each parameter set, then choose the parameter set that will lead to a solution with the best quality (measured by the number of unsatisfied clauses) or the shortest time (measured by the number of flips). This approach is not adequate because solution quality depends highly on starting points and time allowed. Additional metrics are needed to characterize the search behavior.

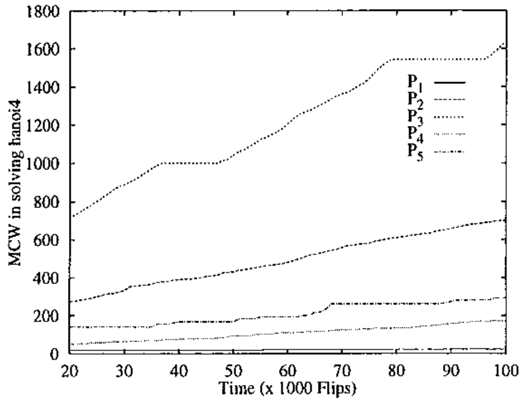
To measure the performance of a parameter set, we divide time into non-overlapping windows, each containing a fixed number of flips. At the end of each window, we compute several performance measures. (In our experiments, the window size was set to 1000 flips.) In total, we have tried eight performance metrics: a) least number of unsatisfied clauses (*LUC*), b) average number of unsatisfied clauses (*AUC*), c) average clause weights (*ACW*), d) maximum clause weights (*MCW*), e) average λ (*AL*), f) maximum λ (*ML*), g) distance (*DIS*) that measures the coverage of the variable space in terms of Hamming distances between the current search point to the previous 10 points found in the last 10 consecutive windows, and h) ratio (*RAT*) that measures the ratio of the number of times of increasing Lagrange multipliers to the number of flips. In these parameters, *LUC* and *AUC* are indicators of solution quality; *ACW* and *MCW* are indicators of times to visit traps; and *AL* and *ML* are indicators of the distribution of unsatisfied clauses;

Figure 2 plots the performance of DLM-99-SAT in terms of two performance metrics when applied to solve *hanoi4*. Some measures, like *AUC* in Figure 2a, are not very informative for identifying the best parameter set because the five curves, each representing the performance of a parameter set, are tangled up. In contrast, measures like *MCW* in Figure 2b are useful in differentiating the performance of different parameter sets.

After trial and error, we chose *LUC*, *MCW*, *DIS*, and *RAT* as our basic performance metrics. Based on statistics gathered on these metrics, we propose a set of rules to automate the selection of the best parameter set according to the following three criteria: a) The parameter set chosen will lead to high quality solutions as compared to other parameter sets. b) It should not get the trajectory trapped more often than other sets. c) It should guide the search to traverse a large area in the search space and not be confined to a small region. Figure 3 outlines the procedure. We present the details



(a) Average number of unsatisfied clauses (AUC)



(b) Maximum clause weight (MCW)

Figure 2. The performance of DLM-99-SAT when applied to solve *hanoi4* using one of five parameter sets. Each point in the graph represents one measurement over a window of 1000 flips.

```
procedure ASPS /* Automatic Selection of Parameter Set */
1.Run DLM-99-SAT for 500,000 flips using each of the
    5 parameter sets; Gather info. on LUC, MCW, DIS,
    RAT in each window;
2.If solution is already found by 1 parameter set, then exit.
3.Compute for each parameter set $i \in [1,5]$ after 500,000 flips
    $MLUC_i$: minimum of $LUC$ over all the windows,
    $MLUC_{min} = \min_{i=1,\dots,5} MLUC_i$,
    $SMCW_i$: sum of $MCW$ over all the windows,
    $SMCW_{max} = \max_{i=1,\dots,5} SMCW_i$,
    $SDIS_i$: sum of $DIS$ over all the windows,
    $SRAT_i$: sum of $RAT$ over all the windows,
4.Initialize flag array: $flag[1] = \cdots = flag[5] = TRUE$;
5.Set $flag[i] = FALSE$ if $MLUC_i > 2 + MLUC_{min}, i \in [1,5]$
6.Set $flag[i] = FALSE$ if $SMCW_i = SMCW_{max}, i \in [1,5]$
7.Set $SDIS_{max} = \max_{i=1,\dots,5} SDIS_i$ where $flag[i] = TRUE$,
  Set $flag[i] = FALSE$ if $SDIS_i < 0.9 \times SDIS_{max}, i \in [1,5]$
8.Set $SRAT_{max} = \max_{i=1,\dots,5} SRAT_i$ where $flag[i] = TRUE$,
  Set selection = i if $SRAT_i = SRAT_{max}, i \in [1,5]$
9.return selection to DLM-99-SAT
end
```

Figure 3. Pseudo code to automatically select an appropriate parameter set.

of each step next.

Line 1 first tries each of the five parameter sets on a given SAT instance for a small number of iterations (determined experimentally to be 500,000 flips in order to remove initial transients). Statistical information on four performance metrics is collected every 1000 flips (called a window).

In Line 2, the procedure exits when the SAT instance is easy and can be solved within 500,000 flips using parameter set P_1 (*aim, par8, ii, jnh,* and *ssa*).

Line 3 computes statistics over all the 500 windows $MLUC_i$, $SMCW_i$, $SDIS_i$, $SRAT_i$, $i = 1, 5$, for each of the five parameter sets if no solution was found in Line 2.

Line 4 sets the flag for each of the five parameter sets to be TRUE initially, since any of them can be the final choice, where *FALSE* means that the corresponding parameter set will not be considered further.

Line 5 eliminates P_i from further consideration if its minimum LUC is worse than the best LUC by 2 in all the five parameter sets. Since the minimization of LUC is one of our objectives, we eliminate a parameter set when its LUC is worse than the best by a threshold of 2. We use a threshold because a search may still have transient behavior after 500,000 iterations, and its LUC may not be the best, even though its parameters were chosen properly.

Line 6 eliminates P_i if $SMCW_i$, the sum of its maximal clause weights over all the windows using P_i, is the largest among the five parameter sets. As discussed in Section 3, a parameter set causing a trajectory to be trapped many times is not desirable.

Line 7 eliminates P_i if its sum of distances over all the windows is less than 90% of the maximum one. Intuitively, a search strategy should have broad coverage of its search space if its parameters were chosen properly, unless a solution was found early in the search. We measure the amount of coverage by computing the distance between the current search point and previously kept search points and by summing all the distances over all the windows. In general, the larger $SDIS$ is, the better the search behaves.

Line 8 chooses the parameter set with the maximum $SRAT$ from the parameter sets left, where a large $SRAT$ means that a relatively large portion of time was spent on adjusting the Lagrange multipliers rather than doing descents in the x space. This choice

Table 3. Test results of applying *ASPS* from 10 random starting points. ("Number Successful" is the number of successful runs that find feasible solutions within 500,000 flips using one of the five parameter sets. "Number Correct" stands for the number of selections that are the best for those runs that did not complete in 500,000 flips. "Number Wrong" is the number of selections that did not pick the best parameter set.) All the experiments were run on a 400 MHz Pentium II computer with Solaris.

Problem ID	Num. Succ.	Num. Corr.	Num. Wrong	Add'l Sec.
aim-200-6-0-yes1-3	10	0	0	0.0
aim-200-6-0-yes1-4	10	0	0	0.0
f600	10	0	0	0.0
f1000	9	1	0	1.00
f2000	3	7	0	21.02
g125-17	3	7	0	112.96
g125-18	10	0	0	0.0
g250-15	10	0	0	0.0
g250-29	8	2	0	194.03
hanoi4	0	10	0	17.58
hanoi4-simple	0	10	0	9.94
par16-1-c	0	10	0	3.74
par16-2-c	1	9	0	3.36
par16-3-c	2	8	0	2.97
par16-4-c	1	9	0	3.31
par16-5-c	3	7	0	2.61
par16-1	2	8	0	3.71
par16-2	0	10	0	4.61
par16-3	0	10	0	4.68
par16-4	1	9	0	4.14
par16-5	0	10	0	4.71

is counter-intuitive because it seems that real work is done by flipping in the x space, whereas updates in the Lagrange-multiplier space are secondary. However, updates in the Lagrange-multiplier space are important because it is the only force that brings a trajectory out of a trap. Consequently, a large *SRAT* indicates that a search can escape from traps.

Line 9 returns the parameter set chosen to DLM-99-SAT that will use it for evaluating the instance.

Note that Lines 5 thru 8 in *ASPS* are only performed when there are two or more TRUE entries in the *flag* array and are skipped when there is only one entry. Further, the numbers used in *ASPS*, like 90% and 2, were decided empirically based on the DIMACS SAT benchmarks and were tuned so that the most suitable parameter set was selected for a specific instance.

4.2 Experimental results of *ASPS*

Table 3 lists the results of applying *ASPS* to select the appropriate parameter set before running DLM-99-SAT. Due to space limitation, we only show the results for a few of the simple SAT problems and those for all the hard SAT problems except *par32*. As indicated before, all the simple problems can be solved within 500,000 flips using P_1, and *ASPS* exits in Line 2.

For each SAT problem in Table 3, we run *ASPS* from 10 randomly generated starting points. The second column reports the number of times that a satisfiable assignment was found in the first 500,000 iterations, thereby terminating the search in Step 2 of *ASPS*. This happens for all the easy problems but not always for the more difficult ones. If *ASPS* continues, the third column lists for each instance the number of times the most suitable parameter set was selected by *ASPS*, and the fourth column, the number of incorrect selections. The best parameter set for each instance was determined by evaluating *ASPS*' performance over all the five parameter sets. Our results indicate that *ASPS* was always able to find the most suitable parameter set. The last column lists the average additional CPU seconds used in applying *ASPS* for all the 10 starting points. The additional time is zero if ASPS was able to solve the problem in the first 500,000 iterations. Our results show that the additional overhead is generally small.

We did not apply *ASPS* to solve the *par32-* problems because the success ratios so far for that class of problems are very low. Consequently, a right choice without a good starting point may not lead to a feasible solution.

4.3 Verification of *ASPS*

To verify the effectiveness of *ASPS* besides the DIMACS SAT benchmarks, we next test it on SAT problems in SATLIB [6]. The library contains problems on random 3-SAT, graph coloring and blocks-world planning.

We have applied DLM-99-SAT with *ASPS* to the most difficult instances in the three classes in SATLIB. We chose the first five instances from *uf200-860* in the random 3-SAT category, the first five instances from *flat100-239* in the graph coloring class, and the largest three instances (*bw_large_b, bw_large_c, bw_large_d*) in the blocks-world planning category. These 13 new SAT instances are the largest and are more difficult as compared to the remaining ones.

All the instances in the first and second classes and *bw_large_b* in the third class were solved during the evaluation stage (Step 2) of *ASPS*. Table 4 shows the experimental results.

For problems in the third class, *ASPS* did not find feasible solutions but made suitable selections on pa-

Table 4. Performance of applying DLM-99-SAT on easy SAT benchmarks in SATLIB [6]. All instances were solved in Step 2 of *ASPS*. Experiments were run a Pentium II 400 MHz computer with Solaris.

Problem ID	Success Ratio	CPU Seconds	Num. of Flips
flat100-1.cnf	10/10	0.18	25797
flat100-2.cnf	10/10	0.23	45198
flat100-3.cnf	10/10	0.09	11356
flat100-4.cnf	10/10	0.18	22574
flat100-5.cnf	10/10	0.15	23682
uf200-01.cnf	10/10	0.15	20362
uf200-02.cnf	10/10	0.16	15520
uf200-03.cnf	10/10	0.08	8248
uf200-04.cnf	10/10	0.17	20436
uf200-05.cnf	10/10	0.19	18437
bw_large_b.cnf	10/10	2.27	51503

Table 5. Performance of DLM-99-SAT on more difficult SAT benchmarks in SATLIB [6] based on the parameter sets selected by *ASPS*. Experiments were run a Pentium II 400 MHz computer with Solaris.

Problem ID	Succ. Ratio	Avg. Sec.	Med. Sec.	Num. Flips	*ASPS* Time
bw_large_c	10/10	352.1	358.1	3341571	18.35
bw_large_d	10/10	2880.5	455.5	43075827	47.5

rameter sets for all the 10 random starting points. Here, "suitable" means that *ASPS*' selections lead to feasible solutions in a relative short amount of time.

Table 5 shows the CPU time in seconds and the number of flips when DLM-99-SAT was applied on the two largest instances in the third class of problems. It also shows the times spent in *ASPS* and the average and medium CPU times spent in DLM-99-SAT after *ASPS* has made its selection. The results demonstrate that *ASPS* was able to make suitable selections without human intervention, and that the performance of DLM-99-SAT is satisfactory. There is, however, one run in *bw_large_d* that has exceptionally large CPU time (23924.57 seconds). This happened because the selection made by *ASPS* for this run was different from that of the remaining nine runs and is demonstrated in the large disparity between the medium and average CPU times.

References

[1] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7:201–215, 1960.

[2] DIMACS SAT benchmark suite. ftp://dimacs.rutgers.edu/pub/challenge/, 1994.

[3] J. Frank. Learning short-term weights for GSAT. *Proc. 15'th Int'l Joint Conf. on AI*, pages 384–391, 1997.

[4] D. G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley Publishing Company, 1984.

[5] P. Morris. The breakout method for escaping from local minima. In *Proc. of the 11th National Conf. on Artificial Intelligence*, pages 40–45, Washington, DC, 1993.

[6] SATLIB suite. http://www.informatik.tu-darmstadt.de/AI/SATLIB.

[7] B. Selman. Private communcation, 1995.

[8] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Proc. of 2nd DIMACS Challenge Workshop on Cliques, Coloring, and Satisfiability, Rutgers University*, pages 290–295, oct 1993.

[9] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proc. of 12th National Conf. on Artificial Intelligence*, pages 337–343, Seattle, WA, 1994.

[10] Y. Shang and B. W. Wah. A discrete Lagrangian based global search method for solving satisfiability problems. *J. Global Optimization*, 12(1):61–99, January 1998.

[11] B. W. Wah and Z. Wu. The theory of discrete lagrange multipliers for nonlinear discrete optimization. *Principles and Practice of Constraint Programming*, (accepted to appear) October 1999.

[12] Z. Wu and B. W. Wah. Solving hard satisfiability problems using the discrete Lagrange-multiplier method. In *Proc. 1999 National Conference on Artificial Intelligence*, pages 673–678. AAAI, July 1999.

[13] Zhe Wu. *Discrete Lagrangian Methods for Solving Nonlinear Discrete Constrained Optimization Problems.* M.Sc. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, May 1998.