Journal of Global Optimization **12:** 61–99, 1998.

© 1998 *Kluwer Academic Publishers. Printed in the Netherlands.*

A Discrete Lagrangian-Based Global-Search Method for Solving Satisfiability Problems *

YI SHANG
Department of Computer Engineering and Computer Science, University of Missouri, Columbia, MO 65211, USA. email: yshang@cecs.missouri.edu

BENJAMIN W. WAH
Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana-Champaign, Urbana, IL 61801, USA. email: b-wah@uiuc.edu

(Received: 25 June 1996; accepted: 2 May 1997)

Abstract. Satisfiability is a class of NP-complete problems that model a wide range of real-world applications. These problems are difficult to solve because they have many local minima in their search space, often trapping greedy search methods that utilize some form of descent. In this paper, we propose a new discrete Lagrange-multiplier-based global-search method (DLM) for solving satisfiability problems. We derive new approaches for applying Lagrangian methods in discrete space, we show that an equilibrium is reached when a feasible assignment to the original problem is found and present heuristic algorithms to look for equilibrium points. Our method and analysis provides a theoretical foundation and generalization of local search schemes that optimize the objective alone and penalty-based schemes that optimize the constraints alone. In contrast to local search methods that restart from a new starting point when a search reaches a local trap, the Lagrange multipliers in DLM provide a force to lead the search out of a local minimum and move it in the direction provided by the Lagrange multipliers. In contrast to penalty-based schemes that rely only on the weights of violated constraints to escape from local minima, DLM also uses the value of an objective function (in this case the number of violated constraints) to provide further guidance. The dynamic shift in emphasis between the objective and the constraints, depending on their relative values, is the key of Lagrangian methods. One of the major advantages of DLM is that it has very few algorithmic parameters to be tuned by users. Besides the search procedure can be made deterministic and the results reproducible. We demonstrate our method by applying it to solve an extensive set of benchmark problems archived in DIMACS of Rutgers University. DLM often performs better than the best existing methods and can achieve an order-of-magnitude speed-up for some problems.

Key words: Discrete Lagrangian methods, global optimization, satisfiability, NP-completeness.

1. Introduction

Satisfiability (SAT) problems belong to an important class of discrete constraint-satisfaction problems (CSP). Many problems in artificial intelligence, logic, computer aided design, database query and planning, etc., can be formulated as SAT

* This research was supported by National Science Foundation Grants MIP 92-18715 and MIP 96-32316. An early version of this paper was presented in the DIMACS Workshop on Satisfiability Problem: Theory and Applications, Rutgers University, NJ, March 1996. The source code of DLM is available at http://manip.crhc.uiuc.edu.

problems. These problems are known to be NP-complete and require algorithms of exponential complexity in the worst case to obtain a satisfying assignment.

Generally, a SAT problem is defined as follows. Given a set of n clauses $\{C_1, C_2, \cdots, C_n\}$ on m variables $x = (x_1, x_2, \cdots, x_m)$, $x_i \in \{0, 1\}$, where each clause C_i consists of only logical-*or* $\vee$ connected variables or negations of variables, and a Boolean formula in conjunctive normal form (CNF), where $\wedge$ is logical *and*.

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

find an assignment of values to the variables so that (1) evaluates to be *true*, or derive its infeasibility if (1) is infeasible. A clause is satisfied if at least one of its member is true. The entire Boolean expression is satisfied if all of its clauses are simultaneously satisfied given a particular assignment to the variables.

Many search methods have been developed in the past for solving this problem. These include resolution, constraint satisfaction, and backtracking. These methods could be computationally expensive for large problem instances.

In addition to the formulation in (1), SAT problems can be formulated as discrete or continuous, constrained or unconstrained, optimization problems. In Section 2, we present five formulations, show the objective and/or constraints for each formulation, and discuss approaches for solving each.

SAT algorithms can be classified as incomplete and complete, depending on whether they can find a random solution or find all solutions. The advantage of complete algorithms is that they can detect infeasibility when a SAT problem is infeasible. However, they are generally computationally expensive for large problem instances. On the other hand, incomplete methods are much faster, but cannot conclude whether a SAT problem is feasible or infeasible when no solution is found within a limited amount of time.

Recently, a class of local (incomplete) search methods were proposed, solving a class of hard SAT problems with size of an order-of-magnitude larger than those solved by complete methods. A major disadvantage of these methods is that they require users to set some problem-specific parameters in order to find solutions efficiently. For this reason, one of our goals in this paper is to design a fast search method whose results can be reproduced easily.

In this paper, we formulate a SAT problem as a discrete constrained optimization problem with a goal of minimizing $N(x)$ subject to a set of constraints.

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

where $U_i = 0$ if C_i is satisfied, and $U_i = 1$ if C_i is not satisfied. We then apply Lagrange multiplier-based methods to solve this problem.

Traditionally, Lagrangian methods have been developed to solve continuous constrained optimization problems. By doing descents in the original variable

space and ascents in the Lagrange-multiplier space, equilibrium is reached when optimal solutions are found. To apply these methods to solve discrete SAT problems, we need to develop discrete Lagrangian operators that can work on discrete values. Our algorithm, called *Discrete Lagrangian method* (DLM), moves a search trajectory towards a local minimum in a direction defined by the Lagrangian function and out of a local minimum in a direction provided by the Lagrange multipliers, without restarting the search. We show that equilibrium is reached when a feasible assignment to the original problem is found. Hence, our method avoids random restarts that are used in existing local search methods, which may bring a search to a completely new search space. It further relies on the balance between the objective and the constraints to define a direction to move, rather than just the weights of violated constraints alone as in existing penalty-based schemes.

We apply DLM to solve SAT problems formulated as (2). Our Lagrangian formulation is based on the objective used in local search schemes and the constraints used in clause-weight schemes. Hence, the analysis of DLM provides a theoretical foundation and generalization of local search schemes and clause-weight schemes. Our discrete formulation allows SAT problems to be solved more efficiently by Lagrangian methods than a continuous constrained formulation.

Our specific Lagrangian formulation of SAT problems can also be considered as a specific form of clause-weight scheme as the objective and the constraints in (2) are related, and the corresponding Lagrangian formula can be rewritten into a clause-weight form. What we have shown in this paper is that it is advantageous to consider the objective and the constraints separately and apply descents in the original-variable space and ascents in the Lagrange-multiplier space to converge to saddle points.

This paper is organized as follows. In Section 2, we summarize previous formulations and algorithms to solve SAT problems. We review in Section 3 general theory for solving continuous constrained optimization problems, and the corresponding discrete optimization formulation and discrete Lagrangian algorithm. In Section 4, we present issues and alternatives in implementing DLM. We show in Section 5 experimental results in applying DLM to solve some SAT benchmark problems from the DIMACS benchmark suite. Finally, concluding remarks are drawn in Section 6

2. Previous Work

In this section, we review previous methods for solving SAT problems. We present various discrete and continuous, constrained and unconstrained, formulations and various algorithms for solving each.

2.1. Discrete Formulations

These can be classified as unconstrained versus constrained, and complete versus incomplete.

(a) *Discrete Constrained Feasibility Formulation*. This is the formulation defined in (1). Methods to solve it can be either complete or incomplete, depending on their ability to prove infeasibility. Complete methods for solving (1) include resolution [41, 10], backtracking [38] and consistency testing [14, 24, 21]. An important resolution method is Davis-Putnam's algorithm [6]. These methods enumerate the search space systematically, and may rely on incomplete methods to find feasible solutions. Their disadvantage is that they are computationally expensive. For instance, Selman *et al.* [44] and Gu [22, 19, 20] have reported that Davis-Putnam's algorithm cannot handle SAT problems with more than 150 variables, and better algorithms today have difficulty in solving SAT problems with more than 400 variables.

(b) *Discrete Unconstrained Formulation*. In this formulation, the goal is to minimize $N(x)$, the number of unsatisfiable clauses. That is,

$$\min_{x \in \{0,1\}^m} N(x) = \sum_{i=1}^{n} U_i(x) \tag{3}$$

where $U_i(x)$ equals 0 if the logical assignment x satisfies C_i, and 1 otherwise. In this case, $N(x)$ equals 0 when all the clauses are satisfied.

Many local search algorithms were designed for this formulation. These algorithms can deal with large SAT problems of thousands of variables. However, they may be trapped by local minima in the search space, where a local minimum is a state whose local neighborhood does not include any state that is strictly better. Consequently, steepest-descent or hill-climbing methods will be trapped there, and random restarts merely bring the search to another local minimum.

Methods designed for (3) are usually incomplete methods, although some mechanisms like backtracking can make them complete. Incomplete methods are usually random methods, relying on ad hoc heuristics to find random solutions quickly. Those that have been applied include multi-start (restart) of descent methods, stochastic methods such as simulated annealing (SA) [31, 2], and genetic algorithms (GA) [27, 34]. They are discussed briefly as follows.

A pure descent method using multi-starts descends in the space of the objective function from an initial point, and generates a new starting point when no further improvement can be found locally. Examples include hill-climbing and steepest descent [14, 51, 36, 44, 45, 35, 19, 54]. For large SAT problems, hill-climbing methods are much faster than steepest descent because they descend in the first direction that leads to improvement, whereas steepest descent methods find the best direction. An example of an objective function suitable to be searched by descent or hill-climbing methods is (3). Pure descent methods are not suitable when there are constraints in the search space as formulated in (2).

Recently, some local search methods were proposed and applied to solve large SAT problems [37, 11, 5, 39]. The most notable ones are those developed independently by Gu and Selman.

Gu developed a group of local search methods for solving SAT and CSP problems. In his Ph.D thesis [14], he first formulated conflicts in the objective function and proposed a discrete relaxation algorithm (a class of deterministic local search) to minimize the number of conflicts in these problems. The algorithms he developed subsequently focused on two components: methods to continue a search when it reaches a local minimum, and methods for variable selection and value assignment. In the first component, he first developed the so-called *min-conflicts* heuristic [14] and showed significant performance improvement in solving large size SAT, n-queen, and graph coloring problems [14, 52, 53, 51, 54]. His methods use various local handlers to escape from local traps when a greedy search stops progressing [15, 22, 23, 16, 24, 17]. Here, a search can continue without improvement when it reaches a local minimum [22] and can escape from it by a combination of backtracking, random restarts, and random swaps. In variable selection and value assignment, Gu and his colleagues have developed random and partial random heuristics [15, 52, 22, 53, 51, 23, 18, 16, 24, 19, 20]. These simple and effective heuristics significantly improve the performance of local search algorithms.

Selman developed GSAT [48, 44, 45, 47, 42, 46] that starts from a randomly generated assignment and performs local search iteratively by flipping variables. Such flipping is repeated until either a satisfiable assignment is found or a pre-set maximum number of flips is reached. When trapped in a local minimum, GSAT either moves up-hill or jumps to another random point. To avoid getting stuck on a *plateau*, which is not a local minimum, GSAT makes side-way moves.

In short, the objective function in (3) may have many local minima that trap local search methods. Consequently, a search in a seemingly good direction may get stuck in a small local minimum, and will rely on random restarts or hill climbing to bring the search out of the local minimum. However, both schemes do not explore the search space systematically, and random restarts may bring the search to a completely different search space.

Stochastic methods, such as GA and SA, have more mechanisms to bring a search out of a local minimum, but are more computationally expensive. Selman *et al.* [45] reported that annealing is not effective for solving SAT problems. To the best of our knowledge, there is no successful application of genetic algorithms to solve SAT problems. In general, stochastic methods based on GA and SA are much slower than descent methods.

(c) *Discrete Constrained Formulation*. There are various forms in this formulation. One approach is to formulate SAT problems as 0-1 integer linear programming (ILP) problems, and apply existing ILP algorithms to solve them [28, 29]. However, this approach is generally computationally expensive.

Another approach is to minimize an objective function $N(x)$, subject to a set of constraints, as defined in (2) and restated as follows.

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

This formulation is better than that in (3) because the constraints provide another mechanism to bring the search out of a local minimum. When a search is stuck in a local minimum, the objective value as formulated in (3) is a discrete integer, and the vicinity of the local minimum may either be worse or be the same. On the other hand, in formulating the problem as in (1), there is very little guidance given to the search algorithm as to which variable to flip when a clause is not satisfied.

Our Lagrange multiplier-based method in this paper is based on formulating a SAT problem as a discrete constrained optimization problem. We show efficient heuristic algorithms that search in discrete space, while satisfying the constraints.

2.2. Continuous Formulations

In formulating a discrete SAT problem in continuous space, we transform discrete variables in the original problem into continuous variables in such a way that solutions to the continuous problem are binary solutions to the original problem. This transformation is potentially beneficial because an objective in continuous space may smooth out some infeasible solutions, leading to smaller number of local minima explored. Unfortunately, continuous formulations require computationally expensive algorithms, rendering them applicable to only small problems. In the following, we show two such formulations.

(a) *Continuous Unconstrained Formulation*.

$$\min_{x\in E^m} f(x) = \sum_{i=1}^{n} c_i(x), \tag{4}$$

where E is the set of real numbers, and $c_i(x)$ is a transformation of clause C_i:

$$c_i(x) = \prod_{j=1}^{m} a_{i,j}(x_j) \tag{5}$$

$$a_{i,j}(x_j) = \begin{cases} (1-x_j)^2 & \text{if } x_j \text{ in } C_i \\ x_j^2 & \text{if } \bar{x_j} \text{ in } C_i \\ 1 & \text{otherwise} \end{cases} \tag{6}$$

Values of x that make $f(x) = 0$ are solutions to the original problem in (1).

Note that the objective is a nonlinear polynomial function. Hence, there may be many local minima in the search space, and descent methods, such as gradient

descent, conjugate gradient and Quasi-Newton, can be trapped by the local minima [33, 17, 18, 19, 20]. Global search techniques, such as clustering methods, generalized gradient methods, Bayesian methods and stochastic methods, can also be applied; however, they are usually much more computationally expensive than descent methods.

To overcome the inefficiency of continuous unconstrained optimization methods, Gu developed discrete bit-parallel optimization algorithms (SAT 14.5 and SAT 14.6) to evaluate continuous objective function [20] and have found significant performance improvements.

(b) *Continuous Constrained Formulation.* This generally involves a heuristic objective function that indicates the quality of the solution obtained (such as the number of clauses satisfied). One formulation similar to (4) is as follows.

$$\begin{aligned} &\min_{x \in E^m} \ f(x) = \sum_{i=1}^{n} c_i(x) \\ &\text{subject to } \ c_i(x) = 0 \quad \forall i \in \{1, 2, \ldots, n\} \end{aligned} \tag{7}$$

where $c_i(x)$ is defined in (5).

The key in this approach lies in the transformation. When the transformation does not smooth out local minima in the discrete space or when the solution density is low, continuous methods are much more computationally expensive to apply than discrete methods.

Since (7) is a continuous constrained optimization problem with a nonlinear objective function and nonlinear constraints, we can apply existing Lagrange-multiplier methods to solve it. Our experience is that a Lagrangian transformation does not reduce the number of local minima, and continuous Lagrangian methods are an order-of-magnitude more expensive to apply than the corresponding discrete algorithms [3].

3. Discrete Lagrangian Methods for Solving SAT Problems

As discussed in the last section, we formulate SAT problems as constrained optimization problems (2) and solve them using Lagrangian methods. In a Lagrangian method, the search tries to minimize the objective function, while relying on unsatisfied clauses in the constraints to provide a force to bring the search out of local minima.

It is important to point out that in our specific formulation of SAT problems in (2), all local minima in the feasible region are globally optimal. This is true because all constraints are satisfied when a local minimum is reached. Hence, Lagrangian methods are only used to find local minima in the feasible region.

In this section we first summarize past work on Lagrangian methods for solving continuous constrained optimization problems. We then extend continuous Lagrangian methods so that they can be used to solve discrete constrained opti-

mization problems. Finally, we show how discrete Lagrangian methods can be applied to solve SAT problems.

3.1. Existing Lagrangian Methods for Solving Continuous Problems

Lagrangian methods are classical methods for solving continuous constrained optimization problems [33, 56, 50]. We review briefly the theory of Lagrange multipliers in this section.

Define an equality constrained optimization problem as follows.

$$\begin{aligned} &\min_{x \in E^m} \ f(x) \\ &\text{subject to } \ g(x) = 0 \end{aligned} \tag{8}$$

where $x = (x_1, x_2, \cdots, x_m)$, and $g(x) = (g_1(x), g_2(x), \cdots, g_n(x))$ are n constraints. Lagrangian function F is defined by

$$F(x, \lambda) = f(x) + \sum_{i=1}^{n} \lambda_i g_i(x) \tag{9}$$

where $\lambda = (\lambda_1, \cdots, \lambda_n)$ are Lagrange multipliers. Here, Lagrange multipliers appear as extra unknowns that transform the constrained problem into an unconstrained problem. Their values may, for example, be obtained through the solution of a saddle-point problem that we introduce later in this section.

DEFINITION 3.1. A point x^* satisfying the constraints $g(x^*) = 0$ is said to be a *regular point* of the constraints if the gradient vectors $\nabla g_i(x^*)$, $i = 1, \cdots, n$, are linearly independent.

The tangent plane at regular points can be characterized in terms of the gradients of the constraint functions.

First-order necessary conditions. Let x^* be a local minimum of f subject to constraints $g(x) = 0$, and x^* be a regular point of these constraints. Then there exists $\lambda \in E^n$ such that

$$\nabla_x F(x^*, \lambda) = 0 \qquad \text{and} \qquad \nabla_\lambda F(x^*, \lambda) = 0 \tag{10}$$

The conditions in (10) are not sufficient to have the existence of a constrained local minimum of $f(x)$ unless second- or higher-order derivatives of $f(x)$ also satisfy certain conditions. An example is the following set of conditions.

Second-order sufficient conditions. Suppose there exists point x^* and $\lambda \in E^n$ such that

$$\nabla_x F(x^*, \lambda) = 0 \qquad \text{and} \qquad \nabla_\lambda F(x^*, \lambda) = 0 \tag{11}$$

Suppose also that matrix $\nabla_x^2 F(x^*, \lambda)$ is positive definite on $M = \{z : \nabla g(x^*) z = 0\}$. Then x^* is a strict local minimum of $f(x)$ subject to $g(x) = 0$.

The necessary conditions in (10) form a system of $n + m$ equations with $n + m$ unknowns. Since the equations are nonlinear, it is difficult to solve them analytically. In this case, a locally optimal solution can be obtained by finding a saddle point of the Lagrangian function.

DEFINITION 3.2. A *saddle-point* (x^*, λ^*) of Lagrangian function $F(x, \lambda)$ is defined as one that satisfies the following condition.

$$F(x^*, \lambda) \leq F(x^*, \lambda^*) \leq F(x, \lambda^*) \tag{12}$$

for all (x^*, λ) and all (x, λ^*) sufficiently close to (x^*, λ^*).

A saddle-point is a local minimum of Lagrangian function $F(x, \lambda)$ in the x space and a local maximum of $F(x, \lambda)$ in the λ space. Hence, one natural way for finding saddle points is to descend in the x space and ascend in the λ space. Lagrange multipliers λ can also be viewed as penalties associated with constraints, and Lagrangian function F corresponds to a penalty function. When certain constraints are not satisfied, the sum of unsatisfied constraints, weighted by the corresponding Lagrange multipliers, are added to the objective function to form a penalty function. Ascents of F in the λ space, therefore, correspond to increasing the penalties associated with unsatisfied constraints. As F is to be minimized, the penalties will eventually increase to a point that pushes the constraints to be satisfied. Likewise, descents of F in the x space find a local minimum when all constraints are satisfied.

In the following, we state a theorem on the relationship between local minima and saddle points based on (8). Although similar proofs have been derived for continuous problems with inequality constraints [33, 56, 50], we present the proof for a special case with equality constraints. This proof will be used when we study discrete optimization problems in Section 3.2.

Saddle-Point Theorem. *x^* is a local minimum to the original problem defined in (8) if there exists λ^* such that (x^*, λ^*) constitutes a saddle point of the associated Lagrangian function $F(x, \lambda)$.*

Proof. Since (x^*, λ^*) is a saddle point, $F(x^*, \lambda) \leq F(x^*, \lambda^*)$ for λ sufficiently close to λ^*. From the definition of the Lagrangian function, this implies

$$\sum_{i=1}^{n} \lambda_i g_i(x^*) \leq \sum_{i=1}^{n} \lambda_i^* g_i(x^*).$$

Our proof is by contradiction. Suppose there exists some k, $1 \leq k \leq n$, $g_k(x^*) \neq 0$. If $g_k(x^*) > 0$, then vector $\lambda = (\lambda_1^*, \cdots, \lambda_k^* + \delta, \cdots, \lambda_n^*)$ would violate the inequality for a positive δ. If $g_k(x^*) < 0$, then vector $\lambda = (\lambda_1^*, \cdots, \lambda_k^* - \delta, \cdots, \lambda_n^*)$ would violate the inequality for a positive δ. Therefore, $g(x^*) = 0$, and x^* is a feasible solution to the problem.

Since (x^*, λ^*) is a saddle point, $F(x^*, \lambda^*) \leq F(x, \lambda^*)$ for x sufficiently close to x^*. From the definition of the Lagrangian function,

$$f(x^*) \leq f(x) + \sum_{i=1}^{n} \lambda_i^* g_i(x).$$

Thus, for any feasible x, $g(x) = 0$, and we have

$$f(x^*) \leq f(x).$$

So x^* is a local minimum. □

Based on the first-order necessary conditions, numerical algorithms have been developed to look for local minima in a search space. One first-order method is to do descents in the original variable space of x and ascents in the Lagrange-multiplier space of λ [1, 33, 4, 57]. The method can be written as a set of ordinary differential equations as follows.

$$\frac{dx}{dt} = -\nabla_x F(x, \lambda) \quad \text{and} \quad \frac{d\lambda}{dt} = \nabla_\lambda F(x, \lambda) \tag{13}$$

where t is an autonomous time variable. This dynamic system evolves over time t and performs gradient descents in the original variable space of x and gradient ascents in the Lagrange-multiplier space of λ. When the system reaches an equilibrium point where all gradients vanish, the point could be a local minimum, a local maximum, or a reflex point subject to constraints. Equilibrium points of local maxima and reflex points are not stable in the sense that a small pertubation will bring the dynamic system away from these points and continue to evolve.

Lagrangian methods are local optimization methods for continuous optimization problems in the sense that they only return a feasible local optimal solution. Therefore, Lagrangian methods do not suffice as a general-purpose global optimization method. The only exception is when there is only one local minimum in the search space or when all local minima are global minima (the formulation of SAT problems in (2)), then Lagrangian methods can find the global minimum.

3.2. Lagrangian Methods for Solving General Discrete Problems

Little work has been done in applying Lagrangian methods to solve discrete constrained combinatorial search problems [12, 26, 9]. The difficulty in traditional Lagrangian methods lies in the requirement of a differentiable continuous space. In order to apply Lagrangian methods to discrete optimization problems, we need to develop a new gradient operator that works in discrete space. Before we introduce the difference gradient operator, we first present some related definitions.

Consider the following equality-constrained optimization problem in discrete space, which is similar to the continuous version in (8),

$$\begin{aligned} &\min_{x \in D^m} \ f(x) \\ &\text{subject to } \ g(x) = 0 \end{aligned} \tag{14}$$

where $x = (x_1, x_2, \cdots, x_m)$, $g(x) = (g_1(x), g_2(x), \cdots, g_n(x))$, and D is the set of integers.

DEFINITION 3.3. A *local minimum* x^* to problem (14) is defined as $g(x^*) = 0$ and $f(x^*) \leq f(x)$ for any feasible x, and x^* and x differ in only one dimension by a magnitude of 1.

For example, if x differs from x^* in the kth dimension, then $|x_k^* - x_k| = 1$. Note that this definition can be extended to the case in which two points can differ by more than one dimensions.

Lagrangian function F has the same form as in the continuous case and is defined as follows.

$$F(x, \lambda) = f(x) + \sum_{i=1}^{n} \lambda_i g_i(x) \tag{15}$$

where λ are Lagrange multipliers that have real values.

A saddle point (x^*, λ^*) of $F(x, \lambda)$ is reached when the following condition is satisfied.

$$F(x^*, \lambda) \leq F(x^*, \lambda^*) \leq F(x, \lambda^*) \tag{16}$$

for all λ sufficiently close to λ^* and all x that differ from x^* in only one dimension by a magnitude 1.

In a way similar to the continuous case, we derive the following theorem specifying the relation between local minima and saddle points.

Discrete Saddle-Point Theorem for Discrete Problems. *x^* is a local minimum solution to the discrete constrained problem in (14) if there exists λ^* such that (x^*, λ^*) constitutes a saddle point of the associated Lagrangian function $F(x, \lambda)$.*

Proof. The proof is similar to that of the continuous case. □

In the continuous case, methods that look for local minimum subject to constraints utilize gradient information. In order for these methods to work in the discrete variable space x of discrete problems, we need to define the counterpart of the gradient operator. Note that λ can remain to be continuous even in the discrete case. In the following, we define a discrete *difference gradient operator* Δ_x. (Note that this operator is not unique, and other operators can be defined to work in a similar way.)

DEFINITION 3.4. *Difference gradient operator* Δ_x is defined with respect to x in such a way that $\Delta_x F(x, \lambda) = (\delta_1, \delta_2, \cdots, \delta_m) \in \{-1, 0, 1\}^m$, $\sum_{i=1}^{m} |\delta_i| = 1$, and $(x - \Delta_x F(x, \lambda)) \in \{0, 1\}^m$, and at most one δ_i is non-zero. For any x' that differs from x by at most value 1 in one dimension, *i.e.*, $\sum_{i=1}^{m} |x'_i - x_i| = 1$,

$$F(x - \Delta_x F(x, \lambda), \lambda) \leq F(x', \lambda).$$

Further, if $\forall x', F(x, \lambda) \leq F(x', \lambda)$, then $\Delta_x F(x, \lambda) = 0$.

Based on this definition, Lagrangian methods for continuous problems can be extended to discrete problems. The basic idea is to descend in the original discrete variable space of x and ascend in the Lagrange-multiplier space of λ. We propose a generic discrete Lagrangian method as follows.

Generic Discrete Lagrangian Method (DLM)

$$x^{k+1} = x^k - D(x^k, \lambda^k) \tag{17}$$
$$\lambda^{k+1} = \lambda^k + g(x^k) \tag{18}$$

where $D(x^k, \lambda^k)$ is a heuristic descent direction for updating x, and k is the iteration index.

$D(x, \lambda)$ is not unique and can be defined by either steepest descent or hill climbing. A steepest-descent approach chooses $D(x, \lambda)$ to be $\Delta_x F(x, \lambda)$, the direction with the maximum gradient. A hill-climbing approach, on the other hand, chooses the first point in the neighborhood of the current x that reduces F. Although both approaches have different descent trajectories, they both can reach equilibrium that satisfies the saddle-point condition. Consequently, they can be considered as alternative approaches to calculate $\Delta_x F(x, \lambda)$.

Given the difference gradient operator $\Delta_x F(x, \lambda)$, the difference equations in discrete space used to seek saddle points are as follows.

$$x^{k+1} = x^k - \Delta_x F(x^k, \lambda^k) \tag{19}$$
$$\lambda^{k+1} = \lambda^k + g(x^k), \tag{20}$$

where x and λ evolve iteratively until a saddle point is reached. The following theorem establishes the relation between saddle points and the termination conditions of (19) and (20).

Discrete Fixed-Point Theorem. *A saddle point (x^*, λ^*) of (15) is reached if and only if (19) and (20) terminate.*

Proof. "$\Rightarrow$" part: If (19) and (20) terminate at (x^*, λ^*), then $g(x^*) = 0$ and $\Delta_x F(x^*, \lambda^*) = 0$. $g(x^*) = 0$ implies that $F(x^*, \lambda) = F(x^*, \lambda^*)$. $\Delta_x F(x^*, \lambda^*) = 0$ implies that $F(x^*, \lambda^*) \leq F(x, \lambda^*)$ for any feasible x in the neighborhood of x^*. Therefore, (x^*, λ^*) is a saddle point, and by the Discrete Saddle Point Theorem, x^* is a local minimum.

"$\Leftarrow$" part: If (x^*, λ^*) is a saddle point, then $g(x^*) = 0$. Further, the condition that $F(x^*, \lambda^*) \leq F(x, \lambda^*)$ for any x in the neighborhood of x^* implies that $\Delta_x F(x^*, \lambda^*) = 0$. Therefore, (19) and (20) terminates at (x^*, λ^*). □

DLM described in this paper is valuable in providing a theoretical framework for better understanding of global search methods to solve general discrete constrained optimization problems.

One of the important features of DLM is that it will continue searching until a solution to the constrained problem is found, independent of its starting point.

Therefore, DLM does not involve random restarts. In contrast, other randomized local search algorithms rely on randomized mechanisms to bring the search out of local minima in the original variable space. When the search reaches a local minimum, it is randomly restarted from a completely new starting point. Consequently, the search may fail to explore the vicinity of the local minimum it has just reached, and the number of unsatisfied constraints at the new starting point is unpredictable.

Another important feature of DLM is that it is a global search algorithm that combines both local search and global search. Its local-search component is based on a descent algorithm in the original variable space, similar to what is used in other local search algorithms. When the search reaches a local minimum, DLM brings the search out of the local minimum using its Lagrange multipliers. This mechanism allows the search to continue in its present trajectory without any breaks. As a result, we often see small fluctuations in the number of unsatisfied constraints as the search progresses, indicating that DLM bypasses small "dents" in the original variable space with the aid of Lagrange multipliers.

One final remark on DLM is that it is an incomplete search method because it does not prove infeasibility. Hence, if there is no feasible solution in the search space or if feasible solutions are very difficult to find, DLM may not terminate. As in continuous Lagrangian methods, the time to find a saddle point can only be determined empirically.

3.3. Discrete Lagrangian Method for Solving SAT Problems

To apply DLM to solve a SAT problem, we introduce an artificial objective $H(x)$ and formulate the problem into a constrained optimization problem as follows.

$$\begin{array}{ll} \min_{x\in\{0,1\}^m} & H(x) \\ \text{subject to} & U_i(x) = 0 \quad \forall i \in \{1, 2, \ldots, n\}, \end{array} \tag{21}$$

where $U_i = 0$ if C_i is satisfied, and $U_i = 1$ if C_i is not satisfied. DLM searches for a saddle point of the Lagrangian formulation of (21).

A saddle point (x^*, λ^*) to the Lagrangian formulation of (21) consists of a feasible solution x^* to the SAT problem in (1). Given the saddle point (x^*, λ^*),

$$\begin{aligned} F(x^*, \lambda) \leq F(x^*, \lambda^*) \;&\Rightarrow\; H(x^*) + \lambda^T U(x^*) \leq H(x^*) + \lambda^{*T} U(x^*) \\ &\Rightarrow\; \lambda^T U(x^*) \leq \lambda^{*T} U(x^*) \end{aligned}$$

for any λ close to λ^*. The condition holds only when $U(x^*) = 0$; *i.e.*, x^* is a feasible solution to the SAT problem in (1).

In order for all feasible solutions to the SAT problem in (1) to become saddle points of (21), the objective function $H(x)$ must satisfy the following condition.

Necessary Condition for Objective function. If a feasible solution x^* to the SAT problem in (1) is a local minima of the objective function $H(x)$, then (x^*, λ^*) is a saddle point of the Lagrangian formulation of (21) for any positive λ^*.

Proof. The Lagrangian function of (21) is

$$F(x, \lambda) = H(x) + \lambda^T U(x)$$

Given a feasible solution x^* of (1), $U(x^*) = 0$. For any positive λ^*,

$$F(x^*, \lambda) = H(x^*) + \lambda^T U(x^*) = H(x^*) = H(x^*) + \lambda^{*T} U(x^*) = F(x^*, \lambda^*)$$

for λ close to λ^*. Since x^* is a local minima of $U(x)$, and $U(x) \geq 0$ for any x, and λ^* is positive,

$$F(x^*, \lambda^*) = H(x^*) + \lambda^{*T} U(x^*) = H(x^*) \leq H(x) + \lambda^{*T} U(x) = F(x, \lambda^*)$$

for x close to x^*. Therefore, we have

$$F(x^*, \lambda) = F(x^*, \lambda^*) \leq F(x, \lambda^*)$$

and (x^*, λ^*) is a saddle point. □

There are many functions that satisfy the necessary condition. Examples are

$$H(x) = \sum_{i=1}^{n} U_i(x) \tag{22}$$

$$H(x) = \sum_{i=1}^{n} l_i U_i(x) \tag{23}$$

$$H(x) = \sum_{i=1}^{n} (l_{max} + 1 - l_i) U_i(x) \tag{24}$$

where l_i is the number of variables appearing in clause C_i, and $l_{max} = \max_{i=1}^{n} l_i$. The first function assigns uniform weights to all clauses. The second function assigns more weights to longer clauses, whereas the third one is the opposite. A feasible solution x^* is a local minimum of these functions because $U_i(x^*) = 0$ and $H(x^*) = 0$. In this paper we use the first function as our objective function, and formulate SAT problems as constrained optimization problems in (2).

The SAT problem defined in (2) is a special case of the discrete constrained optimization problem defined in (14). An important property of the formulation in (2) is that all local minima are also global minima. This is true because, based on (2), x^* is defined as a local minimum if $U(x^*) = 0$, which implies that $N(x^*) = 0$ (a global minimum). This condition is stated more formally as follows.

Necessary and Sufficient Condition for Optimality. x^* is a global minimum of the SAT formulation in (2) if and only if $U(x^*) = 0$.

Proof. Straightforward. □

Due to this property, a SAT problem formulated in (2) can be solved by the difference equations in (19) and (20) that find saddle points of a discrete Lagrangian function. In the rest of this subsection, we show how the general Discrete Lagrangian method (DLM) can be applied to solve discrete SAT problems.

The discrete Lagrangian function for (2) is defined as follows.

$$\begin{aligned} L(x,\lambda) &= N(x) + \lambda^T U(x) = \sum_{i=1}^{n} U_i(x) + \sum_{i=1}^{n} \lambda_i U_i(x) \\ &= \sum_{i=1}^{n} (1+\lambda_i) U_i(x) \end{aligned} \tag{25}$$

where $x \in \{0,1\}^m$, $U(x) = (U_1(x), \ldots, U_n(x)) \in \{0,1\}^n$, and λ^T is the transpose of $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ that denotes the Lagrange multipliers. ((25) gives rise to two possible interpretations that will be discussed at the end of this section.) Due to the specific domain of x, the neighborhood of x is more restricted and is reflected in the definition of the saddle point.

A saddle point (x^*, λ^*) of $L(x,\lambda)$ in (25) is defined as one that satisfies the following condition.

$$L(x^*, \lambda) \leq L(x^*, \lambda^*) \leq L(x, \lambda^*) \tag{26}$$

for all λ sufficiently close to λ^* and for all x whose Hamming distance between x^* and x is 1.

Discrete Saddle-Point Theorem for SAT. *x^* is a global minimum of (2) if and only if there exists some λ^* such that (x^*, λ^*) constitutes a saddle point of the associated discrete Lagrangian function $L(x,\lambda)$.*

Proof. The Discrete Saddle-Point Theorem discussed in Section 3.2 for discrete problems can be applied here. A simpler proof is as follows.

"$\Leftarrow$" part: Since (x^*, λ^*) is a saddle point, $L(x^*, \lambda) \leq L(x^*, \lambda^*)$ for λ sufficiently close to λ^*. From the definition of the Lagrangian function in (25), this implies

$$\sum_{i=1}^{n} \lambda_i U_i(x^*) \leq \sum_{i=1}^{n} \lambda_i^* U_i(x^*).$$

Suppose some $U_k(x^*) \neq 0$, which means $U_k(x^*) = 1$. Then $\lambda = (\lambda_1^*, \cdots, \lambda_k^* + \delta, \cdots, \lambda_n^*)$ would violate the inequality for a positive δ. Therefore, $U(x^*) = 0$, and x^* is a global minimum.

"$\Rightarrow$" part: If x^* is a global minimum, then $U(x^*) = 0$, and $L(x^*, \lambda^*) = L(x^*, \lambda) = 0$. The vector $\lambda^* \geq 0$ makes $L(x^*, \lambda^*) \leq L(x, \lambda^*)$. Therefore, (x^*, λ^*) is a saddle point. □

COROLLARY. If a SAT problem formulated in (2) is feasible, then any algorithm $\mathcal{A}$ that can find a saddle point of $L(x,\lambda)$ defined in (25) from any starting point can find a feasible solution to the SAT problem.

Proof. If a SAT problem is feasible, then its solutions are global minima of (2). These correspond to saddle points of $L(x,\lambda)$ defined in (25). If $\mathcal{A}$ can find a saddle point from any starting point, then $\mathcal{A}$ will find a solution to the problem. □

Since a Lagrangian method only stops at saddle points, this corollary implies that the method will find a saddle point regardless of its starting point (including the origin) if the problem is feasible. Unfortunately, the corollary does not guarantee that it will find a saddle point in a finite amount of time.

To apply DLM to solve SAT problems, we define the *discrete gradient operator* $\Delta_x L(x, \lambda)$ with respect to x such that $\Delta_x L(x, \lambda)$ points to state x' in the Hamming distance-1 neighborhood of the current state x that gives the maximum improvement in $L(x, \lambda)$. If no state in the 1-neighborhood improves $L(x, \lambda)$, then $\Delta_x L(x, \lambda) = 0$.

Next, we propose a method to update (x, λ) so that it will eventually satisfy the optimality condition defined in (26).

Discrete Lagrangian Method (DLM) $\mathcal{A}$ for SAT.

$$x^{k+1} = x^k - \Delta_x L(x^k, \lambda^k) \tag{27}$$

$$\lambda^{k+1} = \lambda^k + U(x^k) \tag{28}$$

It is easy to see that the necessary condition for algorithm $\mathcal{A}$ to converge is when $U(x) = 0$, implying that x is optimal. If any of the constraints in $U(x)$ is not satisfied, then λ will continue to evolve to handle the unsatisfied constraints.

The following theorem establishes the correctness of $\mathcal{A}$ and provides the conditions for termination.

Discrete Fixed-Point Theorem for SAT. *An optimal solution x^* to the SAT problem defined in (2) is found if and only if $\mathcal{A}$ terminates.*

Proof. "$\Rightarrow$" part: If $\mathcal{A}$ terminates, then $U(x) = 0$, which makes $\Delta_x L(x, \lambda) = 0$. Since this is a sufficient condition for optimality, the optimal solution is found.

"$\Leftarrow$" part: If an optimal solution x^* is found, then according to the necessary condition of optimality, $U(x^*) = 0$, implying that $\Delta_x L(x, \lambda) = 0$. Therefore, neither x nor λ will change, leading to the conclusion that $\mathcal{A}$ terminates. □

EXAMPLE. The following simple example illustrates the discrete Lagrangian algorithm. The problem has four variables, $\{x_1, x_2, x_3, x_4\}$, 7 clauses,

$$(x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee \bar{x_2} \vee \bar{x_3}) \wedge (\bar{x_1} \vee \bar{x_2} \vee x_4) \wedge (\bar{x_1} \vee \bar{x_3} \vee \bar{x_4})$$
$$\wedge (x_2 \vee x_3 \vee \bar{x_4}) \wedge (x_2 \vee \bar{x_3} \vee x_4) \wedge (\bar{x_2} \vee x_3 \vee \bar{x_4})\}$$

and 2 solutions, $\{(1, 0, 0, 0), (0, 0, 1, 1)\}$,

Algorithm $\mathcal{A}$ works as follows.

(1) Initially, $x^0 = \{(1, 1, 1, 1)\}$ and $\lambda^0 = \{(0, 0, 0, 0, 0, 0, 0)\}$, and $L(x^0, \lambda^0) = 1$.

(2) The L values of neighboring points of the initial point are

$$L((1, 1, 1, 0), \lambda^0) = L((1, 1, 0, 1), \lambda^0) = L((1, 0, 1, 1), \lambda^0) =$$
$$L((0, 1, 1, 1), \lambda^0) = 1.$$

$\Delta_x L(x^0, \lambda^0) = 0$ since $L(x^0, \lambda^0)$ is less than or equal to the values of neighboring points. As the fourth clause is not satisfied, λ^1 is updated to be $\{0, 0, 0, 1, 0, 0, 0\}$.

Further, x is updated to be $x^1 = x^0$. Note that λ_4, the penalty for the fourth clause, is increased in order to provide a force to pull the search out of the local minimum.

(3) $L(x^1, \lambda^1) = 2$. The L values of x^1's neighboring points are

$$L((1,1,1,0), \lambda^1) = L((1,1,0,1), \lambda^1) = L((0,1,1,1), \lambda^1) = 1 \text{ and } L((1,0,1,1), \lambda^1) = 2.$$

There are three choices of $\Delta_x L(x^1, \lambda^1)$.

- If we choose $\Delta_x L(x^1, \lambda^1) = (0,0,0,1)$, then $x^2 = (1,1,1,0)$ and $\lambda^2 = (0,0,0,2,0,0,0)$.
- If we choose $\Delta_x L(x^1, \lambda^1) = (0,0,1,0)$, then $x^2 = (1,1,0,1)$ and $\lambda^2 = (0,0,0,2,0,0,0)$.
- If we choose $\Delta_x L(x^1, \lambda^1) = (1,0,0,0)$, then $x^2 = (0,1,1,1)$ and $\lambda^2 = (0,0,0,2,0,0,0)$.

Assume that we choose $\Delta_x L(x^1, \lambda^1) = (1,0,0,0)$ in this example.

(4) $x^2 = (0,1,1,1)$ and $L(x^2, \lambda^2) = 1$. The L values of neighboring points are

$$L((1,1,1,1), \lambda^2) = 3,\ L((0,0,1,1), \lambda^2) = 0, \text{ and } L((0,1,0,1), \lambda^2) = L((0,1,1,0), \lambda^2) = 1.$$

Therefore, $\Delta_x L(x^2, \lambda^2) = (0,1,0,0)$. x and λ are updated to be $x^3 = (0,0,1,1)$ and $\lambda^3 = (0,0,0,2,0,1,0)$, respectively.

(5) $U(x^3) = 0$ implies that $\Delta_x L(x^3, \lambda^3) = 0$. Hence, $\mathcal{A}$ terminates and x^3 is a solution.

Note that λ defined in (28) is non-decreasing. Ways to decrease λ are considered in Section 4.2.

There are two ways to interpret DLM when applied to solve SAT problems defined in (2) and formulated in (25).

First, we can consider DLM to provide two counter-acting forces to bring the search to a saddle point. To get out of local minima that do not satisfy all the constraints, DLM increases the penalties on constraints that are violated, recording the history information on constraint violation in the Lagrange multipliers. Eventually as time passes, the penalties on violated constraints will be very large, forcing these constraints to be satisfied. This is similar to what is done in many existing clause-weight schemes, such as Morris' "break-out" strategy [37] and Selman and Kautz's GSAT [44, 45].

The other counter-acting force used in DLM is based on the value of the objective function (in this case the number of violated constraints). When the number of violated constraints is large, then it is important to reduce the number of violated constraints by choosing proper variables to flip. In this case, the weights on the violated constraints may be relatively small as compared to the number of violated constraints and, hence, play a minor role in determining the variables to flip. The advantage of the objective is more apparent in applications in which its value may be large. For instance, in solving MAX-SAT problems, the objective representing the weighted sum of the number of unsatisfied clauses can be large and can provide

better guidance in the search [49]. This part of the search is similar to what is done in many local search methods, such as Gu's local-search methods [18, 19, 20] and GSAT [48, 44, 45, 47, 42, 46], which descends into local minima in the objective space based on (3). Existing local search methods generally use (less effective) random perturbations to escape from local minima. For instance, GSAT uses uphill movements and random restarts to get out of local minima, whereas Gu's local-minimum handler uses stochastic mechanisms to escape from local minima.

The second interpretation of DLM is that it can be treated as a clause-weight scheme, as indicated in (25). This happens because the objective and the constraints are related and can be rewritten into a clause-weight form. What we have shown in this paper is that it is advantageous to consider them separately and apply descents in the original-variable space and ascents in the Lagrange-multiplier space to converge to saddle points.

In the next section, we study new techniques for controlling Lagrange multipliers in order to obtain improved performance. The (new and existing) strategies we consider in Section 4.2 include:

- strategies to determine variables to flip in an iteration;
- strategies based on flat moves [45] and tabu lists [13, 25] to handle plateaus; and
- strategies to periodically reduce the Lagrange multipliers.

The last strategy is particularly critical in reducing the "ruggedness" of the search space as the search progresses, which happens in difficult-to-satisfy but satisfiable SAT problems. The strategy can be considered as systematic restarts (that do not rely on random numbers) in the Lagrangian space.

4. Implementations of DLM to Solve SAT Problems

In this section we discuss issues related to the implementation of DLM and three implementations of the algorithm. There are three components in applying a Lagrangian method: evaluating the derivative of the Lagrangian function, updating the Lagrange multipliers, and evaluating the constraint equations. In the continuous domain, these operations are computationally expensive, especially when the number of variables is large and the function is complex. However, as we show in this and the next sections, implementation in the discrete domain is very efficient, and our method is often faster than other local-search methods for solving SAT problems.

4.1. Algorithmic Design Considerations

The general structure of DLM is shown in Figure 1. It performs descents in the original variable space of x and ascents in the Lagrange-multiplier space of λ. In discrete space, $\Delta_x L(x, \lambda)$ is used in place of the gradient function in continuous

Generic algorithm $\mathcal{A}$

Set initial x and λ
while x is not a solution, *i.e.*, $N(x) > 0$
  update x: $x \longleftarrow x - \Delta_x L(x, \lambda)$
  if condition for updating λ is satisfied **then**
    update λ: $\lambda \longleftarrow \lambda + c \times U(x)$
  end if
end while

Figure 1. Generic discrete Lagrangian algorithm $\mathcal{A}$ for solving SAT problems.

space. We call one *iteration* as one pass through the while loop. In the following, we describe the features of our implementation of $\mathcal{A}$ in Figure 1.

(a) *Descent and Ascent Strategies*. There are two ways to calculate $\Delta_x L(x, \lambda)$: greedy and hill-climbing, each involving a search in the range of Hamming distance one from the current x (assignments with one variable flipped from the current assignment x).

In a *greedy strategy*, the assignment leading to the maximum decrease in the Lagrangian-function value is selected to update the current assignment. Therefore, all assignments in the vicinity need to be searched every time, leading to computation complexity of $O(m)$, where m is number of variables in the SAT problem. In *hill-climbing*, the first assignment leading to a decrease in the Lagrangian-function value is selected to update the current assignment. Depending on the order of search and the number of assignments that can be improved, hill-climbing strategies are generally less computationally expensive than greedy strategies.

We have compared both strategies in solving SAT benchmark problems, and have found hill-climbing to be orders of magnitude faster with solutions of comparable quality. Hence, we have used hill-climbing in our experiments.

(b) *Updating* λ. The frequency in which λ is updated affects the performance of a search. The considerations here are different from those of continuous problems. In a discrete problem, descents based on discrete gradients usually make small changes in $L(x, \lambda)$ in each update of x because only one variable changes. Hence, λ should not be updated in each iteration to avoid biasing the search in the Lagrange-multiplier space of λ over the original variable space of x.

In our implementation, λ is updated in two situations. One is when the search reaches a local minimum; another is decided by parameter T that specifies the number of iterations before λ is updated. T can be changed dynamically according to the value of $L(x, \lambda)$ or when $\Delta_x L(x, \lambda) = 0$. In our experiments, we have found that DLM works better when T is very large. Consequently, λ will be updated infrequently and most likely be updated when $\Delta_x L(x, \lambda) = 0$. When $\Delta_x L(x, \lambda) = 0$, a local minimum in the original variable space is reached, and the search can only escape from it by updating λ. By setting T to infinity, the

strategy amounts to pure descents in the original x variable space, while holding λ constant, until a local minimum is reached. This corresponds to Morris' "break out" strategy [37]. In the breakout algorithm, a *state* is a complete set of assignments for variables. In SAT problems, each clause is associated a weight, and the cost of a state is the sum of weights of unsatisfied clauses of the state. Initially all weights are one. Iterative improvement of state continues until a local minimum is reached. The weight of each current unsatisfied clause is then increased by unit increments until breakout from the local minimum occurs. Iterative improvement resumes afterwards.

A parameter c in the term $c \times U(x)$ in Figure 1 controls the magnitude of changes in λ. Different c's for the whole run is different and is not equivalent to setting c to 1 because the objective is not weighted by c. In general, c can be a vector of real numbers, allowing non-uniform updates of λ across different dimensions and possibly across time. For simplicity, we have used a constant c in our implementation for all λ's. Empirically, $c = 1$ has been found to work well for most of the benchmark problems tested. However, for some larger and more difficult problems, we have used a smaller c in order to reduce the search time.

The last point on λ in Figure 1 is that it is always nondecreasing. This is not true in continuous problems with equality constraints. In applying Lagrangian methods to solve continuous problems, Lagrange multiplier λ of a constraint $g(x) = 0$ increases when $g(x) > 0$ and decreases when $g(x) < 0$. In $\mathcal{A}$ shown in Figure 1, λ is nondecreasing because $U(x)$ is either 0 or 1: when a clause is not satisfied, its corresponding λ is increased; and when a clause is satisfied, its corresponding λ is not changed. For most of the benchmark problems we have tested, this strategy does not worsen search time as these problems are relatively easy to solve. However, for difficult problems that require millions of iterations, λ values can become very large as the search progresses. Large λ's are generally undesirable because they cause large swings in the Lagrangian-function value.

To overcome this problem, we develop in DLM $\mathcal{A}_3$ in Section 4.2 a strategy to reduce λ periodically. Using this strategy, we can solve some of the more difficult benchmark problems that require longer time.

(c) *Starting Points and Restarts*. In contrast to other SAT algorithms that rely on random restarts to bring a search out of a local minimum, DLM will continue to evolve without restarts until a satisfiable assignment is found. This avoids restarting from a new starting point when a search is already in the proximity of a good local minimum. Another major advantage of DLM is that there are very few parameters to be selected or tuned by users, including the initial starting point. This makes it possible for DLM to always start from the origin or from a random starting point generated by a fixed random seed, and find a feasible assignment if one exists.

(d) *Plateaus in the Search Space*. In discrete problems, plateaus with equal values exist in the Lagrangian-function space. Our proposed discrete gradient operator may have difficulties in plateaus because it only examines adjacent points of $L(x, \lambda)$ that differ in one dimension. Hence, it may not be able to distinguish

a plateau from a local minimum. We have implemented two strategies to allow a plateau to be searched.

First, we need to determine when to change λ when the search reaches a plateau. As indicated earlier, λ should be updated when the search reaches a local minimum in the Lagrangian space. However, updating λ when the search is in a plateau changes the surface of the plateau and may make it more difficult for the search to find a local minimum somewhere inside the plateau. To avoid updating λ immediately when the search reaches a plateau, we have developed a strategy called *flat move*. This allows the search to continue for some time in the plateau without changing λ, so that the search can traverse states with the same Lagrangian-function value. How long should flat moves be allowed is heuristic and possibly problem dependent. Note that this strategy is similar to Selman's "sideway-move" strategy [45].

Our second search strategy is to avoid revisiting the same set of states in a plateau. In general, it is impractical to remember every state the search visits in a plateau due to the large storage and computational overheads. In our implementation, we have kept a tabu list to maintain the set of variables flipped in the recent past [13, 25] and to avoid flipping a variable if it is in the tabu list.

To summarize, we employ the following strategies and settings of parameters in our implementations of the generic DLM $\mathcal{A}$.

- A hill-climbing strategy is used for descents in the original variable space.
- T is infinity, which corresponds to updating λ when the search reaches a local minimum in the Lagrangian space.
- The initial λ is 0.
- The initial starting point is either the origin or a randomly-generated starting point obtained by calling random number generator *drand48()* (a publicly available C library function that generates pseudo-random numbers using the well-known linear congruence algorithm and 48-bit integer arithmetic).
- The random number generator uses a fixed initial seed of 101.
- To handle more difficult and complex problems, flat moves and tabu lists may be used in a search to traverse plateaus.

Unless specified otherwise, these strategies and parameters settings are used in our implementations in the following section.

4.2. Three Implementations of the Generic DLM

Figures 2, 3, and 4 show three implementations of the general algorithm $\mathcal{A}$ with increasing complexity.

$\mathcal{A}_1$, DLM Version 1 shown in Figure 2, is the simplest. It has two alternatives to find a variable in order to improve the Lagrangian-function value: flip variables one by one in a predefined order, or flip variables in unsatisfied clauses. Since only variables appearing in unsatisfied clauses can potentially improve the current Lagrangian-function value, it is not necessary to check variables that appear in

DLM $\mathcal{A}_1$

```
Set initial x
Set λ = 0
Set c = 1
Set ϑ = 10
while x is not a solution, i.e., N(x) > 0
    if number of unsatisfied clauses ≤ ϑ, then
        Maintain a list of unsatisfied clauses
        if ∃ variable v in one of the unsatisfied clauses such that
          L(x', λ) < L(x, λ) when flipping v in x to get x' then
          x ⟵ x'
        else
          Update λ: λ ⟵ λ + c · U(x)
        end_if
    else
        if ∃ variable v such that L(x', λ) < L(x, λ) when flipping v
          in a predefined order in x to get x' then
          x ⟵ x'
        else
          Update λ: λ ⟵ λ + c · U(x)
        end_if
    end_if
end_while
```

Figure 2. Discrete Lagrangian Algorithm Version 1, $\mathcal{A}_1$, an implementation of $\mathcal{A}$ for solving SAT problems.

currently satisfied clauses. The first alternative is fast when the search starts. By starting from a randomly generated initial assignment, it usually takes a few flips to find a variable that improves the current Lagrangian-function value. As the search progresses, there are fewer variables that can improve the Lagrangian-function value. At this point, the second alternative should be applied.

$\mathcal{A}_1$ uses parameter ϑ to control switching from the first alternative to the second. We found that $\vartheta = 10$ works well and used this value in all our experiments.

$\mathcal{A}_2$, DLM Version 2 shown in Figure 3, employs another strategy to make local improvements. Initially, it is similar to $\mathcal{A}_1$. As the search progresses, the number of variables that can improve the current Lagrangian-function value is greatly reduced. At this point, a list is created and maintained to contain variables that can improve the current Lagrangian-function value. Consequently, local descent is as simple as flipping the first variable in the list.

$\mathcal{A}_2$ uses κ to control the switching from the first strategy to the second. We found that $\kappa = n/3$ works well and used this value in our experiments. $\mathcal{A}_2$ also has more efficient data structures to deal with larger problems.

DLM $\mathcal{A}_2$

```
Set initial x
Set λ = 0
Set c = 1
Set κ = n/3, where n is the number of variables
while x is not a solution, i.e., N(x) > 0
    if number of iterations ≥ κ then
        Maintain a list, l, of variables such that
          if one of them is flipped, the solution will improve.
        if l is not empty then
          Update x by flipping the first element of l
        else
          Update λ: λ ⟵ λ + c · U(x)
        end_if
    else
        if ∃ variable v such that L(x', λ) < L(x, λ) when flipping v
          in a predefined order in x to get x' then
          x ⟵ x'
        else
          Update λ: λ ⟵ λ + c · U(x)
        end_if
    end_if
end_while
```

Figure 3. Discrete Lagrangian Algorithm Version 2, $\mathcal{A}_2$.

$\mathcal{A}_3$, DLM Version 3 shown in Figure 4, has more complex control mechanisms and was introduced to solve some of the more difficult benchmark problems (such as the "*g*," "*f*" and large "*par*" problems in the DIMACS benchmarks) better than $\mathcal{A}_2$. It is based on $\mathcal{A}_2$ and uses all of A_2's parameters. We have applied strategies based on flat moves and tabu lists to handle plateaus [13, 25]. An important element of $\mathcal{A}_3$ is the periodic scaling down of the Lagrange multipliers in order to prevent them from growing to be very large. This strategy can be considered as systematic restarts (that does not rely on random numbers) in the Lagrangian space. Further, to get better performance, we may have to tune c for each problem instance.

Program efficiency is critical when dealing with SAT problems with a large number of variables and clauses. Since DLM searches by constantly updating state information (current assignment of x, Lagrange-multiplier values and Lagrangian-function value), state update has to be very efficient. In our implementation, we update state information incrementally in a way similar to that in GSAT. In large SAT problems, each variable usually appears in a small number of clauses. Therefore, state changes incurred by flipping a variable are very limited. When flipping a variable, some clauses become unsatisfied while some others become satisfied. The incremental update of the Lagrangian-function value is done by subtracting the part of improvement and adding the part of degradation. This leads to very

DLM $\mathcal{A}_3$

```
Set initial x
Set λ = 0
Set κ = n/3, where n is the number of variables
Set tabu length L_t, e.g., 50
Set flat-region limit L_f, e.g., 50
Set λ reset interval I_λ, e.g., 10000
Set constant c, e.g., 1/2
Set constant r, e.g., 1.5
while x is not a solution, i.e., N(x) > 0
    if number of iterations ≥ κ then
        Maintain a list, l, of variables such that
         if one of them is flipped, the solution will improve.
    end_if
    if number of iterations ≥ κ and l is not empty then
        Update x by flipping the first element of l
    else if ∃ variable v such that L(x', λ) < L(x, λ) when flipping v
         in a predefined order in x to get x' then
         x ⟵ x'
    else if ∃ v such that L(x', λ) = L(x, λ) when flipping v in x to get x'
         and number of consecutive flat moves ≤ L_f
         and v has not been flipped in the last L_t iterations then
        x ⟵ x'                    /* flat move */
    else
        Update λ: λ ⟵ λ + c · U(x),
    end_if
    if iteration index mod I_λ = 0 then
        Reduce λ for all clauses, e.g. λ ⟵ λ/r
    end_if
end_while
```

Figure 4. Discrete Lagrangian Algorithm Version 3, $\mathcal{A}_3$.

efficient evaluation of $L(x, \lambda)$. In a similar way, the computation of $\Delta_x L(x, \lambda)$ can also be done efficiently.

In general Lagrangian methods, Lagrange multipliers introduced in the formulation add extra overhead in computing the Lagrangian function as compared to the original objective function. This overhead in DLM is not significant because an update of λ requires $O(p)$ time, where p is the number of unsatisfied clauses, and $p \ll n$ when n is large.

4.3. Reasons behind the Development of $\mathcal{A}_3$

In our experiments, we found that $\mathcal{A}_2$ has difficulty in solving some of the hard benchmark problems. By looking into the execution profiles of $\mathcal{A}_2$ as illustrated in Figure 5, we have the following interesting observations.

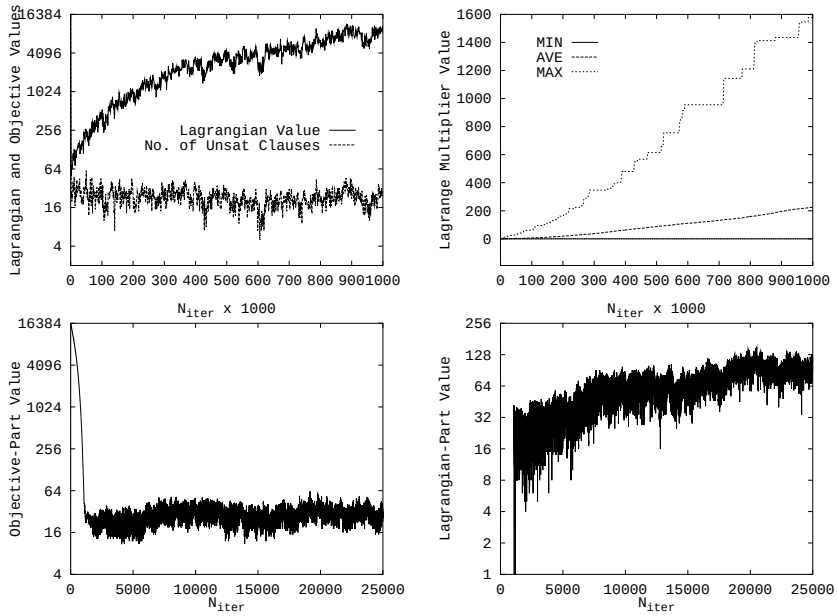

Figure 5. Execution profiles of $\mathcal{A}_2$ on problem "*g125-17*" starting from a random initial point. The top left graph plots Lagrangian-function values and number of unsatisfied clauses against number of iterations sampled every 1000 iterations. The top right graph plots the minimum, average and maximum values of Lagrange multipliers sampled every 1000 iterations. The bottom two graphs plot the iteration-by-iteration objective and Lagrangian-part values for the first 25,000 iterations.

- The sampled Lagrangian-function values decrease rapidly in the first few thousand iterations (bottom left graph of Figure 5), but keep increasing afterwards to become very large at the end. The iteration-by-iteration plot of the Lagrangian part (bottom right graph of Figure 5) shows the same increasing trend.
- Some Lagrange multipliers become very large as indicated in the spread of Lagrange multipliers. This large spread leads to large Lagrangian-function values.
- The number of unsatisfied clauses is relatively stable, fluctuating around 20.

As the Lagrangian-function space is very rugged and difficult to search when there is a large spread in Lagrange-multiplier values, we have developed three strategies in $\mathcal{A}_3$ to address this problem.

First, we introduce in $\mathcal{A}_3$ flat moves to handle flat regions in the Lagrangian-function space. In $\mathcal{A}_2$, Lagrange multipliers are increased whenever the search hits a flat region. Since flat regions in hard problems can be very large, Lagrange multipliers can increase very fast, making the search more difficult. Flat moves

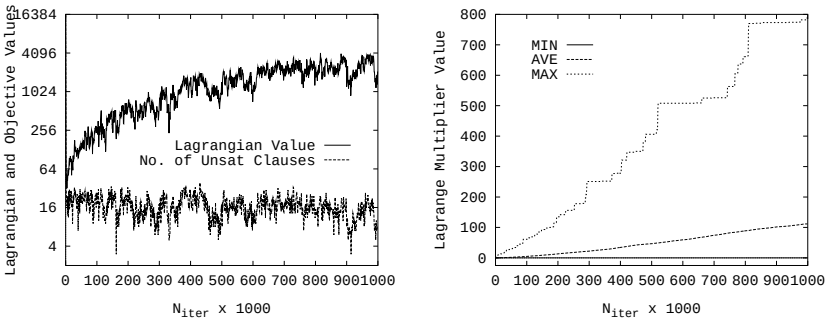

Figure 6. Execution profiles of $\mathcal{A}_3$ using a tabu list of size 30 and flat moves of limit of 50 on problem "*g125-17*" starting from a random initial point. See Figure 5 for further explanation.

allow the search to explore flat regions without increasing the Lagrange multipliers and the Lagrangian-function value.

Second, we introduce a tabu list [13, 25] to store variables that have been flipped in the recent past and to avoid flipping any of them in the near future. This strategy avoids flipping the same set of variables back and forth and revisiting the same state.

The result of applying these two strategies is shown in Figure 6. These graphs show significant reduction in the growth of Lagrangian-function values and Lagrange multipliers as compared to those of $\mathcal{A}_2$. However, these values still grow without bound.

Third, we introduce periodic scale-downs of Lagrange multipliers to control their growth as well as the growth of Lagrangian-function values. Figure 6 shows the result when all Lagrange multipliers are scaled down by a factor of 1.5 every 10,000 iterations. These graphs indicate that periodic scaling, when combined with a tabu list and flat moves, leads to bounded values of Lagrange multipliers and Lagrangian-function values. The bottom right graph in Figure 6 shows the reduction in the Lagrangian part after two reductions of its λ values at 10,000 and 20,000 iterations.

By using these three strategies, $\mathcal{A}_3$ can solve successfully most of the hard problems in the DIMACS benchmark suite.

5. Experimental Results

In this section, we evaluate DLM using SAT benchmarks in the DIMACS archive. The archive is made up of a repository of hundreds of easy and hard SAT problems with many variables and clauses.

Our DLM code was written in C. In our experiments, we have tested DLM on all satisfiable problems in the DIMACS archive. We have compared the performance of DLM with reported results in the literature on the following benchmark problems.

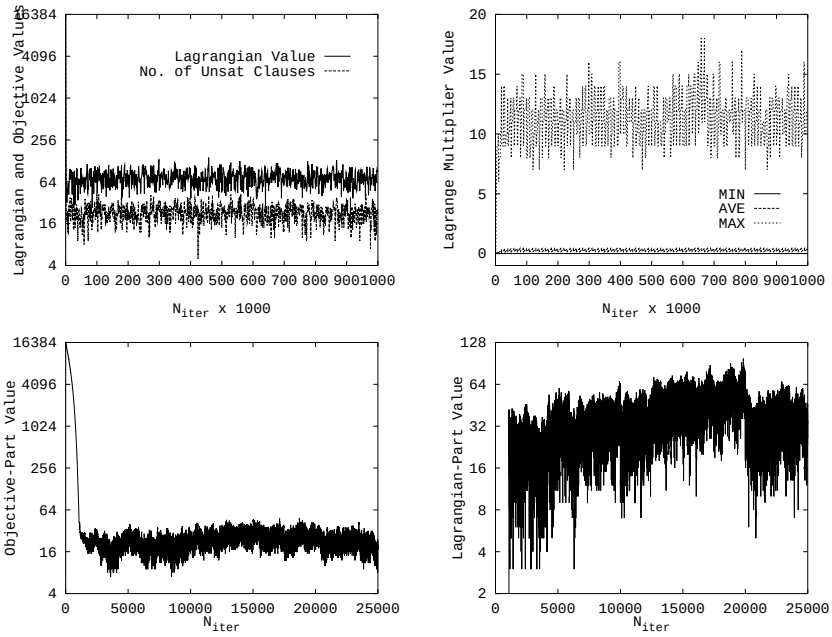

Figure 7. Execution profiles of $\mathcal{A}_3$ with tabu list of size 30, flat moves of limit 50, and periodic scaling of λ by a factor of 1.5 every 10,000 iterations on problem "*g125-17*" starting from a random initial point. Note that the average Lagrange multiplier values are very close to 1 and, therefore, overlap with the curve showing the minimum Lagrange multiplier values. See Figure 5 for further explanation.

- Circuit synthesis problems (*ii*) by Kamath *et al.* [30] — a set of SAT encodings of Boolean circuit-synthesis problems;
- Circuit diagnosis problems (*ssa*) — a set of SAT formulas based on circuit fault analysis;
- Parity learning problems (*par*) — a collection of propositional versions of parity learning problems;
- Artificially generated 3-SAT instances (*aim*);
- Randomly generated SAT instances (*jnh*);
- Large random satisfiable 3-SAT instances (f);
- Hard graph coloring problems (*g*);
- An encoding of the Towers-of-Hanoi problems (*hanoi*); and
- Gu's asynchronous-circuit synthesis benchmarks (*as*) and technology mapping benchmarks (*tm*).

In this section, we show experimental results on the three versions of our DLM implementation described in Section 4.2.

$\mathcal{A}_1$ sets all parameters constant throughout the runs in order to avoid introducing random effects in the program and to allow easy reproduction of results. It works well on the "*aim*" problems, but not as well on others.

Table 1. Execution times of $\mathcal{A}_2$ in CPU seconds on a Sun SparcStation 10/51 for one run of $\mathcal{A}_2$ starting from $x = 0$ (origin) as the initial point on some DIMACS benchmark problems.

Problem Identifier	DLM $\mathcal{A}_2$ Time	DLM $\mathcal{A}_2$ # of Iter.	Problem Identifier	DLM $\mathcal{A}_2$ Time	DLM $\mathcal{A}_2$ # of Iter.
ssa7552-038	0.200	4126	ii16a1	2.100	756
ssa7552-158	0.167	3279	ii16b1	2.100	1313
ssa7552-159	0.167	3802	ii16c1	1.217	916
ssa7552-160	0.167	3409	ii16d1	1.067	595
aim-100-2_0-yes1-1	0.033	982	ii16e1	1.317	1224
aim-100-2_0-yes1-2	0.067	1680	ii32b3	0.883	529
aim-100-2_0-yes1-3	0.000	513	ii32c3	0.533	735
aim-100-2_0-yes1-4	0.367	8510	ii32d3	3.817	1589
par8-2-c	0.317	7841	ii32e3	0.650	518
par8-4-c	0.233	5784			

$\mathcal{A}_2$ has no problem-dependent parameters to be tuned by users and generally works well for all the benchmark problems. However, it has difficulty in solving some larger and more difficult problems, including "*g*," "*f*," large "*par*," and "*hanoi*."

$\mathcal{A}_3$ has some parameters to be tuned in its complex control mechanisms. Although these parameters are problem dependent, we have tried a few sets of parameters and have found one set that generally works well. Results reported are based on the best set of parameters found. $\mathcal{A}_3$ solves some of the more difficult problems better than $\mathcal{A}_2$.

Table 1 shows the experimental results when $\mathcal{A}_2$ was always started from the origin. It shows execution times in CPU seconds and the number of iterations corresponding to the number of flips in GSAT. In each iteration, either one variable was flipped or the λ values were updated. In each experiment, $\mathcal{A}_2$ succeeded in finding a feasible assignment. For most of the test problems, we have found the average time that $\mathcal{A}_2$ spent when started from the origin to be longer than the average time when started from randomly generated initial points. A possible explanation is that the distance between the origin and a local minimum is longer than the average distance between a randomly generated starting point and a nearby local minimum.

We have compared the performance of $\mathcal{A}_2$ with respect to the best known results on these benchmark problems. Most of our timing results were averaged over ten runs with randomly generated initial points, starting from a fixed seed of 101 in our random number generator *drand48()*. Consequently, our results can be reproduced deterministically.

In Table 2, we compare $\mathcal{A}_2$ with WSAT, GSAT, and Davis-Putnam's algorithm in solving the circuit diagnosis benchmark problems. We present average execution

Table 2. Comparison of $\mathcal{A}_2$'s execution times in seconds averaged over 10 runs with respect to published results of WSAT, GSAT, and Davis-Putnam's algorithm [46] on some of the circuit diagnosis problems in the DIMACS archive. ($\mathcal{A}_2$: Sun SparcStation 10/51 and a 150-MHz SGI Challenge with MIPS R4400; GSAT, WSAT and DP: SGI Challenge with a 70 MHz MIPS R4400.)

Problem Identifier	No. of Var.	No. of Clauses	DLM $\mathcal{A}_2$ SUN	DLM $\mathcal{A}_2$ SGI	DLM $\mathcal{A}_2$ # Iter.	WSAT SGI	GSAT SGI	DP SGI
ssa7552-038	1501	3575	0.228	0.235	7970	2.3	129	7
ssa7552-158	1363	3034	0.088	0.102	2169	2	90	*
ssa7552-159	1363	3032	0.085	0.118	2154	0.8	14	*
ssa7552-160	1391	3126	0.097	0.113	3116	1.5	18	*

Table 3. Comparison of $\mathcal{A}_2$'s execution times in seconds averaged over 10 runs with published results on circuit synthesis problems from the DIMACS archive, including the best known results obtained by GSAT, integer programming (IP), and simulated annealing [46]. ($\mathcal{A}_2$: Sun SparcStation 10/51 and a 150-MHz SGI Challenge with MIPS R4400; GSAT and SA: SGI Challenge with a 70 MHz MIPS R4400; Integer Programming: VAX 8700.)

Problem Identifier	No. of Var.	No. of Clauses	DLM $\mathcal{A}_2$ SUN	DLM $\mathcal{A}_2$ SGI	DLM $\mathcal{A}_2$ # Iter.	GSAT SGI	IP Vax	SA SGI
ii16a1	1650	19368	0.122	0.128	819	2	2039	12
ii16b1	1728	24792	0.265	0.310	1546	12	78	11
ii16c1	1580	16467	0.163	0.173	797	1	758	5
ii16d1	1230	15901	0.188	0.233	908	3	1547	4
ii16e1	1245	14766	0.297	0.302	861	1	2156	3

times and average number of iterations of $\mathcal{A}_2$ as well as published average execution times of WSAT, GSAT and Davis-Putnam's method [46]. We did not attempt to reproduce the reported results of GSAT and WSAT, since the results may depend on initial conditions, such as the seeds of the random number generator and other program parameters. We ran $\mathcal{A}_2$ on an SGI Challenge* so that our timing results can be compared to those of GSAT and WSAT. Our results show that $\mathcal{A}_2$ is approximately one order of magnitude faster than WSAT.

In Table 3, we compare $\mathcal{A}_2$ with the published results of GSAT, integer programming and simulated annealing on the circuit synthesis problems [46]. Our results show that $\mathcal{A}_2$ performs several times faster than GSAT.

In Table 4, we compare the performance of the three versions of DLM with some of the best known results of GSAT on circuit-synthesis, parity-learning,

* Based on a single-CPU 150-MHz SGI Challenge with MIPS R4400 at the University of Illinois National Center for Supercomputing Applications, we estimate empirically that it is 15.4% slower than a Sun SparcStation 10/51 for executing $\mathcal{A}_2$ to solve SAT benchmark problems. However, we did not evaluate the speed difference between a 150-MHz SGI Challenge and a 70-MHz SGI Challenge on which GSAT and WSAT were run.

Table 4. Comparison of DLM's execution times in seconds averaged over 10 runs with the best known results obtained by GSAT [43] on the DIMACS circuit-synthesis, parity-learning, artificially generated 3-SAT instances, and graph coloring problems. Results on $\mathcal{A}_3$ were based on a tabu length of 50, flat region limit of 50, λ reset interval of 10,000, and λ reset to be $\lambda/1.5$ when the λ reset interval is reached. For "*g125-18*" and "*g250-15,*" $c = 1/2$; For "*g125-17*" and "*g250-29,*" $c = 1/16$. ($\mathcal{A}_1, A_2, A_3$: Sun SparcStation 10/51; GSAT: SGI Challenge (model unknown))

Problem Identifier	No. of Var.	No. of Clauses	SUN Sec.	Success Ratio	SGI Sec.	Success Ratio
			DLM $\mathcal{A}_1$		GSAT	
aim-100-2_0-yes1-1	100	200	0.19	10/10	1.96	9/10
aim-100-2_0-yes1-2	100	200	0.65	10/10	1.6	10/10
aim-100-2_0-yes1-3	100	200	0.19	10/10	1.09	10/10
aim-100-2_0-yes1-4	100	200	0.10	10/10	1.54	10/10
			DLM $\mathcal{A}_2$		GSAT	
ii32b3	348	5734	0.31	10/10	0.6	10/10
ii32c3	279	3272	0.12	10/10	0.27	10/10
ii32d3	824	19478	1.05	10/10	2.24	10/10
ii32e3	330	5020	0.16	10/10	0.49	10/10
par8-2-c	68	270	0.06	10/10	1.33	10/10
par8-4-c	67	266	0.09	10/10	0.2	10/10
			DLM $\mathcal{A}_3$		GSAT	
g125.17	2125	66272	1390.32	10/10	264.07	7/10
g125.18	2250	70163	3.197	10/10	1.9	10/10
g250.15	3750	233965	2.798	10/10	4.41	10/10
g250.29	7250	454622	1219.56	9/10	1219.88	9/10

some artificially generated 3-SAT, and some of the hard graph coloring problems. The results on GSAT are from [43], which are better than other published results. Our results show that DLM is consistently faster than GSAT on the "*ii*" and "*par*" problems, and that $\mathcal{A}_1$ is an order-of-magnitude faster than GSAT on some "*aim*" problems.

Table 5 also shows the results of $\mathcal{A}_3$ on some "*g*" problems. Recall that $\mathcal{A}_3$ was developed to cope with large flat plateaus in the search space that confuse $\mathcal{A}_2$, which failed to find any solution within 5 million iterations. Hansen [25] and later Selman [45] addressed this problem by using the tabu search strategy. In a similar way, we have adopted this strategy in $\mathcal{A}_3$ by keeping a tabu list to prevent flipping the same variable back and forth. This led to better performance, although the performance is sensitive to the length of the tabu list. $\mathcal{A}_3$ performs comparably to GSAT on these "*g*" problems.

Table 5. Comparison of DLM's execution time in seconds over 10 runs with published results of Grasp on the "aim" problems from the DIMACS archive [39]. (DLM $\mathcal{A}_1$, $\mathcal{A}_2$: Sun SparcStation 10/51; Grasp: SGI Challenge with a 150 MHz MIPS R4400; Success ratio for Grasp is always 10/10.)

Problem Identifier	DLM $\mathcal{A}_2$ Succ. Ratio	DLM $\mathcal{A}_2$ Sun CPU Seconds Avg.	Min.	Max.	DLM $\mathcal{A}_1$ Succ. Ratio	DLM $\mathcal{A}_1$ Sun CPU Seconds Avg.	Min.	Max.	Grasp Avg. SGI Seconds
aim-50-1_6-yes1-1	10/10	0.032	0.000	0.150	10/10	0.042	0.017	0.100	1.12
aim-50-1_6-yes1-2	6/10	0.008	0.000	0.017	10/10	0.020	0.000	0.050	0.12
aim-50-1_6-yes1-3	10/10	0.010	0.000	0.017	10/10	0.027	0.000	0.050	0.49
aim-50-1_6-yes1-4	5/10	0.210	0.017	0.917	10/10	0.030	0.000	0.067	0.46
aim-50-2_0-yes1-1	10/10	0.012	0.000	0.033	10/10	0.035	0.017	0.083	3.67
aim-50-2_0-yes1-2	8/10	0.010	0.000	0.017	10/10	0.035	0.017	0.100	2.82
aim-50-2_0-yes1-3	10/10	0.007	0.000	0.017	10/10	0.077	0.000	0.533	1.76
aim-50-2_0-yes1-4	10/10	0.048	0.000	0.250	10/10	12.603	0.000	125.733	0.62
aim-50-3_4-yes1-1	10/10	0.025	0.000	0.067	10/10	1.528	0.017	13.400	0.39
aim-50-3_4-yes1-2	10/10	0.015	0.000	0.033	10/10	0.162	0.017	0.350	0.58
aim-50-3_4-yes1-3	10/10	0.012	0.000	0.033	10/10	0.142	0.033	0.317	0.49
aim-50-3_4-yes1-4	9/10	0.013	0.000	0.033	10/10	0.057	0.033	0.150	0.91
aim-50-6_0-yes1-1	10/10	0.010	0.000	0.033	10/10	0.027	0.000	0.067	0.06
aim-50-6_0-yes1-2	10/10	0.007	0.000	0.017	10/10	0.028	0.017	0.050	0.14
aim-50-6_0-yes1-3	10/10	0.007	0.000	0.017	10/10	0.035	0.000	0.100	0.05
aim-50-6_0-yes1-4	10/10	0.007	0.000	0.017	10/10	0.027	0.000	0.067	0.05
aim-100-1_6-yes1-1	10/10	0.068	0.033	0.117	10/10	0.092	0.033	0.200	320.40
aim-100-1_6-yes1-2	10/10	0.053	0.017	0.100	10/10	0.098	0.050	0.167	122.27
aim-100-1_6-yes1-3	10/10	0.095	0.050	0.333	10/10	0.142	0.033	0.333	157.15
aim-100-1_6-yes1-4	10/10	0.052	0.000	0.117	10/10	0.095	0.017	0.317	50.10
aim-100-2_0-yes1-1	9/10	0.854	0.017	2.700	10/10	0.193	0.050	0.383	391.79
aim-100-2_0-yes1-2	10/10	0.287	0.050	0.900	10/10	0.652	0.117	1.650	208.40
aim-100-2_0-yes1-3	10/10	0.100	0.017	0.333	10/10	0.187	0.067	0.400	118.22
aim-100-2_0-yes1-4	10/10	0.357	0.033	1.917	10/10	0.097	0.050	0.167	1352.38
aim-100-3_4-yes1-1	10/10	0.450	0.000	2.950	10/10	0.795	0.133	4.417	10.27
aim-100-3_4-yes1-2	10/10	0.195	0.017	1.383	10/10	0.362	0.133	0.817	34.71
aim-100-3_4-yes1-3	10/10	0.050	0.017	0.100	10/10	0.858	0.067	3.800	49.69
aim-100-3_4-yes1-4	10/10	0.038	0.000	0.100	10/10	0.170	0.000	0.317	24.62
aim-100-6_0-yes1-1	10/10	0.020	0.000	0.033	10/10	0.075	0.017	0.133	0.52
aim-100-6_0-yes1-2	10/10	0.018	0.000	0.033	10/10	0.142	0.033	0.467	0.77
aim-100-6_0-yes1-3	10/10	0.017	0.000	0.050	10/10	0.082	0.017	0.233	0.38
aim-100-6_0-yes1-4	10/10	0.018	0.000	0.033	10/10	0.093	0.017	0.267	0.84
aim-200-1_6-yes1-1	10/10	0.958	0.150	2.433	10/10	0.748	0.333	1.583	
aim-200-1_6-yes1-2	6/10	0.786	0.417	1.283	10/10	0.635	0.150	2.350	
aim-200-1_6-yes1-3	10/10	5.498	0.267	43.467	10/10	1.217	0.233	6.950	
aim-200-1_6-yes1-4	2/10	70.042	6.300	133.783	10/10	2.308	0.333	7.183	
aim-200-2_0-yes1-1	1/10	1.283	1.283	1.283	10/10	8.128	0.300	51.483	
aim-200-2_0-yes1-2	4/10	0.538	0.150	1.350	10/10	6.132	0.317	30.000	
aim-200-2_0-yes1-3	7/10	6.355	0.150	28.450	10/10	9.545	0.383	45.717	

Table 5. Continued.

Problem Identifier	DLM $\mathcal{A}_2$ Succ. Ratio	DLM $\mathcal{A}_2$ Sun CPU Seconds Avg.	Min.	Max.	DLM $\mathcal{A}_1$ Succ. Ratio	DLM $\mathcal{A}_1$ Sun CPU Seconds Avg.	Min.	Max.	Grasp Avg. SGI Seconds
aim-200-2_0-yes1-4	0/10				10/10	2.102	0.300	6.950	
aim-200-3_4-yes1-1	8/10	0.469	0.183	1.100	10/10	6.638	0.517	16.500	
aim-200-3_4-yes1-2	10/10	0.547	0.050	3.417	10/10	29.117	0.917	213.467	
aim-200-3_4-yes1-3	10/10	0.838	0.050	5.500	10/10	2.405	0.550	9.467	
aim-200-3_4-yes1-4	9/10	3.122	0.050	22.433	9/10	6.520	0.917	27.150	
aim-200-6_0-yes1-1	10/10	0.075	0.033	0.133	10/10	0.575	0.100	1.717	91.87
aim-200-6_0-yes1-2	9/10	0.209	0.050	0.583	10/10	0.350	0.117	0.933	108.21
aim-200-6_0-yes1-3	10/10	0.102	0.017	0.317	10/10	0.513	0.150	1.250	162.14
aim-200-6_0-yes1-4	10/10	0.218	0.017	0.717	10/10	0.415	0.050	1.000	134.01

In Tables 6, 7, 8 and 9, we compare the performance of DLM with the best results of Grasp on the DIMACS benchmark problems [39]. Grasp is a greedy randomized adaptive search procedure that can find good quality solutions for a wide variety of combinatorial optimization problems [7, 32, 8, 40].

In [39], four implementations of Grasp were applied to solve five classes of DIMACS SAT problems, "aim," "ii," "jnh," "ssa7552," and "par." Comparing to GSAT, Grasp did better on the "aim," "ssa7552," and "par" problems, and worse on the "ii" and "jnh" problems. The results of the most efficient implementation of Grasp, Grasp-A, are used in our comparison. Grasp was run on an SGI Challenge computer with 150 MHz MIPS R4400. The average CPU time of 10 random runs of Grasp are shown in the tables, in which Grasp succeeded in all 10 runs for each problem.

In Table 5, we compare $\mathcal{A}_1$, $\mathcal{A}_2$ and Grasp in solving the whole set of "*aim*" problems, which were randomly generated instances with a single satisfiable solution. For these problems, $\mathcal{A}_1$ performs better than $\mathcal{A}_2$ on the average, and is usually 1 to 3 orders of magnitude faster than Grasp. Grasp does not have results on some of the larger "aim-200" instances.

In Tables 6 and 7, we compare $\mathcal{A}_2$ with Grasp in solving the "ii," "jnh," "par" and "ssa7552" problems. Except for some small instances that both DLM and Grasp solve in no time, DLM is generally 1 to 3 orders of magnitude faster than Grasp in solving the "ii," "jnh" and "ssa7552" problems. For "par" problems, DLM $\mathcal{A}_2$ obtains comparable results to Grasp in solving the "par8-x-c" instances, but is worse in solving the "par8-x" instances.

In addition to the "par8-x" instances, DLM $\mathcal{A}_1$ and $\mathcal{A}_2$ have difficulty in solving the "*par16*," "*par32*," "*hanoi*" and "*f*" problems. Tables 8 shows some preliminary but promising results of $\mathcal{A}_3$ on some of the more difficult but satisfiable DIMACS benchmark problems. Comparing with Grasp, $\mathcal{A}_3$ is better in solving the "par16-x-c" instances, but is worse in solving the "par8-x" and "par16-x" problems.

Table 6. Comparison of DLM $\mathcal{A}_2$'s execution time in seconds over 10 runs with published results of Grasp on the "ii" problems from the DIMACS archive [39]. (DLM $\mathcal{A}_2$: Sun SparcStation 10/51; Grasp: SGI Challenge with a 150 MHz MIPS R4400, and success ratio for Grasp is always 10/10.)

Problem Identifier	Success Ratio	DLM $\mathcal{A}_2$ Sun CPU Seconds Avg.	Min.	Max.	Grasp Avg. SGI Seconds
ii8a1	10/10	0.003	0.000	0.017	0.00
ii8a2	10/10	0.007	0.000	0.017	0.03
ii8a3	10/10	0.013	0.000	0.017	0.10
ii8a4	10/10	0.027	0.017	0.033	0.23
ii8b1	10/10	0.012	0.000	0.017	0.11
ii8b2	10/10	0.028	0.017	0.050	1.67
ii8b3	10/10	0.043	0.017	0.067	35.02
ii8b4	10/10	0.062	0.050	0.083	369.37
ii8c1	10/10	0.013	0.000	0.017	37.26
ii8c2	10/10	0.040	0.033	0.050	8.69
ii8d1	10/10	0.018	0.017	0.033	1.19
ii8d2	10/10	0.043	0.033	0.050	3.23
ii8e1	10/10	0.020	0.017	0.033	1.44
ii8e2	10/10	0.040	0.033	0.067	21.97
ii16a1	10/10	0.122	0.117	0.133	23.46
ii16a2	10/10	0.302	0.200	0.433	1970.58
ii16b1	10/10	0.265	0.217	0.350	449.99
ii16b2	10/10	0.377	0.183	0.717	58.41
ii16c1	10/10	0.163	0.133	0.200	20.83
ii16c2	10/10	0.667	0.133	1.350	43.30
ii16d1	10/10	0.188	0.167	0.217	36.09
ii16d2	10/10	0.618	0.250	1.333	56.32
ii16e1	10/10	0.297	0.267	0.367	74.62
ii16e2	10/10	1.273	0.183	3.350	28.06
ii32a1	10/10	0.337	0.133	1.000	68.36
ii32b1	10/10	0.028	0.017	0.033	0.98
ii32b2	10/10	0.130	0.050	0.517	8.08
ii32b3	10/10	0.305	0.150	0.767	8.21
ii32b4	10/10	0.460	0.167	1.033	28.21
ii32c1	10/10	0.022	0.000	0.033	1.79
ii32c2	10/10	0.050	0.033	0.083	0.41
ii32c3	10/10	0.118	0.083	0.233	2.01
ii32c4	10/10	2.940	0.567	6.217	200.97
ii32d1	10/10	0.065	0.017	0.167	2.04
ii32d2	10/10	0.202	0.083	0.833	17.92
ii32d3	10/10	1.047	0.333	2.750	666.73
ii32e1	10/10	0.022	0.017	0.033	17.67
ii32e2	10/10	0.097	0.050	0.183	1.89
ii32e3	10/10	0.160	0.100	0.450	6.04
ii32e4	10/10	0.190	0.150	0.233	11.53
ii32e5	10/10	0.402	0.250	1.450	16.47

Table 7. Comparison of DLM $\mathcal{A}_2$'s execution time in seconds over 10 runs with published results of Grasp on the "jnh," "par," and "ssa" problems from the DIMACS archive [39]. (DLM $\mathcal{A}_2$: Sun SparcStation 10/51; Grasp: SGI Challenge with a 150 MHz MIPS R4400, and success ratio for Grasp is always 10/10.)

Problem Identifier	Success Ratio	DLM $\mathcal{A}_2$ Sun CPU Seconds Avg.	Min.	Max.	Grasp Avg. SGI Seconds
jnh1	10/10	0.068	0.017	0.150	11.87
jnh7	10/10	0.043	0.017	0.083	3.61
jnh12	10/10	0.155	0.067	0.250	0.84
jnh17	10/10	0.082	0.033	0.167	1.66
jnh201	10/10	0.028	0.017	0.050	1.48
jnh204	10/10	0.172	0.017	0.667	14.64
jnh205	10/10	0.103	0.050	0.183	6.17
jnh207	10/10	0.337	0.050	1.817	3.61
jnh209	10/10	0.433	0.067	2.000	7.45
jnh210	10/10	0.033	0.017	0.067	2.35
jnh212	10/10	5.442	0.033	51.250	70.92
jnh213	10/10	0.083	0.017	0.183	9.43
jnh217	10/10	0.060	0.000	0.133	5.76
jnh218	10/10	0.063	0.017	0.183	1.45
jnh220	10/10	0.387	0.033	2.033	10.17
jnh301	10/10	0.333	0.117	0.950	46.23
par8-1	6/10	3.311	0.033	13.117	0.59
par8-2	6/10	3.981	0.400	7.667	1.78
par8-3	4/10	6.012	0.317	18.017	2.69
par8-4	5/10	6.440	1.083	11.017	0.57
par8-5	6/10	11.478	5.850	18.633	0.86
par8-1-c	10/10	0.075	0.000	0.400	0.07
par8-2-c	10/10	0.058	0.000	0.267	0.14
par8-3-c	10/10	1.998	0.000	9.233	0.35
par8-4-c	10/10	0.088	0.017	0.367	0.55
par8-5-c	10/10	0.477	0.017	2.633	0.17
ssa7552-038	10/10	0.228	0.083	0.933	7.05
ssa7552-158	10/10	0.088	0.050	0.167	3.42
ssa7552-159	10/10	0.085	0.067	0.150	1.72
ssa7552-160	10/10	0.097	0.050	0.183	22.13

Finally, we show in Table 9 results of $\mathcal{A}_2$ on the "as" and "tm" problems in the DIMACS archive. The average time over 10 runs is always under 1 second for these problems.

Table 8. Comparison of DLM $\mathcal{A}_3$'s execution time in seconds over 10 runs with published results of Grasp on some of the more difficult DIMACS benchmark problems from the DIMACS archive [39]. (Success ratio of Grasp is always 10/10.) Program parameters: For all problems, flat region limit = 50; λ reset to $\lambda/1.5$ every 10,000 iterations. For par-16-[1-5] problems: Tabu length = 100, $\lambda = 1$. For the rest of par problems:, Tabu length = 50, $\lambda = \frac{1}{2}$. For f problems: Tabu length = 50, $\lambda = \frac{1}{16}$. For hanoi4 problem: Tabu length = 50, $\lambda = \frac{1}{2}$.
System configuration: DLM $\mathcal{A}_3$: Sun SparcStation 10/51; Grasp: SGI Challenge with a 150 MHz MIPS R4400.

Problem Identifier	Succ. Ratio	DLM $\mathcal{A}_3$ Sun CPU Seconds Avg.	Min.	Max.	Grasp Average SGI Sec.
par8-1	10/10	4.780	0.133	14.383	0.59
par8-2	10/10	5.058	0.100	13.067	1.78
par8-3	10/10	9.903	0.350	21.150	2.69
par8-4	10/10	5.842	0.850	16.433	0.57
par8-5	10/10	14.628	1.167	34.900	0.86
par16-1	5/10	11172.8	4630.6	20489.1	9643.38
par16-2	1/10	856.9	856.9	856.9	8993.89
par16-3	1/10	20281.6	20281.6	20281.6	
par16-4	3/10	3523.1	1015.0	7337.9	
par16-5	1/10	13023.4	13023.4	13023.4	
par16-1-c	10/10	398.1	11.7	1011.9	2235.83
par16-2-c	10/10	1324.3	191.0	4232.3	2179.04
par16-3-c	10/10	987.2	139.8	3705.2	2035.23
par16-4-c	10/10	316.7	5.7	692.66	2071.30
par16-5-c	10/10	1584.2	414.5	3313.2	2566.35
hanoi4	1/10	476.5	476.5	476.5	
f600	10/10	16.9	2.1	37.2	
f1000	10/10	126.8	4.4	280.7	
f2000	10/10	1808.6	174.3	8244.7	

6. Conclusions

In this paper, we have presented a discrete Lagrangian method for solving satisfiability (SAT) problems. Our method belongs to the class of incomplete methods that attempts to find a feasible assignment if one exists, but will not terminate if the problem is infeasible.

We first extend the theory of Lagrange multipliers for continuous problems to discrete problems. With respect to problems in discrete space, we define the concept of saddle points, derive the Discrete Saddle Point Theorem, propose methods to compute discrete gradients, and develop the Discrete Lagrangian Method (DLM)

Table 9. Execution times of DLM $\mathcal{A}_2$ in Sun SparcStation 10/51 CPU seconds over 10 runs on the "as" and "tm" problems from the DIMACS archive.

Problem Identifier	Success Ratio	Sun CPU Seconds Avg.	Min.	Max.	Problem Identifier	Success Ratio	Sun CPU Seconds Avg.	Min.	Max.
as2	10/10	0.020	0.017	0.033	as10	10/10	0.103	0.067	0.150
as3	10/10	0.037	0.017	0.050	as11	10/10	0.047	0.017	0.067
as4	10/10	0.157	0.133	0.200	as12	10/10	0.038	0.017	0.067
as5	10/10	0.988	0.850	1.283	as13	10/10	0.085	0.067	0.117
as6	10/10	0.098	0.050	0.150	as14	10/10	0.017	0.017	0.033
as7	10/10	0.520	0.450	0.633	as15	10/10	0.157	0.117	0.200
as8	10/10	0.047	0.017	0.067					
tm1	10/10	0.238	0.217	0.250	tm2	10/10	0.013	0.000	0.033

to look for saddle points. Finally, we show the Discrete Fixed Point theorem which guarantees that DLM will continue to search until a saddle point is found.

We then apply the DLM to solve SAT problems. We formulate a SAT problem as a discrete constrained optimization problem in which local minima in the Lagrangian space correspond to feasible assignments of the SAT problem. Searching for saddle points, therefore, corresponds to solving the SAT problem. We further investigate various heuristics in implementing DLM.

We have compared the performance of DLM with respect to the best existing methods for solving some SAT benchmark problems archived in DIMACS. Experimental results show that DLM can solve these benchmark problems often faster than other local-search methods. We are still working to solve the remaining DIMACS benchmark problems that include

- *par32-1* thru *par32-5*,
- *par32-1-c* thru *par32-5-c*, and
- *hanoi5*.

We have applied DLM to solve MAX-SAT and weighted MAX-SAT problems and have obtained very positive results [49]. In this case, DLM is able to exploit a more interesting combination of the weighted objective and the constraints. We have also applied DLM to design multiplier-less QMF filter banks, which involves solving highly nonlinear discrete constrained optimization problems whose objectives and constraints are real functions [55].

To summarize, DLM is a generalization of local search schemes that optimize the objective alone and clause-weight schemes that optimize the constraints alone. When the search reaches a local minimum, the Lagrange multipliers in DLM lead the search out of the local minimum and move it in the direction provided by the multipliers. DLM also uses the value of an objective function (in this case the number of violated constraints) to provide further guidance when the number of violations is large. The dynamic shift in emphasis between the objective and the

constraints, depending on their relative values, is the key of Lagrangian methods. DLM improves over existing discrete local- and global-search methods as follows.

- DLM can escape from local minima without random restarts. When a constraint is violated but the search is in a local minimum, the corresponding Lagrange multipliers in DLM provide a force that grows with the amount of time that the constraint is violated, eventually bringing the search out of the local minimum. DLM may rely on systematic restarts (that are not based on random numbers) in the Lagrangian space when the Lagrange multipliers grow large and the search space becomes rugged.
- DLM escapes from local minima in a continuous trajectory, hence avoiding a break in the trajectory as in methods based on restarts. This is advantageous when the trajectory is already in the vicinity of a local minimum, and a random restart may bring the search to a completely different search space.
- DLM is more robust than other local-search methods and is able to find feasible assignments irrespective of its initial starting points. In contrast, descent methods have to rely on properly chosen initial assignments and on a good sampling procedure to find new starting points or by adding noise in order to bring the search out of local minima.

In short, the Lagrangian formulation and its discrete versions presented in this paper are based on a solid theoretical foundation and can be used to develop better heuristic algorithms for solving discrete constrained optimization problems.

References

1. K. J. Arrow and L. Hurwicz. Gradient method for concave programming, I: Local results. In K. J. Arrow, L. Hurwica, and H. Uzawa, editors, *Studies in Linear and Nonlinear Programming*. Stanford University Press, Stanford, CA, 1958.
2. V. Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.
3. Y.-J. Chang and B. W. Wah. Lagrangian techniques for solving a class of zero-one integer linear programs. In *Proc. Computer Software and Applications Conference*, pages 156–161, Dallas, TX, August 1995. IEEE.
4. A. Cichocki and R. Unbehauen. Switched-capacitor artificial neural networks for nonlinear optimization with constraints. In *Proc. of 1990 IEEE Int'l Symposium on Circuits and Systems*, pages 2809–2812, 1990.
5. A. Davenport, E. Tsang, C. Wang, and K. Zhu. Genet: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proc. of the 12th National Conf. on Artificial Intelligence*, pages 325–330, Seattle, WA, 1994.
6. M. Davis and H. Putnam. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7:201–215, 1960.
7. T. Feo and M. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, (8):67–71, 1989.
8. T. Feo and M. Resende. Greedy randomized adaptive search procedures. Technical report, AT&T Bell Laboratories, Murray Hill, NJ 07974, February 1994.
9. B. Gavish and S. L. Hantler. An algorithm for optimal route selection in SNA networks. *IEEE Transactions on Communications*, pages 1154–1161, 1983.
10. M. R. Genesereth and N. J. Nilsson. *Logical Foundation of Artificial Intelligence*. Morgan Kaufmann, 1987.

11. I. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proc. of the 11th National Conf. on Artificial Intelligence*, pages 28–33, Washington, DC, 1993.
12. A. M. Geoffrion. Lagrangian relaxation and its uses in integer programming. *Mathematical Programming Study*, 2:82–114, 1974.
13. F. Glover. Tabu search — Part I. *ORSA J. Computing*, 1(3):190–206, 1989.
14. J. Gu. *Parallel Algorithms and Architectures for Very Fast AI Search*. PhD thesis, Dept. of Computer Science, University of Utah, August 1989.
15. J. Gu. How to solve very large-scale satisfiability (VLSS) problems. Technical Report UCECE-TR-90-002, Univ. of Calgary, Canada, October 1990.
16. J. Gu. Efficient local search for very large-scale satisfiability problems. *SIGART Bulletin*, 3(1):8–12, January 1992.
17. J. Gu. On optimizing a search problem. In N. G. Bourbakis, editor, *Artificial Intelligence Methods and Applications*. World Scientific Publishers, 1992.
18. J. Gu. The UniSAT problem models (appendix). *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(8):865, Aug 1992.
19. J. Gu. Local search for satisfiability (SAT) problems. *IEEE Trans. on Systems, Man, and Cybernetics*, 23(4):1108–1129, 1993.
20. J. Gu. Global optimization for satisfiability (SAT) problems. *IEEE Trans. on Knowledge and Data Engineering*, 6(3):361–381, Jun 1994.
21. J. Gu. *Constraint-Based Search*. Cambridge University Press, New York, to appear.
22. J. Gu and Q.-P. Gu. Average time complexities of several local search algorithms for the satisfiability problem (sat). Technical Report UCECE-TR-91-004, Univ. of Calgary, Canada, 1991.
23. J. Gu and Q.-P. Gu. Average time complexity of the sat1.3 algorithm. Technical report, Tech. Rep., Univ. of Calgary, Canada, 1992.
24. J. Gu and W. Wang. A novel discrete relaxation architecture. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(8):857–865, August 1992.
25. P. Hansen and R. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.
26. M. Held and R. M. Karp. The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 6:62–88, 1971.
27. J. H. Holland. *Adaption in Natural and Adaptive Systems*. University of Michigan Press, Ann Arbor, 1975.
28. J. N. Hooker. Resolution vs. cutting plane solution of inference problems: some computational results. *Operations Research Letters*, 7:1–7, 1988.
29. A. P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende. Computational experience with an interior point algorithm on the satisfiability problem. *Annals of Operations Research*, 25:43–58, 1990.
30. A. P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende. A continuous approach to inductive inference. *Mathematical Programming*, 57:215–238, 1992.
31. S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
32. J. Klincewicz. Avoiding local optima in the p-hub location problem using tabu search and GRASP. *Annals of Operations Research*, (40):283–302, 1992.
33. D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, 1984.
34. Z. Michalewicz. *Genetic Algorithms + Data Structure = Evolution Programs*. Springer-Verlag, 1994.
35. S. Minton, M. D. Johnson, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, Dec 1992.
36. D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. of the 10th National Conf. on Artificial Intelligence*, pages 459–465, 1992.
37. 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.

38. P. W. Purdom. Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, 21:117–133, 1983.
39. M.G.C. Resende and T. Feo. A GRASP for satisfiability. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26, pages 499–520. DIMACS Series on Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996.
40. M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Approximate solution of weighted MAX-SAT problems using GRASP. In *Satisfiability Problem: Theory and Applications. DIMACS Series on Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1997.
41. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. Assoc. Comput. Mach.*, pages 23–41, 1965.
42. R. Sebastiani. Applying GSAT to non-clausal formulas. *Journal of Artificial Intelligence Research*, 1:309–314, 1994.
43. B. Selman, 1995. private communication.
44. B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proc. of the 13th Int'l Joint Conf. on Artificial Intelligence*, pages 290–295, 1993.
45. B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Proc. of the Second DIMACS Challenge Workshop on Cliques, Coloring, and Satisfiability, Rutgers University*, pages 290–295, oct 1993.
46. B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proc. of the 12th National Conf. on Artificial Intelligence*, pages 337–343, Seattle, WA, 1994.
47. B. Selman and H. A. Kautz. An empirical study of greedy local search for satisfiability testing. In *Proc. of the 11th National Conf. on Artificial Intelligence*, pages 46–51, Washington, DC, 1993.
48. B. Selman, H. J. Levesque, and D. G. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of AAAI-92*, pages 440–446, San Jose, CA, 1992.
49. Y. Shang and B. W. Wah. Discrete lagrangian-based search for solving MAX-SAT problems. In *Proc. Int'l Joint Conf. on Artificial Intelligence*. IJCAI, (accepted to appear) Aug. 1997.
50. D. M. Simmons. *Nonlinear Programming for Operations Research*. Prentice-Hall, Englewood Cliffs, NJ, 1975.
51. R. Socic and J. Gu. Fast search algorithms for the N-queen problem. *IEEE Trans. on Systems, Man, and Cybernetics*, 21(6):1572–1576, November 1991.
52. R. Sosič and J. Gu. A polynomial time algorithm for the n-queens problem. *SIGART Bulletin*, 1(3):7–11, October 1990.
53. R. Sosič and J. Gu. 3,000,000 queens in less than one minute. *SIGART Bulletin*, 2(2):22–24, April 1991.
54. R. Sosič and J. Gu. Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Trans. on Knowledge and Data Engineering*, 6(5):661–668, 1994.
55. B. W. Wah, Y. Shang, and Z. Wu. Discrete lagrangian method for optimizing the design of multiplierless QMF filter banks. In *Proc. Int'l Conf. on Application Specific Array Processors*. IEEE, (accepted to appear) July 1997.
56. G. R. Walsh. *Methods of Optimization*. John Wiley and Sons, 1975.
57. S. Zhang and A. G. Constantinides. Lagrange programming neural networks. *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, 39(7):441–452, 1992.