Discrete Lagrangian-Based Search for Solving MAX-SAT Problems *

Benjamin W. Wah and Yi Shang
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street
Urbana, IL 61801, USA
{wah, shang}@manip.crhc.uiuc.edu
URL: http://manip.crhc.uiuc.edu

Abstract

Weighted maximum satisfiability problems (MAX-SAT) are difficult to solve due to the large number of local minima in their search space. In this paper we propose a new discrete Lagrangian based search method (DLM) for solving these problems. Instead of restarting from a new point when the search reaches a local minimum, the Lagrange multipliers in DLM provide a force to lead the search out of the local minimum and move it in a direction provided by the multipliers. Since DLM has very few parameters to be tuned, it can be made deterministic and the results, reproducible. We compare DLM with GRASP in solving a large set of test problems, and show that it finds better solutions and is substantially faster. DLM has a solid theoretical foundation that can be used as a systematic approach for solving general discrete optimization problems.

1 Introduction

The satisfiability (SAT) problem is defined as follows. Given a set of n clauses $\{C_1, C_2, \cdots, C_n\}$ on m variables $x = (x_1, x_2, \cdots, x_m)$, $x_i \in \{0, 1\}$, and a Boolean formula in conjunctive normal form (CNF),

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

find an assignment to the variables so that (1) evaluates to be *true*, or derive its infeasibility if (1) is infeasible.

The *weighted maximum satisfiability problem* (MAX-SAT) is a general case of SAT. In MAX-SAT, each clause C_i is associated with weight w_i. The objective is to find an assignment to the variables that maximizes the sum of the weights of satisfied clauses,

$$\max_{x \in \{0,1\}^m} \sum_{i=1}^{n} w_i S_i(x), \tag{2}$$

where S_i equals 1 if logical assignment x satisfies C_i, and 0 otherwise. This objective is equivalent to minimizing the sum of the weights of unsatisfied clauses.

MAX-SAT problems are difficult to solve for the following reasons. First, they have a large number of local minima in their search space, where a local minimum is a state whose local neighborhood does not include any state that is strictly better. Second, the weights in the objective function of a MAX-SAT problem can lead to a much more rugged search space than the corresponding SAT problem. When a MAX-SAT problem is satisfiable, existing SAT algorithms [Gu, 1989; Selman and Kautz, 1993] developed to find a satisfiable assignment to the corresponding SAT problem can be applied, and the resulting assignment is also optimal for (2). However, this approach does not work when the MAX-SAT problem is not satisfiable. In this case, existing local search SAT methods have difficulties in overcoming the rugged search space.

Methods for solving MAX-SAT can be classified as incomplete and complete, depending on whether they can find the optimal assignment. Complete algorithms can determine optimal assignments. They include mixed integer linear programming methods [Resende *et al.*, 1997] and various heuristics [Hansen and Jaumard, 1990; Goemans and Williamson, 1995; Joy *et al.*, 1997]. They are generally computationally expensive and have difficulties in solving large problems. On the other hand, incomplete methods are usually faster and can solve some large problems that complete methods cannot handle. Many incomplete local search methods have been designed to solve large SAT problems of thousands of variables [Gu, 1989; Selman and Kautz, 1993; Wah and Shang, 1996; Shang and Wah, 1997] and have obtained promising results in solving MAX-SAT problems [Jiang *et al.*, 1995; Resende *et al.*, 1997].

*Research supported by National Science Foundation Grant MIP 96-32316 and National Aeronautics and Space Administration Grant NAG 1-613.

Source code of DLM is at http://manip.crhc.uiuc.edu.

Another approach to solve MAX-SAT is to transform it into continuous formulations. Discrete variables in the original problem are relaxed 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 local minima. Unfortunately, continuous formulations require computationally expensive algorithms, rendering them applicable only to small problems.

A discrete Lagrangian-based search method was developed previously for solving SAT problems [Wah and Shang, 1996; Shang and Wah, 1997]. Traditional Lagrangian theory for solving continuous problems was extended to discrete constrained optimization problems and incorporated in *DLM* (*Discrete Lagrangian method*) that works on discrete variables. Instead of restarting 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 the local minimum and move it in the direction provided by the Lagrange multipliers. Hence, DLM can escape from local minima in a continuous trajectory without restarts, 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. Moreover, DLM has very few algorithmic parameters to be tuned by users, and the search procedure can be made deterministic and the results, reproducible. When applied to the DIMACS SAT benchmarks, DLM generally performs better than the best competing methods and can achieve an order-of-magnitude speedup for many problems.

In this paper we extend the discrete Lagrange-multiplier-based search method to solve MAX-SAT problems. We formulate a MAX-SAT problem in (2) as a discrete constrained optimization problem.

$$\begin{aligned} \min_{x\in\{0,1\}^m} \quad & N(x) = \sum_{i=1}^{n} w_i U_i(x) \\ \text{subject to} \quad & U_i(x) = 0 \quad \forall i \in \{1,2,\ldots,n\}. \end{aligned} \tag{3}$$

where $w_i > 0$, and $U_i(x)$ equals 0 if the logical assignment x satisfies C_i, and 1 otherwise. We propose a mechanism to control the growth rate and magnitude of Lagrange multipliers, which is essential in solving difficult MAX-SAT problems. Our method belongs to the class of incomplete methods that attempt to find approximate solutions in a fixed amount of time.

This paper is organized as follows. Section 2 presents our discrete Lagrangian algorithm. Section 3 discusses issues and alternatives in implementing DLM. Finally, Section 4 presents experimental results in applying DLM to solve a large set of test problems.

2 Discrete Lagrangian Methods (DLM)

In this section we first summarize previous work on Lagrangian methods for solving continuous constrained optimization problems. We then extend continuous Lagrangian methods to discrete constrained optimization problems, and apply it to solve MAX-SAT problems.

2.1 Continuous Lagrangian Methods

Lagrangian methods are classical methods for solving continuous constrained optimization problems [Luenberger, 1984]. Define an equality constrained optimization problem as follows:

$$\begin{aligned} \min_{x\in E^m} \quad & f(x) \\ \text{subject to} \quad & g(x) = 0 \end{aligned} \tag{4}$$

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

$$F(x,\lambda) = f(x) + \sum_{i=1}^{n} \lambda_i g_i(x) \tag{5}$$

where $\lambda = (\lambda_1, \cdots, \lambda_n) \in E^n$ are Lagrange multipliers.

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

$$F(x^*,\lambda) \le F(x^*,\lambda^*) \le F(x,\lambda^*) \tag{6}$$

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

(Continuous) Saddle-Point Theorem [Luenberger, 1984]. x^* is a local minimum to the original problem defined in (4) if and only if there exists λ^* such that (x^*,λ^*) constitutes a saddle point of the associated Lagrangian function $F(x,\lambda)$.

Based on the Saddle Point Theorem, numerical algorithms have been developed to look for saddle points corresponding to local minima in a search space. One typical method is to do descents in the original variable space of x and ascents in the Lagrange-multiplier space of λ. The method can be written as a set of ordinary differential equations as follows:

$$\frac{dx}{dt} = -\nabla_x F(x,\lambda) \quad \text{and} \quad \frac{d\lambda}{dt} = \nabla_\lambda F(x,\lambda) \tag{7}$$

where t is an autonomous variable and ∇ is a gradient.

2.2 General DLM

Little work has been done in applying Lagrangian methods to solve discrete constrained combinatorial search problems. The difficulty lies in the requirement of a differentiable continuous space. To apply Lagrangian methods to discrete optimization problems, we need to develop the counterpart of gradient in discrete space.

Similar to (4), a discrete optimization problem is defined as follows.

$$\begin{array}{ll} \min_{x \in D^m} & f(x) \\ \text{subject to} & g(x) = 0 \end{array} \quad (8)$$

where D is the set of integers. We can also define a discrete Lagrangian function similar to (5).

A saddle point (x^*, λ^*) of $F(x, \lambda)$ is defined as one that satisfies the following condition:

$$F(x^*, \lambda) \leq F(x^*, \lambda^*) \leq F(x, \lambda^*) \quad (9)$$

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

Discrete Saddle-Point Theorem. x^* is a local minimum of (8) if and only if there exists some λ^* such that (x^*, λ^*) constitutes a saddle point of the associated discrete Lagrangian function $F(x, \lambda)$.

The proof is quite straightforward and is based on showing that $U(x^*) = 0$ in both directions. Based on the Discrete Saddle-Point Theorem, we have developed the following DLM to find discrete saddle points.

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

$$x^{k+1} = x^k \ominus \Delta_x F(x^k, \lambda^k) \quad (10)$$

$$\lambda^{k+1} = \lambda^k + U(x^k) \quad (11)$$

where $\ominus$ represents exclusive OR (XOR). We define the *discrete gradient* $\Delta_x F(x, \lambda)$ with respect to x such that $\Delta_x F(x, \lambda) = (\delta_1, \cdots, \delta_m) \in \{0, 1\}^m$ with at most one non-zero δ_i, and it gives $F(x, \lambda)$ the greatest reduction in the neighborhood of x with Hamming distance 1.

DLM provides a theoretical foundation and generalization of local search schemes that optimize the objective alone and clause-weight 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 constraint-weight schemes that rely only on the weights of violated constraints to escape from local minima, DLM also uses the value of the objective function to provide further guidance.

In most existing clause-weight schemes, weights are updated dynamically based on which constraints are violated after each flip or after a series of flips. However, they do not use any objective function that may be important in some stages of the search. For instance, 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, such as the MAX-SAT problems studied in this paper. The dynamic shift in emphasis between the objective and the constraints, depending on their relative values, is the key of DLM.

2.3 DLM Formulation of MAX-SAT

Using the MAX-SAT problem in (3), the Lagrangian function has the same form as in (5):

$$L(x, \lambda) = N(x) + \lambda^T U(x) = \sum_{i=1}^{n} (w_i + \lambda_i) U_i(x) \quad (12)$$

where $x \in \{0, 1\}^m$, $U(x) \in \{0, 1\}^n$, and λ^T (the transpose of $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$) denotes the Lagrange multipliers.

There are three important properties about DLM.

- Since $U(x) = 0$ when $\mathcal{A}$ converges to a discrete saddle point, x is globally (not just locally) optimal when $\mathcal{A}$ converges. If any of the constraints in $U(x)$ is not satisfied, then λ will continue to evolve to handle the unsatisfied constraints, and the search continues.
- Since $U_i(x)$ is weighted by $w_i + \lambda_i$ in (12), the minimization of $L(x, \lambda)$ by DLM while λ is changing is equivalent to perturbing the weights in the original objective function defined in (3). In contrast, existing MAX-SAT methods explore the search space without modifying the weights. This is the major difference between existing methods and DLM.
- The minimization of (12) is equivalent to a clause-weight scheme. This happens because the objective and the constraints in (3) are dependent. In general, when the objective and the constraints are independent, descents in the original-variable space and ascents in the Lagrange-multiplier space form two counteracting forces to converge to saddle points.

3 Implementation Issues of DLM

In this section, we discuss various considerations in implementing DLM. Figure 1 shows the pseudo code of $\mathcal{A}_1$, a generic DLM implementing (10) and (11). Define one *iteration* as one pass through the while loop. In the following, we describe some of our design considerations.

(a) *Initial Points* (Lines 1-2). DLM is started from either the origin or from a random initial point generated using a fixed random seed. Further, λ is always set to zero. The fixed initial points allow the results to be reproducible easily.

1. Set initial x randomly by a fixed random seed
2. Set initial λ to be zero
3. **while** x is not a feasible solution, *i.e.*, $N(x) > 0$
4. update x: $x \longleftarrow x \ominus \Delta_x L(x, \lambda)$
5. update incumbent if $N(x)$ is better than the current incumbent
6. **if** condition for updating λ is satisfied **then**
7. update λ: $\lambda \longleftarrow \lambda + c \times U(x)$
8. **end if**
9. **end while**

gure 1: Generic DLM $\mathcal{A}_1$ for solving SAT problems.

(b) *Descent and Ascent Strategies* (Line 4). There e two ways to calculate $\Delta_x L(x, \lambda)$: greedy and hill-mbing, each involving a search in the range of Ham-ing distance one from the current x. In a *greedy strat-y*, the assignment leading to the maximum decrease in (x, λ) is selected to update the current assignment. In *ill-climbing*, the first assignment leading to a decrease $L(x, \lambda)$ is selected to update the current assignment. Depending on the order of search and the number of as-ignments that can be improved, hill-climbing strategies re generally much faster than greedy strategies.

Among various ways of hill-climbing, we used two al-ernatives in our implementation: flip the variables one by one in a predefined order, or maintain a list of vari-ables that can improve the current $L(x, \lambda)$ and just flip the first variable in the list. The first alternative is fast when the search starts. By starting from a randomly generated initial assignment, it usually takes very few flips to find a variable that improves the current $L(x, \lambda)$. As the search progresses, there are fewer variables that can improve $L(x, \lambda)$. At this point, the second alterna-tive becomes more efficient and should be applied.

(c) *Conditions for updating* λ (Line 6). The fre-quency 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 of the search to avoid biasing the search in the Lagrange-multiplier space of λ over the original variable space of x.

Experimental results show that a good strategy is to update λ only when $\Delta_x L(x, \lambda) = 0$. At this point, a local minimum in the original variable space is reached, and the search can only escape from it by updating λ.

(d) *Amount of update of* λ (Line 7). A parameter c controls the magnitude of changes in λ. In general, c can be a vector of real numbers, allowing non-uniform updates of λ across different dimensions and possibly across time. In our experiments, $c = 1$ has been found to work well for most of the benchmarks tested. How-

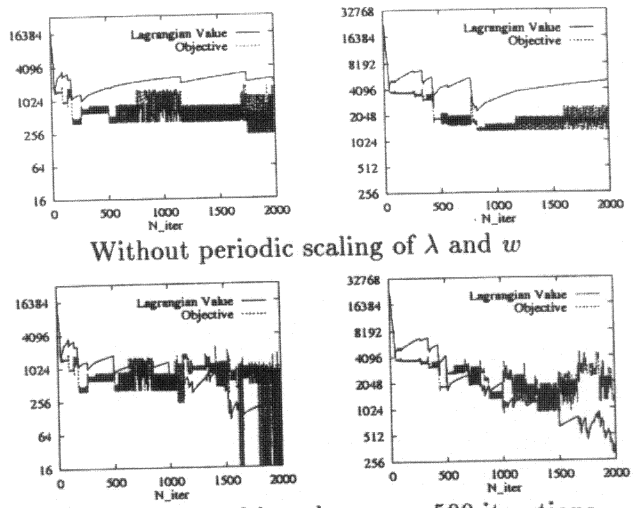

Figure 2: Execution profiles of *jnh1* (left) and *jnh10* (right). *jnh1* is satisfiable and *jnh10* is not.

ever, for some larger problems, a larger c could result in shorter search time and better solutions.

The update rule in Line 7 results in nondecreasing λ because $U(x)$ is either 0 or 1. In contrast, in continuous problems λ_i of constraint $g_i(x) = 0$ increases when $g_i(x) > 0$ and decreases when $g_i(x) < 0$.

From previous experience in solving SAT problems using DLM, some λ values for some difficult problems can become very large after tens of thousands of iterations. At this point, the Lagrangian search space (12) becomes very rugged, and the search has difficulty in identifying an appropriate direction to move. To cope with this problem, λ should be reduced periodically to keep them small and to allow DLM to better identify good solutions.

The situation is worse in MAX-SAT because the weights of clauses, w, can be in a large range, making $L(x, \lambda)$ in (12) even more rugged and difficult to search. Here, $\lambda + w$ in MAX-SAT has a similar role as λ in SAT. Without any mechanism to reduce $\lambda + w$, $L(x, \lambda)$ can become very large and rugged as the search progresses.

This situation is illustrated in the first two graphs of Figure 2 that show the behavior of DLM when it was applied to solve two MAX-SAT problems. The graphs show that the Lagrangian values vary in a very large range, and the Lagrangian space is difficult to search.

One way to overcome this problem is to reduce λ and w periodically. For instance, in the last two graphs of Figure 2, $\lambda + w$ was scaled down by a factor of 2 every 500 iterations. This strategy reduces $L(x, \lambda)$ and restricts the growth of the Lagrange multipliers, leading to faster solutions for many test problems.

Figure 3 shows $\mathcal{A}_2$, our implementation of DLM to

```
Set initial x
Set λ = 0
Set c = 1
Set κ = n/3, where n is the number of variables
Set λ reduction interval I_λ (e.g. 500)
Set reduction ratio r (e.g. 2.0)
Set base λ value λ_b (e.g. 1)
Set base weight value w_b (e.g. 1)
while x is not a feasible 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, L(x, λ) 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 s.t. 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
    if iteration index mod I_λ = 0 then
        Reduce λ and weights for all clauses if possible,
        e.g. λ ⟵ max(λ_b, λ/r), w_i ⟵ max(w_b, w_i/r)
    end_if
    if x is better than incumbent, keep x as incumbent.
end_while
```

Figure 3: $\mathcal{A}_2$: An efficient implementation of DLM.

solve MAX-SAT. $\mathcal{A}_2$ uses the first strategy of hill-climbing in descents in the beginning and switches to the second strategy later. We use κ to control the switch from the first strategy to the second. We found that $\kappa = n/3$ works well and used it in our experiments.

$\mathcal{A}_2$ updates λ only when a local minimum is reached. In our experiments, we have used a simple scheme that increases λ by a constant across all clauses.

$\mathcal{A}_2$ controls the magnitude of $L(x, \lambda)$ by periodically reducing λ and w. More sophisticated adaptive mechanisms can be developed to lead to better performance.

4 Experimental Results

In this section we evaluate DLM using some benchmarks and compare it with previous results obtained by GRASP [Resende and Feo, 1996; Resende *et al.*, 1997]. GRASP is a greedy randomized adaptive search procedure that has been shown to quickly produce good-quality solutions for a wide variety of combinatorial optimization problems, including SAT and MAX-SAT.

The 44 MAX-SAT problems tested by GRASP were derived from DIMACS SAT benchmark *jnh* that have 100 variables and between 800 and 900 clauses. Some of them are satisfiable, while others are not. Their weights are integers generated randomly between 1 and 1000.[1]

Our DLM code was written in C. The experiments were run on a 50-MHz Sun SparcStation 10/51. The code was compiled using *gcc* with the "-O" option.

In our experiments, we first studied the effect of reducing λ and $L(x, \lambda)$. We compared the results with and without periodic reduction of λ. We ran DLM between 5 to 20 times from random initial points, each for 10,000 iterations, for each test problem. We then took the best solution of the multiple runs as our final solution.

When λ was not reduced periodically, DLM found 25 optimal solutions out of the 44 problems (based on the best results of 10 runs). When λ was reduced periodically, we tried several values of I_λ, the reduction interval, and r, the reduction ratio. Using I_λ of 100, 1000, 2000 and 5000 and r of 1.5, 2 and 4, we found significant improvement in solution quality when λ was reduced periodically (except for the case when $I_\lambda = 5000$). The best combination is when $I_\lambda = 500$ and $r = 2$, in which 12 more optimal solutions were found than without λ reduction, 18 test problems were improved, and no solution is worse. Overall, we found optimal solutions in 39 out of the 44 test problems (based on best of 20 runs).

Table 1 compares the solutions obtained by DLM and by GRASP. The results of GRASP are from [Resende *et al.*, 1997] that were run for 10,000 iterations on a 250-MHz SGI Challenge with MIPS R4400. In contrast, DLM was run between 5 and 20 times from random initial points using 10,000 iterations each. Our results show significant improvement over GRASP in 43 of the 44 test problems. It is important to note that, although the differences between GRASP's results and the optimal solutions are relatively small, improvements in this range are the most difficult for any search algorithm.

Table 2 shows the average time in seconds for 10,000 iterations of GRASP and DLM. Since GRASP was run on a faster machine, we estimate that one iteration of DLM takes around three orders of magnitude less time than that of GRASP on the same machine.

GRASP found better results when the number of iterations is increased. Using the ten test problems reported in [Resende *et al.*, 1997] that were run for 100,000 iterations of GRASP, Table 3 compares the results of DLM and GRASP with longer runs. It shows that DLM still found better solutions in all the problems.

Increasing the number of iterations of DLM could improve the best solutions found. DLM was able to find 40 out of 44 optimal solutions when the number of iterations is increased to 100,000 (best of 10 runs).

Our experimental results show that DLM solved these benchmark problems much faster and found better solutions for a majority of the test problems.

[1] The 44 benchmark problems can be obtained from ftp://netlib.att.com/math/people/mgcr/data/maxsat.tar.gz.

Table 1: Comparison of results of DLM and those of GRASP with respect to the optimal solutions. (jnh1-19: 850 clauses, $\sum_i w_i = 420925$; jnh201-220: 800 clauses, $\sum_i w_i = 394238$; jnh301-310: 900 clauses, $\sum_i w_i = 444854$. GRASP was run 10,000 iterations on a 250-MHz SGI Challenge with MIPS R4400 whereas DLM was run 5-20 times, each for 10,000 iterations, from random starting points on a Sun SparcStation 10/51. For DLM, $I_\lambda = 500$ and $r = 2$.)

Problem ID	Deviations from Optimal Sol. # Runs of DLM 5	10	20	GRASP	Optimal Solutions
jnh1	0	0	0	-188	420925
jnh4	-41	-41	-41	-215	420830
jnh5	-164	-131	0	-254	420742
jnh6	0	0	0	-11	420826
jnh7	0	0	0	0	420925
jnh8	0	0	0	-578	420463
jnh9	-7	-7	-7	-514	420592
jnh10	0	0	0	-275	420840
jnh11	-8	0	0	-111	420753
jnh12	0	0	0	-188	420925
jnh13	0	0	0	-283	420816
jnh14	-77	0	0	-314	420824
jnh15	-79	-79	0	-359	420719
jnh16	0	0	0	-68	420919
jnh17	0	0	0	-118	420925
jnh18	-98	0	0	-423	420795
jnh19	0	0	0	-436	420759
jnh201	0	0	0	0	394238
jnh202	0	0	0	-187	394170
jnh203	0	0	0	-310	394199
jnh205	0	0	0	-14	394238
jnh207	0	0	0	-137	394238
jnh208	0	0	0	-172	394159
jnh209	0	0	0	-207	394238
jnh210	0	0	0	0	394238
jnh211	0	0	0	-240	393979
jnh212	0	0	0	-195	394238
jnh214	0	0	0	-462	394163
jnh215	0	0	0	-292	394150
jnh216	-149	0	0	-197	394226
jnh217	0	0	0	-6	394238
jnh218	0	0	0	-139	394238
jnh219	0	0	0	-436	394156
jnh220	0	0	0	-185	394238
jnh301	0	0	0	-184	444854
jnh302	**-510**	**-338**	**-338**	-211	444459
jnh303	-152	-143	-143	-259	444503
jnh304	-106	0	0	-319	444533
jnh305	-196	-194	-194	-609	444112
jnh306	0	0	0	-180	444838
jnh307	0	0	0	-155	444314
jnh308	-103	-60	0	-502	444724
jnh309	0	0	0	-229	444578
jnh310	0	0	0	-109	444391

Table 2: Comparison of average CPU time in seconds for 10,000 iterations of DLM and GRASP.

Problem	# Clauses	GRASP	DLM
jnh1-19	850	799.8	1.7
jnh201-220	800	692.9	1.7
jnh301-310	900	937.8	2.0

Table 3: Comparison of solutions found by DLM and GRASP with longer iterations.

Problem Id	GRASP 10K	100K	DLM 10K iter	Optimal Solutions
jnh1	-188	-77	0	420925
jnh10	-275	-259	0	420840
jnh11	-111	-111	0	420753
jnh12	-188	-54	0	420925
jnh201	0	0	0	394238
jnh202	-187	-141	0	394170
jnh212	-195	-50	0	394238
jnh304	-319	0	0	444533
jnh305	-609	-368	-194	444112
jnh306	-180	-63	0	444838

References

[Goemans and Williamson, 1995] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *JACM*, 42(6):1115–1145, 1995.

[Gu, 1989] J. Gu. *Parallel Algorithms and Architectures for Very Fast AI Search*. PhD thesis, Dept. of Computer Science, University of Utah, August 1989.

[Hansen and Jaumard, 1990] P. Hansen and R. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.

[Jiang *et al.*, 1995] Y. Jiang, H. Kautz, and B. Selman. Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT. In *Proc. of 1st Int'l Joint Workshop on Artificial Intelligence and Operations Research*, Timberline, Oregon, 1995.

[Joy *et al.*, 1997] S. Joy, J. Mitchell, and B. Borchers. A branch and cut algorithm for MAX-SAT and weighted MAX-SAT. In *Satisfiability Problem: Theory and Applications. DIMACS Series on Discrete Mathematics and Theoretical Computer Science*. AMS (to appear), 1997.

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

[Resende and Feo, 1996] 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, AMS, 1996.

[Resende *et al.*, 1997] 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 Appl. DIMACS Series on Discrete Mathematics and Theoretical Computer Science*. AMS, 1997.

[Selman and Kautz, 1993] B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proc. of 13th Int'l Joint Conf. on Artificial Intelligence*, pages 290–295, 1993.

[Shang and Wah, 1997] Y. Shang and B. Wah. Discrete lagrangian methods for solving satisfiability problems. *Journal of Global Optimization (to appear)*, 1997.

[Wah and Shang, 1996] B. W. Wah and Y. Shang. A discrete lagrangian-based global-search method for solving satisfiability problems. In *Proc. DIMACS Workshop on Satisfiability Problem: Theory and Applications*. AMS, March 1996.