GRASP with Path Relinking for the Weighted MAXSAT Problem

PAOLA FESTA
University of Napoli Federico II
PANOS M. PARDALOS
University of Florida
LEONIDAS S. PITSOULIS
Aristotle University of Thessaloniki
and
MAURICIO G. C. RESENDE
AT&T Labs Research

A GRASP with path relinking for finding good-quality solutions of the weighted maximum satisfiability problem (MAX-SAT) is described in this paper. GRASP, or Greedy Randomized Adaptive Search Procedure, is a randomized multistart metaheuristic, where, at each iteration, locally optimal solutions are constructed, each independent of the others. Previous experimental results indicate its effectiveness for solving weighted MAX-SAT instances. Path relinking is a procedure used to intensify the search around good-quality isolated solutions that have been produced by the GRASP heuristic. Experimental comparison of the pure GRASP (without path relinking) and the GRASP with path relinking illustrates the effectiveness of path relinking in decreasing the average time needed to find a good-quality solution for the weighted maximum satisfiability problem.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: Algorithm Design and Analysis

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Algorithms, heuristics, GRASP, path relinking, experimentation, performance, time-to-target plots

Authors' addresses: Paola Festa, Department of Mathematics and Applications, University of Napoli Federico II, Compl. MSA, Via Cintia, 80126, Napoli, Italy; email: paola.festa@unina.it; Panos M. Pardalos, Department of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, FL 32611; email: pardalos@ufl.edu; Leonidas S. Pitsoulis, Department of Mathematical and Physical Sciences, School of Engineering, Aristotle University of Thessaloniki, Thessaloniki, GR54124, Greece; email: pitsouli@gen.auth.gr; Mauricio G. C. Resende, Internet and Network Systems Research Center, AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, NJ 07932; email: mgcr@research.att.com.

1. INTRODUCTION

A propositional formula Φ on a set of n Boolean variables $V = \{x_1, \ldots, x_n\}$ in conjunctive normal form (CNF), is a conjunction on a set of m clauses $\mathbb{C} = \{C_1, \ldots, C_m\}$. Each clause C_i is a disjunction of $|C_i|$ literals, where each literal l_{ij} is either a variable x_j or its negation $\neg x_j$. Formally, we write

$$\Phi = \bigwedge_{i=1}^{m} C_i = \bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{|C_i|} l_{ij} \right)$$

A clause is satisfied if at least one of its literals evaluates to 1 (true), which means that either one of the unnegated Boolean variables has the value of 1 or a negated variable has the value of 0. The propositional formula is said to be satisfied if all of its clauses are satisfied. In the satisfiability problem (SAT), one must decide whether there exists an assignment of values to the variables such that a given propositional formula is satisfied. SAT was the first problem to be shown to be *NP*-complete [Cook 1971]. The maximum satisfiability problem (MAX-SAT) is a generalization of SAT, where given a propositional formula, one is interested in finding an assignment of values to the variables, which maximizes the number of satisfied clauses. Generalizing even further, if we introduce a positive weight w_i, for each clause C_i, then the weighted MAX-SAT problem consists in finding an assignment of values to the variables such that the sum of the weights of the satisfied clauses is maximized.The MAX-SAT has many applications both theoretical and practical, in areas such as complexity theory, combinatorial optimization, and artificial intelligence [Battiti and Protasi 1998]. It is an intractable problem in the sense that no polynomial time algorithm exists for solving it unless P = NP, which is evident, since it generalizes the satisfiability problem [Garey and Johnson 1979].

Because of the computational complexity of the MAX-SAT, there has been an extensive research effort devoted to the development of approximation and heuristic algorithms for solving it. An ϵ-approximate algorithm for the MAX-SAT is a polynomial time algorithm, which finds a truth assignment to the variables that results in a total weight of the satisfied clauses that is at least ϵ times the optimum ($0 < \epsilon < 1$). We will refer to ϵ of an approximation algorithm as its *performance ratio*. The first approximation algorithms for the MAX-SAT were introduced in Johnson [1974], where Johnson presented two algorithms with performance rations $(k-1)/k$ and $(2^k-1)/2^k$, where k is the least number of literals in any clause. For the general case $k = 1$ they both translate to a 1/2-approximation algorithm, while it has been shown in Chen et al. [1997] that the second algorithm is, in fact, a 2/3-approximation algorithm. A 3/4-approximation algorithm, based on network flow theory, was presented by Yannakakis [1992] and also in Goemans and Williamson [1994]. Currently the best deterministic polynomial time approximation algorithm for MAX-SAT achieves a performance ratio of 0.758 and is based on semidefinite programming [Goemans and Williamson 1995], while there is also a randomized algorithm with performance ratio 0.77 [Asano 1997]. Better approximation bounds

for special cases of the problem in which, for instance, we restrict the number of literals per clause or impose the condition that the clauses are satisfiable have also been found [Feige and Goemans 1995; Karloff and Zwick 1997; Trevisan 2000]. With respect to inapproximability results, it is known [Hastad 2001] that unless P = NP there is no approximation algorithm with performance ratio greater than 7/8 for the MAX-SAT in which every clause contains exactly three literals, thereby limiting the general case as well.

Local search is the main ingredient for most of the heuristic algorithms that have appeared in the literature for solving the MAX-SAT, where, in conjunction with various techniques for escaping local optima, they provide solutions which exceed the theoretical upper bound of approximating the problem. We can divide the heuristic algorithms that have appeared in the literature into two main classes. The first class being those heuristics which use the history of the search in order to construct a new solution, such as Tabu Search [Hansen and Jaumard 1990], HSAT [Gent and Walsh 1993], and Reactive Search [Battiti and Protasi 1997], and those that are not history sensitive such as Simulated Annealing [Spears 1996], GSAT [Selman et al. 1992] and GRASP [Resende and Feo 1996; Resende et al. 1997]. Surveys of approximation and heuristic algorithms for solving the MAX-SAT can be found in Battiti and Protasi [1998] and Hansen and Jaumard [1990].

GRASP is a constructive multistart metaheuristic, which has been applied to a wide range of well known combinatorial optimization problems with favorable experimental results [Resende and Pitsoulis 2002]. Each iteration of a generic GRASP procedure consists of two phases: a construction phase, where a solution is constructed in a semigreedy, i.e., randomized greedy, fashion [Hart and Shogan 1987]; and a local search phase, where the local optimum is found in the neighborhood of the constructed solution. GRASP can, therefore, be thought of as a *memoryless* procedure, where past information from previous solutions is not used for the construction of a new solution.

Resende et al. [1997, 2000] describes a GRASP implementation for solving the weighted MAX-SAT and report extensive computational results on a set of weighted SAT benchmark instances [Johnson and Trick 1996], which indicate that the heuristic produces good-quality solutions. In this paper, we show how memory can be incorporated in the GRASP for weighted MAX-SAT proposed in Resende et al. [1997]. At each iteration of the GRASP heuristic, a path of feasible solutions linking the current solution with a solution from a set of elite (or good-quality) solutions previously produced by the algorithm is explored. Path relinking has been used as a memory mechanism in GRASP [Resende and Ribeiro 2005] resulting in faster convergence of the algorithm.

The remainder of the paper is organized as follows. In Section 2, we state the implementation of GRASP for the MAX-SAT from Resende et al. [1997], while in Section 3 we describe how to apply path relinking for the MAX-SAT. Finally, in Section 4, computational results are presented, which demonstrate empirically that path relinking results in faster convergence of GRASP.

```
procedure GRASP(MaxIter,RandomSeed)
1     c_best := 0;
2     do k = 1, ..., MaxIter →
3           x :=ConstructSolution(RandomSeed);
4           x :=LocalSearch(x);
5           if c(x) > c_best →
6                 x_best := x;
7                 c_best := c(x_best);
8           endif;
9     od;
7     return x_best
end GRASP;
```

Fig. 1. Pseudocode of GRASP for maximization problem.

2. GRASP FOR THE WEIGHTED MAX-SAT

The construction and local search phase of GRASP are described in detail in Resende et al. [1997], while in Resende et al. [2000], a complete Fortran implementation is given along with extensive computational runs. In this section, we provide a description in order to facilitate the discussion of path relinking that will follow in the next section. Given a set of clauses $\mathbb{C}$ and a set of Boolean variables V, let us denote by $\mathbf{x} \in \{0, 1\}^n$ the *truth assignment* that corresponds to the truth values assigned to the variables, while let $c(\mathbf{x})$ denote the sum of the weights of the satisfied clauses as implied by $\mathbf{x}$. Without loss of generality, we can assume that all the weights w_i of the clauses are positive integers. Given any two truth assignments $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$, let us denote their *difference set*

$$\Delta(\mathbf{x}, \mathbf{y}) := \{i : x_i \neq y_i, i = 1, \ldots, n\} \tag{1}$$

and their *distance*

$$d(\mathbf{x}, \mathbf{y}) := |\Delta(\mathbf{x}, \mathbf{y})| = \sum_{i=1}^{n} |x_i - y_i| \tag{2}$$

which is the Hamming distance, and will be used as a measure of proximity between two solutions. The GRASP procedure is shown in Figure 1. In the two following sections, we will describe the procedures `Construct Solution(RandomSeed)` and `LocalSearch(`$\mathbf{x}$`)` shown in lines 3 and 4 of Figure 1, respectively.

2.1 Construction Phase

In the construction phase of the algorithm (line 3 of Figure 1), we build a solution one element at a time in a greedy randomized fashion. We maintain a *restricted candidate list* (RCL) throughout the procedure, which contains elements that correspond to assignments of yet-unassigned variables to either 1 (`true`) or 0 (`false`). Choosing an element to be added to a partial solution from the RCL corresponds to assigning the corresponding truth value to the given variable. Given any partial solution, which corresponds to a set of satisfied clauses, we want the next element to be added to the solution to maximize the total weight of the unsatisfied clauses that become satisfied after the assignment. Let

$N = \{1, 2, \ldots, n\}$ and $M = \{1, 2, \ldots, m\}$ be sets of indices for the set of variables and clauses, respectively. Moreover, for $i \in N$, let Γ_i^+ be the set of currently unsatisfied clauses that would become satisfied if variable x_i were to be set to `true`, and Γ_i^- be the set of currently unsatisfied clauses that would become satisfied if variable x_i were to be set to `false`. Let us denote by γ_j^+ and γ_j^- the gain in the objective function value if we set the unassigned variable x_j to 1 and 0, specifically

$$\gamma_i^+ = \sum_{j \in \Gamma_i^+} w_j \quad \text{and} \quad \gamma_i^- = \sum_{j \in \Gamma_i^-} w_j$$

If $X \subseteq V$ is the set of already assigned variables, we compute the best gain

$$\gamma^* := \max\{\gamma_j^+, \gamma_j^- : j \text{ such that } x_j \in V \setminus X\}$$

and keep in the RCL only those assignments with γ_j^+ and γ_j^- that are greater or equal to $\alpha \cdot \gamma^*$ where $0 \leq \alpha \leq 1$ is a parameter. A random choice from the RCL corresponds to a new assignment $x_s = 1$ ($x_s = 0$), which is added to our partial solution, that is, $X := X \cup \{x_s\}$. After each such addition to the partial solution, the sets Γ_i^+, Γ_i^-, as well as the gains γ_j^+ and γ_j^- are updated, a process illustrated in Figure 2, where s is the index of the variable just added to the partial solution. We can recognize two possible cases. If the variable just assigned was set to `true` then Γ^+, Γ^-, γ^+ and γ^- are updated in lines 5, 8, 12, and 13, while if the variable just assigned was set to `false`, then Γ^+, Γ^-, γ^+ and γ^- are updated in lines 19, 22, 26, and 27. The process is repeated until $|X| = n$.

The parameter α reflects the ratio of randomness versus greediness in the construction process, where $\alpha = 1$ corresponds to a pure greedy selection for a new assignment and $\alpha = 0$ to a pure random assignment.

2.2 Local Search

Having completed a truth assignment $\mathbf{x}$, we apply local search (line 4 of Figure 1) in order to guarantee local optimality. The 1-flip neighborhood is used in the local search, which is defined as

$$N_1(\mathbf{x}) := \{\mathbf{y} \in \{0, 1\}^n : d(\mathbf{x}, \mathbf{y}) = 1\} \tag{3}$$

where $\mathbf{x}$ is a local maximum if, and only if, $c(\mathbf{x}) \geq c(\mathbf{y})$ for all $\mathbf{y} \in N_1(\mathbf{x})$. A straightforward implementation of the local search procedure, would require $|N_1| = n$ function evaluations to find the best solution in a given neighborhood, where for a given assignment each function evaluation computes the sum of the weights of the satisfied clauses. If we wish to reach a local maximum, we might need an exponential number of computational steps [Johnson et al. 1988; Krentel 1988]. We can, however, exploit the structure of the neighborhood to reduce the computational effort.

Given an initial solution $\mathbf{x}$ define G_i to be the gain in total weight resulting from flipping variable x_i in $\mathbf{x}$, for all i. Let k be such that $G_k = \max\{G_i \mid i \in N\}$. If $G_k = 0$ then $\mathbf{x}$ is the local maximum and local search ends. Otherwise, the truth assignment resulting from flipping x_k in $\mathbf{x}$, is a local maximum and, hence, we

```
procedure AdaptGreedyFunction(s)
1     if s > 0 →
2           for j ∈ Γ_s^+ →
3                 for k ∈ L_j (k ≠ j) →
4                       if x_k is unnegated in clause j →
5                             Γ_k^+ = Γ_k^+ − {j}; γ_k^+ = γ_k^+ − w_j;
6                       fi;
7                       if x_k is negated in clause j →
8                             Γ_k^− = Γ_k^− − {j}; γ_k^− = γ_k^− − w_j;
9                       fi;
10                rof;
11          rof;
12          Γ_s^+ = ∅; Γ_s^− = ∅;
13          γ_s^+ = 0; γ_s^− = 0;
14    fi;
15    if s < 0 →
16          for j ∈ Γ_−s^− →
17                for k ∈ L_j (k ≠ j) →
18                      if x_k is unnegated in clause j →
19                            Γ_k^+ = Γ_k^+ − {j}; γ_k^+ = γ_k^+ − w_j;
20                      fi;
21                      if x_k is negated in clause j →
22                            Γ_k^− = Γ_k^− − {j}; γ_k^− = γ_k^− − w_j;
23                      fi;
24                rof;
25          rof;
26          Γ_−s^+ = ∅; Γ_−s^− = ∅;
27          γ_−s^+ = 0; γ_−s^− = 0;
28    fi;
29    return
end AdaptGreedyFunction;
```

Fig. 2. `AdaptGreedyFunction` pseudocode.

only need to update the G_i values such that the variable x_i occurs in a clause in which variable x_k occurs (since the remaining G_i values do not change in the new truth assignment). Upon updating the G_i values, we repeat the same process, until $G_k = 0$ where the local search procedure is terminated. The procedure is described in the pseudocode in Figure 3. Initially in line 2, we calculate the G_i values for all the variables in a given assignment $\mathbf{x}$. Given an index k that corresponds to the variable x_k that is flipped, procedure `UpdateGains` is used to update the G_i values returned in an array G. In lines 4 through 7, the procedure finds a local maximum. The value of the local maximum is saved in line 8.

3. PATH RELINKING

Path relinking was originally proposed by Glover [1996] as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search. Given any two elite solutions, their common elements are kept constant, and the space of solutions spanned by these elements is searched with the objective of finding a better solution. The size of the solution space grows exponentially with the distance between the *initial* and *guiding*

```
procedure LocalSearch(x,BestSolutionFound)
1     BestSolutionFound = c(x);
2     GenerateGains(x,G);
3     G_k = max{G_i | i = 1,...,n} ;
4     for G_k ≠ 0 →
5           Flip value of x_k;
6           UpdateGains(x,G,k);
7     rof;
8     BestSolutionFound = c(x);
9     return;
end LocalSearch;
```

Fig. 3. The local search procedure in pseudocode.

solutions and, therefore, only a small part of the space is explored by path relinking. Path relinking has been applied to GRASP as an enhancement procedure in various problems [Aiex et al. 2003, 2005; Canuto et al. 2001; Laguna and Martí 1999; Resende and Ribeiro 2003; Ribeiro et al. 2002], where it can be empirically concluded that it speeds up convergence of the algorithm. A recent survey of GRASP with path relinking is given in Resende and Ribeiro [2005].

We now describe the integration of path relinking into the pure GRASP algorithm described in Section 2. Path relinking will always be applied to a pair of solutions $\mathbf{x}, \mathbf{y}$, where one is the solution obtained from the current GRASP iteration, and the other is a solution from an elite set of solutions. We call $\mathbf{x}$ the *initial solution*, while $\mathbf{y}$ is the *guiding solution*. The set of elite solutions will be denoted by $\mathcal{E}$ and its size will not exceed `MaxElite`. Let us denote the set of solutions spanned by the common elements of $\mathbf{x}$ and $\mathbf{y}$ as

$$S(\mathbf{x}, \mathbf{y}) := \{\mathbf{w} \in \{0,1\}^n : w_i = x_i = y_i, i \notin \Delta(\mathbf{x}, \mathbf{y})\} \setminus \{\mathbf{x}, \mathbf{y}\} \tag{4}$$

where it is evident that $|S(\mathbf{x}, \mathbf{y})| = 2^{d(\mathbf{x},\mathbf{y})} - 2$. The underlying assumption of path relinking is that there exist good-quality solutions in $S(\mathbf{x}, \mathbf{y})$, since this space consists of all solutions, which contain the common elements of two good solutions $\mathbf{x}, \mathbf{y}$. Taking into consideration that the size of this space is exponentially large, we will adopt a greedy search where a path of solutions

$$\mathbf{x} = \mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{d(\mathbf{x},\mathbf{y})} = \mathbf{y}$$

is constructed, such that $d(\mathbf{w}_i, \mathbf{w}_{i+1}) = 1, \quad i = 0, \ldots, d(\mathbf{x}, \mathbf{y}) - 1$, and the best solution from this path is chosen. Note that since both $\mathbf{x}, \mathbf{y}$ are local maxima in some neighborhood N_1 by construction,[1] in order for $S(\mathbf{x}, \mathbf{y})$ to contain solutions that are not contained in the neighborhoods of $\mathbf{x}$ or $\mathbf{y}$, we must have $d(\mathbf{x}, \mathbf{y}) > 3$. Therefore, we need not apply path relinking between any two solutions, which are not sufficiently far apart, since it is certain that we will not find a new solution that is better than both $\mathbf{x}$ and $\mathbf{y}$.

The pseudocode, which illustrates the exact implementation for the path relinking procedure, is shown in Figure 4. We assume that our initial solution will always be the elite set solution, while the guiding solution is the GRASP iterate. This way we allow for greater freedom to search the neighborhood around the

[1]Where the same metric $d(\mathbf{x}, \mathbf{y})$ is used.

```
procedure PathRelinking(y, E)
1     Randomly select a solution x ∈ {z ∈ E : d(y, z) > 4};
2     w_0 := x;
3     w* := x;
4     for k = 0, ..., d(x, y) − 2 →
5           max:= 0
6           for each i ∈ Δ(w_k, y) →
7                 w :=flip(w_k, i);
8                 if c(w) > max →
9                       i* := i;
10                      max:= c(w);
11                fi;
12          rof;
13          w_{k+1} :=flip(w_k, i*);
14          if c(w_{k+1}) > c(w*) → w* := w_{k+1};
15    endfor;
16    return (w*);
end PathRelinking;
```

Fig. 4. Pseudocode of path relinking for maximization problem.

elite solution. In line 1, we select at random among the elite set elements, an initial solution $\mathbf{x}$ that differs sufficiently from our guiding solution $\mathbf{y}$. In line 2, we set the initial solution as $\mathbf{w}_0$, and in line 3 we save $\mathbf{x}$ as the best solution. The loop in lines 4 through 15 computes a path of solutions $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{d(\mathbf{x},\mathbf{y})-2}$, and the solution with the best objective function value is returned in line 16. This is achieved by advancing one solution at a time in a greedy manner, as illustrated in lines 6 through 12, while the function $\texttt{flip}(\mathbf{w}_k, i)$ returns the assignment obtained by flipping the value of the variable w_i in solution $\mathbf{w}_k$. It is noted that the path of solutions never enters the neighborhood of $\mathbf{y}$.

The integration of the path relinking procedure with the pure GRASP is shown in Figure 5, specifically, in lines 6 through 11. The pool of elite solutions is initially empty and, until it reaches its maximum size, no path relinking takes place. After a solution $\mathbf{y}$ is found by GRASP, it is passed to the path relinking procedure to generate another solution. Note here that we may get the same solution $\mathbf{y}$ after path relinking. The procedure $\texttt{AddToElite}(\mathcal{E}, \mathbf{y})$ attempts to add to the elite set of solutions the currently found solution. A solution $\mathbf{y}$ is added to the elite set $\mathcal{E}$ if either one of the following conditions holds:

1. $c(\mathbf{y}) > \max\{c(\mathbf{w}) : \mathbf{w} \in \mathcal{E}\}$,
2. $c(\mathbf{y}) > \min\{c(\mathbf{w}) : \mathbf{w} \in \mathcal{E}\}$ and $d(\mathbf{y}, \mathbf{w}) > \beta n, \quad \forall \mathbf{w} \in \mathcal{E}$, where β is a parameter between 0 and 1 and n is the number of variables.

If $\mathbf{y}$ satisfies either of the above, it then replaces an elite solution $\mathbf{z}$ of weight not greater than $c(\mathbf{y})$ and most similar to $\mathbf{y}$, i.e., $\mathbf{z} = \text{argmin}\{d(\mathbf{y}, \mathbf{w}) : \mathbf{w} \in \mathcal{E}, c(\mathbf{w}) \leq c(\mathbf{y})\}$.

4. COMPUTATIONAL RESULTS

In this section, we report on an experiment designed to determine the effect of path relinking on the convergence of the GRASP for MAX-SAT described in

```
procedure GRASP+PR(MaxIter,RandomSeed)
1     c_best := 0;
2     E := ∅;
3     do k = 1, ..., MaxIter →
4           x :=ConstructSolution(RandomSeed);
5           x :=LocalSearch(x);
6           if |E| =MaxElite →
7                 x :=PathRelinking(x, E);
8                 AddToElite(E, x);
9           else
10                E := E ∪ {x};
11          endif;
12          if c(x) > c_best →
13                x_best := x;
14                c_best := c(x_best);
15          endif;
16    od;
17    return x_best
end GRASP+PR;
```

Fig. 5. Pseudocode of GRASP with path relinking for maximization problem.

Resende et al. [2000].[2] After downloading the Fortran source code, we modified it to enable recording of the elapsed time between the start of the first GRASP iteration and when a solution is found having weight greater than or equal to a given target value. We call this pure GRASP implementation `grasp`. Using `grasp` as a starting point, we implemented path relinking, making use of the local search code in `grasp`. The GRASP with path relinking implementation is called `grasp+pr`. To simplify the path relinking step, we use $\beta = 1$, when testing if a solution can be placed in the elite set. This way only improving solutions are put in the elite set. We were careful to implement independent random number sequences for the pure GRASP and the path relinking portions of the code. This way, if the same random number generator seeds are used for the GRASP portion of the code, the GRASP solutions produced in each iteration are identical for the GRASP and GRASP with path relinking implementations. Consequently, GRASP with path relinking will never take more iterations to find a target value solution than the pure GRASP. Since the time for one GRASP with path relinking iteration is greater than for one pure GRASP iteration, we seek to determine if the potential reduction in number of iterations of GRASP with path relinking will suffice to make the total running time of GRASP with path relinking smaller than that of pure GRASP.

The Fortran programs were compiled with the `g77` compiler, version 3.2.3 with optimization flag `-O3` and run on a SGI Altix 3700 Supercluster running RedHat Advanced Server with SGI ProPack. The cluster is configured with 32 1.5-GHz Itanium-2 processors (Rev. 5) and 245 GB of main memory. Each run was limited to a single processor. User running times were measured with the `etime` system call. Running times exclude problem input.

[2]The Fortran subroutines for the GRASP for MAX-SAT described in Resende et al. [2000] can be downloaded from `http://www.research.att.com/~mgcr/src/maxsat.tar.gz`.

Table I. Test Problems Used in Experiment[a]

Problem	Variables	Clauses	Target	Rel. Error(%)
jnh1	100	800	420739	0.044
jnh10	100	800	420357	0.115
jnh11	100	800	420516	0.056
jnh12	100	800	420871	0.013
jnh201	100	850	394222	0.047
jnh202	100	850	393870	0.076
jnh212	100	850	394006	0.059
jnh304	100	900	444125	0.092
jnh305	100	900	443815	0.067
jnh306	100	900	444692	0.032

[a]For each problem, the table lists its name, number of variables, number of clauses, the target weight used as a stopping criterion, and the percentage deviation of the target from the optimal solution.

We compared both variants on ten test problems previously studied in Resende et al. [2000].[3] Optimal weight values are known for all problems. The target weight values used in the experiments correspond to solutions found in Resende et al. [2000] after 100,000 GRASP iterations and are all near-optimal. Table I shows test problem dimensions, target values, and how close to optimal the targets are.

Since grasp and grasp+pr are both stochastic local search algorithms, we compare their performance by examining the distributions of their running times. For each instance, we make 200 independent runs of each heuristic (using different random number generator seeds) and record the time taken for the run to find a solution with weight at least as large as the given target value. For each instance/heuristic pair, the running times of each heuristic are sorted in increasing order. We associate with the ith sorted running time (t_i) a probability $p_i = (i - \frac{1}{2})/200$, and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 200$. These plots are called the time to target plots and were first introduced in Feo et al. [1994]. These plots display the empirical probability distributions of the random variable *time to target solution*. Figures 6 through 9 are the time to target plots of the test instances.[4]

We make the following observations about the experiments.

- Each heuristic was run a total of 2000 times in the experiments.
- Though the maximum number of GRASP iterations was set to 200,000, both algorithms took much less than that to find truth assignments with total weight at least as large as the target weight on all 200 runs on each instance.
- On all but one instance, the time to target curves for grasp+pr were to the left of the curves for grasp.
- The relative position of the curves implies that, given a fixed amount of computing time, grasp+pr has a higher probability than grasp of finding a target

[3]The test problems can be downloaded from http://www.research.att.com/~mgcr/data/maxsat.tar.gz.

[4]The raw data as well as the plots of the distributions for all of the test problems are available at http://www.research.att.com/~mgcr/exp/gmaxsatpr.

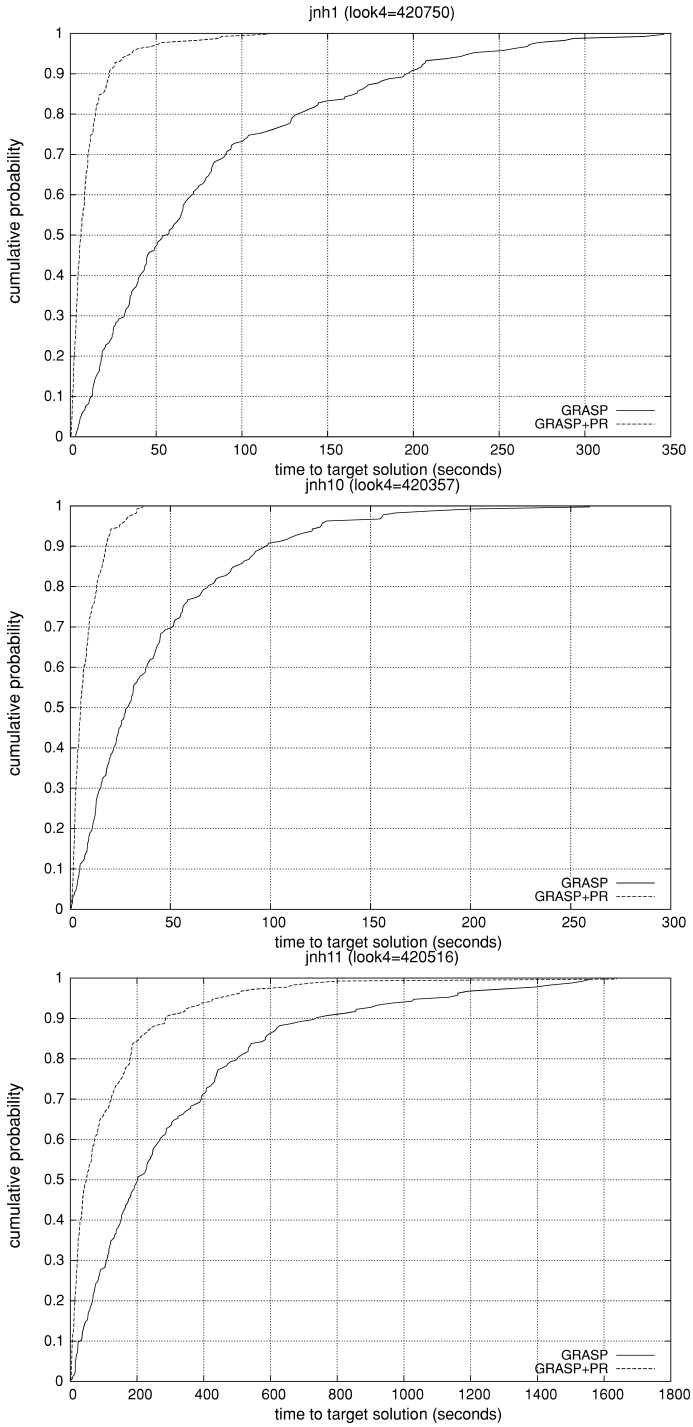

Fig. 6. Time to target distributions comparing `grasp` and `grasp+pr` on instances `jnh1`, `jnh10`, and `jnh11`.

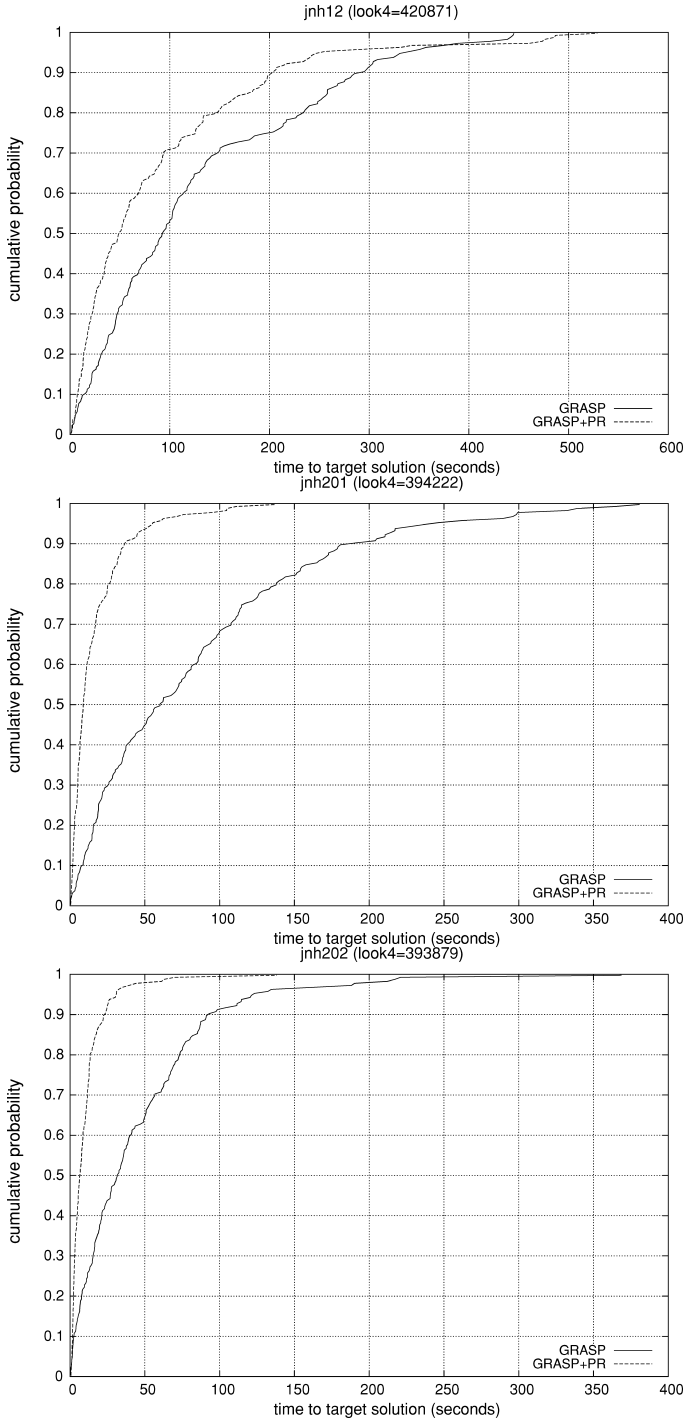

Fig. 7. Time to target distributions comparing `grasp` and `grasp+pr` on instances `jnh12`, `jnh201`, and `jnh202`.

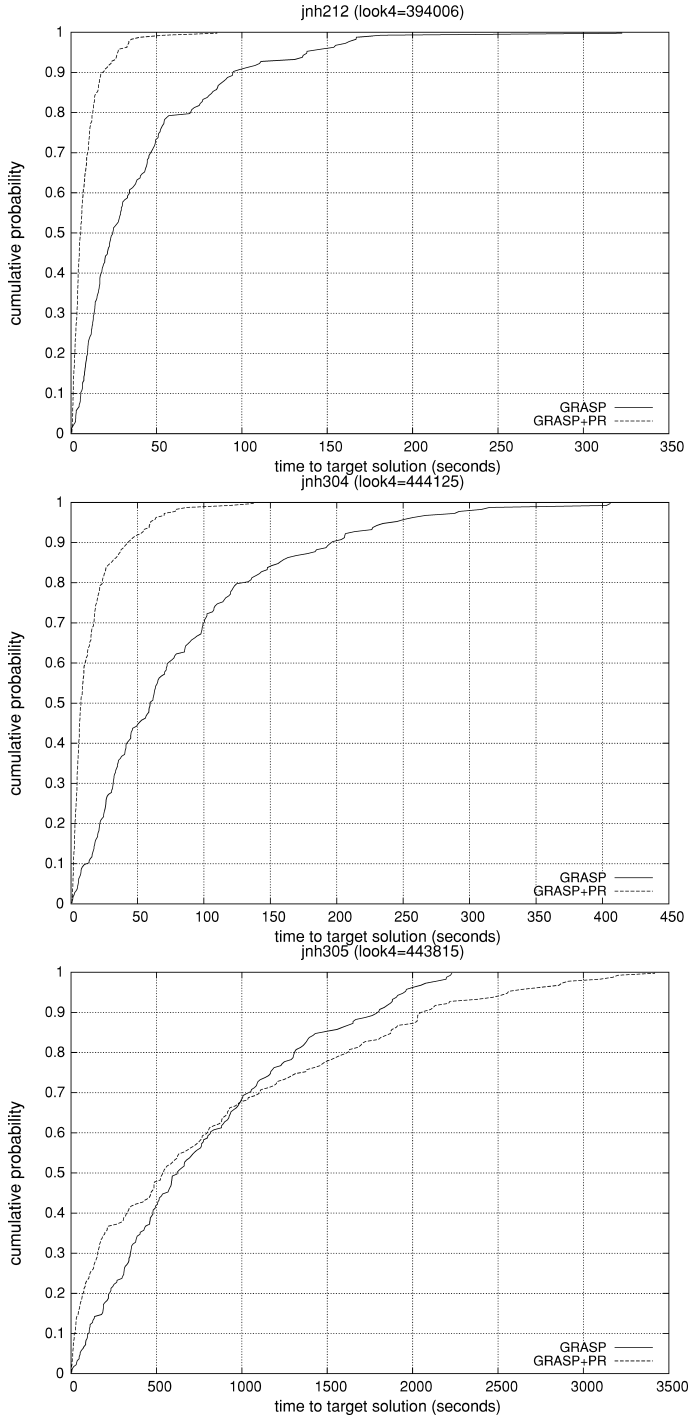

Fig. 8. Time to target distributions comparing `grasp` and `grasp+pr` on instances `jnh212`, `jnh304`, and `jnh305`.

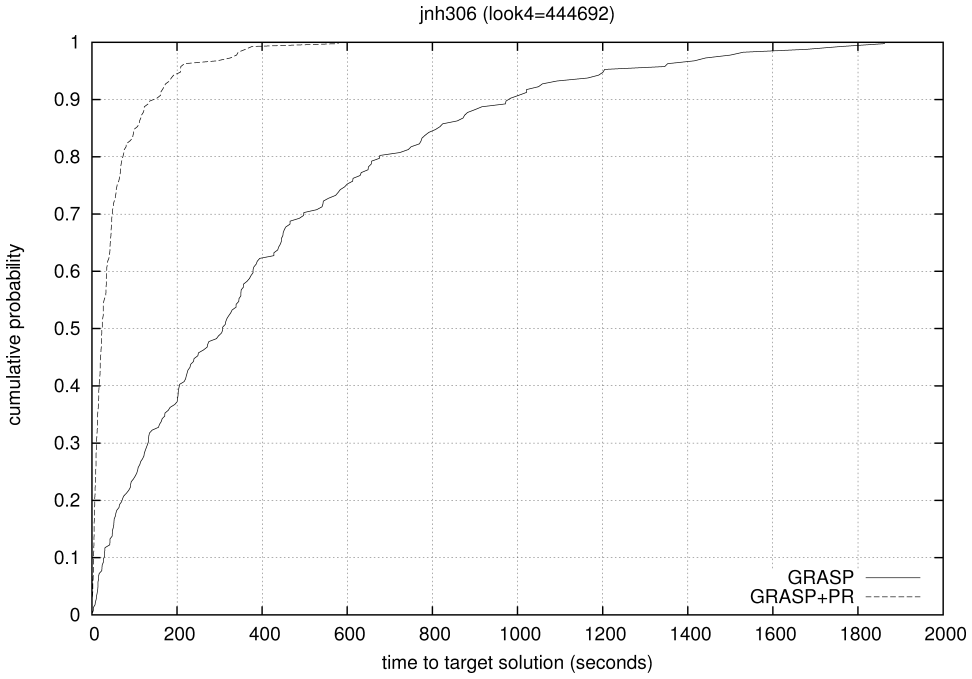

Fig. 9. Time to target distributions comparing `grasp` and `grasp+pr` on instances `jnh306`.

solution. For example, consider instance `jnh1` in Figure 6. The probabilities of finding a target at least as good as 420750 in, at most, 50 s are 48% and 97%, respectively, for `grasp` and `grasp+pr`. In, at most, 100 s, these probabilities increase to 73 and 99%, respectively.

- The relative position of the curves also implies that, given a fixed probability of finding a target solution, the expected time taken by `grasp` to find a solution with that probability is greater than the time taken by `grasp+pr`. For example, consider instance `jnh306` in Figure 9. For `grasp` to find a target solution with 50% probability, we expect it to run for 329 s, while for `grasp+pr` we expect a run of only 25 s. For 90% probability, `grasp` is expected to run for 984 s, while `grasp+pr` only takes 153 s.
- The only instance on which the time to target plots intersect was `jnh305`, where `grasp+pr` took longer to converge than the longest `grasp` run on 21 of the 200 runs, Still, two thirds of the `grasp+pr` were faster than `grasp`.

5. CONCLUSIONS

In this paper we propose a GRASP with path relinking for the weighted maximum satisfiability problem. Although GRASP has been previously implemented for the weighted MAX-SAT problem with favorable computational results in Resende et al. [1997], it did not use past information throughout the search process to improve solution quality or speed up convergence. Path relinking enhances the GRASP heuristic by providing a mechanism to search between

elite solutions, thereby incorporating memory to the procedure. Extensive computational experiments on a standard set of benchmark instances, indicate that for a given amount of computational time, path relinking will significantly increase the probability that a target solution will be found. Equivalently we can state that given that the GRASP heuristic has a certain probability of finding a target solution, path relinking will decrease the expected computational time.

REFERENCES

AIEX, R. M., BINATO, S., AND RESENDE, M. 2003. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing 29*, 393–430.

AIEX, R. M., RESENDE, M. G. C., PARDALOS, P. M., AND TORALDO, G. 2005. GRASP with path relinking for three-index assignment. *INFORMS J. on Computing 17*, 2, 224–247.

ASANO, T. 1997. Approximation algorithms for MAX-SAT: Yannakakis vs. Goemans-Williamson. In *5th IEEE Israel Symposium on the Theory of Computing and Systems*. 24–37.

BATTITI, R. AND PROTASI, M. 1997. Reactive search, a history-sensitive heuristic for MAX-SAT. *ACM Journal of Experimental Algorithms 2*, 2.

BATTITI, R. AND PROTASI, M. 1998. Approximate algorithms and heuristics for the MAX-SAT. In *Handbook of Combinatorial Optimization*, D. Z. Du and P. M. Pardalos, Eds. vol. 1. Kluwer Academic Publ., Boston, MA. 77–148.

CANUTO, S. A., RESENDE, M. G. C., AND RIBEIRO, C. C. 2001. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks 38*, 50–58.

CHEN, J., FRIESEN, D., AND ZHENG, H. 1997. Tight bound on johnson's algorithm for MAX-SAT. In *Proceedings of the 12th Annual IEEE Conference on Computational Complexity*. 274–281.

COOK S. A. 1971. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. 151–158.

FEIGE, U. AND GOEMANS, M. X. 1995. Approximating the value of two proper proof systems, with applications to MAX-2SAT and MAX-DICUT. In *Proceedings of the Third Israel Symposium on Theory of Computing and Systems*. 182–189.

FEO, T. A. RESENDE, M. G. C., AND SMITH, S. H. 1994. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research 42*, 860–878.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco, CA.

GENT, I. P. AND WALSH, T. 1993. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of the 11th National Conference on Artificial Intelligence*. 28–33.

GLOVER, F. 1996. Tabu search and adaptive memory programming: Advances, applications and challenges. In *Interfaces in Computer Science and Operations Research*, R. S. Barr, R. V. Helgason, and J. L. Kennington, Eds. Kluwer Academic Publ. Boston, MA. 1–75.

GOEMANS, M. X. AND WILLIAMSON, D. P. 1994. A new $\frac{3}{4}$ approximation algorithm for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics 7*, 656–666.

GOEMANS, M. X. AND WILLIAMSON, D. P. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of Association for Computing Machinery 42*, 6, 1115–1145.

Hansen, P. and Jaumard, B. 1990. Algorithms for the maximum satisfiability problem. *Computing 44*, 279–303.

HART, J. P. AND SHOGAN, A. W. 1987. Semi greedy heuristics: An empirical study. *Operations Research Letters 6*, 107–114.

HASTAD, J. 2001. Some optimal inapproximability results. *Journal of the ACM 48*, 798–859.

JOHNSON, D. S. 1974. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences 9*, 256–278.

JOHNSON, D. S., PAPADIMITRIOU, C. H., AND YANNAKAKIS, M. 1988. How easy is local search? *Journal of Computer and System Sciences 37*, 79–100.

JOHNSON, D. S. AND TRICK, M. A., EDS. 1996. *Cliques, coloring, and Satisfiability: Second DIMACS Implementation Challenge*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society.

KARLOFF, H. AND ZWICK, U. 1997. A $\frac{7}{8}$-approximation algorithm for MAX-3SAT. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*. 406–415.

KRENTEL, M. W. 1988. The complexity of optimization problems. *Journal of Computer and System Sciences 36*.

LAGUNA, M. AND MARTÍ, R. 1999. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing 11*, 44–52.

RESENDE, M. G. C. AND FEO, T. A. 1996. A GRASP for Satisfiability. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, D. S. Johnson and M. A. Trick, Eds. Number 26 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society. New Providence, Rhode Island. 499–520.

RESENDE, M. G. C. AND PITSOULIS, L. S. 2002. Greedy randomized adaptive search procedures. In *Handbook of Applied Optimization*, P. M. Pardalos and M. Resende, Eds. Oxford University Press, Oxford. 168–183.

RESENDE, M. G. C., PITSOULIS, L. S., AND PARDALOS, P. M. 1997. Approximate solutions of weighted MAX-SAT problems using GRASP. In *Satisfiability Problem: Theory and Applications*, D.-Z. Du, J. Gu, and P. M. Pardalos, Eds. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society. 393–405.

RESENDE, M. G. C., PITSOULIS, L. S., AND PARDALOS, P. M. 2000. Fortran subroutines for computing approximate solutions of weighted MAX-SAT problems using GRASP. *Discrete Applied Mathematics 100*, 95–113.

RESENDE, M. G. C. AND RIBEIRO, C. C. 2003. A GRASP with path-relinking for private virtual circuit routing. *Networks 41*, 104–114.

RESENDE, M. G. C. AND RIBEIRO, C. C. 2005. GRASP and path-relinking: Recent advances and applications. In *Metaheuristics: Progress as Real Problem Solvers*, T. Ibaraki, K. Nonobe, and M. Yagiura, Eds. Springer, New York. 29–63.

RIBEIRO, C. C., UCHOA, E., AND WERNECK, R. F. 2002. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing 14*, 228–246.

SELMAN, B., LEVESQUE, H., AND MITCHELL, D. 1992. A new method for solving hard satisfiability instances. In *Proceedings of the 10th National Conference on Artificial Intelligence*. 440–446.

SPEARS, W. M. 1996. Simulated annealing for hard satisfiability problems. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, D. S. Johnson and M. A. Trick, Eds. Number 26 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society. New Providence, Rhode Island. 533–555.

TREVISAN, L. 2000. Approximating satisfiable satisfiability problems. *Algorithmica 28*, 1, 145–172.

YANNAKAKIS, M. 1992. On the approximation of maximum Satisfiability. In *Proceedings of the Third ACM-SIAM Symposium on Discrete Algorithms*. 1–9.

Received September 2005; revised January 2006; accepted January 2006