GRASP with path-relinking for the weighted maximum satisfiability problem*

Paola Festa[1], Panos M. Pardalos[2], Leonidas S. Pitsoulis[3], and Mauricio G. C. Resende[4]

[1] Department of Mathematics and Applications,
University of Napoli Federico II, Compl. MSA,
Via Cintia, 80126, Napoli, Italy. paola.festa@unina.it
[2] Department of Industrial and Systems Engineering,
University of Florida, 303 Weil Hall,
Gainesville, FL 32611, USA. pardalos@ufl.edu
[3] Department of Mathematical and Physical Sciences,
School of Engineering, Aristotle University of Thessaloniki,
Thessaloniki, GR54124, Greece. pitsouli@gen.auth.gr
[4] Internet and Network Systems Research Center,
AT&T Labs Research, 180 Park Avenue, Room C241,
Florham Park, NJ 07932, USA. mgcr@research.att.com

Abstract. 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 multi-start 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.

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

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

* AT&T Labs Research Technical Report TD-68QNGR. January 17, 2005. The research of Panos M. Pardalos is partially funded by NSF and CRDF grants.

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 [8]. 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 of 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 [5]. 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 [11].

Due to 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 ($0 < \varepsilon < 1$). We will refer to ε of an approximation algorithm as its *performance ratio*. The first approximation algorithms for the MAX-SAT were introduced in [18], 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 [7] 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 in [32] and also in [14] by Goemans and Williamson. Currently the best deterministic polynomial time approximation algorithm for MAX-SAT achieves a performance ratio of 0.758 and is based on semidefinite programming [15], while there is also a randomized algorithm with performance ratio 0.77 [3]. 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 [9, 20, 31]. With respect to inapproximability results, it is known [17] 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 [16], HSAT [12] and Reactive Search [4], and those that are not history sensitive such as Simulated Annealing [30], GSAT [29] and GRASP [22, 24]. Surveys of

approximation and heuristic algorithms for solving the MAX-SAT can be found in [5, 16].

GRASP is a constructive multi-start metaheuristic which has been applied to a wide range of well known combinatorial optimization problems with favorable experimental results [23]. In [24, 25], Resende, Pitsoulis, and Pardalos describe a GRASP implementation for solving the weighted MAX-SAT, and report extensive computational results on a set of weighted SAT benchmark instances [19] that indicate that the heuristic produces good quality solutions. Each iteration consists of two phases: a construction phase where a solution is constructed in a greedy randomized fashion; 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. In this paper, we show how memory can be incorporated in the GRASP for weighted MAX-SAT proposed in [24]. 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 [27] resulting in faster convergence of the algorithm.

The remainder of the paper is organized as follows. In Section 2, we briefly state the implementation of GRASP for the MAX-SAT from [24], 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.

2 GRASP for the weighted MAX-SAT

The construction and local search phase of GRASP are described in detail in [24], while in [25] a complete Fortran implementation is given along with extensive computational runs. In this section, we provide a brief 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* which 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 construction phase of the algorithm (line 3), 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, respectively, and by $X \subseteq V$ the set of already assigned variables. We compute the best gain

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

```
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. Pseudo-code of GRASP for maximization problem.

and keep only those 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 $\gamma_k^+(\gamma_k^-)$ among those best gains corresponds to a new assignment $x_k = 1$ ($x_k = 0$), which is added to our partial solution $X = X \cup \{x_k\}$. After each such addition to the partial solution, the gains γ_j^+ and γ_j^- are updated, and 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. Having completed a truth assignment $\mathbf{x}$, we apply local search (line 4) in order to guarantee local optimality. The 1-flip neighborhood is used in the local search, which is defined as

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

where a depth-first search approach is employed in the sense that the current solution is replaced with a solution in the neighborhood which has greater cost. The search is terminated when the current solution is the local optimum.

3 Path-Relinking

Path-relinking was originally proposed by Glover [13] 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 the distance between the *initial* and *guiding* 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 [1, 2, 6, 21, 26, 28], 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 [27].

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 $|S(\mathbf{x},\mathbf{y})| = 2^{n-d(\mathbf{x},\mathbf{y})} - 2$. The main thesis 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 employ a greedy search where a path of solutions

$$\mathbf{x} = \mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{d(\mathbf{x},\mathbf{y})}, \mathbf{w}_{d(\mathbf{x},\mathbf{y})+1} = \mathbf{y},$$

is build, such that $d(\mathbf{w}_i, \mathbf{w}_{i+1}) = 1, \;\; i = 0, \ldots, d(\mathbf{x},\mathbf{y})$, and the best solution from this path is chosen. Note that since both $\mathbf{x},\mathbf{y}$ are local optima in some neighborhood N_1 by construction[5], in order for $S(\mathbf{x},\mathbf{y})$ to contain solutions which 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 pseudo-code which illustrates the exact implementation for the path-relinking procedure is shown in Figure 2. We employ the strategy 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 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 operation `flip`$(\mathbf{w}_k, i)$ has the effect of negating the variable w_i in solution $\mathbf{w}_k$. It is noted that the path of solutions never enters the neighborhood of $\mathbf{x}$.

The integration of the path-relinking procedure with the pure GRASP is shown in Figure 3, and 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 `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 $d(\mathbf{y},\mathbf{w}) > \beta n, \;\; \forall \mathbf{w} \in \mathcal{E}$, where β is a parameter between 0 and 1 and n is the number of variables.

[5] where the same metric $d(\mathbf{x},\mathbf{y})$ is used

```
procedure PathRelinking(x, 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. 2. Pseudo-code of path-relinking for maximization problem.

```
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. 3. Pseudo-code of GRASP with path-relinking for maximization problem

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. $\mathbf{z} = \text{argmin}\{d(\mathbf{y},\mathbf{w}) : \mathbf{w} \in \mathcal{E} \text{ such that } c(\mathbf{w}) \leq c(\mathbf{y})\}$.

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 [25][6]. 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 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 its iterations take longer, we seek to determine if the longer iterations pay off in terms of a reduction in number of iterations that is enough to result in a reduction in total time.

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.

We compared both variants on ten test problems previously studied in [25] [7]. Optimal weight values are known for all problems. The target weight values used in the experiments correspond to solutions found in [25] after 100,000 GRASP iterations and are all near-optimal. Table 4 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 i-th 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

[6] The Fortran subroutines for the GRASP for MAX-SAT described in [25] can be downloaded from `http://www.research.att.com/~mgcr/src/maxsat.tar.gz`.

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

Table 1. Test problems used in experiment. 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.

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%

[10]. These plots display the empirical probability distributions of the random variable *time to target solution*. Figures 4 and 5 are time to target plots for a subset of the test instances. [8]

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 solution. For example, consider instance `jnh1` in Figure 4. The probabilities of finding a target at least as good as 420750 in at most 50 seconds are 48% and 97%, respectively, for `grasp` and `grasp+pr`. In at most 100 seconds, 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 5. For `grasp` to find a target solution with 50% probability we expect it to run for 329 seconds, while `grasp+pr` we expect a run of only 25 seconds. For 90% probability, `grasp` is expected to run for 984 seconds while `grasp+pr` only takes 153 seconds.
- 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`.

[8] 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`.

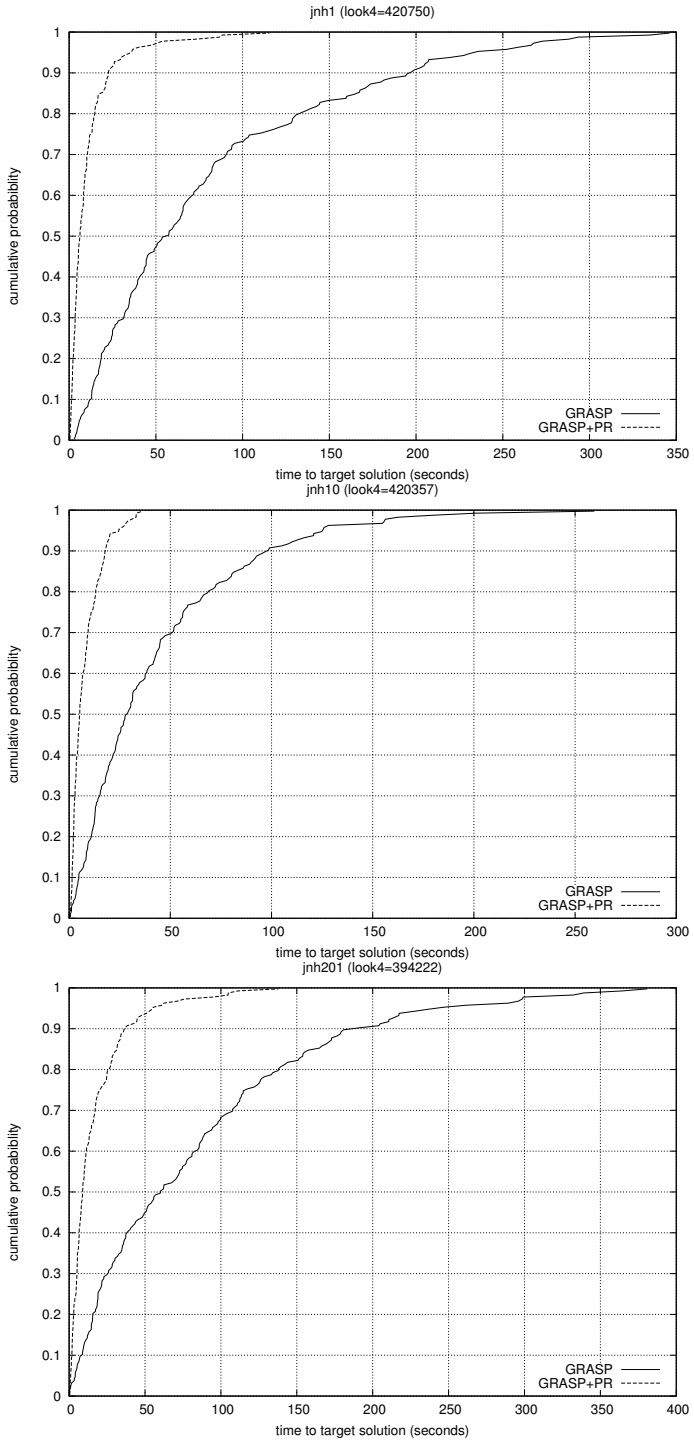

Fig. 4. Time to target distributions comparing `grasp` and `grasp+pr` on instances `jnh1`, `jnh10`, and `jnh201`.

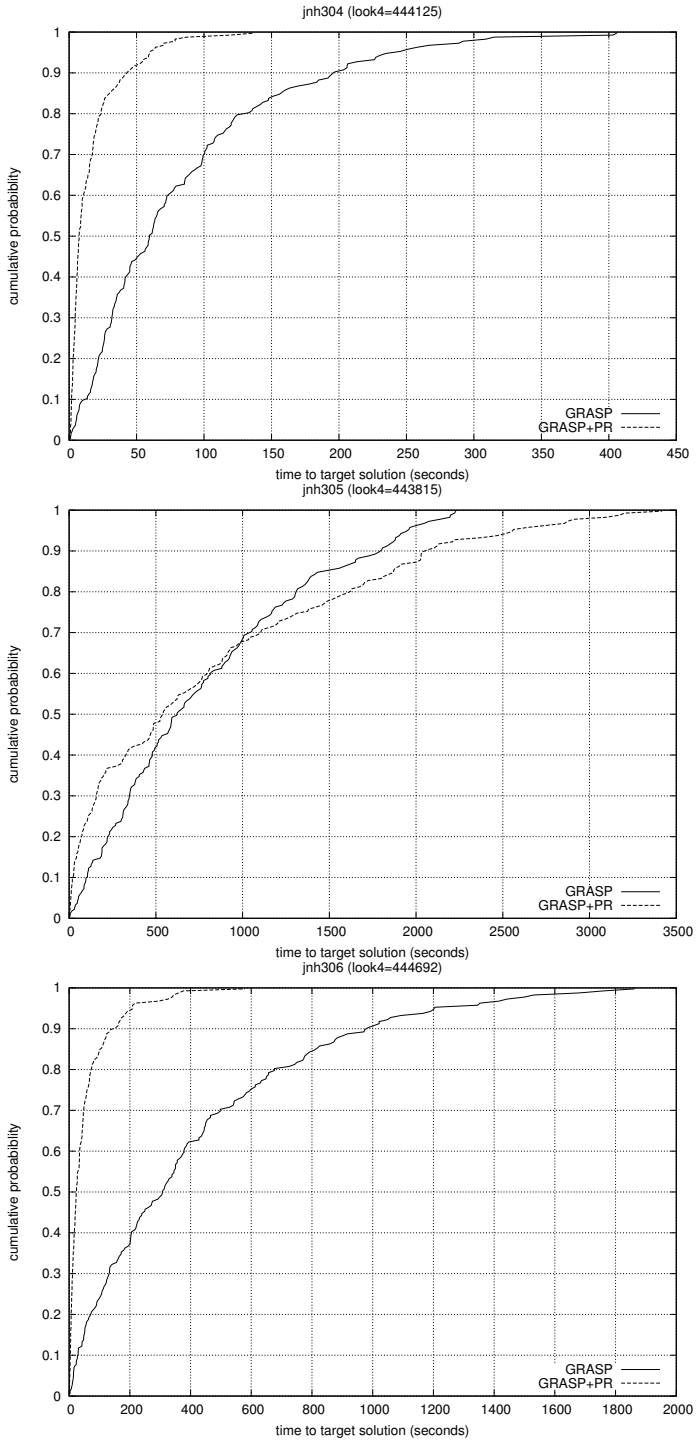

Fig. 5. Time to target distributions comparing `grasp` and `grasp+pr` on instances `jnh304`, `jnh305`, and `jnh306`.

References

1. R.M. Aiex, S. Binato, and M.G.C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29:393–430, 2003.
2. R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with path relinking for three-index assignment. *INFORMS J. on Computing*, 2005. In press.
3. T. Asano. Approximation algorithms for MAX-SAT: Yannakakis vs. Goemans-Williamson. In *5th IEEE Israel Symposium on the Theory of Computing and Systems*, pages 24–37, 1997.
4. R. Battiti and M. Protasi. Reactive search, a history-sensitive heuristic for MAX-SAT. *ACM Journal of Experimental Algorithms*, 2(2), 1997.
5. R. Battiti and M. Protasi. Approximate algorithms and heuristics for the MAX-SAT. In D.Z. Du and P.M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 1, pages 77–148. Kluwer Academic Publishers, 1998.
6. S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38:50–58, 2001.
7. J. Chen, D. Friesen, and H. Zheng. Tight bound on johnson's algorithm for MAX-SAT. In *Proceedings of the 12th Annual IEEE Conference on Computational Complexity*, pages 274–281, 1997.
8. S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.
9. U. Feige and M.X. Goemans. Approximating the value of two proper proof systems, with applications to MAX-2SAT and MAX-DICUT. In *Proceeding of the Third Israel Symposium on Theory of Computing and Systems*, pages 182–189, 1995.
10. T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.
11. M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Company, New York, 1979.
12. I.P. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 28–33, 1993.
13. F. Glover. Tabu search and adaptive memory programming: Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer Academic Publishers, 1996.
14. M.X. Goemans and D.P. Williamson. A new $\frac{3}{4}$ approximation algorithm for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7:656–666, 1994.
15. M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of Association for Computing Machinery*, 42(6):1115–1145, 1995.
16. P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.
17. J. Hastad. Some optimal inapproximability results. *Journal of the ACM*, 48:798–859, 2001.
18. D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.
19. D.S. Johnson and M.A. Trick, editors. *Cliques, coloring, and Satisfiability: Second DIMACS Implementation Challenge*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1996.
20. H. Karloff and U. Zwick. A $\frac{7}{8}$-approximation algorithm for MAX-3SAT. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 406–415, 1997.
21. M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

22. M.G.C. Resende and T.A. Feo. A GRASP for Satisfiability. In D.S. Johnson and M.A. Trick, editors, *Cliques, coloring, and Satisfiability: Second DIMACS Implementation Challenge*, number 26 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 499–520. American Mathematical Society, 1996.
23. M.G.C. Resende and L.S. Pitsoulis. Greedy randomized adaptive search procedures. In P.M. Pardalos and M.G.C. Resende, editors, *Handbook of Applied Optimization*, pages 168–183. Oxford University Press, 2002.
24. M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Approximate solutions of weighted MAX-SAT problems using GRASP. In D.-Z. Du, J. Gu, and P.M. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 393–405. American Mathematical Society, 1997.
25. M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Fortran subroutines for computing approximate solutions of weighted MAX-SAT problems using GRASP. *Discrete Applied Mathematics*, 100:95–113, 2000.
26. M.G.C. Resende and C.C. Ribeiro. A GRASP with path-relinking for private virtual circuit routing. *Networks*, 41:104–114, 2003.
27. M.G.C. Resende and C.C. Ribeiro. GRASP and path-relinking: Recent advances and applications. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*, pages 29–63. Springer, 2005.
28. C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14:228–246, 2002.
29. B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability instances. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 440–446, 1992.
30. W.M. Spears. Simulated annealing for hard satisfiability problems. In D.S. Johnson and M.A. Trick, editors, *Cliques, coloring, and Satisfiability: Second DIMACS Implementation Challenge*, number 26 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 533–555. American Mathematical Society, 1996.
31. L. Trevisan. Approximating satisfiable satisfiability problems. *Algorithmica*, 28(1):145–172, 2000.
32. M. Yannakakis. On the approximation of maximum Satisfiability. In *Proceedings of the Third ACM-SIAM Symposium on Discrete Algorithms*, pages 1–9, 1992.