M. De Santis, P. Festa, G. Liuzzi, S. Lucidi,
F. Rinaldi

A NONMONOTONE GRASP

R. 2, 2014

Marianna De Santis – Fakultät für Mathematik, TU Dortmund, Germany. `msantis@math.tu-dortmund.de`.

Paola Festa – Department of Mathematics and Applications, University of Napoli FEDERICO II, Napoli, Italy. `paola.festa@unina.it`.

Giampaolo Liuzzi – Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti" - CNR, Rome, Italy. `giampaolo.liuzzi@iasi.cnr.it`.

Stefano Lucidi – "Sapienza" Università di Roma, Dipartimento di Ingegneria Informatica Automatica e Gestionale "A. Ruberti", Rome, Italy. `lucidi@dis.uniroma1.it`.

Francesco Rinaldi – Dipartimento di Matematica, University of Padova, Italy. `rinaldi@math.unipd.it`.

ISSN: 1128–3378

Collana dei Rapporti dell'Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", CNR

viale Manzoni 30, 00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: iasi@iasi.cnr.it
URL: http://www.iasi.cnr.it

Abstract

A Greedy Randomized Adaptive Search Procedure (GRASP) is an iterative multistart metaheuristic for difficult combinatorial optimization problems. Each GRASP iteration consists of two phases: a construction phase, in which a feasible solution is produced, and a local search phase, in which a local optimum in the neighborhood of the constructed solution is sought. Repeated applications of the construction procedure yields different starting solutions for the local search and the best overall solution is kept as the result.

The GRASP local search applies iterative improvement until a locally optimal solution is found. During this phase, starting from the current solution an improving neighbor solution is accepted and considered as new current solution.

In this paper, we propose a variant of the GRASP framework that uses a new "nonmonotone" strategy to explore the neighborhood of the current solution. We formally state the convergence of the nonmonotone local search to a locally optimal solution and illustrate the effectiveness of the resulting Nonmonotone GRASP on the Maximum Cut Problem, a classical hard combinatorial optimization problem.

Key words: Combinatorial Optimization, GRASP, Metaheuristics, Local Search, Nonmonotone Line Search, MAX-CUT

1. Introduction

Any combinatorial optimization problem involves a finite number of feasible solutions and is completely defined by a ground set $\mathcal{E} = 1, \ldots, n$, an objective function $f : 2^{\mathcal{E}} \mapsto \mathcal{R}$, and the set of feasible solution $\mathcal{X} \subseteq 2^{\mathcal{E}}$. In case of minimization (resp. maximization), one searches for an optimal solution $x^* \in \mathcal{X}$ such that $f(x^*) \leq f(x)$ (resp. $f(x^*) \geq f(x)$), $\forall\, x \in \mathcal{X}$. To illustrate an example, let us consider the Traveling Salesman Problem, a classical combinatorial optimization problem defined on an undirected edge-weighted graph $G = (V, E)$. In this case, the ground set $\mathcal{E}$ is the set of edges connecting nodes in V to be visited, $\mathcal{X}$ is formed by all edge subsets that determine a Hamiltonian cycle, and the objective function value $f(x)$ is the sum of the costs of all edges in solution x. As a further example of how to define a combinatorial optimization problem through the ground set $\mathcal{E}$, the objective function f, and the set of feasible solution $\mathcal{X}$, let us consider the Maximum Cut Problem, defined on an undirected edge-weighted graph $G = (V, E)$. Here, the ground set $\mathcal{E}$ is the set of all edges in E, $\mathcal{X}$ is the set of all subsets of $\mathcal{E}$ made of edges with endpoints in two different node subsets defining a partition of V, and $f(x)$ is the sum of the weights of edges in solution x.

Combinatorial optimization problems arise in several and heterogenous domains [59], among many others we recall routing, scheduling, production planning, decision making process, and location problems. All these problems have both a theoretical relevance and a practical impact, given their applicability to real-world scenarios [60].

Many combinatorial optimization problems are computationally intractable, in the sense that until now, no polynomial-time algorithm is known to exactly solve them [30]. In the last decades, several optimal seeking methods that do not explicitly examine all feasible solutions have been developed, such as Branch & Bound, Cutting Planes, and Dynamic Programming. Nevertheless, most real-world problems are either computationally intractable by their nature, or sufficiently large so as to preclude the use of exact algorithms. In such cases, heuristic methods are usually employed to find good, but not necessarily guaranteed optimal solutions.

Starting from one of the pioneering paper of Kirkpatrick on Simulated Annealing [47] appeared in 1984, the most promising heuristic methods concentrate their effort in the attempt of avoiding entrapments in local optima and in exploiting the basic structure and properties of the problem they solve. Such techniques include Tabu Search [31, 32, 33, 34], Genetic Algorithms [37], Variable Neighborhood Search [56, 42], and GRASP (Greedy Randomized Adaptive Search Procedures) [15, 16, 26, 24, 25, 27].

A GRASP is a multi-start or iterative process, following in the spirit the pioneering idea proposed in 1973 by Lin and Kernighan for the Traveling Salesman Problem [52]. Each GRASP iteration is usually made up of a construction phase, where a feasible solution is constructed, and a local search phase which starts at the constructed solution and applies iterative improvement until a locally optimal solution is found. Repeated applications of the construction procedure yields diverse starting solutions for the local search and the best overall locally optimal solution is kept as the result. Since its proposal, GRASP has been applied to solve decision and optimization problems arising in several and heterogenous contexts. Recent areas of application include routing [6, 8, 11], logic [21, 63, 64], covering and partition [4, 15, 62], location [12, 13], network optimization [5, 57, 65], assignment [55, 67], timetabling and scheduling [1, 3, 2, 9, 66, 50, 51], graph and map drawing [22, 23, 49, 54, 65], and very recently computational biology [18, 19, 20, 14, 17, 28, 35, 44].

Aim of this paper is to propose a new variant of the classical GRASP framework that uses a nonmonotone strategy to explore the neighborhood of the current solution. Inspired by an idea proposed in 1986 for Newton's method [48], this strategy controls uphill solutions without using a "tabu list" but simply maintaining a set W of a given number of previously computed objective function values. A new solution is accepted if its function value improves the worst value in W.

The remainder of this paper is organized as follows. In Section 2, the main ingredients of a classical GRASP framework are described. Section 3 is devoted to a variant of GRASP, we called Nonmonotone GRASP (NM-GRASP) and whose new local search strategy is described and its properties analyzed. In Section 4, we propose a NM-GRASP for the Maximum Cut Problem (MAX-CUT), and in Section 5, we illustrate its effectiveness comparing it with a classical GRASP on standard test problems from the literature of MAX-CUT. The experiments we have conducted demonstrate empirically that the new described algorithm results in better quality solutions. Concluding remarks are given in the last section.

2. The classical GRASP

Given a combinatorial optimization problem specified by the ground set $\mathcal{E}$, the real-valued objective function f, and the finite set of feasible solution $\mathcal{X}$, a classical GRASP metaheuristic is a multi-start or iterative method, in which each iteration consists of two phases: construction of a solution and local search. The pseudo-code in Figure 1 illustrates the main blocks of a GRASP procedure for minimization, in which `MaxIterations` iterations are performed and `Seed` is used as the initial seed for the pseudorandom number generator.

```
algorithm GRASP($f(\cdot)$, $g(\cdot)$, MaxIterations, Seed)
1   $x_{best}$:=$\emptyset$; $f(x_{best})$:=$+\infty$;
2   for $k = 1, 2, \ldots$,MaxIterations$\rightarrow$
3       $x$:=ConstructGreedyRandomizedSolution(Seed, $g(\cdot)$);
4       if ($x$ not feasible) then
5           $x$:=repair($x$);
6       endif
7       $x$:=LocalSearch($x$, $f(\cdot)$);
8       if ($f(x) < f(x_{best})$) then
9           $x_{best}$:=$x$;
10      endif
11  endfor;
12  return($x_{best}$);
end GRASP
```

Figure 1: Pseudo-code of a classical GRASP for a minimization problem.

The construction phase builds a solution x that can be eventually not feasible (line 3). In this case, the feasibility of the built solution is obtained by invoking a repair procedure in line 5. Once a feasible solution x is obtained, its neighborhood is investigated by the local search until a local minimum is found (line 7). The best overall local optimal solution is kept as the result (line 12).

The pseudo-code in Figure 2 illustrates the main ingredients of a typical GRASP construction phase, which proceeds applying a *greedy*, *randomized*, and *adaptive* criterion. In the spirit the pioneering semi-greedy idea proposed by Hart and Shogan in 1987, the construction procedure starts from an empty solution (line 1) and iteratively, one element at a time, builds a complete

```
procedure ConstructGreedyRandomizedSolution(Seed, $g(\cdot)$)
1   $x$:=$\emptyset$;
2   Sort the candidate elements $i$ in a list $C$ according to their incre-
mental
    costs $g(i)$;
3   while ($x$ is not a complete solution)$\rightarrow$
4       RCL:=MakeRCL($C$);
5       $v$:=SelectIndex(RCL, Seed);
6       $x := x \cup \{v\}$;
7       $C := C \setminus \{v\}$;
8       Resort remaining candidate elements $j \in C$ according to their
        incremental costs $g(j)$;
9   endwhile;
10  return($x$);
end ConstructGreedyRandomizedSolution;
```

Figure 2: Basic GRASP construction phase pseudo-code.

solution (loop in lines 3–9). At each iteration, the choice of the next element to be added to the partial solution under construction is determined by ordering all candidate elements (i.e. those that can be added to the solution) in a candidate list C with respect to a *greedy function* $g : C \rightarrow \mathcal{R}$ that myopically measures the benefit in terms of objective function value of selecting each candidate element. The heuristic is *adaptive* because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element (line 8). The *probabilistic component* of a GRASP construction procedure is characterized by *randomly* choosing one of the best candidates in the list, but not necessarily the top candidate (line 5). The list of best candidates is called the *restricted candidate list* (RCL). This choice technique allows for different solutions to be obtained at each GRASP iteration, but does not necessarily compromise the power of the adaptive greedy component of the method.

```
procedure LocalSearch($x$, $f(\cdot)$)
1   Let $N(x)$ be the neighborhood of $x$;
2   $H$:=$\{y \in N(x) \mid f(y) < f(x)\}$;
3   while ($|H| > 0$)$\rightarrow$
4       $x$:=Select($H$);
5       $H$:=$\{y \in N(x) \mid f(y) < f(x)\}$;
6   endwhile
7   return($x$);
end LocalSearch
```

Figure 3: Pseudo-code of a generic local search procedure.

Once a feasible solution x is obtained, its neighborhood is investigated by the local search until a local minimum is found. Figure 3 illustrates the pseudo-code of a generic local search procedure for a minimization problem.

As any local search algorithm, a typical GRASP local search procedure requires the definition of a proper *neighborhood structure* N for the specific problem to be solved. The *neighborhood structure* N relates a solution x of the problem to a subset of solutions $N(x)$ "close" to x, which is said to be *locally optimal* with respect to $N(x)$ if in $N(x)$ there is no better solution in terms of objective function value.

Once defined and computed a suitable neighborhood $N(x)$ of the current solution x (line 1), a GRASP local search works in an iterative fashion by successively replacing the current solution x by a better solution in the neighborhood $N(x)$. It terminates when no better solution is found in the neighborhood, i.e. when a local minimum is found and returned to the main algorithm.

It can be easily seen that the key to success for a local search algorithm consists of the suitable choice of a neighborhood structure, efficient neighborhood search techniques, and the starting solution.

3. The Nonmonotone GRASP

For finding approximate solutions of hard combinatorial problems, we propose a NonMonotone GRASP (NM-GRASP). The main difference between NM-GRASP and a classical GRASP is in the use of a nonmonotone local search strategy, based on the ideas described in [48].

The pseudo-code in Figure 4 illustrates how the nonmonotone local search works for a minimization problem. As in the classical GRASP local search, the first step of the nonmonotone local search is the computation of a suitable neighborhood $N(x)$ of the current solution x (line 1). The search is then carried out by successively replacing the current solution x by a solution $y \in N(x)$ that improves a given reference objective function value $\bar{f}$ instead of the best value obtained so far. Hence, we have $f(y) < \bar{f}$ which clearly allows for uphill steps thus giving raise to a nonmonotone local search. In order to avoid cycling of the local search routine, the reference value must be updated according to a rigorous criterion. To perform such an update, the routine employs a queue W of maximum size M containing the least recently accepted function values. The queue is managed according to a first-in-first-out policy by means of the following two operations: `push`(f, W), which inserts into W a new function value f, and `pop`(W), which drops from W the least recently inserted function value.

```
procedure NonmonotoneLocalSearch($x$, $f(\cdot)$)
1    Let $M \geq 1$ and let $N(x)$ be the neighborhood of $x$;
2    $W$:=$\{f(x)\}$; $\bar{f}$:=$f(x)$; $x_{min}$:=$x$;
3    $H$:=$\{y \in N(x) \mid f(y) < \bar{f}\}$;
4    while ($|H| > 0$)$\rightarrow$
5        $x$:=Select($H$);
6        if $f(x) < f(x_{min})$ then $x_{min}$:=$x$;
7        if $|W| = M + 1$ then pop($W$);
8        push($f(x), W$); $\bar{f}$:=max$\{f \in W\}$;
9        $H$:=$\{y \in N(x) \mid f(y) < \bar{f}\}$;
10   endwhile
11   return($x_{min}$);
end NonmonotoneLocalSearch
```

Figure 4: Pseudo-code of the nonmonotone local search procedure.

Looking at the pseudo-code in Figure 4, procedure `NonmonotoneLocalSearch` successively updates the current solution x by a new one belonging to the set H of improving solutions with respect to the given reference value $\bar{f}$ and (possibly) updates the reference value $\bar{f}$ itself. The search terminates when there is no solution in the neighborhood that improves $\bar{f}$. More precisely, the nonmonotone local search terminates at x if

$$f(y) \geq \bar{f} \geq f(x), \qquad \forall\, y \in N(x). \tag{1}$$

Note that, condition (1) implies that x is locally optimal with respect to $N(x)$.

In order to show that `NonmonotoneLocalSearch` cannot indefinitely cycle between lines 4 and 10 of the while-cycle, we need to explicitly define the sequences of points and function values generated by the procedure. To this aim, let us denote by x^0 the starting solution of the local search, by x^k and $\bar{f}^k$ the solution and the reference value at the beginning of every iteration of the while-cycle, respectively. Moreover, let be

$$H^k = \{y \in N(x^k) : f(y) < \bar{f}^k\}.$$

Consequently, line 5 of the while-cycle can be written as

$$x^{k+1} := \text{Select}(H^k).$$

We remark that the updating of the reference value performed by the procedure is such that $\bar{f}^k$ can formally be written as

$$\bar{f}^k = \max_{0 \leq i \leq \min\{k,\ M\}} f(x^{k-i}), \tag{2}$$

where $M \in \mathcal{N}^+$ is fixed.

We can now formally introduce the sequences $\{x^k\}$, $\{f(x^k)\}$, and $\{\bar{f}^k\}$ generated by `Nonmonotone-LocalSearch`. Note that, even when

$$f(x^{k+1}) < \bar{f}^k, \qquad \text{for every index } k,$$

it results that the sequence $\{f(x^k)\}$ can be nonmonotone.

Proposition 3.1. *Let $\{x^k\}$ be the sequence of solutions generated by the nonmonotone local search, and $\{\bar{f}^k\}$ be the sequence of reference values defined as in (2).*
Then, `NonmonotoneLocalSearch` *cannot cycle.*

Proof. First, we observe that the sequence $\{\bar{f}^k\}$ is bounded from below, since the number of solutions in the feasible set $\mathcal{X}$ is finite.

Moreover, at each iteration k, we have that

$$f(x^{k+1}) < \bar{f}^k. \tag{3}$$

From (2) and (3), we can write:

$$\bar{f}^{k+1} \leq \bar{f}^k. \tag{4}$$

Then, remembering that $|\mathcal{X}| < \infty$, we can define

$$0 < \delta = \min_{x,y \in \mathcal{X}} \Big\{ |f(x) - f(y)| : f(x) \neq f(y) \Big\},$$

so that we have

$$\bar{f}^{k+M} < \bar{f}^k - \delta. \tag{5}$$

By contradiction, let us assume now that the procedure does not terminate and that a solution $\tilde{x}$ (which is not a local minimum) is generated an infinite number of times. By (4) and (5), there exists an iteration $\tilde{k}$ such that

$$\bar{f}^{\tilde{k}} \leq f(\tilde{x}).$$

Furthermore, as $\tilde{x}$ is generated an infinite number of times, there exists an iteration $\hat{k} \geq \tilde{k}$ such that

$$f(\tilde{x}) < \bar{f}^{\hat{k}}.$$

Hence, we have

$$f(\tilde{x}) < \bar{f}^{\hat{k}} \leq \bar{f}^{\tilde{k}} \leq f(\tilde{x}),$$

which concludes the proof. □

4. A Nonmonotone GRASP for MAX-CUT

We illustrate in this section the main features of a Nonmonotone GRASP (NM-GRASP) for the Maximum Cut Problem (MAX-CUT), a classical hard combinatorial optimization problem.

Given an undirected graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ is the set of vertices and E is the set of edges, and weights w_{ij} associated with the edges $(i, j) \in E$, the MAX-CUT consists in finding a partition $(S, \bar{S})$ of V such that the weight of the cut induced by $(S, \bar{S})$ defined as

$$w(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}} w_{ij}$$

is maximized.

It can be formulated as the following integer quadratic program:

$$\begin{aligned} \max \quad & \frac{1}{2} \sum_{1 \leq i < j \leq n} w_{ij}(1 - y_i y_j) \\ \text{s.t.} \quad & \\ & y_i \in \{-1, 1\}, \quad \forall\, i \in V. \end{aligned}$$

Each set $S = \{i \in V : y_i = 1\}$ induces a cut $(S, \bar{S})$ with weight

$$w(S, \bar{S}) = \frac{1}{2} \sum_{1 \leq i < j \leq n} w_{ij}(1 - y_i y_j).$$

In spite of the very easy statement of this well known combinatorial optimization problem, its decision version has been proved to be NP-complete by Karp already in 1972 [46]. In 1991, it has been showed that MAX-CUT is APX-complete [58], meaning that unless P=NP, it does not allow a polynomial time approximation scheme [69]. Polynomially solvable cases include planar graphs [41], weakly bipartite graphs with nonnegative weights [40], and graphs without K_5 minors [7].

Given the inner intractability of the problem, many researchers have devoted their effort in both deeper studying the inner computational nature of the problem and in designing good approximate and heuristic solution techniques. Along this research line, the idea that the MAX-CUT can be naturally relaxed to a semidefinite programming problem was first observed by Lovász [53] and Shor [68]. Goemans and Williamson [36] proposed a randomized algorithm that uses semidefinite programming to achieve a performance guarantee of 0.87856 if the weights are nonnegative. Since then, many approximation algorithms for NP-hard problems have been devised using SDP relaxations [39, 45, 61].

In the following, we describe a NM-GRASP for finding approximate solutions of the MAX-CUT, based on the instantiation of a classical GRASP proposed by Festa et al. in 2002 [23]. Figure 5 depicts the pseudo-code.

```
algorithm NM-GRASP($w(\cdot)$, $g(\cdot)$, MaxIterations, Seed)
1    $x_{best}$:=$\emptyset$; $w(x_{best})$:=$-\infty$;
2    for $k = 1, \dots,$MaxIterations$\rightarrow$
3        $x$:=ConstructGreedyRandomizedSolution(Seed, $g(\cdot)$);
4        $x$:=NonmonotoneLocalSearch($x$, $f(\cdot)$);
5        if ($w(x) > w(x_{best})$) then
6            $x_{best}$:=$x$;
7        endif
8    endfor;
9   return($x_{best}$);
end NM-GRASP
```

Figure 5: Pseudo-code of a Nonmonotone GRASP (NM-GRASP) for the Maximum Cut Problem.

As the classical GRASP proposed by Festa et al. in 2002 [23], the NM-GRASP we propose for MAX-CUT proceeds in iterations. At each iteration a greedy randomized adaptive solution is built (line 3) and used as starting point in a local search procedure (line 4). The best overall locally optimal solution is returned as an approximation of the global optimal (line 9).

The construction procedure ConstructGreedyRandomizedSolution(Seed, $g(\cdot)$) (line 3) works exactly as the construction procedure described in [23], i.e. making use of an adaptive greedy function, a construction mechanism for the restricted candidate list, and a probabilistic selection criterion. In the case of the MAX-CUT, it is intuitive to relate the greedy function to the sum of the weights of the edges in each cut. More formally, let $(S, \bar{S})$ be a cut. Then, for each vertex $v \notin S \cup \bar{S}$, the following two quantities are computed:

$$\sigma_S(v) = \sum_{u \in S} w_{vu} \quad \text{and} \quad \sigma_{\bar{S}}(v) = \sum_{u \in \bar{S}} w_{vu}.$$

The greedy function, $g(v) = \max\{\sigma_S(v), \sigma_{\bar{S}}(v)\}$, measures how much additional weight will result from the assignment of vertex v to S or $\bar{S}$. The greedy choice consists in selecting the vertex v with the highest greedy function value. If $\sigma_S(v) > \sigma_{\bar{S}}(v)$, then v is placed in $\bar{S}$; otherwise it is placed in S. To define the construction mechanism for the restricted candidate list (RCL), let

$$w_{min} = \min\left\{\min_{v \in V'} \sigma_S(v),\ \min_{v \in V'} \sigma_{\bar{S}}(v)\right\}$$

and

$$\begin{aligned} w^{max} &= \max\left\{\max_{v \in V'} \sigma_S(v),\ \max_{v \in V'} \sigma_{\bar{S}}(v)\right\} \\ &= \max_{v \in V'}\{g(v)\}, \end{aligned}$$

where $V' = V \setminus \{S \cup \bar{S}\}$ is the set of vertices which are still candidate elements, i.e. not yet assigned to either subset S or subset $\bar{S}$. Denoting by $\mu = w_{min} + \alpha \cdot (w^{max} - w_{min})$ the cut-off value, where $\alpha \in [0, 1]$, the RCL is made up by all vertices whose value of the greedy function is greater than or equal to μ. A vertex is randomly selected from the RCL.

Once obtained a greedy, randomized, and adaptive solution x is built, the nonmonotone local search procedure is invoked in line 4. It is based on the same neighborhood structure as in [23] whose elementary move, given the current solution x, consists in moving each vertex from one subset of the cut to the other. More formally, let $(S, \bar{S})$ be the current solution. To each vertex $v \in V$ we associate either the neighbor $(S \setminus \{v\}, \bar{S} \cup \{v\})$ if $v \in S$, or the neighbor $(S \cup \{v\}, \bar{S} \setminus \{v\})$ otherwise. The value

$$\delta(v) = \begin{cases} \sigma_S(v) - \sigma_{\bar{S}}(v), & \text{if } v \in S; \\ \sigma_{\bar{S}}(v) - \sigma_S(v), & \text{if } v \in \bar{S} \end{cases}$$

represents the change in the objective function associated with moving vertex v from one subset of the cut to the other.

In [23], all possible moves are investigated and the acceptance criterion follows a monotone strategy, i.e. the current solution is replaced by its best improving neighbor and the search stops after all possible moves have been evaluated and no improving neighbor was found.

In the Nonmonotone GRASP for MAX-CUT we use, instead,

$$\texttt{NonmonotoneLocalSearch}(x, f(\cdot))$$

introduced in Section 3 for minimization problems. In the case of maximization problems (and this is the case for MAX-CUT) we have the sign $>$ in line 3 and 9. Furthermore, $\bar{f}$ is updated as the minimum among the objective function values in W, and on line 8 we have $\bar{f}$:=min$\{f \in W\}$.

`NonmonotoneLocalSearch` accepts a move if it guarantees an improvement greater than $\bar{f}$. The current solution is then successively replaced and the search stops after all possible moves have been evaluated and no neighbor that improves $\bar{f}$ was found.

5. Computational results

In this section, we present results of the computational experiments with the classical GRASP and the NM-GRASP proposed in this paper.

The experiments were performed on an Intel Xeon E5-2670 processor, running at 2.60 GHz with 64 GB of RAM. All runs were done using a single processor. All codes were written in Fortran 77 and compiled with gfortran compiler. The portable random number generator of Schrage [40] was used. We set the number of maximum iteration equal to 1000 for both heuristics. The parameter M in the nonmonotone local search was set to 10 (since this is a commonly accepted value in the relevant literature, see e.g., [48, 38]).

The following benchmark problem instances[1] were used:

`g` problems. These test problems were used by Fujisawa et al. in [29]. They consist of sparse graphs whose size in terms of number of nodes varies from 10 to 1250.

`sg3dl` problems. Proposed by Burer, Monteiro, and Zhang [10], these instances correspond to cubic lattice graphs modeling Ising spin glasses. The graphs vary in sparsity and sizes, in such a way that the larger is the size in terms of number of nodes (from 1000 to about 3000) the lower is the density (from 0.60% to 0.22%).

[1]The benchmark problem instances are downloadable as a tar file and compressed with gzip from Mauricio G. C. Resende's webpage at `http://www2.research.att.com/ mgcr/data/index.html`

torus problems. This is also a set of instances from the Ising model of spin glasses. The complete problem library is available from the 7th DIMACS Implementation Challenge and is downloadable as a tar file and compressed with gzip from

`http://dimacs.rutgers.edu/Challenges/Seventh/Instances/`

B problems. The graphs in this set of instances are sparse and vary in size from 5000 to 8000 nodes.

G problems. These test problems were created by Helmberg and Rendl [43]. They consist of toroidal, planar, and randomly generated graphs of varying sparsity and sizes. These graphs vary in size from 800 to 3000 nodes and in density from 0.17% to 6.12%.

We performed a first experiment, consisting in comparing the performances of the NM-GRASP and the classical GRASP on average among ten random runs. The results obtained are summarized in Table 1, whose first column reports the name of the instance. The remaining columns report the average number of iterations needed to find the best objective function value (Iter), the average objective function values (Obj) obtained by each algorithm, and the average CPU time in seconds (Time). Looking at the results, we noticed that the NM-GRASP always found cuts greater than or equal to the cut found by the classical GRASP, often finding strictly better cuts.

Table 1: Comparison between NM-GRASP and the classical GRASP (average results over ten runs).

Problem	**NM-GRASP**			**GRASP**		
	Iter	Obj	Time	Iter	Obj	Time
g10.n	1	17	$< 10^{-3}$	1.2	17	$< 10^{-3}$
g20.n	1.9	37	$< 10^{-3}$	4.2	37	$< 10^{-3}$
g25.n	3.4	42	$< 10^{-3}$	2.7	42	$< 10^{-3}$
g30.n	3.4	61	$< 10^{-3}$	8.6	61	$< 10^{-3}$
g50.n	2.5	105	0.001	7.3	105	0
g100.n	9.9	214	0.018	50	214	0.02
g150.n	14	294	0.056	325.9	294	0.22
g200.n	110.2	405	0.798	316.5	402.4	0.38
g250.n	333.4	305	3.212	417.8	302.9	0.63
g500.n	647.1	572.4	25.651	432.3	566.4	2.68
g1000.n	425.4	1703.5	88.837	456.1	1693.5	16.11
g1250.n	532.3	2553.1	199.375	414	2518.5	26.52
sg3dl051000.mc	57.9	110	0.144	88.9	110	0.05
sg3dl052000.mc	217.6	112	0.511	82.2	112	0.05
sg3dl053000.mc	49.1	106	0.116	57	106	0.03
sg3dl054000.mc	106.3	114	0.251	143.1	114	0.08
sg3dl055000.mc	48.3	112	0.117	164.9	112	0.09
sg3dl056000.mc	81.8	110	0.193	110.6	110	0.06
sg3dl057000.mc	434.8	112	0.997	370.3	111.8	0.2
sg3dl058000.mc	53.6	108	0.135	120.4	108	0.07
sg3dl059000.mc	205.3	110	0.477	123.6	110	0.07
sg3dl101000.mc	555.2	881	136.226	515.1	868.4	22.29
sg3dl102000.mc	517.9	892.8	128.384	450.8	880	19.53
sg3dl103000.mc	559.7	878	138.619	554.1	865.8	24.18
sg3dl104000.mc	378.9	889.8	94.143	393.3	875.8	17.17
sg3dl105000.mc	398.7	875.6	95.765	604.5	864.4	26.22

continued on next page

Table 1 – continued from previous page

Problem	NM-GRASP			GRASP		
	Iter	Obj	Time	Iter	Obj	Time
sg3dl106000.mc	466.4	873.2	110.793	266.2	861.6	11.52
sg3dl107000.mc	279.6	891	71.17	278.9	876.8	12.21
sg3dl108000.mc	244.6	873	60.241	553.4	861.2	24.04
sg3dl109000.mc	408.3	887.8	101.35	498.7	875	21.8
sg3dl141000.mc	314.8	2404.2	713.647	385.7	2369.2	143.77
sg3dl142000.mc	493.7	2413.6	1114.955	429.8	2379.4	160.61
sg3dl143000.mc	590.2	2401.6	1338.184	515.8	2370	192.19
sg3dl144000.mc	477.9	2412	1084.964	442.3	2377.4	165.36
sg3dl145000.mc	463.2	2401.2	1058.657	461.9	2368.4	172.06
sg3dl146000.mc	644.5	2415.6	1459.443	351.1	2376.4	131.32
sg3dl147000.mc	420.6	2398.6	961.851	507.4	2368.4	189.57
sg3dl148000.mc	452.6	2405.6	1023.249	493.5	2372.8	183.08
sg3dl149000.mc	489.9	2383.6	1121.737	554.7	2353.4	207.13
sg3dl0510000.mc	245.5	112	0.563	144.2	112	0.08
sg3dl1010000.mc	333	883	82.535	582.9	872.6	25.36
sg3dl1410000.mc	418.3	2419.6	953.918	559.1	2380.8	209.09
torusg3-8.dat	1	3.9995e+7	0.262	1	3.4816e+7	0.01
torusg3-15.dat	1	2.7498e+8	13.587	1	2.439e+8	0.43
toruspm3-8-50.dat	225	452.4	12.435	375.6	449.4	3.93
toruspm3-15-50.dat	400.6	2968.8	1436.373	351.7	2925.2	200.59
B1	3.7	10000	40.84	13.1	10000	16.19
B2	2.1	12000	37.384	25	12000	46.57
B3	1.8	14000	46.177	26.4	14000	68.71
B4	2	16000	69.942	39.2	16000	135.93
G1	287.4	11624	143.49	464.5	11550.3	65.70
G2	629.7	11618.9	314.393	390.4	11548	55.22
G3	246.4	11621.9	124.069	438.1	11566.7	61.55
G4	238.3	11645.9	119.667	529.2	11580.8	74.64
G5	473.6	11630.7	237.006	547.7	11577.4	77.42
G6	536.6	2174.6	285.756	520	2108.2	75.64
G7	660	2001.3	355.281	610.4	1937.9	88.66
G8	481.5	2003.5	259.395	723.7	1945.5	105.96
G9	345.6	2050.2	185.079	421	1984.7	60.96
G10	303.1	1999.3	163.944	449.5	1928.5	65.5
G11	369.8	555.4	47.203	389.3	552.2	7.83
G12	301	549.2	38.424	428.5	544.6	8.62
G13	599.3	575.4	76.576	413.8	572	8.42
G14	459	3050.8	86.243	403.7	3026.9	15.74
G15	560.1	3032.8	103.001	411.3	3006.9	16.01
G16	434.5	3035.7	80.893	369.5	3012.7	14.42
G17	483	3031.6	88.552	457.8	3006.2	17.82
G18	464	982	103.895	764.9	957.5	33.34
G19	432.1	896	98.31	534.2	871.3	23.35
G20	512.1	936.4	114.104	556.7	907.9	24.34
G21	486.1	921.8	108.461	687.6	894	29.99
G22	465.5	13334.7	902.785	421.9	13189.2	192.09
G23	646.7	13317.8	1267.386	444.8	13187.4	203.54
G24	555.7	13311.7	1077.59	420.5	13179.1	191.34
G25	373.3	13311.6	729.414	431.2	13181.4	197.12

continued on next page

Table 1 – continued from previous page

Problem	**NM-GRASP**			**GRASP**		
	Iter	Obj	Time	Iter	Obj	Time
G26	437.7	13302.4	847.347	517.2	13171.1	235.76
G27	503	3305.6	1069.823	450.5	3167.2	216.99
G28	451.2	3266.6	968.743	441.7	3132.9	211.8
G29	479.4	3363.7	1023.348	604.2	3226.6	290.9
G30	489.6	3378.7	1050.762	444.3	3242	214.79
G31	416.9	3275.1	886.558	557.2	3143.5	266.83
G32	541.1	1385	538.142	523.1	1371.2	73
G33	375.2	1359.8	368.493	426	1347	59.69
G34	430.7	1362.2	424.573	377.5	1349.6	52.49
G35	508.6	7629.4	675.347	351	7569.2	96.49
G36	487.9	7624.9	648.174	535.6	7561.8	147.43
G37	581.2	7635.6	770.999	484.4	7572.3	133.54
G38	442.6	7633.4	591.377	438.9	7575.2	120.88
G39	504.6	2359.2	810.084	429	2281.9	132.19
G40	493.5	2349.9	768.65	465.6	2275.7	142.66
G41	484.5	2349.4	764.498	422.1	2276.4	129.55
G42	391.6	2436.2	623.453	506.6	2356.5	155.53
G43	406.3	6656.8	185.258	455.3	6594.4	48.06
G44	471.3	6644.8	215.923	327.1	6590.1	34.60
G45	387.5	6648.3	177.874	422.6	6588.6	44.78
G46	261	6642.4	118.976	540.1	6586.1	57.32
G47	480	6652.9	217.914	535.4	6589.6	56.60
G48	2.2	6000	7.266	16.4	6000	5.98
G49	5.5	6000	14.61	10.6	6000	4.14
G50	421.1	5865.6	918.896	496.5	5862.2	157.14
G51	534.6	3827.3	160.114	525	3797.5	33.16
G52	382	3830.2	114.211	441.3	3800.9	27.85
G53	509.7	3829	154.707	401.7	3800.8	25.47
G54	487.2	3827.9	148.097	522.2	3798.8	33.10
G55	473.6	10192.2	3424.146	367.5	10052.3	461.13
G56	364.3	3896.1	3019.725	394.4	3731.5	550.82
G57	451.3	3426.2	3508.808	452.7	3388	456.50
G58	419.6	19121.1	3987.402	442	18969.1	850.52
G59	488.1	5920.2	5278.781	369.5	5731.2	787.29
G60	488.5	14030.9	7210.677	498.3	13832.1	1252.01
G61	563.1	5616.4	9519.908	480.5	5385	1356.31
G62	482	4777.8	7934.699	510.7	4732	1072.33
G63	386.5	26796.9	7627.619	485.3	26586.4	1936.12
G64	547.9	8484.1	11879.552	486.2	8230.3	2113.55
G65	367.3	5456	8055.086	532.7	5399	1482.73
G66	459.1	6235.2	13156.522	243.7	6172.8	883.88
G67	382.9	6816.4	13927.613	418	6751.8	1893.71

We then performed the following second experiment to help determine whether the integration of the nonmonotone strategy in the local search phase in a classical GRASP is robustly beneficial. For each problem instance, we performed 10 different random runs of the classic GRASP and selected the one with the best objective function value (`best-objf-or`). Then, we performed a single run of the NM-GRASP using the following stopping condition:

$$\texttt{objf-NM} \geq \texttt{best-objf-or}.$$

Table 2 summarizes the results obtained. The NM-GRASP was not able to reach in a single run the value `best-objf-or` for only 6 out of 59 instances (namely, `sg3dl057000.mc`, `torusg3-8.dat`, `torusg3-15.dat`, `torusg3-8-50 .dat`, `G11`, `G50`). Moreover, by taking a better look at the results, it is evident that, apart from those 6 instances, the NM-GRASP always reached `best-objf-or` and, in several cases, it obtained even a better objective function value. Furthermore, the NM-GRASP generally needs a fewer number of iterations (in several cases just one iteration is needed) to reach the value `best-objf-or`. This suggests that the nonmonotone local search strategy improves the ability of the GRASP heuristic in searching for good solutions.

Table 2: One run of the NM-GRASP versus the best over ten runs of the classical GRASP.

Problem	**NM-GRASP**			**GRASP**		
	Iter	Obj	Time	Iter	Obj	Time
g10.n	2	17	$< 10^{-3}$	1	17	$< 10^{-3}$
g20.n	2	37	$< 10^{-3}$	1	37	$< 10^{-3}$
g25.n	1	42	$< 10^{-3}$	1	42	$< 10^{-3}$
g30.n	1	61	$< 10^{-3}$	1	61	$< 10^{-3}$
g50.n	2	105	$< 10^{-3}$	1	105	$< 10^{-3}$
g100.n	1	214	$< 10^{-3}$	1	214	0.01
g150.n	7	294	0.04	131	294	0.15
g200.n	70	405	0.51	550	404	2.32
g250.n	3	303	0.04	133	303	2.38
g500.n	5	569	0.24	363	568	20.55
g1000.n	3	1696	0.59	431	1696	34.71
g1250.n	4	2532	1.68	852	2532	323.6
sg3dl051000.mc	20	110	0.05	5	110	0.03
sg3dl052000.mc	97	112	0.23	1	112	0.18
sg3dl053000.mc	17	106	0.05	6	106	0.02
sg3dl054000.mc	155	114	0.36	29	114	0.96
sg3dl055000.mc	91	112	0.21	20	112	0.05
sg3dl056000.mc	279	110	0.63	9	110	0.1
sg3dl057000.mc	–	–	–	25	112	1.02
sg3dl058000.mc	32	108	0.09	8	108	0.07
sg3dl059000.mc	203	110	0.45	18	110	1.35
sg3dl101000.mc	585	878	140.27	329	878	80.57
sg3dl102000.mc	6	882	1.48	142	882	118.24
sg3dl103000.mc	6	878	1.51	455	872	100.02
sg3dl104000.mc	31	884	7.26	566	878	2.26
sg3dl105000.mc	36	868	8.21	37	868	98.13
sg3dl106000.mc	13	864	2.91	10	864	36.63
sg3dl107000.mc	1	884	0.27	68	882	62.96
sg3dl108000.mc	6	872	1.51	861	866	27.98
sg3dl109000.mc	9	878	2.06	983	878	88.07
sg3dl141000.mc	2	2380	4.5	496	2378	1054.13
sg3dl142000.mc	1	2382	2.35	634	2382	368.66
sg3dl143000.mc	17	2382	37.14	523	2380	1925.46

continued on next page

Table 2 – continued from previous page

Problem	**NM-GRASP**			**GRASP**		
	Iter	Obj	Time	Iter	Obj	Time
sg3dl144000.mc	5	2398	11.03	550	2392	1844.05
sg3dl145000.mc	13	2388	28.13	786	2384	1912.13
sg3dl146000.mc	1	2382	2.17	112	2382	881.3
sg3dl147000.mc	5	2380	11.67	798	2376	1436.56
sg3dl148000.mc	6	2390	13.41	491	2380	335.63
sg3dl149000.mc	5	2376	11.6	862	2362	1637.07
sg3dl0510000.mc	321	112	0.76	50	112	0.04
sg3dl1010000.mc	113	878	26.91	831	878	126.4
sg3dl1410000.mc	11	2398	24.63	765	2390	1057.88
torusg3-8.dat	–	–	–	1	37004013	0.18
torusg3-15.dat	–	–	–	1	253838426	13.05
toruspm3-8-50.dat	–	–	–	244	454	11.2
toruspm3-15-50.dat	1	2954	3.76	329	2932	507.5
B1	5	10000	41.18	8	10000	21.76
B2	1	12000	19.07	27	12000	15.77
B3	1	14000	32.9	14	14000	22.95
B4	1	16000	40.98	1	16000	2.11
G1	2	11596	0.81	207	11580	31.35
G2	1	11602	0.4	255	11561	475.92
G3	3	11612	1.33	91	11607	109.88
G4	1	11603	0.44	241	11591	308.45
G5	1	11590	0.41	639	11585	312.89
G6	7	2133	3.23	737	2132	254.34
G7	1	1981	0.46	800	1958	413.84
G8	1	1969	0.43	928	1957	178.94
G9	3	2022	1.39	804	1998	72.5
G10	1	1971	0.45	730	1939	400.74
G11	–	–	–	217	556	42.05
G12	61	548	7.67	536	548	22.55
G13	204	576	25.26	17	574	44.77
G14	4	3030	0.73	162	3030	6.15
G15	2	3016	0.38	231	3013	184.22
G16	1	3025	0.18	842	3021	31.79
G17	2	3017	0.34	902	3013	102.76
G18	15	969	3	883	967	18.41
G19	7	890	1.5	520	878	36.94
G20	30	925	6.34	835	914	101.82
G21	2	902	0.43	845	901	27.28
G22	1	13259	1.75	125	13206	1183.56
G23	1	13278	1.79	763	13215	1821.67
G24	1	13248	1.64	84	13202	1027.86
G25	1	13248	1.68	344	13198	187.57
G26	1	13251	1.69	519	13188	1426.53
G27	1	3213	2.08	331	3184	1038.97
G28	1	3182	1.98	361	3154	579.41
G29	1	3273	1.86	418	3247	584.06
G30	2	3305	3.97	621	3298	1113.15
G31	1	3189	1.94	537	3177	1252.84
G32	76	1386	74.32	333	1378	230.46

continued on next page

Table 2 – continued from previous page

Problem	**NM-GRASP**			**GRASP**		
	Iter	Obj	Time	Iter	Obj	Time
G33	66	1352	65.29	297	1350	707.62
G34	3	1358	3.13	615	1356	326.82
G35	1	7596	1.28	254	7585	481.51
G36	1	7600	1.22	758	7572	133.79
G37	1	7582	1.36	68	7580	889.55
G38	2	7595	2.37	739	7582	491.59
G39	2	2302	3.11	418	2299	291.4
G40	1	2301	1.45	702	2293	1156.37
G41	1	2304	1.43	411	2295	1076.95
G42	7	2396	10.61	469	2384	1343.65
G43	1	6611	0.38	598	6600	394.43
G44	3	6611	1.29	452	6605	147.3
G45	1	6597	0.36	425	6593	125.55
G46	2	6619	0.85	463	6600	209.7
G47	2	6602	0.81	625	6598	354.01
G48	1	6000	2.54	13	6000	3.39
G49	6	6000	12.59	1	6000	2.93
G50	–	–	–	436	5868	1526.35
G51	1	3807	0.31	631	3807	205.87
G52	3	3813	0.89	769	3809	213.17
G53	2	3814	0.62	280	3808	7.08
G54	2	3820	0.64	422	3805	94.9
G55	1	10149	7.04	360	10062	6444.52
G56	1	3851	7.56	380	3739	2650.62
G57	2	3402	14.21	263	3394	2342.56
G58	1	19029	7.86	680	18977	4868.69
G59	1	5855	11.78	850	5756	9145.78
G60	1	13921	13.48	342	13845	11912.2
G61	1	5541	15.49	473	5405	4230.71
G62	4	4750	57.83	896	4742	1390.64
G63	1	26720	18.92	402	26595	13323.07
G64	1	8460	20.69	333	8262	16665.15
G65	1	5406	20.71	122	5404	7127.78
G66	1	6188	30.69	97	6180	24685.21
G67	3	6778	105.45	100	6756	2438.23

As a further experiment, given the random component of the algorithms, we considered the empirical distributions of the random variable *time-to-target-solution-value* considering instances `g1250.n`, `G40`, `sg3dl142000.mc`, and `toruspm3-15-50` using different target values. We performed 100 independent runs of each heuristic (using random number generator seeds 270001, 270002, . . ., 270100) and recorded the time taken to find a solution at least as good as the target solution. For each run, we considered one hour as time limit.

To plot the empirical distribution, we associate with the i-th sorted running time (t_i) a probability $p_i = (i - \frac{1}{2})/100$, and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 100$.

For the instances `g1250.n`, `G40`, `sg3dl142000.mc`, and `toruspm3-15-50` we fixed as target values 2518, 2275, 2379, and 2925, respectively. These values represent a standard target for both heuristics. As we can see in Figure 6, apart from the instance `toruspm3-15-50` where for 3 runs the classical GRASP is better, we can notice that the NM-GRASP is always superior. It is able to reach the target value in less than 100 seconds CPU time for all the runs, while in several runs the classical GRASP needs more than 1000 seconds.

Figure 7 depicts the empirical distributions of the random variable *time-to-target-solution-value* using as target values 2532, 2293, 2382, and 2932, for the instances `g1250.n`, `G40`, `sg3dl14-2000.mc`, and `toruspm3-15-50`, respectively. These values are the best objective function values found by the classical GRASP over 10 runs. As we can see from the plots, also in this case, the NM-GRASP is able to reach the target value in less than 100 seconds for all the runs. On the other hand, the classical GRASP failed to reach the target solution within the time limit in several runs, especially for instances `g1250.n` and `G40`.

By using instances `g1250.n`, `G40`, `sg3dl142000.mc`, and `toruspm3-15-50`, we plot in Figure 8 the empirical distributions of the random variable *time-to-target-solution-value* using as target values 2556, 2362, 2420, and 2980, respectively. These target values are the best cuts found by the NM-GRASP over 10 runs. In this case, the classical GRASP failed to reach the target solution within the time limit for all runs and all instances. On the contrary, the NM-GRASP is able to reach the target solution for all runs for instances `g1250.n` and `sg3dl142000.mc`.

6. Concluding remarks

In this paper, we introduced a new nonmonotone strategy to explore the neighborhood of the current solution during a local search phase and formally stated the convergence of the resulting nonmonotone local search to a locally optimal solution. To illustrate its effectiveness, we used it as local search procedure in a GRASP framework and compared the resulting Nonmonotone GRASP (NM-GRASP) with a classical GRASP on the Maximum Cut Problem, a classical hard combinatorial optimization problem. The comparison showed that the new proposed approach is very competitive, outperforming the classical GRASP both in solution quality and running times.

The study of the empirical distributions of the random variable time-to-target-solution-value revealed that, given any fixed amount of computing time, NM-GRASP has an empirically higher probability than the classical GRASP of finding a target solution.

References

[1] R. Alvarez-Valdes, F. Parreño, and J. Tamarit, "Reactive GRASP for the strip-packing problem," *Computers & Operations Research*, vol. 35, no. 4, pp. 1065–1083, 2008.

[2] D. Andrade and M. Resende, "GRASP with path-relinking for network migration scheduling," in *Proceedings of the International Network Optimization Conference (INOC 2007)*, 2007.

[3] C. Andres, C. Miralles, and R. Pastor, "Balancing and scheduling tasks in assembly lines with sequence-dependent setup times," *European J. of Operational Research*, vol. 187, no. 3, pp. 1212–1223, 2008.

[4] S. Areibi and A. Vannelli, "A GRASP clustering technique for circuit partitioning," in *Satisfiability problems* (J. Gu and P. Pardalos, eds.), vol. 35 of *DIMACS Series on Discrete*

Mathematics and Theoretical Computer Science, pp. 711–724, American Mathematical Society, 1997.

[5] J. Arroyo, P. Vieira, and D. Vianna, "A GRASP algorithm for the multi-criteria minimum spanning tree problem," *Annals of Operations Research*, vol. 159, pp. 125–133, 2008.

[6] J. Atkinson, "A greedy randomised search heuristic for time-constrained vehicle scheduling and the incorporation of a learning strategy," *J. of the Operational Research Society*, vol. 49, pp. 700–708, 1998.

[7] F. Barahona, "The max-cut problem in graphs not contractible to k_5," *Operations Research Letters*, vol. 2, pp. 107–111, 1983.

[8] J. Bard, L. Huang, P. Jaillet, and M. Dror, "A decomposition approach to the inventory routing problem with satellite facilities," *Transportation Science*, vol. 32, pp. 189–203, 1998.

[9] S. Binato, W. Hery, D. Loewenstern, and M. Resende, "A greedy randomized adaptive search procedure for job shop scheduling," in *Essays and surveys on metaheuristics* (C. Ribeiro and P. Hansen, eds.), pp. 58–79, Kluwer Academic Publishers, 2002.

[10] S. Burer and R. Monteiro., "Rank-two relaxation heuristics for max-cut and other binary quadratic programs," *SIAM J. on Optimization*, vol. 12, pp. 503–521, 2001.

[11] C. Carreto and B. Baker, "A GRASP interactive approach to the vehicle routing problem with backhauls," in *Essays and surveys on metaheuristics* (C. Ribeiro and P. Hansen, eds.), pp. 185–200, Kluwer Academic Publishers, 2002.

[12] I. Contreras and J. Díaz, "Scatter search for the single source capacitated facility location problem," *Annals of Operations Research*, vol. 157, pp. 73–89, 2008.

[13] G. Cravo, G. Ribeiro, and L. N. Lorena, "A greedy randomized adaptive search procedure for the point-feature cartographic label placement," *Computers and Geosciences*, vol. 34, no. 4, pp. 373–386, 2008.

[14] A. Facchiano, P. Festa, A. Marabotti, L. Milanesi, and F. Musacchia, "Solving biclustering with a GRASP-like metaheuristic: Two case-studies on gene expression analysis," vol. 7548 of *Lecture Notes in Computer Science*, pp. 253–267, Springer-Verlag, 2012.

[15] T. Feo and M. Resende, "A probabilistic heuristic for a computationally difficult set covering problem," *Operations Research Letters*, vol. 8, pp. 67–71, 1989.

[16] T. Feo and M. Resende, "Greedy randomized adaptive search procedures," *J. of Global Optimization*, vol. 6, pp. 109–133, 1995.

[17] D. Ferone, P. Festa, and M. Resende, "Hybrid metaheuristics for the far from most string problem," in *Proceedings of HM 2013*, vol. 7919 of *Lecture Notes in Computer Science*, pp. 174–188, Springer-Verlag, 2013.

[18] P. Festa, "On some optimization problems in molecular biology," *Mathematical Bioscience*, vol. 207, no. 2, pp. 219–234, 2007.

[19] P. Festa, "A biased random-key genetic algorithm for data clustering," *Mathematical Bioscience*, vol. 245, no. 1, pp. 76–85, 2013.

[20] P. Festa and P. Pardalos, "Efficient solutions for the far from most string problem," *Annals of Operations Research*, vol. 196, no. 1, pp. 663–682, 2012.

[21] P. Festa, P. Pardalos, L. Pitsoulis, and M. Resende, "GRASP with path-relinking for the weighted MAXSAT problem," *ACM J. on Experimental Algorithmics*, vol. 11, pp. 1–16, 2006.

[22] P. Festa, P. Pardalos, and M. Resende, "Algorithm 815: FORTRAN subroutines for computing approximate solution to feedback set problems using GRASP," *ACM Transactions on Mathematical Software*, vol. 27, pp. 456–464, 2001.

[23] P. Festa, P. Pardalos, M. Resende, and C. Ribeiro, "Randomized heuristics for the MAX-CUT problem," *Optimization Methods and Software*, vol. 17, no. 6, pp. 1033–1058, 2002.

[24] P. Festa and M. G. C. Resende, "An annotated bibliography of GRASP – Part I: algorithms," *International Transactions in Operational Research*, vol. 16, no. 1, pp. 1–24, 2009.

[25] P. Festa and M. G. C. Resende, "An annotated bibliography of GRASP – Part II: applications," *International Transactions in Operational Research*, vol. 16, no. 2, pp. 131–172, 2009.

[26] P. Festa and M. Resende, "GRASP: An annotated bibliography," in *Essays and Surveys on Metaheuristics* (C. Ribeiro and P. Hansen, eds.), pp. 325–367, Kluwer Academic Publishers, 2002.

[27] P. Festa and M. Resende, "GRASP: Basic components and enhancements," *Telecommunication Systems*, vol. 46, no. 3, pp. 253–271, 2011.

[28] R. Frinhani, R. Silva, G. Mateus, P. Festa, and M. Resende, "GRASP with path-relinking for data clustering: A case study for biological data," vol. 6630 of *Lecture Notes in Computer Science*, pp. 410–420, Springer-Verlag, 2011.

[29] K. Fujisawa, M. Fukuda, M. Fojima, and K. Nakata, "Numerical evaluation of SDPA (Semidefinite Programming Algorithm," in *High performance optimization*, pp. 267–301, Kluwer Academic Publishers, 2000.

[30] M. Garey and D. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Company, New York, 1979.

[31] F. Glover, "Tabu search – Part I," *ORSA J. on Computing*, vol. 1, pp. 190–206, 1989.

[32] F. Glover, "Tabu search – Part II," *ORSA J. on Computing*, vol. 2, pp. 4–32, 1990.

[33] F. Glover, "Tabu search and adaptive memory programing – Advances, applications and challenges," in *Interfaces in Computer Science and Operations Research* (R. Barr, R. Helgason, and J. Kennington, eds.), pp. 1–75, Kluwer, 1996.

[34] F. Glover and M. Laguna, *Tabu Search*. Kluwer Academic Publishers, 1997.

[35] A. Goëffon, J.-M. Richer, and J.-K. Hao, "Progressive tree neighborhood applied to the maximum parsimony problem," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 5, no. 1, pp. 136–145, 2008.

[36] M. Goemans and D. Williams, "Improved approximation algorithms for Max-Cut and Satisfiability Problems using Semidefinite Programming," *J. of the ACM*, vol. 42, pp. 1115–1145, 1995.

[37] D. Goldberg, *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley, 1989.

[38] L. Grippo, F. Lampariello, and S. Lucidi, "A class of nonmonotone stabilization methods in unconstrained optimization," *Numerische Mathematik*, vol. 59, no. 1, pp. 779–805, 1991.

[39] L. Grippo, L. Palagi, M. Piacentini, V. Piccialli, and G. Rinaldi, "Speedp: an algorithm to compute sdp bounds for very large max-cut instances," *Mathematical Programming*, vol. 136, no. 2, pp. 353–373, 2012.

[40] M. Grötschel and W. Pulleyblank, "Weakly bipartite graphs and the max-cut problem," *Operations Research Letters*, vol. 1, pp. 23–27, 1981.

[41] F. O. Hadlock, "Finding a maximum cut of a planar graph in polynomial time," *SIAM Journal on Computing*, vol. 4, pp. 221–225, 1975.

[42] P. Hansen and N. Mladenović, "Developments of variable neighborhood search," in *Essays and Surveys in Metaheuristics* (C. Ribeiro and P. Hansen, eds.), pp. 415–439, Kluwer Academic Publishers, 2002.

[43] C. Helmberg and F. Rendl, "A spectral bundle method for semidefinite programming," *SIAM J. on Optimization*, vol. 10, pp. 673–696, 2000.

[44] M. Hirsch, C. Meneses, P. Pardalos, M. Ragle, and M. Resende, "A continuous GRASP to determine the relationship between drugs and adverse reactions," in *Data mining, systems analysis, and optimization in biomedicine* (O. Seref, O. Kundakcioglu, and P. Pardalos, eds.), vol. 953 of *AIP Conference Proceedings*, pp. 106–121, Springer, 2007.

[45] S. Karisch, F. Rendl, and J. Clausen, "Solving graph bisection problems with semidefinite programming," *SIAM J. on Computing*, vol. 12, pp. 177–191, 2000.

[46] R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations* (R. Miller and J. Thatcher, eds.), pp. 85–103, Plenum Press, NY, 1972.

[47] S. Kirkpatrick, "Optimization by simulated annealing: Quantitative studies," *J. of Statistical Physics*, vol. 34, pp. 975–986, 1984.

[48] S. L. L. Grippo, F. Lampariello, "A nonmonotone line search technique for Newton's method," *SIAM Journal on Numerical Analysis*, vol. 23, pp. 707–716, 1986.

[49] M. Laguna and R. Martí, "A GRASP for coloring sparse graphs," *Computaional Optimization and Applications*, vol. 19, pp. 165–178, 2001.

[50] R. D. Leone, P. Festa, and E. Marchitto, "A bus driver scheduling problem: A new mathematical model and a GRASP approximate solution," *Journal of Heuristics*, vol. 17, no. 4, pp. 441–466, 2011.

[51] R. D. Leone, P. Festa, and E. Marchitto, "Solving a bus driver scheduling problem with randomized multistart heuristics," *International Transactions in Operational Research*, vol. 18, no. 6, pp. 707–727, 2011.

[52] S. Lin and B. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, pp. 498–516, 1973.

[53] L. Lovász, "On the Shannon capacity of a graph," *IEEE Trans. of Information Theory*, vol. IT-25, pp. 1–7, 1979.

[54] R. Martí and M. Laguna, "Heuristics and meta-heuristics for 2-layer straight line crossing minimization," *Discrete Applied Mathematics*, vol. 127, no. 3, pp. 665–678, 2003.

[55] T. Mavridou, P. Pardalos, L. Pitsoulis, and M. Resende, "A GRASP for the biquadratic assignment problem," *European J. of Operational Research*, vol. 105, pp. 613–621, 1998.

[56] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers and Operations Research*, vol. 24, pp. 1097–1100, 1997.

[57] I. Osman, B. Al-Ayoubi, and M. Barake, "A greedy random adaptive search procedure for the weighted maximal planar graph problem," *Computers and Industrial Engineering*, vol. 45, no. 4, pp. 635–651, 2003.

[58] C. H. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," *Journal of Comput. System Science*, vol. 43, no. 3, pp. 425–440, 1991.

[59] C. Papadimitriou and K. Steiglitz, *Combinatorial optimization: Algorithms and complexity.* Prentice-Hall, 1982.

[60] P. Pardalos and M. Resende, eds., *Handbook of Applied Optimization.* Oxford University Press, 2002.

[61] S. Poljak, F. Rendl, and H. Wolkowicz, "A recipe for semidefinite relaxation for 0-1 quadratic programming," *J. of Global Optimization*, vol. 7, pp. 51–73, 1995.

[62] G. Pu, Z. Chong, Z. Qiu, Z. Lin, and J. He, "A hybrid heuristic algorithm for HW-SW partitioning within timed automata," in *Proceedings of Knowledge-based Intelligent Information and Engineering Systems*, vol. 4251 of *Lecture Notes in Artificial Intelligence*, pp. 459–466, Springer-Verlag, 2006.

[63] M. Resende and T. Feo, "A GRASP for satisfiability," in *Cliques, Coloring, and Satisfiability: The Second DIMACS Implementation Challenge* (D. Johnson and M. Trick, eds.), vol. 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 499–520, American Mathematical Society, 1996.

[64] M. Resende, L. Pitsoulis, and P. Pardalos, "Approximate solution of weighted MAX-SAT problems using GRASP," in *Satisfiability problems* (J. Gu and P. Pardalos, eds.), vol. 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 393–405, American Mathematical Society, 1997.

[65] M. Resende and C. Ribeiro, "A GRASP for graph planarization," *Networks*, vol. 29, pp. 173–189, 1997.

[66] C. Ribeiro and S. Urrutia, "Heuristics for the mirrored traveling tournament problem," *European J. of Operational Research*, vol. 179, pp. 775–787, 2007.

[67] A. Robertson, "A set of greedy randomized adaptive local search procedure (GRASP) implementations for the multidimensional assignment problem," *Computational Optimization and Applications*, vol. 19, pp. 145–164, 2001.

[68] N. Shor, "Quadratic optimization problems," *Soviet J. of Computer and Systems Science*, vol. 25, pp. 1–11, 1987.

[69] L. Trevisan, G. B. Sorkin, M. Sudan, and D. P. Williamson, "Gadgets, approximation, and linear programming," *SIAM Journal on Computing*, vol. 29, no. 6, pp. 2074–2097, 2000.

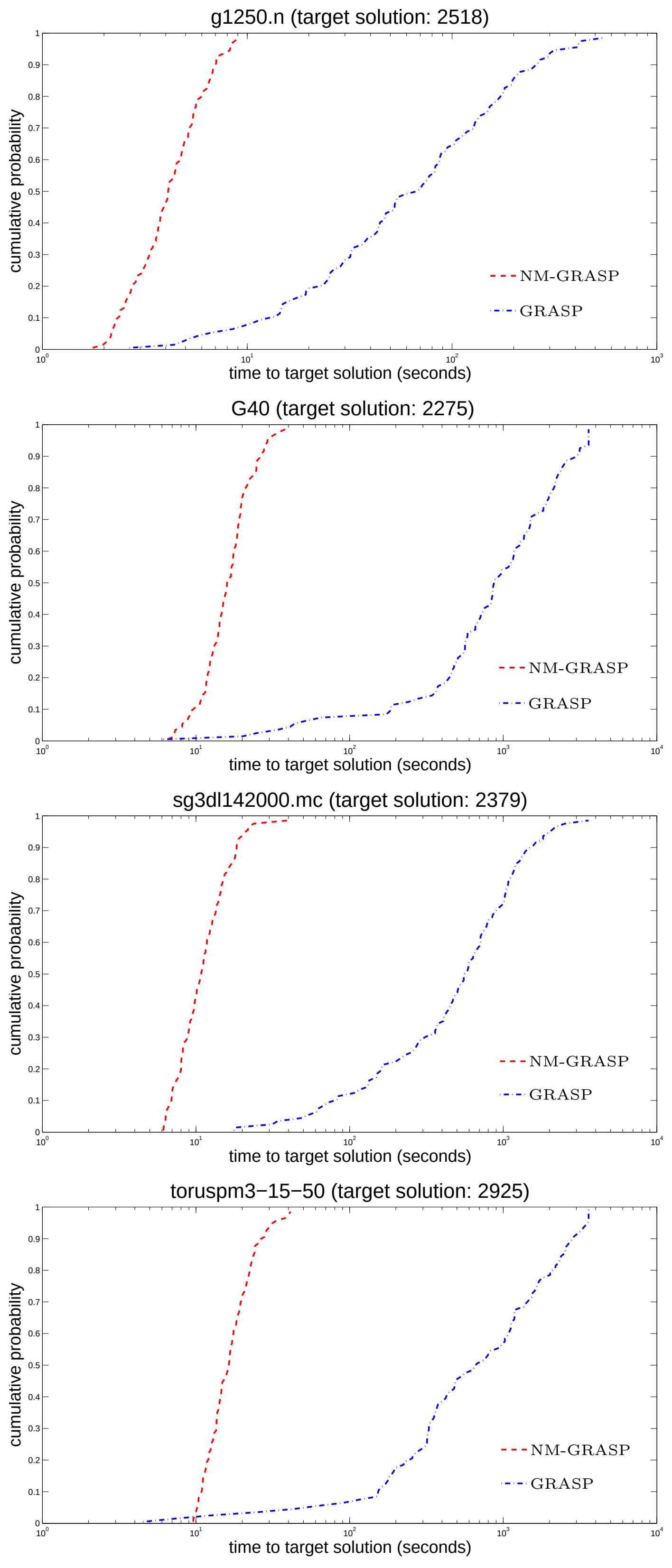

Figure 6: TTTplots for the easy targets.

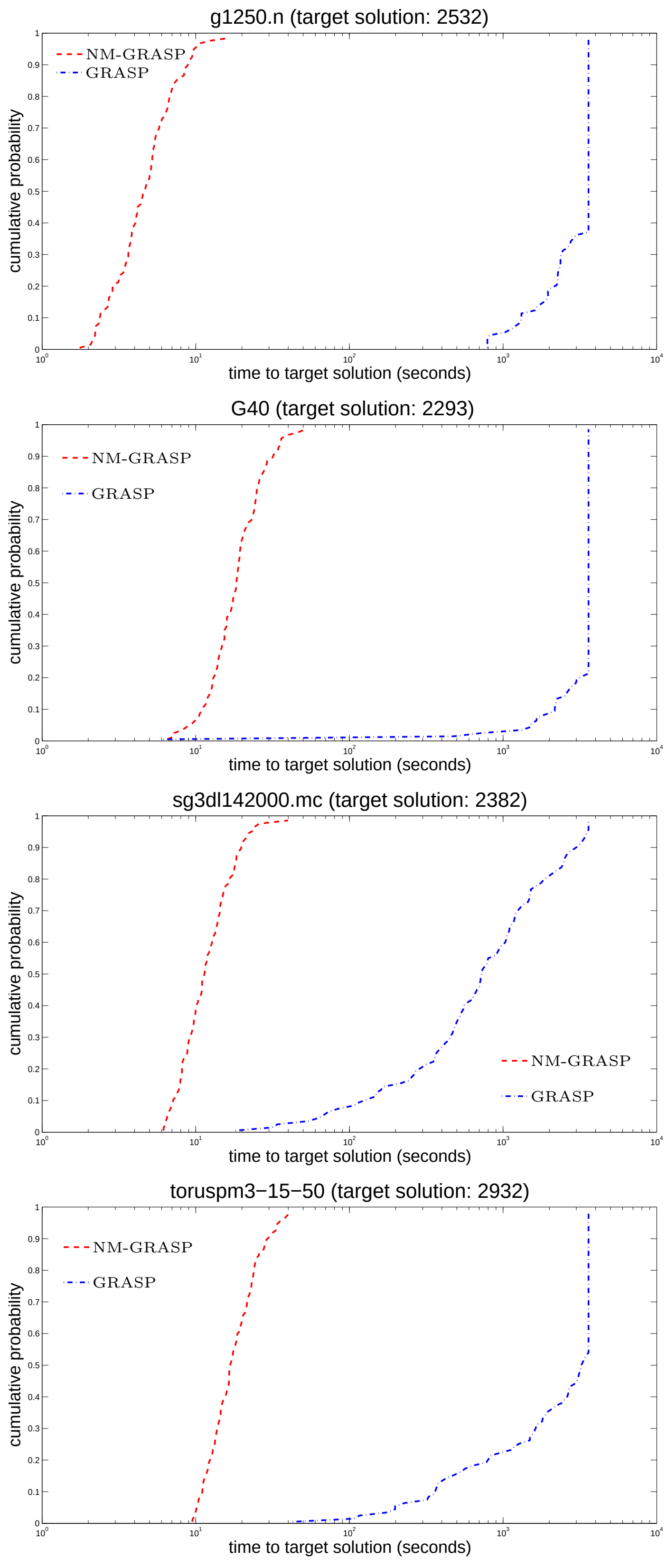

Figure 7: TTTplots for the classical GRASP targets.

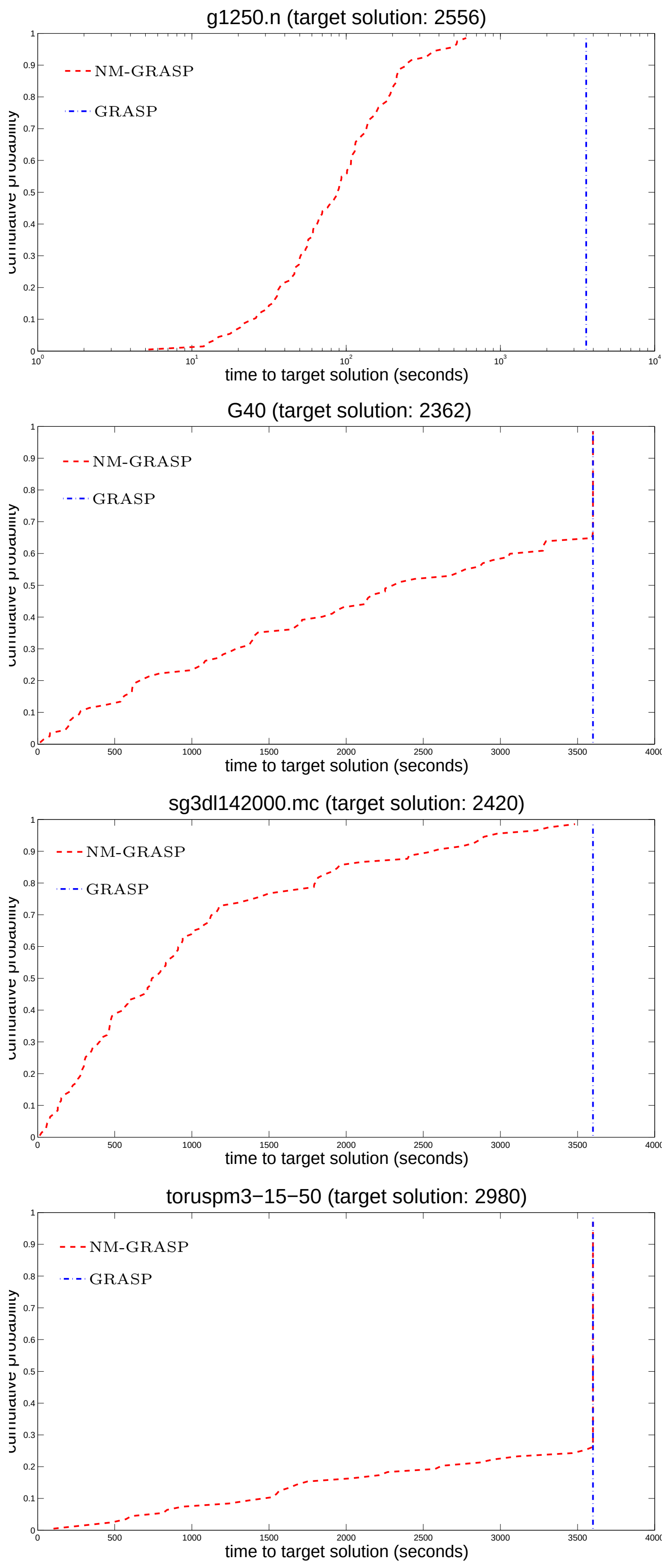

Figure 8: TTTplots for the Nonmonotone GRASP targets.