Noname manuscript No.
(will be inserted by the editor)

A Nonmonotone GRASP

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

Received: date / Accepted: date

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 the 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 so-

M. De Santis
Fakultät für Mathematik, TU Dortmund, Germany.
E-mail: msantis@math.tu-dortmund.de

P. Festa [Corresponding author]
Department of Mathematics and Applications, University of Napoli FEDERICO II, Napoli, Italy.
Phone/Fax +39081675605
E-mail: paola.festa@unina.it

G. Liuzzi
Istituto di Analisi dei Sistemi ed Informatica - CNR, Rome, Italy.
E-mail: giampaolo.liuzzi@iasi.cnr.it

S. Lucidi
Dipartimento di Ingegneria Informatica Automatica e Gestionale, "La Sapienza" University of Rome, Italy.
E-mail: lucidi@dis.uniroma1.it

F. Rinaldi
Dipartimento di Matematica, University of Padova, Italy.
E-mail: rinaldi@math.unipd.it

lution. 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 three classical hard combinatorial optimization problems: the Maximum Cut Problem (MAX-CUT), the Weighted Maximum Satisfiability Problem (MAX-SAT), and the Quadratic Assignment Problem (QAP).

Keywords Combinatorial Optimization, GRASP, Metaheuristics, Local Search, Nonmonotone Line Search, MAX-CUT, MAX-SAT, QAP.

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 $f : 2^{\mathcal{E}} \mapsto \mathbb{R}$, 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 [87], 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 [89].

Many combinatorial optimization problems are computationally intractable, in the sense that until now, no polynomial-time algorithm is known to exactly solve them [45]. 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 papers of Kirkpatrick on Simulated Annealing [69] which 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 [47–50], Genetic Algorithms [54], Variable Neighborhood Search [83,59], and GRASP (Greedy Randomized Adaptive Search Procedure) [30,31,39–42].

A GRASP is a multi-start or iterative process introduced by Feo and Resende [30], following in the spirit of the pioneering idea proposed in 1973 by Lin and Kernighan for the Traveling Salesman Problem [77]. 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 contexts. Recent areas of application include routing [9,11,19], logic [36,93,94], covering and partition [6,30,92], location [23,25], network optimization [7,85,97], assignment [82,99], timetabling and scheduling [3,5,4,14,98,73,74], graph and map drawing [37, 38,72,80,97], and very recently computational biology [33–35,28,32,43,51,64].

The 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 [55], 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. In Section 3, we illustrate a GRASP for three selected hard combinatorial optimization problems, i.e., the MAX-CUT, MAX-SAT, and QAP problem. Section 4 is devoted to the description and analysis of a nonmonotone variant of GRASP (NM-GRASP). In Section 5, we illustrate the effectiveness of our Nonmonotone GRASP by comparing it with the classical GRASP on standard test problems (from the literature) for the three combinatorial optimization problems described in Section 3. The experiments empirically show that the new described GRASP framework 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(·), g(·), MaxIterations, Seed)
1    x_best:=∅; f(x_best):=+∞;
2    for k = 1, 2, ...,MaxIterations→
3        x:=ConstructGreedyRandomizedSolution(Seed, g(·));
4        if (x not feasible) then
5            x:=repair(x);
6        endif
7        x:=LocalSearch(x, f(·));
8        if (f(x) < f(x_best)) then
9            x_best:=x;
10       endif
11   endfor;
12   return(x_best);
end GRASP
```

Fig. 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).

```
procedure ConstructGreedyRandomizedSolution(Seed, g(·))
1    x:=∅;
2    Sort the candidate elements i in a list C according to their incremental
     costs g(i);
3    while (x is not a complete solution)→
4        RCL:=MakeRCL(C);
5        v:=SelectIndex(RCL, Seed);
6        x := x ∪ {v};
7        C := C \ {v};
8        Resort remaining candidate elements j ∈ C according to their
         incremental costs g(j);
9    endwhile;
10   return(x);
end ConstructGreedyRandomizedSolution;
```

Fig. 2 Basic GRASP construction phase pseudo-code.

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 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* $g : C \to \mathbb{R}$ 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(·))
1    Let N(x) be the neighborhood of x;
2    H:={y ∈ N(x) | f(y) < f(x)};
3    while (|H| > 0)→
4        x:=Select(H);
5        H:={y ∈ N(x) | f(y) < f(x)};
6    endwhile
7    return(x);
end LocalSearch
```

Fig. 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 within $N(x)$ there is no better solution in terms of objective function value.

Once a suitable neighborhood $N(x)$ of the current solution x has been defined and computed (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 A GRASP for the MAX-CUT, the MAX-SAT, and the QAP

We briefly illustrate in this section the main features of state-of-the-art classical GRASP proposed for three classical hard combinatorial optimization problems: the MAX-CUT, the MAX-SAT, and the QAP problem.

3.1 MAX-CUT

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.

The problem can be formulated as the following integer quadratic program:

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

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

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

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 [68]. In 1991, it has been showed that MAX-CUT is APX-complete [86], meaning that unless P=NP, it does not allow a polynomial time approximation scheme [108]. Polynomially solvable cases include planar graphs [58], weakly bipartite graphs with nonnegative weights [57], and graphs without K_5 minors [10].

Given the inner intractability of the problem, many researchers have devoted their effort to both more deeply studying the inner computational nature of the problem and in designing good approximate and heuristic solution techniques (see e.g. [13,22]). Along this research line, the idea that the MAX-CUT

can be naturally relaxed to a semidefinite programming problem was first observed by Lovász [79] and Shor [103]. Goemans and Williamson [53] 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 [56,66,91].

In the following, we describe a classical GRASP proposed by Festa et al. in 2002 [38]. As any GRASP, it proceeds in iterations. At each iteration, a greedy randomized adaptive solution is built and used as starting point in a local search procedure. The best overall locally optimal solution is returned as an approximation of the global optimal.

The construction procedure uses 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 a greedy, randomized, and adaptive solution x is built, the local search procedure is invoked. Given the current solution x, it implements an elementary move, that 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 [38], 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 is found.

3.2 MAX-SAT

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 $\neg x_j$. Φ can formally be written as

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

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. Φ 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 [24]. The 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 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 [12]. 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 [45].

The first approximation algorithms for the MAX-SAT were introduced in [65], 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 [20] 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 [110] and also in [52] by Goemans and Williamson. Currently, one among the best deterministic polynomial time approximation algorithm for the MAX-SAT achieves a performance ratio of 0.758 and is based on semidefinite programming [53], while there is also a randomized algorithm with performance ratio 0.77 [8]. 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 [29,67,107]. With respect to inapproximability results, it is known [60] 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. In 1997, to heuristically solve the problem a GRASP has been proposed [95]. In [96] a complete Fortran implementation of the algorithm is given along with extensive computational runs. In the following, we provide a brief description of the main ingredients of the classical GRASP for the MAX-SAT.

Given a set of clauses $\mathbb{C}$ and a set of Boolean variables V, let $\mathbf{x} \in \{0,1\}^n$ be a *truth assignment* (i.e., the vector of truth values assigned to the variables) and let $c(\mathbf{x})$ be the sum of the weights of the satisfied clauses as implied by $\mathbf{x}$. Without loss of generality, all the weights w_i of the clauses are assumed to be positive integers. Given any two truth assignments $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ let $\Delta(\mathbf{x}, \mathbf{y})$ denote their *difference set*, i.e.,

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

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. As detailed in [95], in the construction phase of the algorithm a solution is built one element at a time in a greedy randomized fashion, by maintaining a 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 setting the respective truth value to the given variable. Given any partial solution, which corresponds to a set of satisfied clauses, the next element to be added to the solution is chosen taking into account the total weight of the unsatisfied clauses that become satisfied after the assignment to the just chosen element. More formally, 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 unassigned clauses that would become satisfied if variable x_i were to be set to `true`, and Γ_i^- be the set of unassigned clauses that would become satisfied if variable x_i were to be set to `false`. Let γ_j^+ and γ_j^- be the gain in the objective function value if we set

the unassigned variable x_j to 1 and 0, respectively. Formally, they are defined as follows:

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

If $X \subseteq V$ is the set of already assigned variables, the best gain γ^* is computed as

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

and RCL keeps 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 $X = X \cup \{x_s\}$. After each such addition to the partial solution, Γ_i^+, Γ_i^-, γ_j^+, and γ_j^- are consequently updated, in an adaptive fashion. The process is repeated until $|X| = n$.

Having completed a truth assignment $\mathbf{x}$, a local search is applied in order to guarantee local optimality. The 1-flip neighborhood is used in the local search, which is defined as

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

If $w(\mathbf{x})$ is the total weight of the satisfied clauses for the truth assignment $\mathbf{x}$, then $\mathbf{x}$ is a local maximum if and only if $w(\mathbf{x}) \geq w(\mathbf{y})$ for all $\mathbf{y} \in N_1(\mathbf{x})$.

3.3 QAP

Given a set $N = \{1, 2, \ldots, n\}$, the set Π_N of all permutations of the elements in N and two $n \times n$ matrices F and D, such that, for $i, j \in \{1, 2, \ldots, n\}$, $f_{ij}, d_{ij} \in \mathbb{R}^+$, the QAP aims at finding a permutation $\pi^* \in \Pi_N$ such that

$$\pi^* = \arg\min_{p \in \Pi_N} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \cdot d_{\pi(i)\pi(j)}.$$

The QAP was first proposed already in 1957 by Koopmans and Beckman [70] while studying the plant location problem. In the location theory context, one is given a set $\mathcal{F} = \{\{_1, \ldots, \{_n\}$ of n facilities and a set N of n locations. Matrices F and D represent the flow matrix and the distance matrix, respectively, and the objective is to determine to which location each facility must be assigned so as to minimize the total cost of the assignment. Since its first formulation, the QAP has appeared in several practical applications, including economy [61,62], scheduling [46], and numerical analysis [15]. Recent surveys on the QAP are given in [78] and in [26], besides two nice and comprehensive books ([90], [17]).

The QAP is one of the most difficult combinatorial optimization problems. In 1976, Sahni and Gonzales [101] had shown that it is NP-hard and that, unless P = NP, it is not possible to find an ϵ-approximation algorithm, for any constant ϵ and this result stands even under the hypotheses that F and D

are symmetric coefficient matrices. Due to its high computation complexity, to find in reasonable running times good quality solutions for the QAP Li et al. [76] proposed a GRASP, whose Fortran implementation has been described in [88].

In the GRASP for QAP, the construction phase performs two stages. In the first stage, only two assignments are produced. Once sorted inter-site distances in increasing order and inter-facility flows in decreasing order, the idea in this first stage is to assign facilities with high interaction (i.e., having high f_{ij} values) to nearby sites (i.e., sites with low d_{kl} values). Coherently, among the pairs of assignments having the smallest $d_{kl} \cdot f_{ij}$ products and inserted in the RCL a pair is selected at random and the corresponding assignment established. The remaining $n-2$ facility-site assignments are then made sequentially in the second stage. The idea now is to favor assignments that have small interaction cost with the set of previously-made assignments. To do this, at each iteration, the procedure keeps all costs of unassigned facility-site pairs sorted in increasing order. More specifically, let $\Gamma = \{(i_1, k_1), \ldots, (i_q, k_q)\}$ be the set of q assignments at a certain iteration of the construction phase. Then, the cost c_{jl} of assigning facility j to site l, with respect to the already-made q assignments is computed as follows:

$$c_{jl} = \sum_{(i,k)\in\Gamma} d_{kl} \cdot f_{ij}.$$

The pairs having the least $\alpha \cdot |\Gamma|$, $\alpha \in (0, 1]$ costs are inserted in the RCL and one of them is selected at random and added to Γ. The procedure is repeated until $n-1$ assignments are made. The remaining facility is then assigned to the remaining site. In the local search phase, a simple 2-exchange neighborhood structure is defined and the local improvement procedure considers all possible 2-swaps of facility-locations. If a swap improves the cost of the assignment, it is accepted. The procedure continues until no swap improves the solution value.

4 A 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 [55].

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: $\texttt{push}(f, W)$, which inserts into W a new function value f, and $\texttt{pop}(W)$, which drops from W the least recently inserted function value.

```
procedure NonmonotoneLocalSearch(x, f(·))
1    Let M ≥ 1 and let N(x) be the neighborhood of x;
2    W:={f(x)}; f̄:=f(x); x_min:=x;
3    H:={y ∈ N(x) | f(y) < f̄};
4    while (|H| > 0)→
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); f̄:=max{f ∈ W};
9        H:={y ∈ N(x) | f(y) < f̄};
10   endwhile
11   if x ≠ x_min then x:=x_min; goto 2;
12   return(x_min);
end NonmonotoneLocalSearch
```

Fig. 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

$$f(y) \geq \bar{f} \geq f(x), \qquad \forall\, y \in N(x). \tag{4}$$

Note that, condition (4) 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, and by x^k and $\bar{f}^k$ the solution and the reference value at the beginning of each 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

$$\bar{f}^k = \max_{0 \le i \le \min\{k,\ M\}} f(x^{k-i}), \tag{5}$$

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

We can now formally introduce the sequences $\{x^k\}$, $\{f(x^k)\}$, and $\{\bar{f}^k\}$ generated by `NonmonotoneLocalSearch`. 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 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 (5).*

Then, `NonmonotoneLocalSearch` *cannot cycle and produces a local minimum point.*

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

$$f(x^{k+1}) < \bar{f}^k. \tag{6}$$

From (5) and (6), we can write:

$$\bar{f}^{k+1} \le \bar{f}^k. \tag{7}$$

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

$$\bar{f}^{k+M} < \bar{f}^k - \delta. \tag{8}$$

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 (7) and (8), there exists an iteration $\tilde{k}$ such that

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

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

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

Hence, we have

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

which shows that the local search procedure cannot cycle. Then, the point produced is such that $|H| = 0$, therefore we have

$$f(x) \leq \bar{f} \leq f(y), \qquad \text{for all } y \in N(x),$$

which implies that x is a local minimum. □

For the three combinatorial optimization problem considered in section 3, namely the MAX-CUT, the MAX-SAT, and the QAP, we have designed a Nonmonotone GRASP (NM-GRASP), that applies the procedure

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

`NonmonotoneLocalSearch` is based on the same neighborhood structure as in the classical GRASPs. In the case of maximization problems (and this is the case for the MAX-CUT and the MAX-SAT), 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 our computational experience on the MAX-CUT, the MAX-SAT, and the QAP. In order to compare the performance of the classical GRASP and the NM-GRASP we tested the two heuristics on benchmark test problems from the literature. The instances used for the tests on the MAX-CUT and the MAX-SAT problems are downloadable from Mauricio G.C. Resende's webpage at `http://mauricio.resende.info/data/index.html`. The instances used for the tests on the QAP problem are downloadable from `http://anjos.mgi.polymtl.ca/qaplib/inst.html#BO`.

As for the MAX-CUT, the following problem instances were used:

`g` problems. These test problems were used by Fujisawa et al. in [44]. 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 [16], 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%).

`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 [63]. 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%.

As for the MAX-SAT, the instances have been generated from the jnh SAT problems class of the 2nd DIMACS Implementation Challenge by randomly generating clause weights uniformly between 1 and 1000. In these instances, the number of variables is 100, while the number of clauses ranges from 800 to 900.

For the QAP, benchmark problem instances have been proposed by Burkard et al.[18] and are known as QAPLIB - A Quadratic Assignment Problem Library[1]:

chr problems. These test problems were used by Christofides and Benavent in [21]. They are characterized by a $n \times n$ adjacency matrix of a weighted tree and a $n \times n$ adjacency matrix of a complete graph, with n varying from 12 to 25.

els19 problem. In this instance, the data describe the distances of $n = 19$ different facilities of a hospital and the flow of patients between those locations. It has been used by Mautor in [81].

esc problems. These test problems were used by Eschermann and Wunderlich in [27] in a computer science application where to minimize the amount of additional hardware needed for the testing of self-testable sequential circuits. In these instances, n varies from 16 to 128.

kra problems. These are real–world instances used to plan the Klinikum Regensburg in Germany and described by Krarup and Pruzan in [71]. Here, $n = 30$.

lipa problems. These test problems were randomly generated by Li and Pardalos [75]. They are asymmetric instances with known optimal solutions and n ranging from 20 to 90.

nug problems. These test problems were used by Negent et al. in [84]. They are characterized by a distance matrix containing Manhattan distances of rectangular grids. The size n ranges from 14 to 30.

rou problems. These instances were used by Roucairol in [100]. The entries of the matrices are randomly generated between 1 and 100 and $n = \{12, 15, 20\}$.

scr, tho, and wil problems. In all these instances, the entries of the matrices are rectangular. It only changes the size. In the scr problems $n = \{12, 15, 20\}$ and they were used by Scriabin and Vergin in [102]. In the tho

[1] The QAPLIB - A Quadratic Assignment Problem Library has an online version at http://anjos.mgi.polymtl.ca/qaplib/.

problems, $n = \{30, 40, 150\}$ and they were used by Thonemann and A. Bölte in [106]. Finally, in the `wil` problems, $n = \{50, 100\}$ and they were used by Wilhelm and Ward in [109].

`sko` problems. In these instances, the entries of the distance matricx are rectangular, the entries in the flow matrix are pseudorandom numbers, and n ranges from 42 to 100. They were used by Skorin-Kapov in [104].

`ste` problems. They refer to the backboard wiring problem and have size $n = 36$. Totally, they constitute a set of three instances characterized by data representing Manhattan, squared Euclidean, and Euclidean distances. They were used by Steinberg in [105].

We performed ten random runs for each instance considered. For each run, with the time limit of one hour, we stored the solution found with the best objective function value, the CPU time, and the iteration in which such solution was found.

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.

About the fine tuning of the parameter M used in the nonmonotone local search, the best value resulting in our experiments has been 10 for the MAX-CUT and the QAP instances and 5 for the MAX-SAT instances.

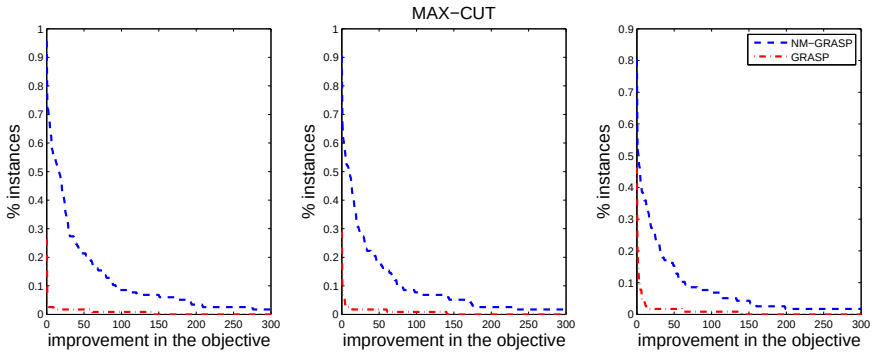

Fig. 5 Performance comparison between NM-GRASP (dashed line) and the original GRASP (dash-dot line) on the MAX-CUT problems (Worst-left; Average-center; Best-right).

Figure 5, Figure 6, and Figure 7 plot the performance of the NM-GRASP and the classical GRASP in terms of objective function value for the MAX-CUT, the MAX-SAT, and the QAP, respectively. The dashed line gives on the y-axis the percentage of instances in which the absolute improvement in terms of objective function value of NM-GRASP with respect to GRASP is greater than or equal to the value given in the x-axis. Furthermore, the dash-

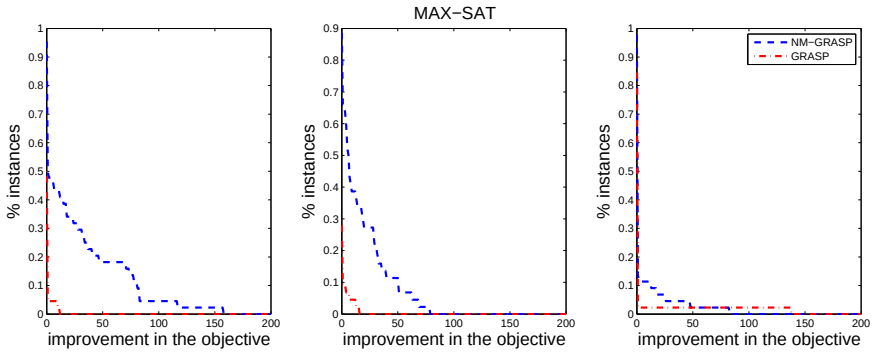

Fig. 6 Performance comparison between NM-GRASP (dashed line) and the original GRASP (dash-dot line) on Max-sat problems (Worst-left; Average-center; Best-right).

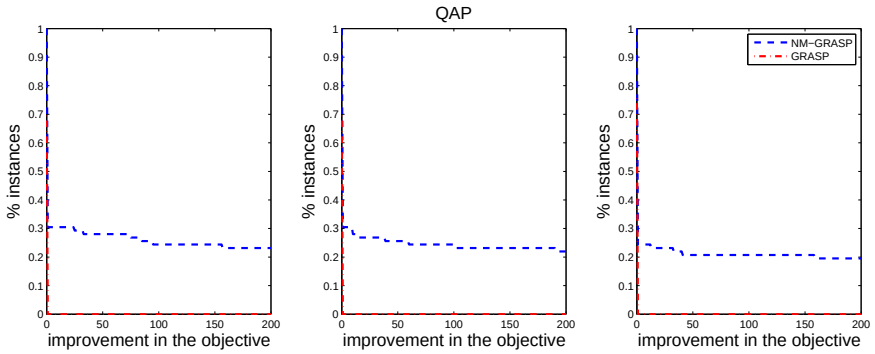

Fig. 7 Performance comparison between NM-GRASP (dashed line) and the original GRASP (dash-dot line) on quadratic assignment problems (Worst-left; Average-center; Best-right).

dot line gives on the y-axis the percentage of instances in which the absolute improvement of GRASP with respect to NM-GRASP is greater or equal than the value given in the x-axis. More specifically, let f_i^{NM} and f_i^{OR} be the objective function values found by NM-GRASP and the classical GRASP on instance i, respectively. Let N be the total number of instances considered for each problem. The dashed line for the MAX-CUT and the MAX-SAT is the plot of

$$y(x) = \frac{|V(x)|}{N}, \tag{9}$$

where $V(x) = \{i : f_i^{NM} - f_i^{OR} \geq x\}$. The dash-dot line for the MAX-CUT and the MAX-SAT is the plot of

$$y(x) = \frac{|W(x)|}{N}, \tag{10}$$

where $W(x) = \{i : \ f_i^{OR} - f_i^{NM} \geq x\}$. For what concerns the QAP, since it is a minimization problem, we have that the dashed line is the plot of (10) while the dash-dot line is the plot of (9).

In every figure, we report: 1) on the left, the plot related to the worst objective function value obtained among the ten runs; 2) on the center, the one related to the average of the objective function value obtained on the ten runs, and 3) on the right, the plot related to the best objective function value obtained among the ten runs by the two heuristics.

On the basis of the figures, we notice that NM-GRASP is generally able to guarantee better performances than GRASP in all three scenarios (worst, average, and best scenario).

To deeper investigate and confirm the better performance of NM-GRASP, Table 1, Table 2, and Table 3 summarize the details of the results obtained by comparing the two algorithms on the benchmark instances of the three selected combinatorial optimization problems. The first column of the three tables reports the name of the instance. The remaining columns report for each of the two approaches the average CPU time (Time), the average number of iterations (Iter), the average objective function value (Obj) with the standard deviation in brackets, and the best objective function value obtained over the ten runs (Best Obj) with the number of times the best value is obtained in brackets.

As for the 117 benchmark instances of the MAX-CUT, the 44 instances of the MAX-SAT, and the 82 instances of the QAP, we notice that the NM-GRASP found solutions whose objective function value is better than or equal to the objective function value of the solution found by the classical GRASP (often strictly better) for the great majority of problems. Moreover, the number of times NM-GRASP found the best objective function value over only ten runs is higher for all instances for the QAP, and for almost all instances except for a very small percentage of cases that is below 5% for the MAX-CUT and below 11% for the MAX-SAT.

In order to see if there exist significant differences in the results, in terms of solution quality, between NM-GRASP and the original GRASP, we apply the Friedman non-parametric statistical test followed by the post-hoc test on the results from the tables. The post-hoc analysis shows that NM-GRASP statistically outperforms the original GRASP on both MAX-CUT and the QAP instances with p-values of 1.05973563e-10 and 3.84942067e-09, respectively. The performance between the two algorithms is statistically less significant for the MAX-SAT, with a p-value of 6.47928279e-02.

As additional comparison between GRASP and NM-GRASP, we consider their performance with respect to the average CPU time. We recall that, for each run, we stored the CPU time needed to find the solution with the best objective function value in that run. In Figure 8, we report the box plots related to the distribution of the average CPU time over the ten runs for each problem (MAX-CUT on the left, MAX-SAT in the center and QAP on the right). On each box, the central mark is the median, the edges of the box are the 25th

and 75th percentiles, the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. From each plots, we see that the median of the NM-GRASP is lower than the median of the classical GRASP. For the QAP problem, the NM-GRASP find its best solutions generally much faster than the classical GRASP.

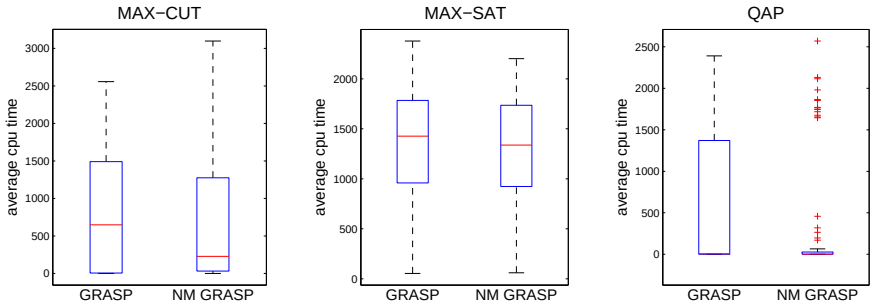

Fig. 8 Box plots related to the CPU time (average results over ten runs).

As a further experiment for the MAX-CUT problem, we considered the empirical distributions of the random variable *time-to-target-solution-value* (see [1,2] for further details) considering instances `g1250.n`, `G40`, `sg3dl142000.mc`, and `toruspm3-15-50` using different target values. We performed 100 independent runs of each heuristic and recorded the time needed to find a solution at least as good as the target solution. For each run, we considered one hour as the time limit. The Time-To-Target-plot analysis is reported in Appendix B.

Summarizing, our computational experience shows that considering a nonmonotone local search in the GRASP heuristic often gives a significant improvement in the quality of the solution, and this improvement is achieved without deteriorating the CPU time.

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 three selected hard combinatorial optimization problems: the MAX-CUT, the MAX-SAT, and the QAP. The comparison showed that the new proposed approach is very competitive, outperforming the classical GRASP.

A Detailed Tables for MAX-CUT, MAX-SAT and QAP

In this appendix, we report the detailed tables related to the comparison between NM-GRASP and the original version of GRASP for MAX-CUT, MAX-SAT and QAP. The first column of the three tables reports the name of the instance. The remaining columns report for each of the two approaches the average CPU time (Time), the average number of iterations (Iter), the average objective function value (Obj) with the standard deviation in brackets, and the best objective function value obtained over the ten runs (Best Obj) with the number of times the best value is obtained in brackets.

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

Problem	**GRASP**				**NM-GRASP**			
	Time	Iter	Obj	Best Obj	Time	Iter	Obj	Best Obj
g10.n	$< 10^{-3}$	1	17(0)	17(10)	10^{-3}	1	17(0)	17(10)
g20.n	$< 10^{-3}$	1.3	37(0)	37(10)	10^{-3}	1	37(0)	37(10)
g25.n	$< 10^{-3}$	1	42(0)	42(10)	10^{-3}	1	42(0)	42(10)
g30.n	$< 10^{-3}$	1.5	61(0)	61(10)	10^{-3}	1	61(0)	61(10)
g50.n	$< 10^{-3}$	1.2	105(0)	105(10)	10^{-3}	1	105(0)	105(10)
g100.n	$< 10^{-3}$	4.4	214(0)	214(10)	0.03	1	214(0)	214(10)
g150.n	0.06	7.5	294(0)	294(10)	0.05	1	294(0)	294(10)
g200.n	1.46	98.3	405(0)	405(10)	0.28	5	405(0)	405(10)
g250.n	6.19	286.2	305(0)	305(10)	3.39	33	305(0)	305(10)
g500.n	922.66	13643.1	573.5(0.53)	574(5)	320.58	818	573(0)	573(10)
g1000.n	828.69	3056	1704.3(1.16)	1706(1)	102.88	63	1708(0)	1708(10)
g1250.n	1848.67	1740.5	2546.1(3.38)	2551(1)	2865.98	507	2555(0)	2555(10)
sg3dl051000.mc	0.02	4.6	110(0)	110(10)	0.03	3	110(0)	110(10)
sg3dl052000.mc	0.03	7	112(0)	112(10)	0.25	16	112(0)	112(10)
sg3dl053000.mc	0.01	3.3	106(0)	106(10)	0.02	1	106(0)	106(10)
sg3dl054000.mc	$< 10^{-3}$	2.8	114(0)	114(10)	0.05	4	114(0)	114(10)
sg3dl055000.mc	$< 10^{-3}$	3.4	112(0)	112(10)	0.09	6	112(0)	112(10)
sg3dl056000.mc	0.01	4.4	110(0)	110(10)	10^{-3}	1	110(0)	110(10)
sg3dl057000.mc	0.03	8.3	112(0)	112(10)	0.05	4	112(0)	112(10)
sg3dl058000.mc	$< 10^{-3}$	2.5	108(0)	108(10)	0.01	1	108(0)	108(10)
sg3dl059000.mc	$< 10^{-3}$	2.9	110(0)	110(10)	0.01	1	110(0)	110(10)
sg3dl101000.mc	96.01	4368.9	890.4(3.10)	894(3)	1368.74	1772.3	891(1.05)	892(5)
sg3dl102000.mc	118.33	3092.1	898.6(1.35)	900(4)	1112.46	1451.2	894(0)	894(10)
sg3dl103000.mc	686.75	5064.4	885(2.87)	890(2)	1288.65	1028.4	886(0)	886(10)
sg3dl104000.mc	145.84	3043.4	896(1.33)	898(2)	95.67	283.8	898(0)	898(10)
sg3dl105000.mc	648.32	3363.4	881(1.70)	882(7)	180.24	261.2	882.2(0.63)	884(1)
sg3dl106000.mc	142.68	4354.6	884.6(2.84)	888(3)	448.53	413.8	880.2(0.63)	882(1)
sg3dl107000.mc	37.83	2538.7	895.8(2.20)	900(1)	537.59	584.2	892.2(0.63)	894(1)
sg3dl108000.mc	164.99	2334	879.6(1.58)	882(2)	4.66	456.4	880.2(0.63)	882(1)

continued on next page

Table 1 – continued from previous page

Problem	GRASP				NM-GRASP			
	Time	Iter	Obj	Best Obj	Time	Iter	Obj	Best Obj
sg3dl109000.mc	369.95	4004.9	896.2(1.48)	900(1)	267.67	579	898(0)	898(10)
sg3dl141000.mc	757.58	705.1	2395.4(4.12)	2402(1)	550.67	196.1	2414.4(1.58)	2416(4)
sg3dl142000.mc	1014.72	704.3	2403.4(3.89)	2410(1)	220.4	22	2426(0)	2426(10)
sg3dl143000.mc	1222.57	520.6	2394.4(4.70)	2400(1)	938.15	231.6	2413.8(2.39)	2418(1)
sg3dl144000.mc	1139.55	744.9	2398.8(2.70)	2404(1)	13.32	97.1	2419.4(1.35)	2422(1)
sg3dl145000.mc	1245.02	573.9	2392(4.90)	2400(2)	659.22	83.5	2410.2(0.63)	2412(1)
sg3dl146000.mc	856.59	577.6	2402.6(3.78)	2410(1)	1064.98	125.8	2417(3.16)	2426(1)
sg3dl147000.mc	824.15	733.6	2387.6(3.98)	2398(1)	272.13	176.6	2414(2.49)	2418(2)
sg3dl148000.mc	1699.91	611	2396.4(5.23)	2404(2)	1373.20	121	2426.8(1.03)	2428(4)
sg3dl149000.mc	440.21	749.7	2374.6(4.72)	2384(1)	1536.99	161	2398.8(1.03)	2400(4)
sg3dl0510000.mc	0.02	5.1	112(0)	112(10)	0.02	4	112(0)	112(10)
sg3dl1010000.mc	166.29	4875.4	890.2(1.99)	892(5)	193.73	255.5	890(0)	890(10)
sg3dl1410000.mc	607.91	726.3	2408.6(7.18)	2420(1)	943.40	134.6	2426.4(0.84)	2428(2)
torusg3-8.dat	0.01	1	35322378.4(> 100)	36589003(1)	0.23	1	40200626(0)	40200626(10)
torusg3-15.dat	0.8	1	244427176.8(> 100)	254963285(1)	11.87	1	276413144(0)	276413144(10)
toruspm3-8-50.dat	296.99	6969.9	457.2(1.03)	458(6)	83.10	2058.9	456.4(0.84)	458(2)
toruspm3-15-50.dat	799.81	299.6	2950.2(5.12)	2960(1)	570.56	34	2974(0)	2974(10)
B1	11.96	2	10000(0)	10000(10)	37.05	1	10000(0)	10000(10)
B2	16.08	1.6	12000(0)	12000(10)	54.58	1	12000(0)	12000(10)
B3	23.46	1.8	14000(0)	14000(10)	53.54	2	13860(0)	13860(10)
B4	33.48	1.9	16000(0)	16000(10)	111.12	1	16000(0)	16000(10)
G1	2027.46	3002.4	11610.2(5.59)	11619(1)	2212.27	637	11624(0)	11624(10)
G2	1291.09	2155.4	11602.8(7.96)	11620(1)	1568.53	410	11620(0)	11620(10)
G3	2491.38	4004.8	11605.9(5.67)	11613(1)	105.00	29	11622(0)	11622(10)
G4	1769.57	2638.2	11636.5(5.46)	11646(1)	228.21	67	11646(0)	11646(10)
G5	2338.39	3517	11617(5.75)	11627(1)	2930.24	916	11631(0)	11631(10)
G6	1599.98	2556.7	2161.1(5.63)	2174(1)	49.65	15	2178(0)	2178(10)
G7	1769.32	2602.7	1994.5(5.02)	2001(1)	426.72	114	2000(0)	2000(10)
G8	1807.99	2672.7	1991.4(4.58)	1996(2)	212.22	68	2005(0)	2005(10)
G9	1850.68	2707.9	2037.3(8.17)	2048(1)	145.90	43	2054(0)	2054(10)
G10	1460.44	2236.4	1990.7(5.27)	1999(1)	262.57	80	2000(0)	2000(10)

continued on next page

Table 1 – continued from previous page

Problem	**GRASP**				**NM-GRASP**			
	Time	Iter	Obj	Best Obj	Time	Iter	Obj	Best Obj
G11	0.28	495.4	564(0)	564(10)	0.80	62	564(0)	564(10)
G12	1.98	2652.1	554.8(1.03)	556(4)	2.05	569.5	556(0)	556(10)
G13	2.91	2950.5	580.6(1.35)	582(4)	86.63	504.9	581.6(0.84)	582(8)
G14	587.78	6033.4	3052.1(3.60)	3059(1)	823.84	462	3058(0)	3058(10)
G15	155.31	4884.1	3040(4)	3045(1)	516.06	282	3039(0)	3039(10)
G16	784.78	4703.3	3039(3.71)	3048(1)	1473.27	954	3050.2(0.42)	3051(2)
G17	542.66	5171.1	3034.8(2.82)	3039(2)	194.13	764.3	3038(0)	3038(10)
G18	210.25	4614	987.2(2.39)	991(1)	1623.37	780	988(0)	988(10)
G19	672.3	4417.1	898.7(3.06)	905(1)	1362.69	960	903.3(0.48)	904(3)
G20	359.49	3125.5	940(1.89)	941(7)	3098.75	1440	927(0)	927(10)
G21	91.02	3451.8	925.3(5.14)	931(3)	63.31	36	921(0)	921(10)
G22	1643.63	396.9	13242.8(11.04)	13259(1)	647.59	33	13289(0)	13289(10)
G23	1795.59	384.7	13247.5(11.22)	13267(1)	1731.60	94	13317(0)	13317(10)
G24	1325.41	341.6	13250.8(11.23)	13271(1)	1380.06	73	13303(0)	13303(10)
G25	1490.15	282.4	13248.4(12.48)	13270(1)	502.05	30	13322(0)	13322(10)
G26	2187.87	415.7	13229.6(9.36)	13249(1)	44.91	50.8	13293(0)	13293(10)
G27	1184.28	267.6	3239.3(8.90)	3251(1)	1902.83	104	3306(0)	3306(10)
G28	1412.62	267.3	3199.9(8.44)	3214(1)	1316.64	67	3282(0)	3282(10)
G29	1533.36	283	3297.2(7.71)	3306(2)	1130.35	66	3404(0)	3404(10)
G30	1747.62	324	3314.4(6.17)	3324(1)	2764.04	147	3388(0)	3388(10)
G31	2006.52	387.6	3206.6(7.31)	3218(1)	82.56	7	3277(0)	3277(10)
G32	2.59	1182.6	1396.4(2.46)	1402(1)	3.56	296.5	1401.2(1.40)	1404(1)
G33	2.94	1588.2	1370(3.53)	1376(1)	6.44	233.9	1371.8(1.99)	1376(1)
G34	4.04	849	1371(2.71)	1376(1)	54.36	509.7	1377.4(1.65)	1380(1)
G35	1579.03	617.4	7608.5(5.10)	7617(1)	1269.66	92	7639(0)	7639(10)
G36	2063.38	639.3	7597.5(4.14)	7606(1)	168.15	15	7637(0)	7637(10)
G37	708.69	699.4	7610.6(6.24)	7620(1)	1313.85	93	7624(0)	7624(10)
G38	1835.19	674.5	7612.2(4.66)	7619(1)	2422.95	194	7644(0)	7644(10)
G39	1436.41	756.9	2326.1(13.14)	2358(1)	1750.33	99	2356(0)	2356(10)
G40	1028.03	742.5	2319.2(8.92)	2330(1)	2817.61	176	2366(0)	2366(10)
G41	1149.98	637.8	2318.7(6.57)	2328(2)	1554.23	85	2352(0)	2352(10)

continued on next page

Table 1 – continued from previous page

Problem	**GRASP**				**NM-GRASP**			
	Time	Iter	Obj	Best Obj	Time	Iter	Obj	Best Obj
G42	1039.49	752.1	2400.2(8.59)	2413(1)	1825.54	98	2446(0)	2446(10)
G43	1607.26	1831.3	6643.1(4.61)	6651(1)	1068.16	252	6656(0)	6656(10)
G44	2557.72	2771.7	6630(2.49)	6634(1)	2647.95	638	6649(0)	6649(10)
G45	1491.39	1616.1	6634.4(5.27)	6646(1)	94.59	24	6647(0)	6647(10)
G46	1998.32	2600.7	6633.9(6.15)	6649(1)	81.82	19	6647(0)	6647(10)
G47	1820.72	2167	6639.6(8.03)	6653(1)	61.08	17	6655(0)	6655(10)
G48	3.09	1.4	6000(0)	6000(10)	12.96	1	6000(0)	6000(10)
G49	5.4	3.7	6000(0)	6000(10)	8.35	2	5940(0)	5940(10)
G50	2.46	17.8	5880(0)	5880(10)	45.31	2	5880(0)	5880(10)
G51	770.93	4350	3828.1(4.75)	3835(1)	1506.60	551	3834(0)	3834(10)
G52	399.11	4160.1	3832.8(3.36)	3839(1)	1032.60	359	3835(0)	3835(10)
G53	500.85	3859.9	3831.5(5.78)	3847(1)	2769.37	937	3836(0)	3836(10)
G54	746.42	4137.7	3826.3(2.31)	3830(2)	388.44	136.4	3836(0)	3836(10)
G55	1412.81	51.3	10091.8(12.05)	10113(1)	2865.13	20	10168(0)	10168(10)
G56	1778.39	48	3792.8(11.01)	3816(1)	357.66	1	3930(0)	3930(10)
G57	35.05	268.7	3419.8(4.66)	3432(1)	105.50	62.8	3439.4(2.50)	3444(1)
G58	1688.03	69.2	19024.8(9.98)	19041(1)	1107.63	11	19122(0)	19122(10)
G59	1690.64	78	5788(25.69)	5826(2)	2123.25	20	5959(0)	5959(10)
G60	1901.51	26.8	13875.5(9.41)	13892(1)	948.27	5	14051(0)	14051(10)
G61	1619.66	21.9	5453.9(11.38)	5476(1)	374.96	3	5627(0)	5627(10)
G62	24.22	127.4	4757(4.74)	4764(1)	211.17	26.6	4788.6(1.90)	4792(1)
G63	756.44	31.8	26643(16.06)	26673(1)	976.63	4	26786(0)	26786(10)
G64	1201.17	28.6	8248.7(24.78)	8284(1)	1411.91	4	8482(0)	8482(10)
G65	30.3	100.3	5427.2(9.99)	5446(1)	61.65	17	5470.4(3.63)	5476(2)
G66	30.44	70.5	6193.2(5.90)	6202(1)	133.21	29.6	6252(0.94)	6254(1)
G67	44.43	38.2	6768.2(6.56)	6778(1)	322.77	16	6823(3.56)	6826(4)

Table 2: Comparison between NM-GRASP and the classical GRASP on MAX-SAT instances (average results over ten runs).

Problem	**GRASP**				**NM-GRASP**			
	Time	Iter	Obj	Best Obj	Time	Iter	Obj	Best Obj
jnh1.sat	1328.3	142441	420877.7(25.68)	420909(2)	1326.92	108931.6	420882(22.26)	420909(1)
jnh4.sat	1353.85	136301.3	420778.5(5.82)	420789(1)	1810.57	144416.1	420785.4(4.65)	420789(6)
jnh5.sat	719.09	66549.1	420663.8(52.43)	420742(2)	1008.64	73242.4	420742(0)	420742(10)
jnh6.sat	379.3	36290.1	420826(0)	420826(10)	260.3	19562.3	420826(0)	420826(10)
jnh7.sat	70.28	6887.3	420925(0)	420925(10)	60.11	4669.6	420925(0)	420925(10)
jnh8.sat	1532.76	139218.2	420345.9(88.72)	420463(3)	2006.08	143432.5	420378(76.95)	420463(4)
jnh9.sat	2177.85	223394.3	420395.1(65.75)	420505(1)	1870.48	148042.7	420446(62.32)	420522(2)
jnh10.sat	1114.76	100633.3	420699.2(71.31)	420758(5)	1737.95	123013.5	420767.7(60.04)	420840(3)
jnh11.sat	1584.34	158770.2	420664.2(33.01)	420728(1)	1637.46	128420.3	420695.6(42.36)	420740(3)
jnh12.sat	720.94	70902.8	420886.2(24.51)	420925(1)	686.21	52092.2	420886.2(24.51)	420925(1)
jnh13.sat	1222.77	113488.8	420805.2(9.30)	420816(4)	1349.27	98403.2	420810.6(8.69)	420816(7)
jnh14.sat	1730.13	169101.2	420740.4(42.45)	420824(1)	1682.6	130176.1	420744.9(40.37)	420824(1)
jnh15.sat	1549.57	144399.3	420654.7(41.11)	420719(2)	1252.27	91906	420683.3(39.50)	420719(5)
jnh16.sat	983.73	97357.6	420900.7(13.50)	420914(4)	1781.32	136140.1	420905.1(11.38)	420914(4)
jnh17.sat	1622.75	164583.5	420918.1(11.11)	420925(7)	1287.66	102744.7	420925(0)	420925(10)
jnh18.sat	1898.91	174732.2	420716.8(67.24)	420795(3)	864.33	62599.7	420779(37.97)	420795(8)
jnh19.sat	1304.29	130933.3	420591(72.67)	420759(1)	1190.26	94283.7	420609.2(74.37)	420759(1)
jnh201.sat	116.17	14155.6	394238(0)	394238(10)	80.55	7428.8	394238(0)	394238(10)
jnh202.sat	1153	127764.7	394023.8(34.92)	394099(1)	1188.47	103056.7	394024.7(44.01)	394100(1)
jnh203.sat	1839.14	184293.7	394103.3(34.08)	394135(2)	2031.74	160281.2	394122.7(11.93)	394135(4)
jnh205.sat	1979.56	210258.2	394233.4(6.96)	394238(5)	2154.06	181150.8	394236.4(4.38)	394238(7)
jnh207.sat	1576.35	177935.5	394210.7(24.79)	394238(1)	1510.65	133834.2	394213.9(24.78)	394238(1)
jnh208.sat	1702.74	176684.2	394104.1(19.29)	394159(1)	1811.67	147571.9	394104.1(19.29)	394159(1)
jnh209.sat	1832.04	202283.3	394216.7(18.02)	394238(3)	1361.69	117263.8	394212.8(17.10)	394238(2)
jnh210.sat	53.46	6181.5	394238(0)	394238(10)	61.66	5539.4	394238(0)	394238(10)
jnh211.sat	2037.73	201069.7	393884.9(48.88)	393954(3)	1804.86	141947.5	393897.8(52.47)	393979(1)
jnh212.sat	1318.64	145918.8	394209.6(26.93)	394227(6)	1561.99	134291.2	394209.3(27.13)	394227(6)
jnh214.sat	1734.67	183986.4	394137.3(19.92)	394163(2)	1364.32	113672.1	394154.9(14.59)	394163(7)
jnh215.sat	1742.73	185872.8	394030(44.37)	394091(2)	1826.16	154193.6	394037.4(40.97)	394091(2)

continued on next page

Table 2 – continued from previous page

Problem	GRASP				NM-GRASP			
	Time	Iter	Obj	Best Obj	Time	Iter	Obj	Best Obj
jnh216.sat	316.95	31514.3	394156.5(27.91)	394226(1)	1145.18	90451.2	394185(43.22)	394226(5)
jnh217.sat	819.28	93640.7	394238(0)	394238(10)	558.24	49253.8	394238(0)	394238(10)
jnh218.sat	935.46	102156.5	394237.4(1.90)	394238(9)	962.95	81950.3	394238(0)	394238(10)
jnh219.sat	2378.6	249815.8	393937.3(93.33)	394111(1)	1627.03	134223.1	393966.3(79.15)	394111(1)
jnh220.sat	1456.98	167317.4	394188.4(29.67)	394238(1)	1489.63	132691.9	394190.7(32.01)	394238(1)
jnh301.sat	780.52	66748.3	444792.2(8.28)	444807(1)	884.31	59613.6	444805.6(24.32)	444854(1)
jnh302.sat	1431.94	125142.8	444398.1(63.37)	444459(4)	1168.27	79954.4	444437.4(45.54)	444459(8)
jnh303.sat	1967.02	160225.3	444372.9(47.93)	444503(1)	1119.74	72104.4	444366(0)	444366(10)
jnh304.sat	1423.02	124770	444507(44.74)	444533(7)	1281.37	89894.4	444491.6(48.51)	444533(5)
jnh305.sat	1027.68	85927.6	443985.8(105.90)	444112(3)	725.16	47584.1	444035.8(66.72)	444112(4)
jnh306.sat	2367.91	223524	444802(31.83)	444838(4)	2202.9	161105.4	444808.5(31.39)	444838(5)
jnh307.sat	1827.27	158754.1	444305.5(16.67)	444314(7)	778.54	53181.8	444293.5(20.61)	444314(4)
jnh308.sat	1379.2	118159.7	444530.5(56.21)	444568(4)	1733.12	116605.2	444538.8(36.22)	444568(4)
jnh309.sat	572.28	49806.7	444578(0)	444578(10)	292.8	19910.3	444578(0)	444578(10)
jnh310.sat	1941.09	161933.4	444343.4(33.50)	444391(3)	1619.11	106253.2	444378.8(25.72)	444391(8)

Table 3: Comparison between NM-GRASP and the classical GRASP on QAP instances (average results over ten runs).

Problem	**GRASP**				**NM-GRASP**			
	Time	Iter	Obj	Best Obj	Time	Iter	Obj	Best Obj
chr12a.dat	0.02	135	9552(0)	9552(10)	0.02	7.8	9552(0)	9552(10)
chr12b.dat	$< 10^{-3}$	25.9	9742(0)	9742(10)	$< 10^{-3}$	2.1	9742(0)	9742(10)
chr12c.dat	0.12	693	11156(0)	11156(10)	0.11	38.1	11156(0)	11156(10)
chr15a.dat	0.59	2045.3	9896(0)	9896(10)	0.52	109.2	9896(0)	9896(10)
chr15b.dat	0.26	800.7	7990(0)	7990(10)	0.16	27	7990(0)	7990(10)
chr15c.dat	1.1	4015.8	9504(0)	9504(10)	0.49	98.7	9504(0)	9504(10)
chr18a.dat	8.73	18842.6	11098(0)	11098(10)	0.28	24.4	11098(0)	11098(10)
chr18b.dat	0.19	328.5	1534(0)	1534(10)	0.08	10.3	1534(0)	1534(10)
chr20a.dat	64.08	98401.6	2192(0)	2192(10)	8.74	972.8	2192(0)	2192(10)
chr20b.dat	1012.67	1574763.6	2298(0)	2298(10)	27.88	2727.8	2298(0)	2298(10)
chr20c.dat	1.6	2282	14142(0)	14142(10)	1.15	121.1	14142(0)	14142(10)
chr22a.dat	391.24	436752	6156(0)	6156(10)	2.79	205.8	6156(0)	6156(10)
chr22b.dat	1218.59	1338050.7	6194(0)	6194(10)	11.95	803.1	6194(0)	6194(10)
chr25a.dat	204.32	144587.4	3796(0)	3796(10)	9.46	479.5	3796(0)	3796(10)
els19.dat	0.33	395.9	17212548(0)	17212548(10)	0.79	113.4	17212548(0)	17212548(10)
esc16a.dat	$< 10^{-3}$	2.7	68(0)	68(10)	$< 10^{-3}$	1.5	68(0)	68(10)
esc16b.dat	0	1	292(0)	292(10)	$< 10^{-3}$	1	292(0)	292(10)
esc16c.dat	$< 10^{-3}$	2.1	160(0)	160(10)	$< 10^{-3}$	1.3	160(0)	160(10)
esc16d.dat	$< 10^{-3}$	2	16(0)	16(10)	$< 10^{-3}$	1.4	16(0)	16(10)
esc16e.dat	$< 10^{-3}$	3.6	28(0)	28(10)	$< 10^{-3}$	3.2	28(0)	28(10)
esc16f.dat	$< 10^{-3}$	1	0(0)	0(10)	$< 10^{-3}$	1	0(0)	0(10)
esc16g.dat	$< 10^{-3}$	2.5	26(0)	26(10)	$< 10^{-3}$	1.7	26(0)	26(10)
esc16h.dat	$< 10^{-3}$	1	996(0)	996(10)	$< 10^{-3}$	1	996(0)	996(10)
esc16i.dat	$< 10^{-3}$	1.1	14(0)	14(10)	$< 10^{-3}$	1	14(0)	14(10)
esc16j.dat	$< 10^{-3}$	1.7	8(0)	8(10)	$< 10^{-3}$	1.5	8(0)	8(10)
esc32a.dat	114.39	39049.4	130(0)	130(10)	5.31	857	130(0)	130(10)
esc32b.dat	0.75	217.6	168(0)	168(10)	0.37	39.6	168(0)	168(10)
esc32c.dat	$< 10^{-3}$	1.8	642(0)	642(10)	$< 10^{-3}$	1.2	642(0)	642(10)
esc32d.dat	0.1	25.7	200(0)	200(10)	0.07	9.3	200(0)	200(10)

continued on next page

Table 3 – continued from previous page

Problem	**GRASP**				**NM-GRASP**			
	Time	Iter	Obj	Best Obj	Time	Iter	Obj	Best Obj
esc32e.dat	$< 10^{-3}$	1	2(0)	2(10)	$< 10^{-3}$	1	2(0)	2(10)
esc32g.dat	$< 10^{-3}$	1	6(0)	6(10)	$< 10^{-3}$	1	6(0)	6(10)
esc32h.dat	0.42	123.6	438(0)	438(10)	0.12	21.9	438(0)	438(10)
esc64a.dat	0.12	2.9	116(0)	116(10)	0.14	3.3	116(0)	116(10)
esc128.dat	6.58	19.5	64(0)	64(10)	5.06	14.4	64(0)	64(10)
kra30a.dat	15.03	5456.4	88900(0)	88900(10)	0.98	55.6	88900(0)	88900(10)
kra30b.dat	105.81	38123.1	91420(0)	91420(10)	2.62	154.9	91420(0)	91420(10)
lipa20a.dat	0.26	303	3683(0)	3683(10)	0.07	5.6	3683(0)	3683(10)
lipa20b.dat	0.01	12.8	27076(0)	27076(10)	0.03	1.1	27076(0)	27076(10)
lipa30a.dat	6.64	2389.6	13178(0)	13178(10)	0.34	7.9	13178(0)	13178(10)
lipa30b.dat	0.1	23.7	151426(0)	151426(10)	0.15	1.4	151426(0)	151426(10)
lipa40a.dat	1234.74	169235.3	31547.4(29.73)	31538(9)	1.58	12.4	31538(0)	31538(10)
lipa40b.dat	0.35	33.4	476581(0)	476581(10)	0.24	1	476581(0)	476581(10)
lipa50a.dat	1843.91	114771.4	62618.4(27.18)	62572(1)	4.42	15.1	62093(0)	62093(10)
lipa50b.dat	1.44	73.9	1210244(0)	1210244(10)	0.88	1.8	1210244(0)	1210244(10)
lipa60a.dat	1428.77	47338.4	108111.8(15.46)	108076(1)	64.76	119.7	107218(0)	107218(10)
lipa60b.dat	12.14	362.9	2520135(0)	2520135(10)	2.12	2.6	2520135(0)	2520135(10)
lipa70a.dat	1367.48	25895.6	171017.9(30.47)	170950(1)	196.54	209.2	169755(0)	169755(10)
lipa70b.dat	16.11	273.4	4603200(0)	4603200(10)	3.37	2.6	4603200(0)	4603200(10)
lipa80a.dat	1636.88	19263.4	254887.8(23.05)	254845(1)	1718.03	1027.4	253942.6(647.10)	253195(4)
lipa80b.dat	171.88	1811.4	7763962(0)	7763962(10)	11.46	5.1	7763962(0)	7763962(10)
lipa90a.dat	1849.8	14190.6	362912.4(31.84)	362840(1)	1865.28	739.6	361812.2(816.67)	360630(3)
lipa90b.dat	287.74	1970.3	12490441(0)	12490441(10)	12.22	3.8	12490441(0)	12490441(10)
nug12.dat	$< 10^{-3}$	72	578(0)	578(10)	$< 10^{-3}$	3.7	578(0)	578(10)
nug15.dat	0.01	35.1	1150(0)	1150(10)	0.05	11.2	1150(0)	1150(10)
nug20.dat	0.35	420	2570(0)	2570(10)	0.15	10.6	2570(0)	2570(10)
nug30.dat	359.59	121705.1	6124(0)	6124(10)	1.11	27.8	6124(0)	6124(10)
rou12.dat	0.02	118.7	235528(0)	235528(10)	0.02	4.9	235528(0)	235528(10)
rou15.dat	0.086	222.7	354210(0)	354210(10)	0.06	7.7	354210(0)	354210(10)
rou20.dat	5.09	7049.4	725522(0)	725522(10)	3	195.9	725522(0)	725522(10)
scr12.dat	$< 10^{-3}$	17.3	31410(0)	31410(10)	0.01	3.4	31410(0)	31410(10)

continued on next page

Table 3 – continued from previous page

Problem	GRASP				NM-GRASP			
	Time	Iter	Obj	Best Obj	Time	Iter	Obj	Best Obj
scr15.dat	0.04	97.8	51140(0)	51140(10)	0.03	4.4	51140(0)	51140(10)
scr20.dat	1.58	2060	110030(0)	110030(10)	0.27	16.4	110030(0)	110030(10)
sko42.dat	1317.99	137418	15821.2(11.75)	15812(3)	5.19	44.6	15812(0)	15812(10)
sko49.dat	2077.55	124989.8	23445.4(12.19)	23426(1)	318.15	1744.7	23386(0)	23386(10)
sko56.dat	1505.45	56054.1	34559.4(32.39)	34490(1)	458.64	1539.2	34458(0)	34458(10)
sko64.dat	2111.54	48512.6	48688.4(25.83)	48656(1)	264.69	577.1	48498(0)	48498(10)
sko72.dat	1601.34	24047.8	66597(30.26)	66552(1)	1764.75	2609	66260.2(3.33)	66256(2)
sko81.dat	1613.38	15695.3	91469.2(63.12)	91374(1)	2570.36	2545.1	91012.8(12.73)	90998(2)
sko90.dat	1310.89	8720	116241.4(91.13)	116082(1)	1981.96	1391.9	115562.8(17.21)	115534(1)
sko100a.dat	1803.86	8085.8	152828.6(85.59)	152708(1)	2131.23	1038.7	152050.8(35.35)	152002(3)
sko100b.dat	2391.59	10723.6	154699(129.49)	154494(1)	1853.61	894.8	153914.8(19.28)	153890(2)
sko100c.dat	1918.08	8569.3	148653.2(130.13)	148500(1)	1769.97	863.7	147883.2(15)	147862(1)
sko100d.dat	1803.04	8118.8	150522.8(129.14)	150292(1)	1676.29	811	149607(17.37)	149576(1)
sko100e.dat	1767.35	7878.1	150084.2(139.80)	149856(1)	1655.6	798.8	149166.4(11.81)	149150(2)
sko100f.dat	1448.53	6558	150044.6(112.92)	149802(1)	2116.36	1035	149093(39.26)	149036(1)
ste36a.dat	1373.93	250878.7	9540(10.28)	9526(2)	16.45	276.8	9526(0)	9526(10)
ste36b.dat	168.29	29626.6	15852(0)	15852(10)	1.2	16.4	15852(0)	15852(10)
ste36c.dat	1370.83	246808.8	8247337.2(6728.75)	8239110(2)	3.76	50.8	8239110(0)	8239110(10)
tho40.dat	2347.46	291488.4	240749.6(246.39)	240516(1)	169.75	1230.6	240516(0)	240516(10)
tho150.dat	1829.53	1806.5	8202955.2(9233.59)	8191890(1)	1745.69	147.2	8140210.6(3253.11)	8133398(1)
wil50.dat	2095.1	116930.9	48854.2(19.79)	48828(1)	59.05	315.3	48816(0)	48816(10)
wil100.dat	1978.65	8924.6	273908.6(114.40)	273748(1)	1646.01	836.4	273075.2(18.48)	273038(1)

B Time To Target-Plots analysis on MAX-CUT problems

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 9, 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 10 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`, `sg3-dl142000.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 11 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`.

References

1. R. M. Aiex, M. G. C. Resende, and C. C. Ribeiro. Probability distribution of solution time in grasp: An experimental investigation. *Journal of Heuristics*, 8:343–373, 2002.
2. R. M. Aiex, M. G. C. Resende, and C. C. Ribeiro. Ttt plots: a perl program to create time-to-target plots. *Optimization Letters*, 1:355–366, 2007.
3. R. Alvarez-Valdes, F. Parreño, and J.M. Tamarit. Reactive GRASP for the strip-packing problem. *Computers & Operations Research*, 35(4):1065–1083, 2008.
4. D.V. Andrade and M.G.C. Resende. GRASP with path-relinking for network migration scheduling. In *Proceedings of the International Network Optimization Conference (INOC 2007)*, 2007.
5. C. Andres, C. Miralles, and R. Pastor. Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. *European J. of Operational Research*, 187(3):1212–1223, 2008.
6. S. Areibi and A. Vannelli. A GRASP clustering technique for circuit partitioning. In J. Gu and P.M. Pardalos, editors, *Satisfiability problems*, volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 711–724. American Mathematical Society, 1997.
7. J.E.C. Arroyo, P.S. Vieira, and D.S. Vianna. A GRASP algorithm for the multi-criteria minimum spanning tree problem. *Annals of Operations Research*, 159:125–133, 2008.
8. 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.
9. J.B. Atkinson. A greedy randomised search heuristic for time-constrained vehicle scheduling and the incorporation of a learning strategy. *J. of the Operational Research Society*, 49:700–708, 1998.
10. F. Barahona. The max-cut problem in graphs not contractible to k_5. *Operations Research Letters*, 2:107–111, 1983.
11. J.F. Bard, L. Huang, P. Jaillet, and M. Dror. A decomposition approach to the inventory routing problem with satellite facilities. *Transportation Science*, 32:189–203, 1998.

12. 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.
13. U. Benlic and J.-K. Hao. Breakout local search for maximum clique problems. *Computers & Operations Research*, 40(1):192–206, 2013.
14. S. Binato, W.J. Hery, D. Loewenstern, and M.G.C. Resende. A greedy randomized adaptive search procedure for job shop scheduling. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys on metaheuristics*, pages 58–79. Kluwer Academic Publishers, 2002.
15. M.J. Brusco and S. Stahl. Using quadratic assignment methods to generate initial permutations for least-squares unidimensional scaling of symmetric proximity matrices. *Journal of Classification*, 17(2):197–223, 2000.
16. S. Burer and R.D.C. Monteiro. Rank-two relaxation heuristics for max-cut and other binary quadratic programs. *SIAM J. on Optimization*, 12:503–521, 2001.
17. R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. SIAM Press, 2009.
18. R.E. Burkard, S.E. Karisch, and F. Rendl. `QAPLIB` - A Quadratic Assignment Problem Library. *Journal of Global Optimization*, 10:391–403, 1997.
19. C. Carreto and B. Baker. A GRASP interactive approach to the vehicle routing problem with backhauls. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys on metaheuristics*, pages 185–200. Kluwer Academic Publishers, 2002.
20. 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.
21. N. Christofides and E. Benavent. An exact algorithm for the quadratic assignment problem. *Operations Research*, 37(5):760–768, 1989.
22. C.W. Commander. Maximum cut problem, MAX-CUT. In C.A. Floudas and P.M. Pardalos, editors, *Encyclopedia of Optimization*, pages 1991–1999. Springer, 2009.
23. I.A. Contreras and J.A. Díaz. Scatter search for the single source capacitated facility location problem. *Annals of Operations Research*, 157:73–89, 2008.
24. 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.
25. G.L. Cravo, G.M. Ribeiro, and L.A. Nogueira Lorena. A greedy randomized adaptive search procedure for the point-feature cartographic label placement. *Computers and Geosciences*, 34(4):373–386, 2008.
26. Z. Drezner, P.M. Hahn, and É.D. Taillard. Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. *Annals of Operations Research*, 139:65–94, 2005.
27. B. Eschermann and H.J. Wunderlich. Optimized synthesis of self-testable finite state machines. In *20th International Symposium on Fault-Tolerant Computing (FFTCS 20)*, pages 390–397, 1990.
28. 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. volume 7548 of *Lecture Notes in Computer Science*, pages 253–267. Springer-Verlag, 2012.
29. 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.
30. T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.
31. T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *J. of Global Optimization*, 6:109–133, 1995.
32. D. Ferone, P. Festa, and M.G.C. Resende. Hybrid metaheuristics for the far from most string problem. In *Proceedings of HM 2013*, volume 7919 of *Lecture Notes in Computer Science*, pages 174–188. Springer-Verlag, 2013.
33. P. Festa. On some optimization problems in molecular biology. *Mathematical Bioscience*, 207(2):219–234, 2007.
34. P. Festa. A biased random-key genetic algorithm for data clustering. *Mathematical Bioscience*, 245(1):76–85, 2013.

35. P. Festa and P.M. Pardalos. Efficient solutions for the far from most string problem. *Annals of Operations Research*, 196(1):663–682, 2012.
36. P. Festa, P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. GRASP with path-relinking for the weighted MAXSAT problem. *ACM J. on Experimental Algorithmics*, 11:1–16, 2006.
37. P. Festa, P.M. Pardalos, and M.G.C. Resende. Algorithm 815: FORTRAN subroutines for computing approximate solution to feedback set problems using GRASP. *ACM Transactions on Mathematical Software*, 27:456–464, 2001.
38. P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the MAX-CUT problem. *Optimization Methods and Software*, 17(6):1033–1058, 2002.
39. P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys on Metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.
40. P. Festa and M.G.C. Resende. An annotated bibliography of GRASP – Part I: algorithms. *International Transactions in Operational Research*, 16(1):1–24, 2009.
41. P. Festa and M.G.C. Resende. An annotated bibliography of GRASP – Part II: applications. *International Transactions in Operational Research*, 16(2):131–172, 2009.
42. P. Festa and M.G.C. Resende. GRASP: Basic components and enhancements. *Telecommunication Systems*, 46(3):253–271, 2011.
43. R.M.D. Frinhani, R.M.A. Silva, G.R. Mateus, P. Festa, and M.G.C. Resende. GRASP with path-relinking for data clustering: A case study for biological data. volume 6630 of *Lecture Notes in Computer Science*, pages 410–420. Springer-Verlag, 2011.
44. K. Fujisawa, M. Fukuda, M. Fojima, and K. Nakata. Numerical evaluation of SDPA (Semidefinite Programming Algorithm. In *High performance optimization*, pages 267–301. Kluwer Academic Publishers, 2000.
45. 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.
46. A.M. Geoffrion and G.W. Graves. Scheduling parallel production lines with changeover costs: Practical applications of a quadratic assignment/LP approach. *Operations Research*, 24:595–610, 1976.
47. F. Glover. Tabu search – Part I. *ORSA J. on Computing*, 1:190–206, 1989.
48. F. Glover. Tabu search – Part II. *ORSA J. on Computing*, 2:4–32, 1990.
49. F. Glover. Tabu search and adaptive memory programing – 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, 1996.
50. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
51. 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*, 5(1):136–145, 2008.
52. 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.
53. 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.
54. D.E Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.
55. L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton's method. *SIAM Journal on Numerical Analysis*, 23:707–716, 1986.
56. 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*, 136(2):353–373, 2012.
57. M. Grötschel and W.R. Pulleyblank. Weakly bipartite graphs and the max-cut problem. *Operations Research Letters*, 1:23–27, 1981.
58. F. O. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4:221–225, 1975.
59. P. Hansen and N. Mladenović. Developments of variable neighborhood search. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 415–439. Kluwer Academic Publishers, 2002.

60. J. Hastad. Some optimal inapproximability results. *Journal of the ACM*, 48:798–859, 2001.
61. D.R. Heffley. The quadratic assignment problem: A note. *Econometrica*, 40(6):1155–1163, 1972.
62. D.R. Heffley. Decomposition of the koopmansbeckmann problem. *Regional Science and Urban Economics*, 10(4):571–580, 1980.
63. C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM J. on Optimization*, 10:673–696, 2000.
64. M.J. Hirsch, C.N. Meneses, P.M. Pardalos, M.A. Ragle, and M.G.C. Resende. A continuous GRASP to determine the relationship between drugs and adverse reactions. In O. Seref, O.E. Kundakcioglu, and P.M. Pardalos, editors, *Data mining, systems analysis, and optimization in biomedicine*, volume 953 of *AIP Conference Proceedings*, pages 106–121. Springer, 2007.
65. D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.
66. S.E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. *SIAM J. on Computing*, 12:177–191, 2000.
67. 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.
68. R.M. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, NY, 1972.
69. S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *J. of Statistical Physics*, 34:975–986, 1984.
70. T.C. Koopmans and M.J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.
71. J. Krarup and P.M. Pruzan. Computer-aided layout design. *Mathematical Programming Study*, 9:75–94, 1978.
72. M. Laguna and R. Martí. A GRASP for coloring sparse graphs. *Computaional Optimization and Applications*, 19:165–178, 2001.
73. R. De Leone, P. Festa, and E. Marchitto. A bus driver scheduling problem: A new mathematical model and a GRASP approximate solution. *Journal of Heuristics*, 17(4):441–466, 2011.
74. R. De Leone, P. Festa, and E. Marchitto. Solving a bus driver scheduling problem with randomized multistart heuristics. *International Transactions in Operational Research*, 18(6):707–727, 2011.
75. Y. Li and P.M. Pardalos. Generating quadratic assignment test problems with known optimal permutations. *Computational Optimization and Applications*, 1:163–184, 1992.
76. Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994.
77. S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21:498–516, 1973.
78. E.M. Loiola, N.M. Maia de Abreu, P.O. Boaventura-Netto, P. Hahn, and Tania Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176:657–690, 2007.
79. L. Lovász. On the Shannon capacity of a graph. *IEEE Trans. of Information Theory*, IT-25:1–7, 1979.
80. R. Martí and M. Laguna. Heuristics and meta-heuristics for 2-layer straight line crossing minimization. *Discrete Applied Mathematics*, 127(3):665–678, 2003.
81. T. Mautor. *Contribution à la résolution des problèmes d'implanation: algorithmes séquentiels et parallèles pour l'affectation quadratique*. PhD thesis, Université Pierre et Marie Curie, Paris, France. In French., 1992.

82. T. Mavridou, P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A GRASP for the biquadratic assignment problem. *European J. of Operational Research*, 105:613–621, 1998.
83. N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.
84. C.E. Nugent, T.E. Vollman, and J. Ruml. An experimental comparison of techniques for the assignment of facilities to locations. *Operations Research*, 16:150–173, 1968.
85. I.H. Osman, B. Al-Ayoubi, and M. Barake. A greedy random adaptive search procedure for the weighted maximal planar graph problem. *Computers and Industrial Engineering*, 45(4):635–651, 2003.
86. C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Comput. System Science*, 43(3):425–440, 1991.
87. C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: Algorithms and complexity*. Prentice-Hall, 1982.
88. P.M. Pardalos, P.S. Pitsoulis, and M.G.C. Resende. Algorithm 769: Fortran subroutines for approximate solution of sparse quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software*, 23:196–208, 1997.
89. P.M. Pardalos and M.G.C. Resende, editors. *Handbook of Applied Optimization*. Oxford University Press, 2002.
90. P.M. Pardalos and H. Wolkowicz. Quadratic assignment and related problems. In P.M. Pardalos and H. Wolkowicz, editors, *High performance optimization*. American Mathematical Society, 1994.
91. S. Poljak, F. Rendl, and H. Wolkowicz. A recipe for semidefinite relaxation for 0-1 quadratic programming. *J. of Global Optimization*, 7:51–73, 1995.
92. G.G. Pu, Z. Chong, Z.Y. Qiu, Z.Q. Lin, and J.F. He. A hybrid heuristic algorithm for HW-SW partitioning within timed automata. In *Proceedings of Knowledge-based Intelligent Information and Engineering Systems*, volume 4251 of *Lecture Notes in Artificial Intelligence*, pages 459–466. Springer-Verlag, 2006.
93. 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: The Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 499–520. American Mathematical Society, 1996.
94. M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Approximate solution of weighted MAX-SAT problems using GRASP. In J. Gu and P.M. Pardalos, editors, *Satisfiability problems*, volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 393–405. American Mathematical Society, 1997.
95. 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.
96. 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.
97. M.G.C. Resende and C.C. Ribeiro. A GRASP for graph planarization. *Networks*, 29:173–189, 1997.
98. C.C. Ribeiro and S. Urrutia. Heuristics for the mirrored traveling tournament problem. *European J. of Operational Research*, 179:775–787, 2007.
99. A.J. Robertson. A set of greedy randomized adaptive local search procedure (GRASP) implementations for the multidimensional assignment problem. *Computational Optimization and Applications*, 19:145–164, 2001.
100. C. Roucairol. *Du sequentiel au parallele: la recherche arborescente et son application a la programmation quadratique en variables 0 et 1*. PhD thesis, Université Pierre et Marie Curie, Paris, France. In French., 1987.
101. S. Sahni and T. Gonzales. P-complete approximation problems. *Journal of the Association for Computing Machinery*, 23:555–565, 1976.
102. M. Scriabin and R.C. Vergin. Comparison of computer algorithms and visual based methods for plant layout. *Management Science*, 22:172–187, 1975.

103. N.Z. Shor. Quadratic optimization problems. *Soviet J. of Computer and Systems Science*, 25:1–11, 1987.
104. J. Skorin-Kapov. Tabu search applied to the quadratic assingnment problem. *ORSA Journal on Computing*, 2(1):33–45, 1990.
105. L. Steinberg. The backboard wiring problem: a placement algorithm. *SIAM Review*, 3:37–50, 1961.
106. U.W. Thonemann and A. Bölte. An improved simulated annealing algorithm for the quadratic assignment problem. Technical report, Department of Production and Operations Research, 1994.
107. L. Trevisan. Approximating satisfiable satisfiability problems. *Algorithmica*, 28(1):145–172, 2000.
108. L. Trevisan, G. B. Sorkin, M. Sudan, and D. P. Williamson. Gadgets, approximation, and linear programming. *SIAM Journal on Computing*, 29(6):2074–2097, 2000.
109. M.R. Wilhelm and T.L. Ward. Solving quadratic assignment problems by simulated annealing. *IIE Transaction*, 19/1:107–119, 1987.
110. M. Yannakakis. On the approximation of maximum Satisfiability. In *Proceedings of the Third ACM-SIAM Symposium on Discrete Algorithms*, pages 1–9, 1992.

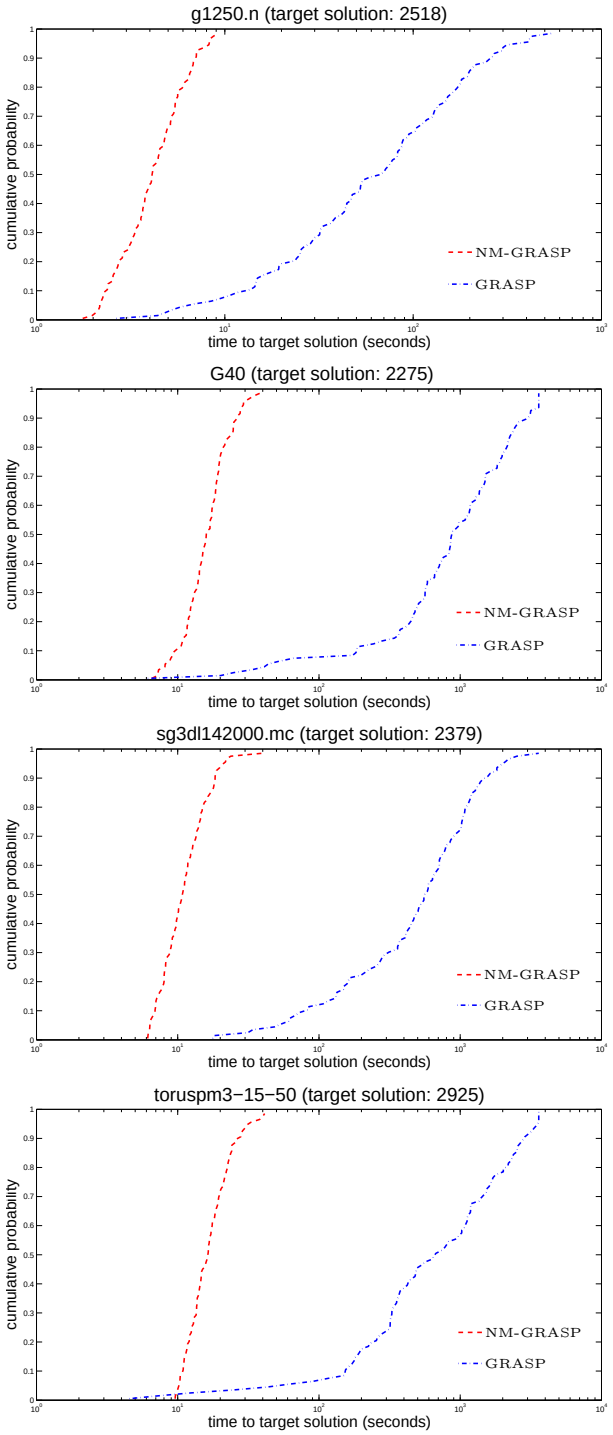

Fig. 9 TTTplots for the easy targets.

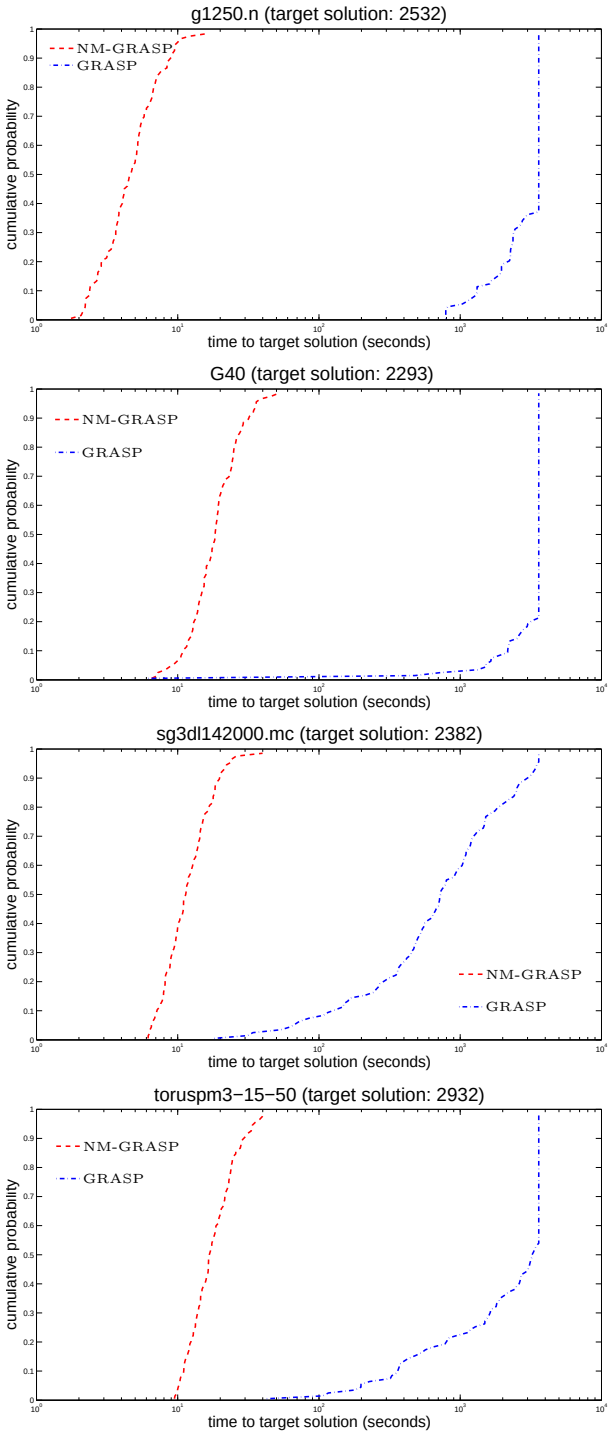

Fig. 10 TTTplots for the classical GRASP targets.

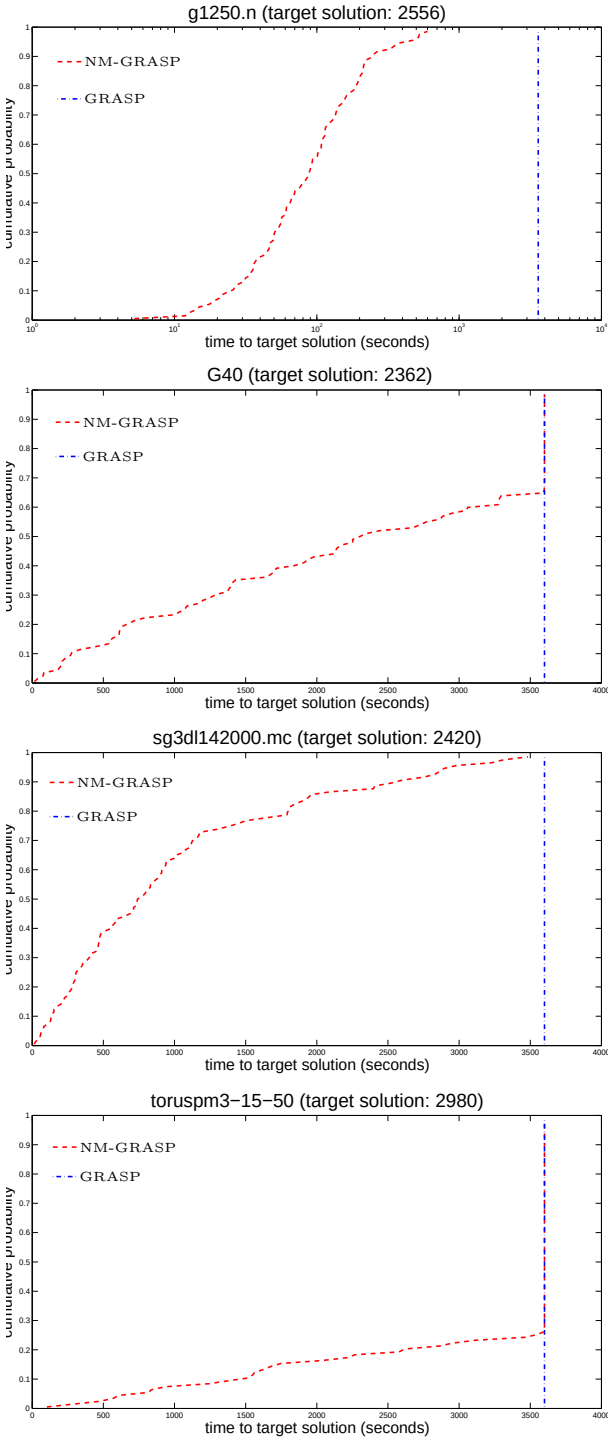

Fig. 11 TTTplots for the Nonmonotone GRASP targets.