# ADVANCED GRASP FOR THE OBNOXIOUS $p$-MEDIAN PROBLEM

J. MANUEL COLMENAR, PETER GREISTORFER, RAFAEL MARTÍ, AND ABRAHAM DUARTE

Abstract. The Obnoxious $p$-Median problem consists in selecting a subset of $p$ facilities from a given set of possible locations, in such a way that the sum of the distances between each customer and its nearest facility is maximized. The problem is $\mathcal{NP}$-hard and can be formulated as an integer linear program. It was introduced in the 1990s, and a branch and cut method coupled with a tabu search has been recently proposed. In this paper, we propose a heuristic method – based on the GRASP methodology – for finding approximate solutions to this optimization problem. In particular, we consider an advanced GRASP design in which a filtering mechanism avoids applying the local search method to low quality constructed solutions. Empirical results indicate that the proposed implementation compares favorably to previous methods. This fact is confirmed with non-parametric statistical tests.

## 1. Introduction

Facility location has been of practical and theoretical interest for more than the half of a century (Cohen, 1973; Klose and Drexl, 2005). First linear programming (LP) formulations origin in the late fifties and were soon followed by solution techniques, which later on became well-known as Branch and Bound (B&B) or Mixed Integer Programing (MIP). The classical so-called warehouse location problem (WLP), also well-known as (simple) facility or plant location problem, is an uncapacitated optimization problem, which can be modeled as an LP (Balinski, 1965) or network problem (Drezner and Hamacher, 2004; Melkote and Daskin, 2001). It is a site-selecting location-allocation model with a min-sum objective, where from a number of potential facility or warehouse sites the set of costumers has to be serviced, while minimizing the total fixed site-costs (location) plus the total variable customer assignment costs (allocation). The WLP is $\mathcal{NP}$-hard (Garey and Johnson, 1979; Papadimitriou and Yannakakis, 1991) and, as such, has been in the focus of researchers interested in developing specialized LP-approaches (Körkel, 1999) or approximative constructive and local search or improvement algorithms, including standard add- or drop-heuristics, and Lagrangian approaches (Kuehn and Hamburger, 1963; Beasley, 1993), which then were followed by more advanced metaheuristics like genetic algorithms or tabu search and its derivatives (Kratica et al., 2001; Michel and Hentenryck, 2004; Greistorfer and Rego, 2006). Comparing the WLP with the $p$-median problem, pMP (Hakimi, 1964; 1965), one identifies two differences: (1) there are no fixed site-costs involved and (2) the number of finally opened sites, $p$, is no longer a decision variable, but becomes included in

*Key words and phrases.* Obnoxious location, Diversity problem, Metaheuristics, GRASP, Filter solutions.

the model. The pMP can be solved in polynomial time for fixed values of $p$, but is strongly $\mathcal{NP}$-hard for variable values of $p$ (Garey and Johnson, 1979; Megiddo and Supowit, 1984; Current et al., 2004). Consequently, any pMP is a special case of the general class of WLPs. Such as for the WLP, a variety of solution procedures, exact approaches, primal-dual approaches and metaheuristics, has been introduced for the pMP (Tansel et al., 1983; Reese, 2006; Mladenović et al., 2007). A most recent paper, Batta et al. (2014), is recommend for a classical as well as modern view (dispersion, population, and equity criteria) on locational developments, offering an additional special focus on pMP.

This work relates to location type problems like the WLP and pMP, additionally taking account of so-called obnoxious or semi-obnoxious effects. Such effects often occur when interesting services of some provider are based on unwanted, but inevitably necessary locations, which do not add additional value to the product. Quite contrary, pure obnoxiousness clearly devalues the production process. Obnoxious problems were firstly coined in Church and Garfinkel (1978) who located a facility in a network using an exact method and also introduced the term of so-called bottleneck points for a network.

Erkut and Neuman (1989) used the terms *disservice* and *service* of the people in the vicinity of an (semi-)obnoxious facility that occurs during the processing of a product. In general, such services include a type of hazardous material, waste disposal, water treatment, nuclear power or chemical plants as well as big public facilities like airports. Regularly, these problems arise in the context of urban settings when the network may consist of noisy or polluting roads, transportation corridors, or rail lines (Segal, 2003).

The present work deals with the Obnoxious $p$-Median (OpM) problem. It can be formally defined as follows. Let $I$ be a set of clients, $J$ a set of facilities, and $d_{ij}$ the distance between the client $i \in I$ and the facility $j \in J$. The OpM problem consists in finding a set $S$ with $p$ facilities (with $S \subseteq J$ and $p < |J|$), such that the sum of the minimum distance between each client and the set of facilities is maximized. In mathematical terms:

$$\begin{aligned} &\max \quad \sum_{i \in I} \min\{d_{ij} : j \in S\} \\ &\text{subject to} \\ &\qquad S \subseteq J \ / \ |S| = p \end{aligned}$$

Facilities in $S$ are called open facilities, while facilities in $J \setminus S$ are known as closed or unopened facilities.

Table 1 shows an example of pair-client distances, where there are 9 clients $I = \{i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8, i_9\}$ and 6 potential facilities $J = \{j_1, j_2, j_3, j_4, j_5, j_6\}$. Suppose that $p = 3$. Then, a solution of the OpM consists of selecting 3 facilities out of 6 in $J$. Table 2a shows a solution with $S = \{j_2, j_5, j_6\}$. The minimum distance between each client and the corresponding facility is highlighted with bold font. For example, the distances from client $i_1$ to each facility are: $d(i_1, j_2) = 2$, $d(i_1, j_5) = 4$, and $d(i_1, j_8) = 8$. Then, $j_2$ is the closest facility to client $i_1$. The value of the objective function of this solution, denoted as $f(S)$, is the sum of the

| | $j_1$ | $j_2$ | $j_3$ | $j_4$ | $j_5$ | $j_6$ |
|---|---|---|---|---|---|---|
| $i_1$ | 3 | 2 | 10 | 13 | 4 | 8 |
| $i_2$ | 14 | 14 | 11 | 3 | 15 | 12 |
| $i_3$ | 5 | 4 | 6 | 12 | 14 | 6 |
| $i_4$ | 3 | 1 | 1 | 3 | 10 | 9 |
| $i_5$ | 13 | 9 | 2 | 10 | 9 | 4 |
| $i_6$ | 9 | 5 | 11 | 14 | 4 | 11 |
| $i_7$ | 4 | 3 | 4 | 13 | 4 | 2 |
| $i_8$ | 12 | 2 | 9 | 3 | 15 | 13 |
| $i_9$ | 9 | 11 | 1 | 2 | 7 | 4 |
| $\sum$ | 72 | 51 | 55 | 73 | 82 | 69 |

Table 1. Matrix of distances. Last row shows the sum of all the client distances for each facility.

minimum distances from each client to the set of facilities: $f(S) = 2+12+4+1+4+4+2+2+4 = 35$.

Table 2b shows a different solution $S' = \{j_2, j_3, j_4\}$, where the value of the objective function is $f(S') = 23$. Considering that the OpM is a maximization problem, solution $S$ is better than solution $S'$. In other words, the minimum distances among the clients and facilities in $S$ has a sum that is larger than the one in $S'$.

| | $j_2$ | $j_5$ | $j_6$ |
|---|---|---|---|
| $i_1$ | **2** | 4 | 8 |
| $i_2$ | 14 | 15 | **12** |
| $i_3$ | **4** | 14 | 6 |
| $i_4$ | **1** | 10 | 9 |
| $i_5$ | 9 | 9 | **4** |
| $i_6$ | 5 | **4** | 11 |
| $i_7$ | 3 | 4 | **2** |
| $i_8$ | **2** | 15 | 13 |
| $i_9$ | 11 | 7 | **4** |
| | | | $f(S) = 35$ |

(a) $S = \{j_2, j_5, j_6\}$

| | $j_2$ | $j_3$ | $j_4$ |
|---|---|---|---|
| $i_1$ | **2** | 10 | 13 |
| $i_2$ | 14 | 11 | **3** |
| $i_3$ | **4** | 6 | 12 |
| $i_4$ | **1** | 1 | 3 |
| $i_5$ | 9 | **2** | 10 |
| $i_6$ | **5** | 11 | 14 |
| $i_7$ | **3** | 4 | 13 |
| $i_8$ | **2** | 9 | 3 |
| $i_9$ | 11 | **1** | 2 |
| | | | $f(S) = 23$ |

(b) $S' = \{j_2, j_3, j_4\}$

Table 2. Example of solutions with $p = 3$.

In this paper we explore the adaptation of the Greedy Randomized Adaptive Search Procedure methodology, GRASP, introduced in Feo and Resende (1989), to solve the OpM problem. Each GRASP iteration consists of constructing a trial solution and then applying an improvement procedure to find a local optimum. We explore different designs for both phases, construction and improvement. In particular, in Section 3 we propose two different constructive methods and in Section 4, we describe two local search algorithms. Additionally, we propose an efficient strategy to update the objective function that considerably reduces the running time of GRASP. In Section 5, we present a filtering strategy intended to discard low-quality solutions and to selectively apply the local search only to promising solutions. This mechanism reduces the computing time without deteriorating the quality of the final solution. Section 6 reports on an extensive computational experience to validate the proposed algorithm by comparing its performance with those in the current state of the art. Finally, Section 7 summarizes the main conclusions of our research.

## 2. Literature review

There exists an extensive literature treating obnoxious and semi-obnoxious situations, modeled on the plane or in a graph, using max-min, max-max and often bi-criteria objectives (Erkut and Neuman, 1989; Cappanera, 1999; Segal, 2003; Conceição Fonseca and Captivo, 2007; Farahani et al., 2010; Batta et al., 2014). This goes along with the vast research on (obnoxious) $p$-median, $p$-center and similar location problems. Some of these sources are highlighted subsequently.

In Drezner and Wesolowsky (1980) a planar, Euclidean model of a 1-facility-problem is presented in which the shortest weighted distance to a point is maximized. Simultaneously, a side constraint must hold, i.e., the facility must be within a pre-specified distance from each point. The problem is solved using a graphical circles-based approach. A continuation of this work can be found in Drezner and Wesolowsky (1983), where a constrained obnoxious problem is considered for which a weighted rectangular-metric objective has to be maximized. The authors propose two algorithms, a so-called boundary and segment search and an LP-based approach. A semi-obnoxious scenario is solved in Melachrinoudis (1985). It uses a max-min model and seeks for a point on a convex two-dimensional bounded region, which maximizes the minimum weighted distance from that point to a given set of existing points in the region to be considered. As before, a circles-based, straightforward geometrical approach as well as an exact algorithm, using Kuhn-Tucker boundary points, is introduced. Complexity issues regarding the placement of several facilities in an obnoxious setting, a $p$ max-min problem, were considered by Tamir (1991). It is shown that even the finding of an approximate solution is $\mathcal{NP}$-hard. Again, Drezner and Wesolowsky (1996) deal with an Euclidean network and an obnoxious urban situation in which a location has to be determined inside the convex hull of a number of nodes in order to maximize the minimum weighted distance between this point and the nodes and arcs of the network. Among others, the authors offer an $\epsilon$-approximation scheme, which was later improved in Segal (2003). Welch and Salhi (1997) present three heuristics to solve the max-min formulation for siting $p$ facilities on a network considering pollution dispersion equation and additionally facility interaction. Starting with $p = 1$, a graphical method approximates the pollution dispersion by the use of polygons. Afterwards, the general case for $p$ and the combination of both the $p$ max-min and $p$ max-sum objectives, using a lexicographic approach, are investigated. Moreover, a simulated annealing algorithm is used to evaluate the base-algorithms.

Ohsawa and Tamura (2003) study the placement of a semi-obnoxious facility in a continuous plane underlying the bi-objective of maximizing the distance to the nearest inhabitant and minimizing the sum of distances to all users. In doing so, an elliptic max-min criterion and the rectangular min-sum criterion are used to model the push- and pull-aspect, respectively. For determining an efficient set and according trade-off curves, polynomial-time algorithms are presented. The obnoxious topic is considerably extended in Cappanera et al. (2004), who additionally consider a routing component, in which sites and transportation links may be affected by a hazardous single commodity. The resulting flow model of this obnoxious facility location problem is then solved by a Lagrangian heuristic and possibly improved by a subsequent B&B-algorithm. A multi-objective model for the location of landfills is discussed in Rakas et al. (2004). They use a combined weighted objective and solve the overall problem based on a composition of several MIP-solutions. Highlighted are the use of fuzzy arithmetic rules with triangular fuzzy numbers to cover the aspect of uncertainty and the coverage of a real-case-study. Tamir (2006) focuses on the problem of finding the location of two new and obnoxious facilities in the plane. The objective is to maximize the minimum of all weighted distances between the existing customer facilities and the two new facilities, and the weighted distance between the pair of new facilities. For the Euclidean and rectilinear versions of this 2-obnoxious facility location problem efficient logarithmic running time algorithms are presented, and the rectilinear case is extended to also handle general (non-convex) planar and compact polygonal sets. Yapicioglu et al. (2007) use a single- and a bi-objective objective particle swarm optimizer, well-known from evolutionary computation for their semi-obnoxious facility location problem. The idea is to simultaneously minimize transportation costs and undesirable effects, represented by a piecewise distance function depicting the degree of obnoxiousness. A computational analysis approximates the linear complexity in effort with increasing problem size. Berman and Wang (2007; 2008) consider the problem of locating single and multiple semi-obnoxious facilities while expropriating the nearest demand nodes under the restriction of a given total budget hold by the developer. The objective is to maximize the minimum weighted distance to the non-expropriated demand nodes. The algorithms introduced are based on dominating sets, Lagrangian relaxation and B&B. In Berman et al. (2008) the base topic is extended to the possible expropriation of population centers in the vicinity of selected routes of a transportation model, thus introducing a routing effect, e.g., when avoiding the transportation of hazardous material in populated resident zones. Solution approaches for the single- and multiple-flow-case are based on a greedy heuristic and on column generation or branch and price, respectively.

A multi-objective obnoxious facility location model is treated in Bhattacharya (2011), in which an obnoxious facility interacts with a given set of existing costumers on the plane and has to be located according to a max-min criterion, while at the same time maximizing the number of existing facility points covered. The algorithm proposed is a rectangle decomposition, a step-wise analytical approach that embeds linear as well as non-linear programming sub-procedures. Ortigosa et al. (2011) describe heuristic methods, based on Pareto-approaches, for generating semi-obnoxious locations. In doing so, they use a bi-objective convex/non-convex approach and several randomized sampling methods to optimize the desirable and the non-desirable aspect of the model, i.e. the min-sum pull-objective and the obnoxious push-objective. Coutinho-Rodrigues et al. (2012) deal with an empiric case of an urban waste collection problem in which a number of waste containers has to be located in a way that combines push and pull aspects of being serviced by a location too near (visual aesthetics, smell, nuisance, attractiveness) and by a location too far away (walking effort to deposit waste), respectively. Again, a bi-objective model approach is used, this time to minimize both, the container investment cost and a so-called weighted average customer dissatisfaction. The solution approach is a MIP-model, letting the decision-maker the final choice from non-dominated locations. Lozano et al. (2012) study the so-called separation or dual bandwidth problem. This $\mathcal{NP}$-hard problem uses a max-min objective function to optimize transmitters in radio frequency assignment by assigning different frequencies in such a way that physically neighboring transmitters have as different frequencies as possible. To solve this type of layout problem, different metaheuristic approaches are studied, e.g., ejection chains, tabu search and variable neighborhood search. The interesting point in Heydari and Melachrinoudis (2012), who describe a semi-obnoxious facility with elliptic max-min and network min-sum objectives, is the use of two different metric measures. To find the efficient set of this bi-objective problem three phases are implemented: a network redefinition (of the transportation network due to bottleneck points and Voronoi-clustering), elimination (of inefficient edges) and construction (of the efficient set and its non-dominated trade-off curve). In Plastria et al. (2013) a continuous, i.e., Euclidean, semi-obnoxious facility location problem is in the focus of interest. The analytical method offered, based on machine learning, gives the necessary conditions for optimality, and, using the latter, develops a polynomial enumeration on the set of dominating solutions.

In this paper, we target the Obnoxious $p$-Median (OpM) problem, which consists of selecting a subset of $p$ facilities from a given set of possible locations, in such a way that the sum over all customers of the distances between each customer and its nearest facility is maximized. OpMP was introduced in the 1990s (Eiselt and Laporte, 1995; Welch and Salhi, 1997; Cappanera, 1999) and, as of today, has only gained relatively few attention. It is $\mathcal{NP}$-hard (Tamir, 1991) and can be formulated as a binary LP. Note that Burkard et al. (2007) proved that the special case of the OpM problem on a tree can be solved in linear time. A branch and cut method coupled with a tabu search has been recently suggested by Belotti et al. (2007). We include both methods, the branch and cut and the tabu search in our computational experimentation reported in Section 6.

## 3. Constructive methods

The GRASP construction phase is an iterative, greedy, randomized and adaptive procedure that constructs solutions from scratch. It is iterative since the solution is built by considering one element at a time. It is greedy because the addition of each element to the solution under construction is guided by a greedy function. It is randomized because it performs a random selection from a list of good candidates. Finally, the method is adaptive in the sense of updating relevant information from a construction step to the next one.

A complete solution of the OpM is a set $S \subset J$ with exactly $p$ elements ($|S| = p$). Let $S \subset J$ be a *partial solution* to the OpM problem (i.e., $0 \leq |S| < p$). For each client $i \in I$, we define the minimum distance between $i$ and the facilities in $S$, denoted as $\delta_i$, as follows:

$$\forall i \in I \rightarrow \delta_i = \min_{j \in S} d_{ij} \tag{1}$$

The value of this partial solution, $f(S)$, can be directly computed as:

$$f(S) = \sum_{i \in I} \delta_i \tag{2}$$

We propose a greedy function that calculates the change in the objective function when a new facility is added to a partial solution. In particular, if the facility $j$ is included in the partial solution $S$, producing a new solution $S' = S \cup \{j\}$, the corresponding $\delta'$-value for each client is updated as follows:

$$\forall i \in I \rightarrow \delta'_i = \min\{\delta_i, d_{ij}\}$$

Instead of directly computing the change of the objective function, we determine the contribution of the new facility to the objective function. This is obtained with the greedy function $g(S, j)$:

$$g(S, j) = \sum_{i \in I} \min\{0, (d_{ij} - \delta_i)\} \tag{3}$$

where $(d_{ij} - \delta_i) < 0$ indicates that the facility $j$ is closer to $i$ than the remaining facilities in $S$. Then, the inclusion of $j$ reduces $\delta_i$. On the other hand, if $(d_{ij} - \delta_i) \geq 0$, the inclusion of the facility $j$ does not affect to client $i$. We use $g(S, j)$ to determine the best potential facility, $j^\star$, to be included in the partial solution. In mathematical terms:

$$j^\star = \arg \max_{j \in J \setminus S} \{g(S, j)\}$$

Let us illustrate the computation of the proposed greedy function with the example shown in Section 1. Considering that the OpM problem consists of finding a subset of facilities distant from the clients, we first identify the facility with the largest sum of distances. In this example, the sum of distances from all clients to each facility are 72, 51, 55, 73, 82 and 69. Then, the most distant facility with respect to all clients is $j_5$ with a sum of distances equal to 82. Therefore, the greedy strategy selects this facility, obtaining the partial solution $S = \{j_5\}$ and the minimum distance from each client to $j_5$ is $[4, 15, 14, 10, 9, 4, 4, 15, 7]$ respectively.

Table 3 calculates the objective function change when we try to include a new facility, showing for each client $i$, $d_{ij} - \delta_i$. For example, if we selected $j_1$ to be included in the partial solution, the situation of clients $i_1, i_2, i_3, i_4$, and $i_8$ would be deteriorated since $j_1$ is closer to these clients than $j_5$. On the other hand, $i_5, i_6, i_7$, and $i_9$ are not affected for the inclusion of this new facility. The facility which presents the best (higher) evaluation of the greedy function is, in fact, $j_1$, with a value equal to -21 according to Equation 3. We then update the partial solution to $S = \{j_1, j_5\}$ and the array of distances to $[3, 14, 5, 3, 9, 4, 4, 12, 7]$, respectively.

Similarly, Table 4 shows the values of the greedy function when we try to add a new facility to the current solution. The best option now is to include $j_6$, which obtains a value of -12. We then include $j_6$ in the partial solution, obtaining $S = \{j_1, j_5, j_6\}$ and the distances to elements: $[3, 12, 5, 3, 4, 4, 2, 12, 4]$. Considering that $p = 3$, $S$ is a feasible solution with an objective function value of 49 (i.e., the sum of minimum distances).

Now we describe our two constructive algorithms $C1$ and $C2$ for the OpM problem. $C1$ implements a typical GRASP construction where each candidate element

| | $g(S, j_1)$ | $g(S, j_2)$ | $g(S, j_3)$ | $g(S, j_4)$ | $g(S, j_6)$ |
|---|---|---|---|---|---|
| $i_1$ | 3-4 = -1 | 2-4= -2 | 10-4= 6 | 13-4= 9 | 8-4 = 4 |
| $i_2$ | 14-15= -1 | 14-15= -1 | 11-15=-4 | 3-15= -12 | 12-15= -3 |
| $i_3$ | 5-14=-9 | 4-14=-10 | 6-14=-8 | 12-14=-2 | 6-14=-8 |
| $i_4$ | 3-10= -7 | 1-10= -9 | 1-10= -9 | 3-10= -7 | 9-10= -1 |
| $i_5$ | 13-9=4 | 9-9=0 | 2-9=-7 | 10-9=1 | 4-9=-5 |
| $i_6$ | 9-4=5 | 5-4=1 | 11-4=7 | 14-4=10 | 11-4=7 |
| $i_7$ | 4-4 =0 | 3-4 = -1 | 4-4 =0 | 13-4= 9 | 2 -4 = -2 |
| $i_8$ | 12-15=-3 | 2-15=-13 | 9-15=-6 | 3-15=-12 | 13-15=-2 |
| $i_9$ | 9-7=2 | 11-7=4 | 1-7=-6 | 2-7=-5 | 4-7=-3 |
| | **-21** | -36 | -40 | -38 | -24 |

Table 3. Greedy evaluation with respect to the partial solution $S = \{j_5\}$

| | $g(S, j_2)$ | $g(S, j_3)$ | $g(S, j_4)$ | $g(S, j_6)$ |
|---|---|---|---|---|
| $i_1$ | 2-3= -1 | 10-3= 7 | 13-3= 10 | 8-3 = 5 |
| $i_2$ | 14-14= 0 | 11-14=-3 | 3-14= -11 | 12-14= -2 |
| $i_3$ | 4-5=-1 | 6-5=1 | 12-5=7 | 6-5=1 |
| $i_4$ | 1-3= -2 | 1-3= -2 | 3-3= 0 | 9-3= 6 |
| $i_5$ | 9-9=0 | 2-9=-7 | 10-9=1 | 4-9=-5 |
| $i_6$ | 5-4=1 | 11-4=7 | 14-4=10 | 11-4=7 |
| $i_7$ | 3-4 = -1 | 4-4 =0 | 13-4= 9 | 2 -4 = -2 |
| $i_8$ | 2-12=-10 | 9-12=-3 | 3-12=-9 | 13-12=1 |
| $i_9$ | 11-7=4 | 1-7=-6 | 2-7=-5 | 4-7=-3 |
| | -15 | -21 | -25 | **-12** |

Table 4. Greedy evaluation with respect to the partial solution $S = \{j_1, j_5\}$

is initially evaluated by a greedy function to construct a Restricted Candidate List (RCL), and one element is selected at random from the RCL. Algorithm 1 shows the pseudo-code for $C1$. It initially creates a list of candidates (CL) which contains the elements that can be added to the partial solution under construction (Step 2). In order to favor the diversity of the constructed solutions, the method randomly selects the first facility from CL (Step 3) and includes it in the partial solution (Step 4). The method thus iterates until it obtains a solution with $p$ facilities (Steps 6 to 13). In each iteration, $C1$ calculates the minimum ($g_{\min}$) and maximum ($g_{\max}$) values of the greedy function $g(S, j)$ (Steps 7 to 8). After that, $C1$ constructs a restricted candidate list (RCL) with all the elements (facilities) whose greedy value exceeds a percentage $\alpha$ of the best greedy value (Step 9). Finally, in Step 10, the method selects at random one facility from the RCL and adds it to the solution, updating CL (Steps 11 to 12).

We now describe $C2$, based on the construction strategy introduced in Resende and Werneck (2004), in which randomization takes place before the greedy selection in each construction step. In $C2$, we first randomly choose candidates and then evaluate them to make the greedy choice. $C2$ first constructs a restricted candidate list RCL with a fraction  (with $0 \leq \alpha \leq 1$) of the elements in the CL selected

**Algorithm 1:** Greedy constructive algorithm ($C1$)

```
1: $S \leftarrow \emptyset$
2: $\text{CL} \leftarrow J$
3: $j_0 \leftarrow \texttt{SelectRandom}(\text{CL})$
4: $S \leftarrow S \cup \{j_0\}$
5: $\text{CL} \leftarrow \text{CL} \setminus \{j_0\}$
6: while $|S| < p$ do
7:     $g_{min} \leftarrow \min_{j \in \text{CL}} g(S, j)$
8:     $g_{max} \leftarrow \max_{j \in \text{CL}} g(S, j)$
9:     $\text{RCL} \leftarrow \{j \in \text{CL} \mid g(S, j) \geq g_{min} + \alpha \cdot (g_{max} - g_{min})\}$
10:    $j \leftarrow \texttt{SelectRandom}(\text{RCL})$
11:    $S \leftarrow S \cup \{j\}$
12:    $\text{CL} \leftarrow \text{CL} \setminus \{j\}$
13: end while
14: return $S$
```

at random. Then, it evaluates all of them, computing the greedy function for all elements in the RCL, and selects the best one. Algorithm 2 shows the pseudo-code for $C2$. As it can be seen, the initial steps are the very same than those in Algorithm 1. The main difference between both methods resides on the instructions in the main loop (Steps 6 to 12 in Algorithm 2). In particular, it randomly selects a number of facilities from the CL. This number is determined by $\alpha$, the percentage of the current size of the CL. Notice that we ensure that the value of $size$ ranges from 1 to |CL| (see Step 7). The restricted candidate list is built with $size$ elements of the CL, selected at random (Step 8). Then, the method selects the element $j^\star$, which presents the best value according to the greedy function $g$ (Step 9). Similarly to $C1$, the constructive procedure adds the selected element to $S$ (Step 10) and removes it from CL (Step 11).

**Algorithm 2:** Greedy constructive algorithm ($C2$)

```
1: $S \leftarrow \emptyset$
2: $\text{CL} \leftarrow J$
3: $j_0 \leftarrow \texttt{SelectRandom}(\text{CL})$
4: $S \leftarrow S \cup \{j_0\}$
5: $\text{CL} \leftarrow \text{CL} \setminus \{j_0\}$
6: while $|S| < p$ do
7:     $size \leftarrow \max\{\lfloor \alpha \cdot |\text{CL}| \rfloor, 1\}$
8:     $\text{RCL} \leftarrow \texttt{SelectRandom}(\text{CL}, size)$
9:     $j^\star \leftarrow \arg\max_{j \in \text{RCL}} g(S, j)$
10:    $S \leftarrow S \cup \{j^\star\}$
11:    $\text{CL} \leftarrow \text{CL} \setminus \{j^\star\}$
12: end while
13: return $S$
```

The $\alpha$ parameter controls the greediness/randomness of the GRASP constructive procedures. Specifically, if $\alpha = 0$, the corresponding method would be totally random. On the other hand, if $\alpha = 1$, the constructive procedure would be a pure greedy method. In Section 6 we investigate the influence of this parameter on the performance of $C1$ and $C2$, selecting the value that obtains the best result.

## 4. Local search

The second stage of a GRASP algorithm consists in improving the constructed solution by applying a local search method to obtain a local optimum. Given a solution $S$, a neighbor solution $S'$ can be obtained by applying a move, which can be viewed as a perturbation in the solution $S$. The set of all neighbor solutions of $S$, called neighborhood, is denoted as $N(S)$. In particular, we define a move operator that interchanges a facility $j \in S$ with a facility $j' \in J \setminus S$, producing a new feasible solution $S' = S \setminus \{j\} \cup \{j'\}$. For the sake of simplicity, we denote this move as $S' \leftarrow exchange(S, j, j')$. Therefore, the neighborhood of a solution can be defined as follows:

$$N(S) = \{S' \leftarrow exchange(S, j, j') : j \in S, j' \in J \setminus S\}$$

The neighborhood can be explored with two different strategies: best improvement, in which all the solutions are examined (i.e., the associated neighborhood is completely explored), to identify the best; and first improvement, that tries to avoid the time complexity of exploring the whole neighborhood by performing the first improving move encountered during the exploration of the corresponding neighborhood. Therefore, the order in which the neighbors are inspected has a significant influence on the search. Notice that the order of exploration in the best improvement strategy is irrelevant since the corresponding neighborhood is fully explored.

We propose two different local search procedures based on the first improvement strategy. The difference between them resides in the neighborhood scanning strategy. In particular, $LS1$ explores the neighborhood at random. Algorithm 3 shows the pseudo-code of this procedure. It receives as input argument a feasible solution $S$, which is improved (Steps 2 to 20). The local search uses working copies of the set of facilities $J$, which are given as $J_c$ (opened sites, Step 4) and $J'_c$ (unopened sites, Step 5), which are both randomly sorted. The neighborhood is, therefore, explored at random by selecting an opened facility (Step 7) and an unopened facility (Step 9). Specifically, in Step 10 a neighbor solution is visited. Then, $LS1$ tests whether $S'$ improves upon $S$ (Step 11) or not (Step 14). If so, the exploration of the current neighborhood is abandoned, updating the incumbent solution (Steps 12 and 13); otherwise, the copied sets are updated (Steps 15 and 16). The local search ends when no further improvement is found.

The second local search, $LS2$, sorts the facilities in the solution in ascending order of the $\delta$-values (see Section 3). For the sake of simplicity we do not include the pseudo-code of this method since is similar to the one presented in Algorithm 3. The only difference is that facilities $j$ and $j'$ are selected according to their $\delta$-value (instead of selecting them at random). We will study the performance of both methods in the computational experience.

One of the key elements in designing an effective local search method is the definition of the move and the associated move value (change in the objective function

**Algorithm 3:** Random selection local search ($LS1$)

```
1: $improve \leftarrow true$
2: while $improve$ do
3:   $improve \leftarrow false$
4:   $J_c \leftarrow$ RandomlySort($S$)
5:   $J'_c \leftarrow$ RandomlySort($J \setminus S$)
6:   while ($|J_c| \neq \emptyset$) & ($improve \neq$ true) do
7:     $j \leftarrow$ next($J_c$)
8:     while ($|J'_c| \neq \emptyset$) & ($improve \neq$ true) do
9:       $j' \leftarrow$ next($J'_c$)
10:      $S' \leftarrow$ exchange($S, j, j'$)
11:      if $f(S') > f(S)$ then
12:        $S \leftarrow S'$
13:        $improve \leftarrow true$
14:      else
15:        $J_c \leftarrow J_c \setminus \{j\}$
16:        $J'_c \leftarrow J'_c \setminus \{j'\}$
17:      end if
18:    end while
19:  end while
20: end while
21: return $S$
```

value). A move produces a change in the objective function, called *move_value*. More precisely, let $S$ be a solution whose associated cost is $f(S)$, let $j_1 \in S$ be a opened facility, and let $j_2 \in J \setminus S$ be an unopened facility. Then, *exchange*$(S, j_1, j_2)$ produces a new solution $S' = S \setminus \{j_1\} \cup \{j_2\}$ with cost $f(S')$. The change in the objective function (due to the move) can be computed as:

$$move\_value = f(S') - f(S)$$

We say that *exchange*$(S, j_1, j_2)$ is an improving move if *move_value* $> 0$, since the OpM is a maximization problem.

The local search performs a sequence of moves to reach the final solution (local optimum). To perform a single move, it evaluates many candidates in the neighborhood. Therefore, the move evaluation is a critical part when designing a GRASP, since it requires a considerable computational effort. In a straightforward implementation, once the improvement method executes a move, it has to be evaluated, computing its corresponding *move_value*. Normally, this computation requires to scan all the clients and all the facilities (i.e., the computational complexity of such a straightforward move evaluation is $\Theta(|I||J|)$). Therefore, the larger the number of clients or facilities, the longer the execution time. However, we can considerably improve and reduce this complexity by maintaining a list for each client, which stores the currently best link to the closest facility. We conveniently separate these distances in two independent data structures. The first one contains the distances from all clients to the opened facilities. In particular, given a solution $S$, we maintain for each client $i$ the list $\Delta_S(i)$ with those distances from $i$ to each facility $j \in S$. This list is sorted is ascending order. For example, given the matrix of distances shown in Table 1 and the solution $S = \{j_2, j_5, j_6\}$, the corresponding data structure is depicted in Table 5a, where each row contains the information of each client. For instance, the list of distances for client $i_3$ is $\Delta_S(i_3) = \{(4, j_2), (6, j_6), (14, j_5)\}$, which means that the closest facility is $j_2$ (where the distance from $i_3$ to $j_2$ is 4), the second closest facility is $j_6$ (distance 6), and the farthest facility is $j_5$ (distance 14).

We construct a similar structure with the distances from clients to unopened facilities (i.e., those in $J \setminus S$). Specifically, we store for each client $i$ the ordered list $\Delta_{J\setminus S}(i)$ with the distances from $i$ to each facility $j \in J \setminus S$. Table 5b shows the list of distances for each client considering again the example described above (i.e. $J \setminus S = \{j_1, j_3, j_4\}$).

| | |
|---|---|
| $\Delta_S(i_1)$ | $\{(2, j_2), (4, j_5), (8, j_6)\}$ |
| $\Delta_S(i_2)$ | $\{(12, j_6), (14, j_2), (15, j_5)\}$ |
| $\Delta_S(i_3)$ | $\{(4, j_2), (6, j_6), (14, j_5)\}$ |
| $\Delta_S(i_4)$ | $\{(1, j_2), (9, j_6), (10, j_5)\}$ |
| $\Delta_S(i_5)$ | $\{(4, j_6), (9, j_2), (9, j_5)\}$ |
| $\Delta_S(i_6)$ | $\{(4, j_5), (5, j_2), (11, j_6)\}$ |
| $\Delta_S(i_7)$ | $\{(2, j_6), (3, j_2), (4, j_5)\}$ |
| $\Delta_S(i_8)$ | $\{(2, j_2), (13, j_6), (15, j_5)\}$ |
| $\Delta_S(i_9)$ | $\{(4, j_6), (7, j_5), (11, j_2)\}$ |

(a) Facilities in $S$.

| | |
|---|---|
| $\Delta_{J\setminus S}(i_1)$ | $\{(3, j_1), (10, j_3), (14, j_4)\}$ |
| $\Delta_{J\setminus S}(i_2)$ | $\{(3, j_4), (11, j_3), (14, j_1)\}$ |
| $\Delta_{J\setminus S}(i_3)$ | $\{(5, j_1), (6, j_3), (12, j_4)\}$ |
| $\Delta_{J\setminus S}(i_4)$ | $\{(1, j_3), (3, j_1), (3, j_4)\}$ |
| $\Delta_{J\setminus S}(i_5)$ | $\{(2, j_3), (10, j_4), (13, j_1)\}$ |
| $\Delta_{J\setminus S}(i_6)$ | $\{(9, j_1), (11, j_3), (14 j_4)\}$ |
| $\Delta_{J\setminus S}(i_7)$ | $\{(4, j_1), (4, j_3), (13, j_4)\}$ |
| $\Delta_{J\setminus S}(i_8)$ | $\{(3, j_4), (9, j_3), (12, j_1)\}$ |
| $\Delta_{J\setminus S}(i_9)$ | $\{(1, j_3), (2, j_4), (9, j_1)\}$ |

(b) Facilities in $J \setminus S$.

Table 5. List of distances for each client considering the example in Table 1 and solution $S = \{j_2, j_5, j_6\}$.

The evaluation of a move is considerably improved by using these two data structures. In particular, if we execute the move $S' \leftarrow exchange(S, j, j')$, the value $f(S')$ can be obtained in $\mathcal{O}(|I|)$, since it is only required to access the first element of each list. In addition, the aforementioned data structures can be updated in $\mathcal{O}(|I| \log(|J|))$, since this operation requires to insert the elements $j$ and $j'$ in $2 * |I|$ ordered lists (that can be performed in $\mathcal{O}(\log(|J|))$). Therefore, *move_value* can be computed in $\mathcal{O}(|I| \log(|J|)) + \mathcal{O}(|I|) = \mathcal{O}(|I| \log(|J|))$, which is considerably better than the straightforward implementation. In Section 6 we will study the influence of this strategy that we denote as incremental evaluation.

## 5. GRASP with Filtering

GRASP repeatedly applies a construction followed by a local search. However, as mentioned in Section 4, the local search is time consuming, so we implement a filtering mechanism to discard low-quality constructed solutions and skip the local search. Filtering low-quality solutions was proposed in the early paper by Feo et al. (1994). We implement here the specific design proposed in Laguna and Martí (1999) and applied in Martí et al. (2011).

The percentage of improvement achieved by the application of the local search can be estimated as follows:

$$P(i) = \frac{f(S'_i) - f(S_i)}{f(S'_i)}$$

where $i$ indicates the $i$-th iteration, $S_i$ is the solution generated by the constructive algorithm at iteration $i$, and $S'_i$ is the solution obtained after applying the local search method to $S_i$. Considering that the OpM problem is a maximization problem the value of this percentage ranges from $P(i) = 0$ (i.e., the improvement method is not able to improve the constructed solution) to $P(i) < 1$.

The filtering strategy requires a "warming up" phase to compute the average improvement obtained by the local search, as well as the standard deviation of the improvement. This phase corresponds to the first $k$ iterations of the GRASP algorithm, where $k$ is an input parameter of the algorithm. After that phase, the mean $\mu_P$ and standard deviation $\sigma_P$ of the percentage of improvement $P$ can be estimated as:

$$\hat{\mu}_P = \frac{\sum\limits_{i=1}^{k} P(i)}{k}$$

$$\hat{\sigma}_P = \sqrt{\frac{\sum\limits_{i=1}^{k} (P(i) - \hat{\mu}_P)^2}{k-1}}$$

These two statistics can be used to determine whether it is "likely" that the constructed solution at iteration $i$ (with $i > k$) can be improved enough to outperform the current best solution, $S^\star$. To do this, we calculate the minimum percentage improvement that a constructed solution $S_i$ needs to be improved in order to be better than $S^\star$. This value, denoted as $imp(i)$, is obtained with the following expression:

$$imp(i) = \frac{f(S^\star) - f(S_i)}{f(S_i)}$$

If this value is close to $\hat{\mu}_P$, we can assume that the improvement method could improve the current best solution. On the other hand, if $imp(i)$ is considerably different than the estimation of the mean, it is unlikely that the local search method is able to obtain a solution better than the current one. In particular we only apply the local search method if and only if $imp(i)$ is smaller than the estimated mean plus $q$ times the estimation of the standard deviation (to consider a conservative rule). In mathematical terms:

$$imp(i) < \hat{\mu}_P + q \cdot \hat{\sigma}_P$$

where $q$ is a search parameter representing a threshold on the number of standard deviations away from the estimated mean percentage improvement. It is well-known that in a GRASP implementation the local search method consumes most of the running time. Therefore, we can considerably reduce it if we only improve promising constructed solutions. In Section 6, we will study the effect of different $q$ values on solution quality and running time.

The pseudo-code of the GRASP approach with the filtering strategy is shown in Algorithm 4. It performs $N$ independent iterations as it is customary in GRASP. The algorithm starts each iteration by constructing a solution (Step 3) with any of the procedures described in Section 3 ($C1$ or $C2$). If the number of iterations $i$ is lower than $k$, the constructed solution is improved (Step 5) with any of the local search procedures introduced in Section 4 ($LS1$ or $LS2$). Additionally, the percentage of improvement with respect to the best solution found are also updated (Steps 6 and 7). After $k$ iterations (out of $N$), the method decides whether to improve a new constructed solution or not. In particular, the average mean, standard deviation and the percentage of improvement are computed in Steps 9 to 11. Notice that these computations are performed in an incremental way. Additionally, these three values are updated during the entire execution of the procedure. This strategy has the effect of being more restrictive in the application of the local search procedure as soon as the number of iterations increases.

**Algorithm 4:** GRASP with filtering strategy.

```
1:  $S^\star \leftarrow \emptyset$
2:  for $i = 1$ to $N$ do
3:      $S_i \leftarrow$ Constructive()
4:      if $(i < k)$ then
5:          $S'_i \leftarrow$ LocalSearch($S_i$)
6:          $P(i) = \frac{f(S'_i) - f(S_i)}{f(S'_i)}$
7:          $S^\star \leftarrow$ UpdateBest($S'_i$)
8:      else
9:          $\hat{\mu}_P = \frac{\sum_{x=1}^{i} P(x)}{i}$
10:         $\hat{\sigma}_P = \sqrt{\frac{\sum_{x=1}^{i} (P(x) - \hat{\mu}_P)^2}{i-1}}$
11:         $imp(i) = \frac{f(S^\star) - f(S_i)}{f(S_i)}$
12:         if $(imp(i) < (\hat{\mu}_P + q \cdot \hat{\sigma}_P))$ then
13:             $S'_i \leftarrow$ LocalSearch($S_i$)
14:             $P(i) = \frac{f(S'_i) - f(S_i)}{f(S'_i)}$
15:             $S^\star \leftarrow$ UpdateBest($S'_i$)
16:         end if
17:     end if
18: end for
19: return $S^\star$
```

## 6. Experimental Results

In this section we first describe the preliminary experiments that guided us in the selection of the values of the different key search parameters and strategies of both constructive and local search algorithms. Then, we show the results of our GRASP method compared with the state-of-the-art methods, i.e., branch & cut and tabu search procedures describe in Belotti et al. (2007). All the experiments, including the algorithms reported in Belotti et al. (2007), were executed on the same computer, an Intel i5 660 processor running at 3.3 GHz with 8 Gb of RAM using GNU/Linux. In addition, all the results related to execution times will be displayed in seconds.

We have used a set of instances previously used in Belotti et al. (2007). In particular, they were generated by considering 24 instances (from `pmed17` to `pmed40`) of the well know $p$-median problem[1], where the number of nodes ranges from 400

[1]`http://people.brunel.ac.uk/~mastjjb/jeb/orlib/pmedinfo.html`

to 900. In order to transform a $p$-median instance into an obnoxious $p$-median one, Belotti et al. (2007) described the following procedure. Given the original instance with $n$ nodes, this method selects $n/2$ nodes at random to be the set of clients. The remaining $n/2$ become the set of facilities. Additionally, for each original instance, Belotti et al. (2007) derived three new instances for the OpM by considering three values of $p$: $\lfloor n/2 \rfloor$, $\lfloor n/4 \rfloor$ and $\lfloor n/8 \rfloor$. Table 6 reports the main characteristics of the new set of of 72 instances, where $n$ indicates the number of nodes, $|I|/|J|$ represents the number of clients/facilities, and $p$ the number of required facilities.

| Instance | $n$ | $\lvert I\rvert/\lvert J\rvert$ | $p$ |
|---|---|---|---|
| pmed17-p100 | 400 | 200 | 100 |
| **pmed17-p25** | **400** | **200** | **25** |
| pmed17-p50 | 400 | 200 | 50 |
| pmed18-p100 | 400 | 200 | 100 |
| pmed18-p25 | 400 | 200 | 25 |
| pmed18-p50 | 400 | 200 | 50 |
| pmed19-p100 | 400 | 200 | 100 |
| pmed19-p25 | 400 | 200 | 25 |
| pmed19-p50 | 400 | 200 | 50 |
| pmed20-p100 | 400 | 200 | 100 |
| pmed20-p25 | 400 | 200 | 25 |
| **pmed20-p50** | **400** | **200** | **50** |
| pmed21-p125 | 500 | 250 | 125 |
| pmed21-p31 | 500 | 250 | 31 |
| pmed21-p62 | 500 | 250 | 62 |
| pmed22-p125 | 500 | 250 | 125 |
| pmed22-p31 | 500 | 250 | 31 |
| **pmed22-p62** | **500** | **250** | **62** |
| pmed23-p125 | 500 | 250 | 125 |
| pmed23-p31 | 500 | 250 | 31 |
| pmed23-p62 | 500 | 250 | 62 |
| pmed24-p125 | 500 | 250 | 125 |
| pmed24-p31 | 500 | 250 | 31 |
| pmed24-p62 | 500 | 250 | 62 |
| pmed25-p125 | 500 | 250 | 125 |
| pmed25-p31 | 500 | 250 | 31 |
| pmed25-p62 | 500 | 250 | 62 |
| pmed26-p150 | 600 | 300 | 150 |
| pmed26-p37 | 600 | 300 | 37 |
| pmed26-p75 | 600 | 300 | 75 |
| pmed27-p150 | 600 | 300 | 150 |
| pmed27-p37 | 600 | 300 | 37 |
| pmed27-p75 | 600 | 300 | 75 |
| pmed28-p150 | 600 | 300 | 150 |
| pmed28-p37 | 600 | 300 | 37 |
| **pmed28-p75** | **600** | **300** | **75** |

| Instance | Nodes | $\lvert I\rvert/\lvert J\rvert$ | $p$ |
|---|---|---|---|
| pmed29-p150 | 600 | 300 | 150 |
| pmed29-p37 | 600 | 300 | 37 |
| pmed29-p75 | 600 | 300 | 75 |
| pmed30-p150 | 600 | 300 | 150 |
| pmed30-p37 | 600 | 300 | 37 |
| pmed30-p75 | 600 | 300 | 75 |
| pmed31-p175 | 700 | 350 | 175 |
| pmed31-p43 | 700 | 350 | 43 |
| pmed31-p87 | 700 | 350 | 87 |
| pmed32-p175 | 700 | 350 | 175 |
| pmed32-p43 | 700 | 350 | 43 |
| pmed32-p87 | 700 | 350 | 87 |
| pmed33-p175 | 700 | 350 | 175 |
| pmed33-p43 | 700 | 350 | 43 |
| **pmed33-p87** | **700** | **350** | **87** |
| pmed34-p175 | 700 | 350 | 175 |
| pmed34-p43 | 700 | 350 | 43 |
| pmed34-p87 | 700 | 350 | 87 |
| pmed35-p100 | 800 | 400 | 100 |
| pmed35-p200 | 800 | 400 | 200 |
| pmed35-p50 | 800 | 400 | 50 |
| **pmed36-p100** | **800** | **400** | **100** |
| pmed36-p200 | 800 | 400 | 200 |
| pmed36-p50 | 800 | 400 | 50 |
| pmed37-p100 | 800 | 400 | 100 |
| pmed37-p200 | 800 | 400 | 200 |
| pmed37-p50 | 800 | 400 | 50 |
| pmed38-p112 | 900 | 450 | 112 |
| pmed38-p225 | 900 | 450 | 225 |
| pmed38-p56 | 900 | 450 | 56 |
| **pmed39-p112** | **900** | **450** | **112** |
| pmed39-p225 | 900 | 450 | 225 |
| pmed39-p56 | 900 | 450 | 56 |
| pmed40-p112 | 900 | 450 | 112 |
| **pmed40-p225** | **900** | **450** | **225** |
| pmed40-p56 | 900 | 450 | 56 |

Table 6. Instances generated from the OR-Library (Beasley, 1990).

We have designed two constructive algorithms based on the same greedy function (see Section 3). These procedures, namely $C1$ and $C2$ are parametrized by $\alpha$, which controls the trade-off between randomness and greediness. In the first experiment, we evaluate the influence of this parameter over the performance of $C1$ and $C2$ by considering four different values: 0.25, 0.5, 0.75, and random, where random indicates that the method randomly selects an $\alpha$ value in the range $[0, 1]$ for each construction. In order to avoid the over-fitting of our methods, we consider a representative subset of 10% (i.e., 8 instances) of the whole set of instances, with different sizes and properties. Notice that the remaining 90 % instances (64 out of 72) are reserved for the final comparison with the state-of-the-art procedures. Table 6 shows with bold font the 8 instances that form the representative subset.

| Constr. ($\alpha$) | Avg. Cost | Avg. Time | Dev. (%) | Div. | # Best |
|---|---|---|---|---|---|
| C1(0.25) | 3851.63 | 7.56 | 7.9 | **6164.82** | 0 |
| C1(0.5) | 4063.75 | 13.33 | 8.17 | 6072.53 | 0 |
| C1(0.75) | 4745.13 | 10.91 | 7.77 | 5362.49 | 0 |
| C1(random) | 5837.38 | 7.02 | 30.38 | 5912.39 | 0 |
| C2(0.25) | 5892.38 | 6.46 | 2.98 | 1495.35 | 2 |
| C2(0.5) | 5896.75 | 8.86 | 2.65 | 1422.15 | 1 |
| C2(0.75) | **5904.5** | 11.53 | **2.53** | 1355.42 | **3** |
| C2(random) | 5902.38 | **4.69** | 3.89 | 1740.97 | **3** |

Table 7. Comparison of different variants of the constructive algorithms. The best results are highlighted with bold font.

Table 7 shows the corresponding results when generating 100 independent constructions averaged over the subset of 8 instances. We report the average best cost (Avg. Cost); average computing time (Avg. Time); deviation with respect to the best result in this experiment (Dev.(%)); the diversity of the solutions (Div.), computed as the sum of the different facilities found in the constructions generated on each run, divided by the number of constructions; and the number of best results (#Best), computed as the number of times the algorithm is able to match the best solution in this experiment. Attending to these results, we can assert that $C2$ outperforms $C1$ in terms of effectiveness, because it obtains the best results regarding the average cost, deviation and number of best solutions found, emerging $C2$ with $\alpha = 0.75$ as the best variant. On the other hand, $C1$ presents better results in terms of diversity, being $C1$ with $\alpha = 0.25$ the best variant. Finally, $C2$ with random $\alpha$ is the fastest algorithm, although the average cost and deviation are worse than $C2(0.75)$.

At this point, we thought that it could be interesting to select both, the most effective constructive algorithm, $C2(0.75)$, and the algorithm that produces the most diverse solutions, $C1(0.25)$. The first one feeds the local search with high-quality solutions, although they share most of their facilities. On the contrary, the second one provides diverse solutions to the local search to drive the search to different areas in the search space, but the quality of this solutions is considerably worse. It is well-known that the design of efficient metaheuristics mainly relies on a balance between search intensification and diversification. Therefore, we cannot anticipate which would result in better outcomes when coupled with the local search method.

In the next experiment we analyzed this combination by considering the two local search procedures defined in Section 4, namely $LS1$ and $LS2$ with the best two constructive procedures aforementioned (i.e., $C1(0.25)$ and $C2(0.75)$). Table 8 shows the results of the four derived GRASP variants in terms of cost, time, deviation, and number of best solutions. We do not include in this experiment the diversity computation since it is not useful for further improvements.

As shown in Table 8, the GRASP variants that use $C2(0.75)$ as constructive procedure obtain better results than the ones that consider $C1(0.25)$. We can then conclude that in this problem is more relevant to produce high quality solutions, although it requires to sacrifice the diversity (at least in the set of instances considered in this experiment). Additionally, $LS1$ obtains slightly better results both

| GRASP | Avg. Cost | Avg. Time | Dev. (%) | # Best |
|---|---|---|---|---|
| C1(0.25) + LS1 | 5870.82 | 3581.15 | 1.29 | **4** |
| C1(0.25) + LS2 | 5822.89 | 730.11 | 1.93 | 1 |
| C2(0.75) + LS1 | **5892.81** | 2207.63 | 0.91 | **4** |
| C2(0.75) + LS2 | 5888.66 | **461.63** | **0.84** | 1 |

Table 8. Local search methods coupled with two constructive procedures. The best results are highlighted with bold font.

in average cost and number of best results, spending longer computing times than $LS2$. This fact can be partially explained because $LS2$ sorts the elements in the solution according to the $\delta$-values (see Section 4), while $LS1$ scans the elements in a random way. Given that the local search is based on the first improvement strategy, $LS2$ reaches improved solutions faster than $LS1$.

In the next experiment, we study the computing time of the best GRASP method when (1) using a direct cost computation (DCC) and (2) using the incremental cost computation (ICC) presented in Section 4. Figure 1 depicts a bar diagram where the $X$-axis represents the name of the 8 instances considered in the preliminary experiments and the $Y$-axis gives the computing time required to obtain a local optimum for both methods, respectively, (DCC and ICC) in the corresponding instance. The figure clearly shows that the saving in computing time is significant for ICC. Specifically, for these 8 instances DCC needs 2364.18 seconds on average to obtain a local optimum, while ICC requires 316.75 seconds on average to obtain the same optimum value, i.e., more than 7 times faster which implies a reduction in the execution time is over 86% on average.

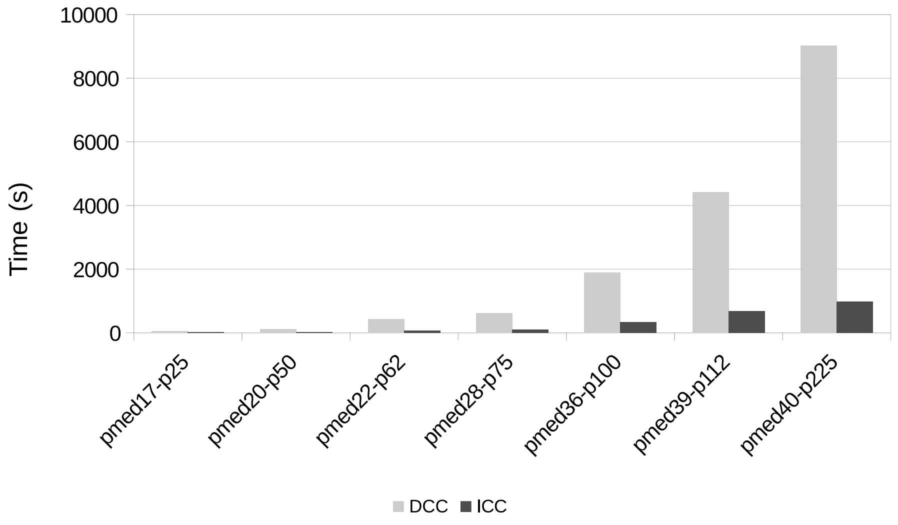

Figure 1. Computing time comparison for direc (DCC) and incremental (ICC) cost computation.

| $q$ (# Constr.) | Avg. Cost | Avg. Time (% vs Max) | Dev. (%) | # Best | Avg. # Skip |
|---|---|---|---|---|---|
| 0.5 (100) | 5889.51 | 1.51 (0.76%) | 0.074 | 38 | **53.17** |
| 1.0 (100) | 5889.51 | 1.41 (0.71%) | 0.077 | 38 | 41.04 |
| 1.5 (100) | 5889.22 | **1.38 (0.69%)** | 0.085 | 39 | 30.56 |
| 2.0 (100) | 5890.75 | 1.70 (0.85%) | **0.051** | **47** | 21.01 |
| 2.5 (100) | 5890.21 | 1.65 (0.83%) | 0.063 | 41 | 14.06 |
| GRASP(20) | 5884.53 | 1.72 (0.86%) | 0.176 | 27 | - |
| GRASP(100) | **5890.81** | 198.99 (100%) | 0.053 | 45 | - |

Table 9. Performance of the GRASP when filtering solutions with different values of $q$. Two GRASP versions without the filtering strategies are included as a baseline. The best results are highlighted with bold font.

In the next preliminary experiment we test the influence of the filter strategy described in Section 5. This mechanism skips the improvement method when the value of the constructed solution does not reach a minimum quality. The parameter $q$ controls the number of standard deviations away from the estimated mean percentage improvement. In order to determine the effectiveness of this strategy and to select the best value of $q$, we configure the GRASP algorithm with $k = 20$ (warming up phase) and $N = 100$, applying the filter strategy from iteration 21 to 100 (see Algorithm 4).

Table 9 shows the results of this experiment, where five values of $q$, from 0.5 to 2.5, are studied. In order to analyze the improvement with respect to traditional GRASP implementations, we include two versions without the filter strategy (last two rows of the table). The number between parenthesis indicates the number of iterations of each variant.

We additionally include the percentage time reduction with respect to the slowest algorithm (% vs Max), and the average number of skipped improvements (Avg. #Skip). As expected, the lower the value of $q$, the larger the number of skipped improvements, and the larger the reduction in the computing time. It is important to remark that the quality of the obtained results (in terms of the average percentage deviation and the number of best solutions found) does not present significant changes across different values of $q$, thus indicating the robustness of the filter (i.e. solutions skipped for improvement hardly modify the final result).

The reduction in the execution time is also remarkable. In particular, the slowest algorithm, GRASP(100), employs 198.99 s on average, while the filter variants need less than 2 seconds to find similar results. The filter strategy described in Section 5 reduces the computing time in more than 99%. We select the variant with $q = 2$ since it presents the best results in terms of quality and with a really competitive computing time (1.70 s). Notice that a meaningful consequence of the filter approach is the fact that it allows us to run the GRASP algorithm for a larger number of iterations.

As described in Section 2, the best identified procedures in the related literature are the branch & cut (B&C) and the tabu search method (XTS) described in Belotti et al. (2007)[2]. As it was aforementioned, the B&C is an exact procedure that, executed for unlimited time, guarantees the optimality of the solution found. However, considering that we are proposing heuristic procedures, we set the maximum computing time of this exact method to 3600 s. If after that time the B&C has not

[2]The authors kindly provided us with the source code of their algorithms

| Algorithm | Avg. Cost | Avg. Time | Dev. (%) | # Best |
|---|---|---|---|---|
| GRASP(1000) | 5893.15 | **237.34** | 0.76% | 34 |
| GRASP(5000) | **5894.51** | 749.09 | **0.73%** | **47** |
| B&C | 5764.35 | 3600.00 | 3.27% | 11 |
| XTS | 5847.08 | 511.35 | 1.57% | 25 |

Table 10. Comparison of the best GRASP method over the whole set of instances.

found the optimal solution, we interrupt its execution, returning the best solution found. The XTS is configured with the best parameters described in Belotti et al. (2007). We compare these methods with our best GRASP variant (constructive $C2$ with $\alpha = 0.75$, local search $LS1$, incremental computation of the cost, and filter strategy with $q = 2$) executed for two time horizons (1000 and 5000 iterations).

Table 10 presents the performance of each algorithm over the whole group of 72 instances. We report the same statistics used above. These results show the merit of both GRASP approaches. In particular, the fastest version (1000 iterations) clearly outperforms B&C and XTS in all the considered metrics. In particular, it obtains the best results in 34 instances (out of 72), while the competitors obtain 11 and 25, respectively. In the same line, GRASP(1000) presents a remarkable average percentage deviation of 0.76 %, which compares favorably to the 3.27 % and 1.57 %, achieved by B&C and XTS. Notice that OpM instances used in this experimentation presents a large value of the objective function (see the column Avg. Cost in Table 10). Therefore, improving the value of the cost in hundreds barely affects to the value of the average deviation (this problem is even aggravated in large instances). Then, the reduction of 0.81 % in this set of instances can be considered as a success. GRASP(1000) obtains these results in considerably shorter computing time. It is important to remark that all algorithms were executed in the same computer.

In order to show how the performance of the GRASP procedure is affected by the number of iterations, we include a version executed for 5000 iterations. It improves upon the results of the fastest version, obtaining the best solution in 47 instances (out of 72) with an average percentage deviation of 0.73 %. In this case, GRASP(5000) is slower than XTS but faster than B&C.

We finally compare our best method, GRASP(1000), with the best previous heuristic, XTS, with two well-known non-parametric tests for pairwise comparisons: the Wilcoxon test and the Sign test. The Wilcoxon test answers the question: Do the two samples (solutions obtained with both methods in our case) represent two different populations? The resulting $p$-value of 0.017 clearly indicates that the values compared come from different methods (using a typical significance level $\alpha = 0.05$ as the threshold for reject or not the null hypothesis). On the other hand, the Sign test computes the number of instances on which an algorithm supersedes another one. The resulting $p$-value of 0.013 indicates that there are significant differences between both algorithms, confirming the superiority of our GRASP method.

## 7. Conclusions

In this work we have studied the application of a GRASP approach to solve the Obnoxious $p$-median problem. In particular, we have developed two constructive algorithms based on a greedy function that calculates the contribution of a facility to the current solution. Our experimentation showed that one of them emphasizes search intensification, while the other one mainly diversifies the search into different regions.

We also proposed two different local search procedures to be executed after the constructive phase. Knowing that the local search changes only one facility in a given solution, an incremental move evaluation of the solutions was proposed, which averages an 86% reduction in the execution time in relation to a full evaluation of the solutions. In line with our objective of designing an efficient method, we implemented a filtering mechanism that is able to skip the application of the improvement phase in not promising constructed solutions. In this way, this technique saves an average of 91.4% of the computation time for the GRASP with 100 constructions, obtaining similar results in terms of cost than the approach with no filter.

Finally, before engaging in competitive testing, we performed a series of scientific preliminary tests to determine the contribution of the various elements that we have designed. We believe that the reader can find them very useful since valuable lessons can be learned from them, and applied to other problems.

The experimental results show that our GRASP algorithm is able to outperform the current state-of-the-art methods in both short and long time horizons.

As future work, we will incorporate additional cost functions to enhance the quality of the solutions. The idea is to deal with a multi-objective optimization scenario for the OpM problem.

## References

M. Balinski. Integer programming: Methods, uses, computation. *Mgt. Sci.*, 12: 253–313, 1965.

R. Batta, M. Lejeuneb, and S. Prasad. Public facility location using dispersion, population, and equity criteria. *EJOR*, 234(3):819–829, 2014.

J. E. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

J.E. Beasley. Lagrangean heuristics for location problems. *EJOR*, 65:383–399, 1993.

P. Belotti, M Labbé, F Maffioli, and M. Ndiaye. A branch-and-cut method for the obnoxious p-median problem. *4OR*, 5(4):299–314, 2007. ISSN 1619-4500.

O. Berman and Q. Wang. Locating a semi-obnoxious facility with expropriation. *J. Opl Res. Soc.*, 58:378–390, 2007.

O. Berman and Q. Wang. Locating a semi-obnoxious facility with expropriation. *Comput. & Ops Res.*, 35(2):392–403, 2008.

O. Berman, Z. Drezner, Q. Wang, and G.O. Wesolowsky. The route expropriation problem. *IIE Transactions*, 40:468–477, 2008.

U.K. Bhattacharya. A multi-objective obnoxious facility location model on a plane. *American J. Ops Res.*, 1:29–45, 2011.

R.E. Burkard, J. Fathali, and H. Taghizadeh Kakhki. The p-maxian problem on a tree. *Ops Res. Letters*, 35(3):331–335, 2007.

P. Cappanera. A survey on obnoxious facility location problems. Technical report, Dipartimento di Informatica, Università di Pisa, 1999.

P. Cappanera, G. Gallo, and F. Maffioli. Discrete facility location and routing of obnoxious activities. *Disc. Appl. Math.*, 133:3–28, 2004.

R.L. Church and R.S. Garfinkel. Locating an obnoxious facility on a network. *Trans. Sci.*, 12(2):107–118, 1978.

J.J. Cohen. A survey on the warehouse location problem. Operations research center working paper, OR 022-73, Massachusetts Institute of Technology (MIT), 1973.

M. da Conceição Fonseca and M.E. Captivo. Analysis of some models for semiobnoxious facility location. Technical report, Universidade de Lisboa, Faculdade de Ciências, Centro de Investigação Operacional, CIO Working Paper 16, 2007.

J. Coutinho-Rodrigues, L. Tralhão, and L. Alçada-Almeida. A bi-objective modeling approach applied to an urban semi-desirable facilitylocation problem. *EJOR*, 223(1):203213, 2012.

J. Current, M. Daskin, and D. Shilling. Discrete network location models. In Zvi Drezner and Horst W. Hamacher, editors, *Facility Location: Applications and Theory*, Springer series in operations research, chapter 3, pages 83–120. Springer Science & Business Media, 2004.

Z. Drezner and H.W. Hamacher. *Facility Location: Applications and Theory.* Springer, 2004.

Z. Drezner and G.O. Wesolowsky. A maximin location problem with maximum distance constraints. *AIIE Transactions*, 12(3):249–252, 1980.

Z. Drezner and G.O. Wesolowsky. The location of an obnoxious facility with rectangular distances. *J. Regional Science*, 23(2):241–248, 1983.

Z. Drezner and G.O. Wesolowsky. Obnoxious facility location in the interior of a planar network. *J. Regional Science*, 35(4):675–688, 1996.

H.A. Eiselt and G. Laporte. Objectives in location problems. In Z. Drezner, editor, *Facility Location. A Survey of Applications and Methods.*, pages 151–180. Springer, New York, 1995.

E. Erkut and S. Neuman. Analytical models for locating undesirable facilities. *EJOR*, 40(3):275–291, 1989.

R.Z. Farahani, M. SteadieSeifi, and Nasrin Asgari. Multiple criteria facility location problems: A survey. *Appl. Math. Modelling*, 34:1689–1709, 2010.

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.

T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.

M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–Completeness.* W.H. Freeman and Co., New York, 1979.

P. Greistorfer and C. Rego. A simple filter-and-fan approach to the facility location problem. *Comput. & Ops Res.*, 33(9):2590–2601, 2006.

S.L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Ops Res.*, 12(3):450–459, 1964.

S.L. Hakimi. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Ops Res.*, 13(3):462–475, 1965.

R. Heydari and E. Melachrinoudis. Location of a semi-obnoxious facility with elliptic maximin and network minisum objectives. *EJOR*, 223:452–460, 2012.

A. Klose and A. Drexl. Facility location models for distribution system design. *EJOR*, 162(1):429, 2005.

M. Körkel. *Effiziente Verfahren zur Lösung unkapazitierter Standort-Probleme*. Verlag für Wissenschaft und Forschung, Berlin, 1999.

J. Kratica, D. Tošić, V. Filipović, and I. Ljubic. Solving the simple plant location problems by genetic algorithm. *RAIRO Operations Research*, 35:127–142, 2001.

A.A. Kuehn and M.J. Hamburger. A heuristic program for locating warehouses. *Mgt. Sci.*, 9:643–666, 1963.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11(1):44–52, 1999.

M. Lozano, A. Duarte, F. Gortázar, and R. Martí. Variable neighborhood search with ejection chains for the antibandwidth problem. *J. of Heuristics*, 18:919–938, 2012.

R. Martí, J.J. Pantrigo, A. Duarte, V. Campos, and F. Glover. Scatter search and path relinking: A tutorial on the linear arrangement problem. *Int. J. Swarm. Intell. Res.*, 2(2):1–21, April 2011. ISSN 1947-9263.

N. Megiddo and K.J. Supowit. On the complexity of some common geometric location problem. *Siam J. Comput.*, 13(1):182 – 196, February 1984.

E. Melachrinoudis. Determining an optimum location for an undesirable facility in a workroom environment. *Appl. Math. Modelling*, 9:365–369, 1985.

S. Melkote and Mark S. Daskin. An integrated model of facility location and transportation network design. *Transp. Res. Part A*, 35:515–538, 2001.

L. Michel and P. Van Hentenryck. A simple tabu search for warehouse location. *EJOR*, 157(3):576–591, 2004.

N. Mladenović, J. Brimberg, P. Hansen, and J.A. Moreno-Pérez. The p-median problem: A survey of metaheuristic approaches. *EJOR*, 179(3):927–939, 2007.

Y. Ohsawa and K. Tamura. Efficient location for a semi-obnoxious facility. *Annals of OR*, 123:173–188, 2003.

P.M. Ortigosa, E.M.T. Hendrix, and J.L. Redondo. On methods for generating semi-obnoxious locations heuristically. Technical report, Wageningen School of Social Sciences, WASS Working PAPER No. 2, 2011.

C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *J. of Computer and System Sciences*, 43:425–440, 1991.

F. Plastria, J. Gordillo, and E. Carrizosa. Locating a semi-obnoxious covering facility with repelling polygonal regions. *Disc. Appl. Math.*, 161(16–17):2604–2623, 2013.

J. Rakas, D. Teodorović, and T. Kim. Multi-objective modeling for determining location of undesirable facilities. *Transp. Res. Part D*, 9:125–138, 2004.

J. Reese. Solution methods for the p-median problem: An annotated bibliography. *Networks*, 48(3):125–142, 2006.

M.G.C. Resende and R.F. Werneck. A hybrid heuristic for the p-median problem. *Journal of Heuristics*, 10(1):59–88, 2004. ISSN 1381-1231.

M. Segal. Placing an abnoxious facility in geometric networks. *Nordic J. of Computing*, 10(3):224–237, 2003.

A. Tamir. Obnoxious facility location on graphs. *SIAM J. Discrete Math.*, 2(4): 550–567, 1991.

A. Tamir. Locating two obnoxious facilities using the weighted maximin criterion. *Ops Res. Letters*, 34:97–105, 2006.

B.C. Tansel, R.L. Francis, and T.J. Lowe. Location on networks: A survey. Part I: The p-center and p-median problems. *Mgt. Sci.*, 29(4):482–497, 1983.

S.B. Welch and S. Salhi. The obnoxious p facility network location problem with facility interaction. *EJOR*, 102:302–319, 1997.

H. Yapicioglu, A.E. Smith, and G. Dozier. Solving the semi-desirable facility location problem using bi-objective particle swarm. *EJOR*, 177:733749, 2007.

(J.M. Colmenar) Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.

*E-mail address*: `josemanuel.colmenar@urjc.es`

(P. Greistorfer) Institut für Produktion und Logistik, Karl-Franzens-Universität Graz, Austria

*E-mail address*: `peter.greistorfer@uni-graz.at`

(R. Martí) Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain

*E-mail address*: `rafael.marti@uv.es`

(A. Duarte) Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.

*E-mail address*: `abraham.duarte@urjc.es`