INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Solving Large p-Median Problems with a Radius Formulation

Sergio García, Martine Labbé, Alfredo Marín,

To cite this article:
Sergio García, Martine Labbé, Alfredo Marín, (2011) Solving Large p-Median Problems with a Radius Formulation. INFORMS Journal on Computing 23(4):546-556. http://dx.doi.org/10.1287/ijoc.1100.0418

INFORMS Journal on Computing
Vol. 23, No. 4, Fall 2011, pp. 546–556
ISSN 1091-9856 | EISSN 1526-5528 | 11 | 2304 | 0546

http://dx.doi.org/10.1287/ijoc.1100.0418

Solving Large *p*-Median Problems with a Radius Formulation

Sergio García
Departamento de Estadística, Universidad Carlos III de Madrid, 28911 Leganés (Madrid), Spain, sergio.garcia@uc3m.es

Martine Labbé
Departément d'Informatique, Université Libre de Bruxelles, B-1050 Brussels, Belgium, mlabbe@ulb.ac.be

Alfredo Marín
Departamento de Estadística e Investigación Operativa, Universidad de Murcia, 30100 Espinardo (Murcia), Spain, amarin@um.es

By means of a model based on a set covering formulation, it is shown how the *p*-median problem can be solved with just a column generation approach that is embedded in a branch-and-bound framework based on dynamic reliability branching. This method is more than competitive in terms of computational times and size of the instances that have been optimally solved. In particular, problems of a size larger than the largest ones considered in the literature up to now are solved exactly in this paper.

Key words: discrete location; *p*-median; column-and-row generation
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received October 2009; revised May 2010, July 2010; accepted August 2010. Published online in *Articles in Advance* December 29, 2010.

1. Introduction

Significant research efforts have been devoted to discrete location problems because of both their importance for practice and their theoretical interest. These models are not only used to optimally locate facilities but also appear as subproblems of a wider spectrum of logistic problems (Ghiani and Musmanno 2004), and under a different appearance, they can be identified in several scientific fields where equivalent problems are studied. Indeed, the *p*-median problem introduced in Hakimi (1964), together with the simple plant location problem, could be considered the two best-studied problems in Discrete Location (see ReVelle and Eiselt 2005 for more details).

Given a set of n customers (nodes, vertices) and nonnegative costs c_{ij}, $1 \le i, j \le n$, the *p*-median problem consists of choosing a subset of p elements (*medians*) of the n nodes where facilities are established and the nonmedian nodes to these medians are allocated in such a way that the total allocation cost is minimized. The *p*-median problem is NP-hard (Kariv and Hakimi 1979), and several successful approaches have been used to solve instances with up to a few thousand nodes (Briant and Naddef 2004, Avella et al. 2007).

This paper describes a column-and-row generation algorithm for the *p*-median problem. It is based on a formulation where all but one of the constraints are in the shape of set covering inequalities. In this model, it is possible to solve the initial *reduced* linear relaxation, which considers just a very small part of the whole formulation (both in terms of variables and constraints), and to then add new constraints sequentially, each containing a single new variable. The final formulation is still small and guarantees an optimal solution for the relaxed problem. By embedding this process in a branch-and-bound framework, an exact algorithm to solve the original *p*-median problem is obtained. This approach performs extremely well in terms of both computational times and the sizes of the instances that have been successfully solved.

The rest of this paper is organized as follows. Section 2 shows the classical formulation for the *p*-median problem. A reduced formulation for this problem is given in §3, where some other existing reduced formulations are also mentioned. The resolution algorithm is detailed in §4, and §5 contains the computational study. Some final remarks are given in §6.

2. The *p*-Median Problem

Consider a cost matrix $C = (c_{ij})$, $i, j = 1, \ldots, n$, where $c_{jj} = 0 \ \forall j$ and $c_{ij} > 0 \ \forall i \neq j$. Given an integer number p, $1 \le p \le n-1$, the *p*-median problem (pM) looks for a subset $P \subseteq \{1, \ldots, n\}$ with cardinality p that minimizes the value

$$\sum_{i=1}^{n} \min\{c_{ij}\}_{j \in P}.$$

The classical formulation for pM is

$$
\begin{aligned}
\text{(CF)} \quad \min \ & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{s.t.} \ & \sum_{j=1}^{n} x_{ij} = 1, 1 \le i \le n, && (1)\\
& x_{ij} \le x_{jj}, 1 \le i, j \le n,\ i \ne j, && (2)\\
& \sum_{j=1}^{n} x_{jj} = p, && (3)\\
& x_{jj} \in \{0,1\}, 1 \le j \le n,\\
& x_{ij} \ge 0, 1 \le i,\ j \le n,\ i \ne j,
\end{aligned}
$$

where x_{jj} takes a value of one if and only if $j \in P$. Note that there is an optimal solution where x_{ij} takes a value of one for a median j such that $c_{ij} = \min\{c_{ik}\}_{k \in P}$. Variables x_{jj} are called *location variables* and variables x_{ij} $(i \ne j)$ are called *allocation variables*. The simple plant location problem (SPLP, a variant of the *p*-median problem where, instead of opening a fixed number of facilities, an opening cost must be paid for every facility to be opened) is obtained from (CF) by deleting constraint (2) and introducing positive costs associated to variables x_{jj} in the objective function.

Literature on the *p*-median problem is vast and extensive, and it is not the aim of this paper to give an exhaustive list of related works. Among the many existing papers, the interested reader is referred to Reese (2006) for a survey on methods for solving pM or Mladenović et al. (2007) for a review on metaheuristic techniques.

3. The Reduced Formulation and Related Work

First, we explain an alternative formulation for the *p*-median problem. Then, we focus on papers where either a *set covering formulation* for the *p*-median problem is proposed or a reduction in the formulation size is used to try to solve larger instances (i.e., papers that can be considered precursors of this study). Because the SPLP is closely related to the *p*-median problem, some of the cited papers below will refer to SPLP.

3.1. The Set Covering Approach *p*-Median Reformulation

We cite several papers that introduce and analyze formulations for the *p*-median problem and/or the SPLP with the common property that some allocation variables are enforced to take a value of one if there is no location variable that takes a value of one in a certain set. This differs from the classical point of view where allocation variables are forced to take a value of zero when location variables are zero.

The seminal paper by Cornuéjols et al. (1980) where these ideas were introduced and analyzed gives the so-called *canonical representation* (CR) for the SPLP and is adapted here to the *p*-median problem as follows: for each i, $1 \le i \le n$, construct a vector $D_i = (D_{i1}, \ldots, D_{iG_i})$ by sorting the different entries of the ith row of the cost matrix $C = (c_{ij})$ and by removing possible multiplicities:

$$0 = D_{i1} < D_{i2} < \cdots < D_{iG_i} = \max\{c_{ij}\}_{j=1}^{n}.$$

Notice that because $c_{ii} = 0$, then $D_{i1} = 0$ for all i. Now, define binary variables z_{ik} (called *cumulative variables*), $1 \le i \le n$, $1 \le k \le G_i$. Variable z_{ik} takes a value of one if and only if the allocation cost of customer i is at least D_{ik} (no matter which median it is allocated to) and zero otherwise. Also, variables y_i, $1 \le i \le n$, are defined to take a value of one if node i is a median and zero otherwise.

The *p*-median problem is now formulated in the following way:

$$
\begin{aligned}
\text{(CR)} \quad \min \ & \sum_{i=1}^{n}\sum_{k=2}^{G_i} (D_{ik} - D_{i,k-1}) z_{ik} \\
\text{s.t.} \ & \sum_{i=1}^{n} y_i = p,\\
& z_{ik} + \sum_{\{j \setminus c_{ij} < D_{ik}\}} y_j \ge 1, && (4)\\
& \quad 1 \le i \le n,\ 2 \le k \le G_i,\\
& y_i \in \{0,1\}, \quad 1 \le i \le n,\\
& z_{ik} \ge 0, \qquad 1 \le i \le n,\ 2 \le k \le G_i.
\end{aligned}
$$

Constraints (5) enforce variables z_{ik} to take a value of one if there is no open facility at less than distance D_{ik}, and positive coefficients in the objective function guarantee that z_{ik} takes a value of zero otherwise. Thus, they can be relaxed to be positive continuous variables. Notice that variables z_{i1} are not included in any constraint because the corresponding inequality would be $z_{i1} \ge 1$, which can be omitted. Moreover, since $c_{ii} = 0$ for all i and the remaining costs are strictly positive, inequalities $z_{i2} + y_i \ge 1$ hold as equalities indeed. Thus, the model can be reduced by doing the substitution $y_i = 1 - z_{i2}$ (which is useful when coding the solving algorithm).

It was proven in Cornuéjols et al. (1980) for the SPLP that the lower bound obtained by solving the linear relaxation of (CR) is equal to the bound obtained from the linear relaxation of (CF). This result can be trivially extended to the *p*-median problem.

There is an important advantage of formulation (CR) over formulation (CF) regarding the number of variables and constraints. There are n^2 x variables in (CF) and n^2 constraints in (1) and (2). In a worst-case

situation (that is, all the positive values in each row of C are different), the number of variables and constraints will be n^2 also in (CR). Otherwise, for each repeated value in a same row, there will be one less variable and one less constraint in (CR).

3.2. Literature Review on Reduced Formulations

Although formulation (CR) presents the clear advantage of a smaller size over formulation (CF), it was used in just a few papers and then almost forgotten. Kolen (1983) used a version of formulation (CR) to solve the SPLP problem in polynomial time on a tree (i.e., the set of customers is the set of nodes of a tree graph, and c_{ij} represents the length of a shortest path from i to j). Simão and Thizy (1989) solved the linear relaxation of (CR) using a specialized dual simplex algorithm. (CR) formulation was included in two different reviews in the same book (Cornuéjols et al. 1990, Kolen and Tamir 1990). Also, Dearing et al. (1992) developed a Boolean formulation for the SPLP and obtained a version of (CR) when linearizing it. Xu and Lowe (1993) explored the connection between the method of Simão and Thizy (1989) and DUALOC (Erlenkotter 1978), which is probably the most successful method to solve the SPLP to date. No attempt to design an algorithm to solve the SPLP or pM was done in any of these papers. The Boolean approach has also been used in Goldengorin et al. (2004) to develop a branch-and-peg algorithm for the SPLP (an algorithm that uses a rule to determine whether plants will or will not be located at certain sites of the current subproblem before branching).

Recently, Elloumi et al. (2004) developed a formulation for the p-center problem, the aim of which is to minimize the value of

$$\max\{\min\{c_{ij}\}_{j\in P}\}_{i=1}^{n}. \tag{5}$$

They inspired in the same idea to use cumulative variables z_k that take a value of one if the maximum (5) is at least equal to D_k (the kth positive value in matrix C). Instances of the p-center problem with up to 1,817 points were efficiently solved by means of a specialized method based on this formulation.

Apart from (CR), other reduced formulations have been developed during the last years in an attempt to efficiently solve large instances of the SPLP and/or pM. In Avella and Sassano (2001), (CF) is modified to delete variables $\{x_{ii}\}_{i=1}^{n}$. Avella and Sassano obtain several families of valid inequalities for this new formulation and generate some tentative results for medium-sized instances that prove the inequalities to be useful to close the duality gap.

The following refinement of (CF) is proposed in Church (2003). For the sake of simplicity, it is assumed here that all entries of each row of C are different. Given two pairs (i, j) and (i', j) such that $\{a\colon c_{ia} \le c_{ij}\} = \{a\colon c_{i'a} \le c_{i'j}\}$, it can be easily derived that $x_{ij} = x_{i'j}$ in an optimal solution. Therefore, one of these two variables can be replaced by the other one, deleting it from the formulation. The same idea is proposed in a Boolean framework in Dearing et al. (1992), where these pairs of equal allocation variables are substituted by the same product of binary location variables.

It must be said that, although it is a property that will not be used in this paper, the idea of Church (2003) can be easily generalized, as shown in Cánovas et al. (2007) in what the authors call *dominance constraints* by developing inequalities $x_{ij} \le x_{i'j}$ when the previous equality of sets is weakened to be an inclusion in one single way.

Moreover, Avella et al. (2007) use a delayed column-and-row generation strategy that they combine with cuts obtained from the study of the polyhedral structure of (CF) as well as with a Lagrangian relaxation technique. The final result is a very well-performing branch-and-cut-and-price algorithm that solves problems with up to 3,795 nodes under a CPU time limit of 100 hours. As far as we know, this is the best exact algorithm to date for solving p-median instances.

Last, Elloumi (2010) uses (CR) to develop another alternative reduced formulation. This formulation is used on the most difficult instances from the OR-Library (http://people.brunel.ac.uk/~mastjjb/jeb/info.html), but it performs computationally much worse than the algorithm we propose in this paper.

4. The Resolution Algorithm

We now explain how to take advantage of the particular structure of model (CR) to efficiently solve large p-median instances. In §4.1, we discuss the main idea that motivates our method, which is later detailed in §4.2. Our method could be called an "enlarge-and-branch" algorithm: we begin with a very small model and then progressively add new rows and columns.

4.1. The Basic Idea

Consider formulation (CR) introduced in the previous section. For any optimal solution vector z, given a customer $i \in \{1, \ldots, n\}$, subvector $z_{i\cdot} = (z_{i1}, z_{i2}, \ldots, z_{iG_i})$ has the following structure:

$$z_{i\cdot} = (1, 1, \ldots, 1, 0, 0, \ldots, 0).$$

That is, it begins with a value of one in all its components until it reaches an index where the first zero appears. From there, all the components have a value of zero. This property holds for every customer i, and it is an immediate consequence of the hierarchical structure of the constraints because

$$\{j\colon c_{ij} < D_{ik}\} \subsetneq \{j\colon c_{ij} < D_{i,k+1}\}.$$

Now, consider the linear relaxation (LP) of formulation (CR). First, we remove all the constraints (5) for $k \geq 3$, which gives us a relaxation (LP_0) of the linear relaxation (LP). Then, because the coefficients in the objective function are all positive and it is a minimization problem, it is trivial to see that variables z_{ik}, $k \geq 3$, can be fixed to zero. Once (LP_0) has been solved to optimality, if $z_{i2} = 0$ for all i, then the solution is also optimal for (LP). Otherwise, some new z_{i3} variables must be added to obtain a new relaxation (LP_0) of (LP). The process is repeated until an optimal solution of (LP) is obtained. This is detailed in the following algorithm.

Algorithm 1

Step 1. Let (LP_0) be a relaxation of (LP)—the linear relaxation of formulation (CR)—obtained by keeping only variables $\{y_i\}_{i=1}^n$ and $\{z_{i2}\}_{i=1}^n$. The remaining constraints are those including the considered variables and the objective function is also reduced accordingly.

Define $t_i = 2$, $1 \leq i \leq n$.

Step 2. Solve (LP_0) and let (y^*, z^*) be its optimal solution.

Step 3. Let $\Gamma = \{i\colon z^*_{it_i} > 0\}$. If $\Gamma = \emptyset$, then (y^*, z^*) is optimal for (LP). Stop.

Otherwise, go to Step 4.

Step 4. For every index $t_i \in \Gamma$:

1. create variable z_{i,t_i+1},
2. incorporate constraint

$$z_{i,t_i+1} + \sum_{\{j \setminus c_{ij} < D_{i,t_i+1}\}} y_j \geq 1$$

to formulation (LP_0),

3. add term $(D_{i,t_i+2} - D_{i,t_i+1})z_{i,t_i+1}$ to the objective function of (LP_0), and
4. update $t_i := t_i + 1$.

Go to Step 2.

Because this algorithm provides us with an optimal solution of the linear relaxation of (LP), to solve the integer problem, it is embedded in a branch-and-bound framework where the starting subproblem of every child node is the final formulation of the parent node with the branching y_j variable fixed to either zero or one. This exact method allows us to solve the p-median problem with a reduced formulation, that is, using much less constraints and variables than the classical formulation (CF).

Remark. In every iteration of the previous algorithm, the LP optimum yields a valid lower bound of the subproblem's optimal solution. This gives rise to the following observation. Suppose we have a feasible solution $(\bar{y}, \bar{z})$ with objective value $\bar{b}$. After Step 2 in Algorithm 1, let b^* be the objective value of solution (y^*, z^*). If $b^* \geq \bar{b}$, then the node can be pruned because it cannot give a better solution than the one already known. This property is very important because it allows to discard nodes before we even solve the linear relaxation to optimality.

4.2. The Enlarge-and-Branch Algorithm

The first fact to be stated to understand our algorithm is its purpose: to solve very large instances. This must be taken into account to understand the trade-off that our approach must face: to balance efficiency (perform well) and efficacy (solve much larger instances than the ones solved up to now). For the sake of simplicity when speaking about our algorithm, we will be cite it as *Zebra* (*Z-E*nlarge-and-*BR*anch-*A*lgorithm).

As has been previously explained, the procedure of solving enlarged linear relaxations until feasibility is reached is embedded in a branch-and-bound framework with several particularities to improve efficiency. Its main characteristics, described throughout, are as follows:

1. It incorporates an existing heuristic method as an auxiliary tool.
2. The size of the starting formulation is automatically decided depending on some properties of the instance.
3. Memory management is optimized (the entry data are not stored except in very few occasions).
4. A best-first search strategy is used for choosing the node. This strategy for selecting the node was the best with regard to average running time.
5. A dynamic hybrid strategy of pseudocosts and strong branching is used to choose the branching variable.
6. Although several parameters are used in the algorithm, they are automatically and univocally determined by the data of the instance being solved as per the rules described in this paper (see the rest of this section and the Online Supplement, available at http://joc.pubs.informs.org/ecompanion.html/).

4.2.1. The Heuristic Algorithmic Tool. For those instances of the ensuing computational study (§5) with up to 5,934 nodes, the heuristic method described in Resende and Werneck (2004) is used to find an initial heuristic solution (x^h, y^h) and an initial upper bound z^h. This algorithm (called Popstar) can be considered one of the best heuristic methods in recent literature for solving the p-median problem.

The reason why this heuristic is not used for larger instances is memory limitation: Popstar is not able to deal with such instances because it runs out of memory. This already points to a strong characteristic of *Zebra*: not only is it an exact method, it uses much less memory than a very efficient and contrasted heuristic.

Value z^h is used for pruning nodes in the branching tree while no better solution is found. Solution (x^h, y^h) is used to establish the value of parameter *sdz* described below.

Notice that the role of Popstar could be perfectly played by any other heuristic algorithm providing a feasible solution for the problem (that is, not only must the heuristic give an upper bound but it must also show how the location allocation is done).

4.2.2. The Starting Depth for z-Variables. In the previous section we proposed to initialize *Zebra* with only variables $\{z_{i2}\}_{i=1}^{n}$. We say that the *starting depth for z variables* is 2; that is, a parameter *sdz* is defined with a value of two.

Nevertheless, a value $sdz > 2$ could be considered to start the model with all the variables $\{z_{ik}\}_{\substack{i=1,\ldots,n,\\ k=2,\ldots,sdz}}$. The reason for starting with $sdz = 2$ is the hope of using as few variables as possible. However, the smaller p is, the more variables z_{ik} are needed (customers are allocated to more and more expensive medians). This means that the initial formulation will be reoptimized more and more times, as long as more variables z_{ik} are added.

To illustrate this idea, Figure 1 shows resolution times (in seconds) when considering different values of *sdz* and p in instance pcb3038 from the TSPLIB library (http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/) with no auxiliary heuristic solution. Although there is no monotonous increase or decrease in time when *sdz* grows up, we notice a general trend: the larger p is, the smaller the value of *sdz* should be. Also, there is always a certain value of *sdz* sufficiently large from the initial model that is too large to be solved efficiently.

Hence, the following dilemma must be faced: On the one hand, starting with few variables implies that RAM memory usage is minimized and that small linear relaxations can be quickly reoptimized (of course, at the expense of having to reoptimize a high number of problems). On the other hand, if we start with a large number of variables, we need much fewer reoptimizations. However, bigger models bring much longer solving and reoptimization times and more memory consumption. Moreover, perhaps not all of these extra variables are really useful; that is, we are solving a larger model than the "right" size to solve the linear relaxation optimally.

As can be seen, there must be a balance between both criteria. That is why value *sdz* is established according to what we can "expect" from the optimal solution—basically, the larger p is, the fewer z_{ik} variables are necessary. Two techniques will be used, depending on the size of the instance.

- *Technique 1 (with a feasible solution).* This approach can be used as long as there is a known feasible solution (x^h, y^h) for (CF).

From this solution, we can easily get a solution (y^h, z^h) for (CR). Let s_i be the number of ones used by subvector z_i^h; that is, $z_{i1}^h = z_{i2}^h = \cdots = z_{is_i}^h = 1$ and $z_{i,\,s_i+1}^h = 0$. Then, the starting depth for z is set to the average number of z variables per customer in the incumbent heuristic solution (rounded up):

$$sdz := \left\lceil \frac{s_1 + \cdots + s_n}{n} \right\rceil.$$

Our hope is that either the incumbent heuristic solution will indeed be optimal or the true optimal solution will not be too different from it.

As an ad hoc rule to improve efficiency, the true value for the starting depth is set to

$$sdz := \left\lfloor 1.1 \left\lceil \frac{s_1 + \cdots + s_n}{n} \right\rceil \right\rfloor.$$

The reason behind this 10% increase is that, because we are solving linear relaxations, many fractional variables appear. This leads to the need for a higher number of variables until we reach a linear relaxation where the optimal solution is also feasible for the original problem. It might also be that our heuristic solution is not optimal so that we need to be flexible.

- *Technique 2 (without any feasible solution).* If no information about how a feasible solution looks is available, then the starting depth is defined as $sdz = \bar{G}/p$, where $\bar{G} = (G_1 + \cdots + G_n)/n$. Value $\bar{G}/p$ can be considered an attempt to estimate the average number of z variables per customer that will be needed.

As an ad hoc rule, *sdz* is doubled for small instances and reduced whenever too many constraints would be generated at the initial model. More specifically, let r be the initial number of constraints. If $r \leq 10{,}000$, then *sdz* is doubled; otherwise, the parameter is multiplied by $3/\log_{10} r$ and rounded up.

Because obtaining values $G_1, \ldots, G_n$ (that is, for every customer, sorting the n costs and remove multiplicities) can be quite time consuming for the larger instances (e.g., about a half hour for instances with 13,509 nodes using the quicksort method), a uniform sample $\{i_1, \ldots, i_t\}$ from the nodes ($t = \lceil n/100 \rceil$) is used to get their respective values $\{G_{i_1}, \ldots, G_{i_t}\}$ and to estimate $\bar{G}$ with the sample average of this reduced subset. This strategy provides with very good estimations while significantly reducing running time (just 1% of the original effort).

Both of these described techniques (with and without heuristic solution) are very useful because they do not allow values D_{ik} to be stored in memory (after all, most of them will not be used) unless they are actually used in the model. Therefore, a lot of time is gained and a considerable amount of memory is saved.

4.2.3. Node Selection. A best-first strategy is used for selecting the next node to branch on—the one with the lowest lower bound.

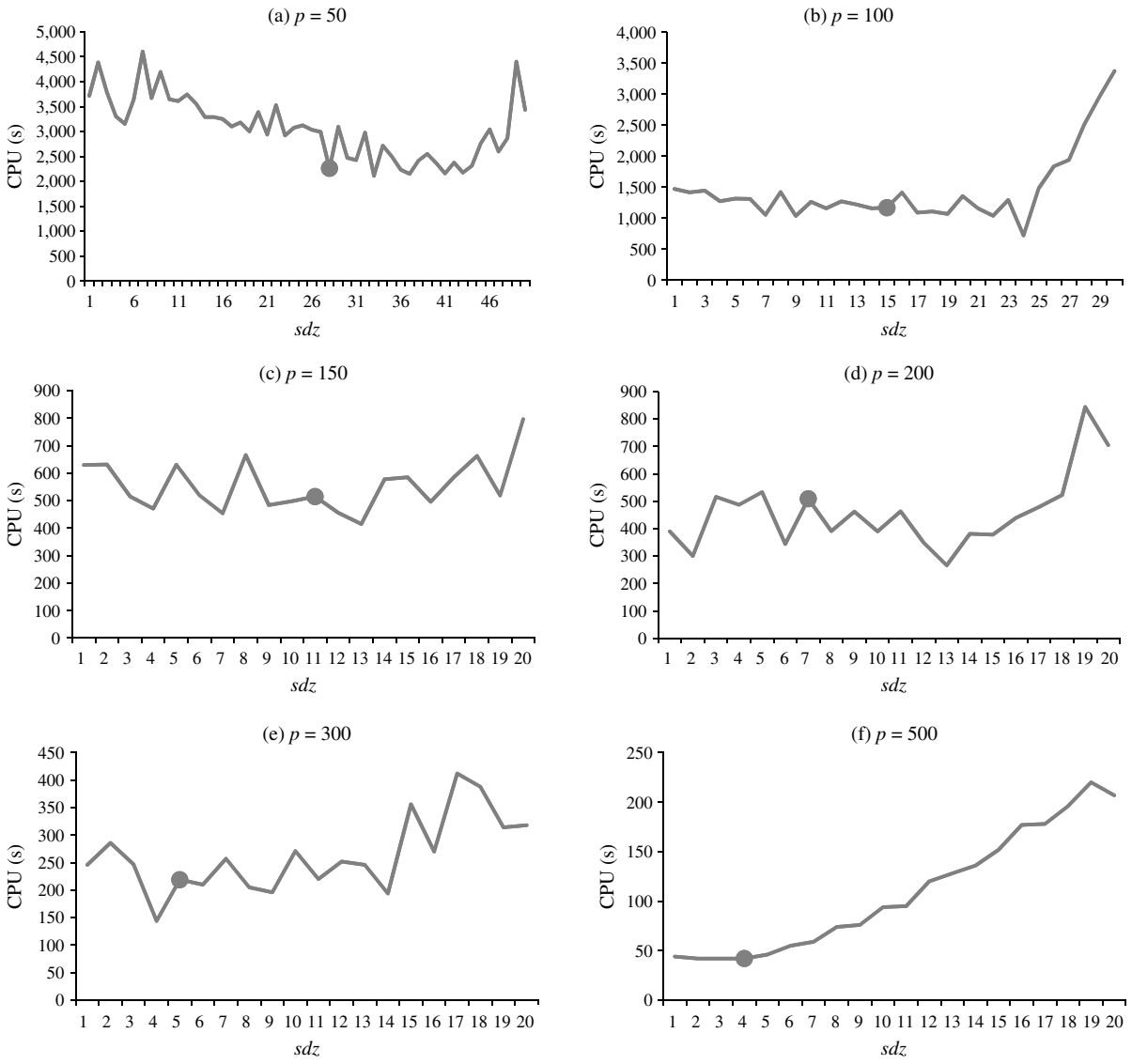

Figure 1 Comparison of CPU Times for Instance pcb3038 with Different *sdz* Values

4.2.4. Branching Strategy. As a branching rule, we use a dynamic version of the so-called *reliability branching strategy* introduced in Achterberg et al. (2005). See the Online Supplement for the details.

5. Computational Study

A computational study was carried out to test the solution method. The efficiency of *Zebra* is compared with the state of the art for the *p*-median problem that is given by Avella, Sassano, and Vasil'ev (Avella et al. 2007; ASV hereafter).

The two types of instances for the computational study are those of the OR-Library already mentioned in §3.2 and several TSP instances from the TSPLIB library (http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/). For the OR-Library instances, we compare the most difficult ones (although, in truth, neither of them is extremely difficult to solve). For the TSP instances, we compare rl1304, fl1400, u1432, vm1748, d2103, pcb3038, fl3795, and rl5934. In addition, the test bed will consider some instances never used before: usa13509, sw24978, ch71009, and pla85900. As in Avella et al. (2007), the time limit is 100 hours per problem.

The computational study for ASV was carried out on a Compaq EVO W4000 personal computer with a Pentium IV 1.8 GHz processor and 1 GB RAM. The code was written in Microsoft Visual C++ 6.0, and the LP solver was CPLEX 8.0. Our computational study was carried out on an Intel Core 2 CPU 6600 with two 2.4 GHz processors (although just one was used) and 3 GB RAM. The code was written in Microsoft Visual C++ 2005 and the LP solver was CPLEX 11.0. Unfortunately, it was not possible to compare ASV on our computer because the authors

were not able to provide us with their code. Therefore, we only cite the computational times shown in Avella et al. (2007).

Throughout the following comments on the computational results, instances have been divided into four groups depending on their size: small, medium, large, and huge. Notice that already the small problems are highly difficult for CPLEX (Avella et al. 2007). Hence, a direct application of CPLEX to solve the classical formulation has not been included in this paper.

The abbreviations on Tables 1–4 have the following meanings:

- Instance: the problem name.
- n: number of vertices.
- p: number of medians.
- Time: CPU time (in seconds) for solving the problem.
- BB: number of the nodes of the branching tree.
- *sdz*: starting depth for z variables.
- BLB: best lower bound.
- BUB: best upper bound. If an upper bound appears in bold, then it means that the problem has been optimally solved and this is the optimal value.

In the last row of each table, we provide the means for the CPU times and the number of branching nodes. For comparison to be as fair as possible, only the instances solved optimally by both AVS and *Zebra* have been included. For the larger instances where AVS cannot solve any instance, only the information for *Zebra* is provided.

Finally, a star (∗) means that the computer ran out of memory while solving the problem. When available, the best information up to that moment is shown.

Table 1 Computational Results for the *Zebra* Method: Small Instances

			AVS		*Zebra*				
Instance	n	p	BUB	Time	Time	BB	BLB	BUB	*sdz*
pmed26	600	5	**9,917**	187	12	17	9,917	**9,917**	12
pmed27	600	10	**8,307**	47	5	3	8,307	**8,307**	8
pmed28	600	60	**4,498**	3	1	1	4,498	**4,498**	4
pmed29	600	120	**3,033**	2	1	1	3,033	**3,033**	3
pmed30	600	200	**1,989**	2	1	1	1,989	**1,979**	2
pmed31	700	5	**10,086**	106	11	11	10,086	**10,086**	11
pmed32	700	10	**9,297**	65	5	3	9,297	**9,297**	9
pmed33	700	70	**4,700**	3	2	1	4,700	**4,700**	4
pmed34	700	140	**3,013**	2	2	1	3,013	**3,013**	3
pmed35	800	5	**10,400**	189	15	17	10,400	**10,400**	11
pmed36	800	10	**9,934**	453	69	155	9,934	**9,934**	9
pmed37	800	80	**5,057**	3	2	1	5,057	**5,057**	4
pmed38	900	5	**11,060**	320	30	31	11,060	**11,060**	11
pmed38	900	10	**9,431**	402	27	39	9,431	**9,431**	8
pmed38	900	20	**7,839**	41	9	13	7,839	**7,839**	6
pmed38	900	50	**5,892**	32	5	9	5,892	**5,892**	4
pmed38	900	100	**4,450**	8	2	1	4,450	**4,450**	4
pmed38	900	200	**2,905**	8	3	1	2,905	**2,905**	3
pmed38	900	300	**1,972**	8	3	1	1,972	**1,972**	2
pmed38	900	400	**1,305**	7	3	1	1,305	**1,305**	2
pmed38	900	500	**836**	7	3	1	836	**836**	2
pmed39	900	5	**11,069**	177	28	25	11,069	**11,069**	11
pmed39	900	10	**9,423**	271	20	37	9,423	**9,423**	8
pmed39	900	20	**7,894**	17	3	1	7,894	**7,894**	6
pmed39	900	50	**5,941**	37	6	7	5,941	**5,941**	5
pmed39	900	100	**4,461**	8	4	3	4,461	**4,461**	4
pmed39	900	200	**2,918**	8	3	1	2,918	**2,918**	3
pmed39	900	300	**1,968**	7	3	1	1,968	**1,968**	2
pmed39	900	400	**1,303**	7	3	1	1,303	**1,303**	2
pmed39	900	500	**821**	7	3	1	821	**821**	2
pmed40	900	5	**12,305**	124	14	7	12,305	**12,305**	12
pmed40	900	10	**10,491**	160	14	15	10,491	**10,491**	9
pmed40	900	20	**8,717**	72	7	5	8,717	**8,717**	7
pmed40	900	50	**6,518**	66	8	21	6,518	**6,518**	5
pmed40	900	90	**5,128**	9	2	1	5,128	**5,128**	4
pmed40	900	200	**3,132**	8	3	1	3,132	**3,132**	3
pmed40	900	300	**2,106**	8	3	1	2,106	**2,106**	2
pmed40	900	400	**1,398**	9	3	1	1,398	**1,398**	2
pmed40	900	500	**900**	8	3	1	900	**900**	2
Mean				74	14	11			

5.1. Small Instances

It is clear by looking at Table 1 that *Zebra* is much more efficient than AVS. Indeed, because of the recent advances in technology, these problems can be described as quite easy to solve (when using a specialized algorithm).

5.2. Medium Instances

If we analyze the CPU times in Table 2, it can be seen than neither of the two methods clearly outperforms the other one. It could perhaps be said that AVS is slightly better on average although it is much worse in some instances: for example, instance u1432 with $p = 50$ needed almost eight hours against less than eight minutes needed by *Zebra*.

The weak point of our algorithm can be seen in these instances: in general, because of the way the model has been defined, the smaller the size of p is, the higher the number of variables z_{ik} are needed, and the higher the probability that the method performs badly.

Finally, it must be remarked that *Zebra* is able to solve instance fl1400 with $p = 400$. This could not be solved before.

5.3. Large Instances

The performance of both algorithms for large instances of size up to 3,795 nodes is similar, but it can be seen that our algorithm is slightly better at this point. It is clear that AVS is much better for small values of p, but *Zebra* is better for higher numbers of instances and much better for some of them. Consider, for example, instances d2103 with $p = 100$ and fl3795 with $p = 150$. When dealing with small values of p, *Zebra* gets some out-of-memory cases at the root node because, as has been said before, the smaller the size of p, the more constraints *Zebra* needs (high value of *sdz*).

Instance rl5934 is stated explicitly by Avella et al. (2007) as intractable because of the large amount of memory needed. With *Zebra*, thanks in part to the

Table 2 Computational Results for the *Zebra* Method: Medium Instances

			AVS		*Zebra*				
Instance	*n*	*p*	BUB	Time	Time	BB	BLB	BUB	*sdz*
rl1304	1,304	5	**3,099,073**	32	3,579	1	3,099,073	**3,099,073**	114
rl1304	1,304	10	**2,134,295**	1,614	4,015	61	2,134,295	**2,134,295**	61
rl1304	1,304	20	**1,412,108**	22	170	1	1,412,108	**1,412,108**	31
rl1304	1,304	50	**795,012**	14	24	1	795,012	**795,012**	13
rl1304	1,304	100	**491,639**	40	16	5	491,639	**491,639**	7
rl1304	1,304	200	**268,573**	16	10	3	268,573	**268,573**	4
rl1304	1,304	300	**177,326**	18	11	3	177,326	**177,326**	3
rl1304	1,304	400	**128,332**	14	12	1	128,332	**128,332**	3
rl1304	1,304	500	**97,024**	20	13	3	97,024	**97,024**	2
fl1400	1,400	5	**174,877**	45	598	1	174,877	**174,877**	52
fl1400	1,400	10	**100,601**	33	140	1	100,601	**100,601**	33
fl1400	1,400	20	**57,191**	24	24	1	57,191	**57,191**	16
fl1400	1,400	50	**28,486**	18	11	1	28,486	**28,486**	8
fl1400	1,400	100	**15,962**	378	20	11	15,962	**15,962**	5
fl1400	1,400	200	**8,806**	191	868	9,227	8,806	**8,806**	3
fl1400	1,400	300	6,111	*	112,501*	585,779	6,103	6,123	3
fl1400	1,400	400	4,648	*	137,549	1,994,737	4,648	**4,648**	2
fl1400	1,400	500	3,764	*	360,000	3,034,955	3,761	3,766	2
u1432	1,432	5	**1,210,126**	41	412	1	1,210,126	**1,210,126**	39
u1432	1,432	10	**849,759**	26	172	1	849,759	**849,759**	22
u1432	1,432	20	**588,766**	101	60	3	588,766	**588,766**	13
u1432	1,432	50	**362,072**	28,257	323	327	362,072	**362,072**	6
u1432	1,432	100	**243,793**	119	16	7	243,793	**243,793**	4
u1432	1,432	200	**159,887**	58	22	7	159,987	**159,887**	3
u1432	1,432	300	**123,689**	43	21	13	123,689	**123,689**	2
u1432	1,432	400	103,979	*	360,000	2,502,105	103,636	104,102	2
u1432	1,432	500	**93,200**	36	10	1	93,200	**93,200**	2
v1748	1,748	5	**4,479,421**	59	2,870*	1	4,178,344	4,479,421	146
v1748	1,748	10	**2,983,645**	478	4,245	7	2,983,645	**2,983,645**	77
v1748	1,748	20	**1,899,680**	341	956	3	1,899,680	**1,899,680**	39
v1748	1,748	50	**1,004,331**	36	55	3	1,004,331	**1,004,331**	16
v1748	1,748	100	**636,515**	136	38	7	636,515	**636,515**	8
v1748	1,748	200	**390,350**	22	20	1	390,350	**390,350**	5
v1748	1,748	300	**286,039**	24	24	3	286,039	**286,039**	4
v1748	1,748	400	**221,526**	155	22	5	221,526	**221,526**	3
v1748	1,748	500	**176,986**	74	22	3	176,986	**176,986**	3
Mean				1,046	514	313			

optimized management of memory, these instances cannot only be loaded into CPLEX but also solved for different values of p. Thus, when we check the results for the different values of p considered, we see that the performance of our algorithm is much better.

It should be remarked that some new instances not solved before in literature can be solved to optimality by *Zebra*, including d2103 ($p = 100$), pcb3038 ($p = 20, 50$), and many rl5934 instances ($p \geq 200$). In addition, the best-known solution for instance d2103 and $p = 50$ is improved.

Finally, the two observations regarding AVS are as follows.

1. The optimal value for instance d2103 with $p = 400$ is not the same for AVS (75,356) than for *Zebra* (75,324). We assume that the *Zebra* solution is the right one because besides having checked its feasibility, we know that 75,324 is also the optimal value given in Beltrán et al. (2006).

2. Some instances in Avella et al. (2007) are only tackled in an heuristic way (e.g., pcb3038 with $p = 20$ and 50, and all problems for instance rl5934, except $p = 1{,}500$), which explains their small computational times.

5.4. Huge Instances

Some very large instances with up to 85,900 nodes have been solved (e.g., usa13509, sw24978). Although for these instances we could not use the Popstar heuristic (it cannot solve the problems because they are too large), our exact method was able to solve an important number of cases (from $p = 300$ for usa13509 and from $p = 1{,}000$ for sw24978; see Table 4). On the other hand, there are some instances where, after a large number of nodes in the branching tree, *Zebra* runs out of memory without even obtaining a feasible solution.

Table 3 Computational Results for the *Zebra* Method: Large Instances

			AVS		*Zebra*				
Instance	*n*	*p*	BUB	Time	Time	BB	BLB	BUB	*sdz*
d2103	2,103	5	**1,005,136**	96	*2,872	1	943,629	1,005,136	106
d2103	2,103	10	**687,321**	260	3,143	5	687,321	**687,321**	53
d2103	2,103	20	**482,926**	733	1,759	41	482,926	**482,926**	28
d2103	2,103	50	302,337	*	360,000	41,277	302,075	302,223	14
d2103	2,103	100	194,920	*	34,614	17,633	194,664	**194,664**	7
d2103	2,103	200	**117,753**	1,828	55	15	117,753	**117,753**	4
d2103	2,103	300	**90,471**	1,133	305	607	90,471	**90,471**	3
d2103	2,103	400	**75,356**	235	8,917	118,965	75,324	**75,324**	3
d2103	2,103	500	**64,006**	5,822	511	4,437	64,006	**64,006**	3
pcb3038	3,038	5	**1,777,835**	1,114	*109	1	0	1,777,835	213
pcb3038	3,038	10	**1,211,704**	134	*64	1	0	1,211,704	118
pcb3038	3,038	20	839,635	247	17,207	25	839,494	**839,494**	64
pcb3038	3,038	50	506,339	223	3,047	41	506,339	**506,339**	27
pcb3038	3,038	100	**351,500**	7,492	1,225	79	351,500	**351,500**	15
pcb3038	3,038	150	**280,128**	3,057	562	71	280,128	**280,128**	11
pcb3038	3,038	200	**237,399**	2,562	564	177	237,399	**237,399**	7
pcb3038	3,038	300	**186,833**	2,977	274	189	186,833	**186,833**	5
pcb3038	3,038	400	**156,276**	454	106	39	156,276	**156,276**	4
pcb3038	3,038	500	**134,798**	704	115	51	134,798	**134,798**	4
fl3795	3,795	5	**1,052,627**	362	*364	1	0	1,052,627	174
fl3795	3,795	10	**520,940**	200	*1,437	1	473,375	520,940	69
fl3795	3,795	20	**319,722**	184	2,484	1	319,722	**319,722**	41
fl3795	3,795	50	**150,940**	103	339	1	150,940	**150,940**	17
fl3795	3,795	100	**88,299**	140	122	1	88,299	**88,299**	9
fl3795	3,795	150	**65,868**	346,396	4,257	4,063	65,868	**65,868**	7
fl3795	3,795	200	**53,928**	84,047	5,731	7,017	53,928	**53,928**	6
fl3795	3,795	300	**39,586**	53,352	12,897	24,973	39,586	**39,586**	4
fl3795	3,795	400	**31,354**	6,761	3,422	11,985	31,354	**31,354**	4
fl3795	3,795	500	**25,976**	770	125	1	25,976	**25,976**	3
rl5934	5,934	10	9,795,785	2,870	*547	1	0	9,792,218	246
rl5934	5,934	20	6,718,043	1,534	*252	1	0	6,716,365	124
rl5934	5,934	50	4,030,518	789	*6,766	1	3,925,161	4,030,263	52
rl5934	5,934	200	1,806,693	295	9,056	653	1,805,530	**1,805,530**	14
rl5934	5,934	300	1,392,419	143	742	45	1,392,419	**1,392,419**	9
rl5934	5,934	400	1,143,962	199	2,906	1,271	1,143,940	**1,143,940**	7
rl5934	5,934	500	972,799	128	357	17	972,799	**972,799**	6
rl5934	5,934	600	847,301	106	376	75	847,301	**847,301**	5
rl5934	5,934	700	751,131	101	361	75	751,131	**751,131**	4
rl5934	5,934	800	675,963	115	415	97	675,958	**675,958**	4
rl5934	5,934	900	612,629	99	320	35	612,629	**612,629**	4
rl5934	5,934	1,000	558,167	110	1,335	2,431	558,167	**558,167**	3
rl5934	5,934	1,100	511,247	103	294	29	511,192	**511,192**	3
rl5934	5,934	1,200	469,775	93	290	11	469,747	**469,747**	3
rl5934	5,934	1,300	433,060	88	311	77	433,060	**433,060**	3
rl5934	5,934	1,400	401,370	88	299	7	401,370	**401,370**	3
rl5934	5,934	1,500	**373,566**	87	310	1	373,566	**373,566**	3
Mean				24,719	2,249	74			

For very large values of p, it is shown that our algorithm can tackle very huge problems of up to 85,900 nodes. It is true that these values of p are extremely large, but it is already an important fact that it is possible to approach them.

For these huge instances, we have the handicap that Popstar cannot be used, and as a consequence, we cannot benefit from an upper bound to reduce the number of nodes in the branching tree. The development of an efficient heuristic for these much larger instances could be an interesting future line of research.

6. Conclusions

The algorithm proposed in this paper, based on a reduced formulation, performs very well in general and is particularly useful for solving instances with large values of p because of both the smaller amount of variables and constraints needed, as well as the efficient management of memory. As a consequence,

Table 4 Computational Results for the *Zebra* Method: Huge Instances

			Zebra				
Instance	*n*	*p*	Time	BB	BLB	BUB	*sdz*
usa13509	13,509	300	35,525	111	59,340,915	**59,340,915**	20
usa13509	13,509	400	53,589	721	50,538,905	**50,538,905**	15
usa13509	13,509	500	84,357	2,131	44,469,860	**44,469,860**	12
usa13509	13,509	600	36,0,000	16,055	39,951,879	39,953,070	10
usa13509	13,509	700	*27,8,455	19,930	36,469,125	$+\infty$	8
usa13509	13,509	800	17,832	1,129	33,635,127	**33,635,127**	7
usa13509	13,509	900	*32,4,464	41,932	31,274,975	$+\infty$	6
usa13509	13,509	1,000	28,9,968	44,831	29,268,216	**29,268,216**	6
usa13509	13,509	2,000	1,700	407	18,230,856	**18,230,856**	2
usa13509	13,509	3,000	3,953	4,437	13,098,935	**13,098,935**	1
usa13509	13,509	4,000	503	333	9,905,715	**9,905,715**	2
usa13509	13,509	5,000	242	209	7,608,605	**7,608,605**	1
sw24978	24,978	1,000	66,352	2,421	1,841,938	**1,841,938**	3
sw24978	24,978	2,000	49,851	12,489	1,197,645	**1,197,645**	1
sw24978	24,978	3,000	18,509	6,447	912,070	**912,070**	1
sw24978	24,978	4,000	*76,077	37,337	738,518	$+\infty$	1
sw24978	24,978	5,000	*61,945	57,547	618,507	$+\infty$	1
sw24978	24,978	6,000	24,031	19,951	528,202	**528,202**	1
sw24978	24,978	7,000	17,968	21,431	456,766	**456,766**	1
sw24978	24,978	8,000	*13,430	20,968	398,436	$+\infty$	1
sw24978	24,978	9,000	*24,757	37,793	348,667	$+\infty$	1
sw24978	24,978	10,000	*26,369	54,817	307,342	$+\infty$	1
ch71009	71,009	50,000	2,228	755	465,220	**465,220**	1
ch71009	71,009	60,000	1,339	473	168,886	**168,886**	1
pla85900	85,900	70,000	650	1	18,977,475	**18,977,475**	1
pla85900	85,900	80,000	605	1	4,512,752	**4,512,752**	1
Mean			54,169	7,070			

we are able to solve much larger problems than the ones solved up to now: 24,978 nodes against 3,795 in a very efficient way and many more nodes for large values of p. Therefore, it can be concluded that our proposed method is competitive with the state-of-the-art algorithms.

Although the reduced formulation is a very useful idea, its success is also attributed to an efficient branch-and-bound framework that makes use of reliability branching (see the Online Supplement) and uses some parameters in a dynamic way.

Now, to solve even larger instances, the development of valid inequalities that strengthen the existing formulation could be considered for future research.

Acknowledgments

The research by the first and third authors was funded by Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica (I+D+I) (Project MTM2009-14039-C06-04 and RDEF funds) and Fundación Séneca (Project 08716/PI/08). The research by the first author was also supported by Comunidad de Madrid and Universidad Carlos III (Project CCG07-UC3M/ESP-3389). The research by the second author was funded in part by Communauté Française de Belgique-Actions de Recherche Concertées (ARC). The authors thank the two anonymous referees for their interesting comments and suggestions, which helped to improve the quality of this paper.

References

Achterberg, T., T. Koch, A. Martin. 2005. Branching rules revisited. *Oper. Res. Lett.* **33**(1) 42–54.

Avella, P., A. Sassano. 2001. On the *p*-median polytope. *Math. Programming A* **89**(3) 395–411.

Avella, P., A. Sassano, I. Vasil'ev. 2007. Computational study of large scale *p*-median problems. *Math. Programming A* **109**(1) 89–114.

Beltrán, C., C. Tadonki, J. P. Vial. 2006. Solving the *p*-median problem with a semi-Lagrangian relaxation. *Comput. Optim. Appl.* **35**(2) 239–260.

Briant, O., D. Naddef. 2004. The optimal diversity management problem. *Oper. Res.* **52**(4) 515–526.

Cánovas, L., S. García, M. Labbé, A. Marín. 2007. A strengthened formulation for the simple plant location problem with order. *Oper. Res. Lett.* **35**(2) 141–150.

Church, R. L. 2003. COBRA: A new formulation of the *p*-median location problem. *Ann. Oper. Res.* **122**(1–4) 103–120.

Cornuéjols, G., G. L. Nemhauser, L. A. Wolsey. 1980. A canonical representation of simple plant location problems and its applications. *SIAM J. Algebraic Discrete Methods* **1**(3) 261–272.

Cornuéjols, G., G. L. Nemhauser, L. A. Wolsey. 1990. The uncapacitated facility location problem. P. B. Mirchandani, R. L. Francis, eds. *Discrete Location Theory*. John Wiley & Sons, New York, 119–171.

Dearing, P. M., P. L. Hammer, B. Simeone. 1992. Boolean and graph theoretic formulations of the simple plant location problem. *Transportation Sci.* **26**(2) 138–148.

Elloumi, S. 2010. A tighter formulation of the *p*-median problem. *J. Combin. Optim.* **19**(1) 69–83.

Elloumi, S., M. Labbé, Y. Pochet. 2004. New formulation and resolution method for the *p*-center problem. *INFORMS J. Comput.* **16**(1) 84–94.

Erlenkotter, D. 1978. A dual-based procedure for uncapacitated facility location. *Oper. Res.* **26**(6) 992–1009.

Ghiani, G., G. Laporte, R. Musmanno. 2004. *Introduction to Logistics Systems Planning and Control*. John Wiley & Sons, Chichester, UK.

Goldengorin, B., D. Ghosh, G. Siersksma. 2004. Branch and peg algorithms for the simple plant location problem. *Comput. Oper. Res.* **31**(2) 241–255.

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

Kariv, O., S. L. Hakimi. 1979. An algorithmic approach to network location problems II: The *p*-medians. *SIAM J. Appl. Math. Oper. Res.* **37**(3) 539–560.

Kolen, A. 1983. Solving covering problems and the uncapacitated plant location problem on trees. *Eur. J. Oper. Res.* **12**(3) 266–278.

Kolen, A., A. Tamir. 1990. Covering problems. P. B. Mirchandani, R. L. Francis, eds. *Discrete Location Theory*. John Wiley & Sons, New York, 263–304.

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

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

Resende, M. G. C., R. F. Werneck. 2004. A hybrid heuristic for the *p*-median problem. *J. Heuristics* **10**(1) 59–88.

ReVelle, C. S., H. A. Eiselt. 2005. Location analysis: A synthesis and survey. *Eur. J. Oper. Res.* **165**(1) 1–19.

Simão, H. P., J. M. Thizy. 1989. A dual simplex algorithm for the canonical representation of the uncapacitated facility location problem. *Oper. Res. Lett.* **8**(5) 279–286.

Xu, N., T. J. Lowe. 1993. On the equivalence of dual methods for two location problems. *Transportation Sci.* **27**(2) 194–199.