Les Cahiers du GERAD ISSN: 0711–2440

A scalable exact algorithm for the vertex p-center problem

C. Contardo, M. Iori, R. Kramer

G–2018–15

March 2018

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

A scalable exact algorithm for the vertex p-center problem

Claudio Contardo [a]
Manuel Iori [b]
Raphael Kramer [b]

[a] *GERAD & Département de management et technologie, ESG UQÀM, Montréal (Québec), Canada, H2X 3X2*

[b] *Dipartimento di Scienze e Metodi dell'Ingegneria, UNIMORE, 42122 Reggio Emilia, Italy*

claudio.contardo@gerad.ca
manuel.iori@unimore.it
raphael.kramer@unimore.it

March 2018
Les Cahiers du GERAD
G–2018–15

Abstract: The vertex p-center problem consists in selecting p centers among a finite set of candidates and assigning a set of clients to them, with the aim of minimizing the maximum dissimilarity between a client and its associated center. State-of-the-art algorithms for the problem are based on the solution of a series of covering subproblems, and are particularly efficient when additional reduction rules are used to limit the size of the subproblems. In fact, these algorithms do not scale well when the cardinality of the dissimilarity matrix grows above a few million entries, as the time and space required to compute and store the matrix start dominating those required for solving the subproblems. We introduce a scalable relaxation-based iterative algorithm that does not rely on the computation of the entire matrix, but rather relies on the computation of a typically much smaller sub-matrix that is only enlarged if deemed necessary. This method can solve to proven optimality p-center problems derived from the TSP library and containing up to one million clients for small but realistic values of p.

Keywords: p-center problem, clustering, facility location, relaxation algorithm

Acknowledgments: This research was partially supported by *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq/Brazil) under Grant GDE 201222/2014-0, *Natural Sciences and Engineering Research Council of Canada* (NSERC/Canada) under *Discovery Grant* 435824-2013, and by project *FAR 2017 "Modellazione Multiscala nelle Scienze, nell'Industria e nella Società"*, Università degli Studi di Modena e Reggio Emilia.

1 Introduction

Facility location is a classical family of combinatorial optimization problems that requires to locate a set of facilities and supply a set of customers from the chosen locations with the objective of minimizing location and supply costs. The problem is of interest because has many applications in fields such as logistics and data mining. The location of intermediate consolidation centers (Grangier et al., 2016), firehouses (Plane and Hendrick, 1977), hospitals (Chu and Chu, 2000) or mailboxes (Labbé and Laporte, 1986), are all examples arising in logistics management. In computer science, and most notably in data mining, several clustering problems can be modeled as pure facility location applications (Hansen et al., 2009).

The *vertex p-center problem* (PCP) is a particular facility location problem that has applications in both of these fields. It can be defined as follows. Given an integer $p \geq 2$, a set $N = \{1, 2, \ldots, n\}$ of clients, a set $M = \{1, 2, \ldots, m\}$ of candidate centers (with $m > p$), and a dissimilarity matrix $d : N \times M \longrightarrow \mathbb{R}_+$, find a subset $V \subset M$ of cardinality p such that the maximum dissimilarity between a client and its closest center in V is minimized. In facility location problems, the dissimilarity between two points (clients or facilities) is usually measured in terms of distance. Consequently, in this paper, the term "distance" is also adopted as synonym of "dissimilarity". Note, however, that in many applications, such as in clustering problems, the dissimilarity matrices do not need to satisfy the conditions of distance metrics.

Classical exact solution methods for the PCP (e.g., Daskin, 1995) are built on a *binary search* (BS) algorithm that exploits the following observation. For a fixed radius r, the problem of deciding whether it is possible to find a subset $V \subset M$ of cardinality p such that every point in N (a client) is at a distance r or less from its closest point in V (selected center) can be stated as an instance of the well-known *set covering problem* (SCP). Let `SetCover`(N, M, p, r) be an exact algorithm for this particular SCP that returns a feasible solution $V \subset M$ of cardinality p (if feasible), or $\emptyset$ (if infeasible). The BS algorithm for the PCP is provided in Algorithm 1. First, a lower bound l and an upper bound u is computed. Then, set $r = \lfloor (l+u)/2 \rfloor$ and let $V \leftarrow$ `SetCover`(N, M, p, r). If $V = \emptyset$ (meaning that the decision subproblem is infeasible), we let $l \leftarrow r + 1$, otherwise (i.e., if the decision subproblem is feasible) we let $u \leftarrow r$. In both cases we iterate using the updated values of l and u. The procedure stops whenever $l = u$, in which case the last set V found is *an* optimal solution of the problem. In Algorithm 1, `PCPlowerbound` and `PCPheuristic` denote any two procedures used to instantiate the values of l and u, respectively. We highlight the fact that the PCP may contain multiple optimal solutions, as this plays an important role in the performance of our method.

Algorithm 1 Binary search algorithm for the PCP

Require: N, M, p
Ensure: An optimal set of centers V^* of cardinality p
1: $l \leftarrow$ `PCPlowerbound`(N, M, p) ▷ Lower bound procedure
2: $(r^*, V^*) \leftarrow$ `PCPheuristic`(N, M, p) ▷ Heuristic procedure
3: $u \leftarrow r^*$
4: **while** $l < u$ **do**
5: $r \leftarrow \lfloor (l + u)/2 \rfloor$
6: $V \leftarrow$ `SetCover`(N, M, p, r) ▷ Exact SCP procedure
7: **if** $V = \emptyset$ **then**
8: $l \leftarrow r + 1$
9: **else**
10: $u \leftarrow r^* \leftarrow r$
11: $V^* \leftarrow V$
12: **end if**
13: **end while**
14: **return** (r^*, V)

Solving the decision set-covering subproblem as an *integer linear program* (ILP) requires the precomputation of a covering matrix $cov : N \times M \longrightarrow \{0, 1\}$, where for every pair (u, v) of points, with $u \in N$ and $v \in M$,

$$cov(u, v) = \begin{cases} 1 & \text{if } d(u, v) \leq r \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

For every client $u \in N$, we further define $ng(u) = \{v \in M : cov(u, v) = 1\}$ as the set of candidate centers that are at a distance r or less from u. Let, for every $v \in M$, x_v be a binary variable that takes the value 1 iff v is used as a center. The ILP is then

$$\min \sum_{v \in M} x_v \tag{2}$$

$$\sum_{v \in ng(u)} x_v \geq 1 \qquad u \in N, \tag{3}$$

$$x_v \in \{0, 1\} \qquad v \in M. \tag{4}$$

Storing the matrix *cov* has the advantage of allowing the efficient use of the reduction rules for the solution of a potentially much smaller ILP, but requires $\mathcal{O}(nm)$ time and space complexity. Even if one decides not to explicitly store the covering matrix *cov*, constraints (3) still require to at least compute the dissimilarities among each pair of points. This quadratic complexity may not be an issue in PCPs containing a few hundred or a few thousand points, but may rapidly become prohibitive for larger instances. This is a relevant issue in clustering problems arising from data mining applications, where potentially millions of data points might need to be classified.

In this paper, we propose an exact algorithm that aims at filling the gap that we have just described when massive datasets containing potentially millions of points need to be classified according to the PCP. The proposed algorithm is based on a row generation approach that iteratively solves small subproblems until an optimal solution for the original problem is found (see, e.g., Chen and Handler 1987; Aloise and Contardo 2018). It improves the algorithm of Chen and Chen (2009), by using an efficient strategy that mitigates the negative impact of primal degeneracy that usually occurs when solving PCPs by means of relaxation-based algorithms. Moreover, the proposed algorithm does not rely on the pre-computation or storage of the entire covering matrix *cov* at each iteration. It is then scalable in the sense that *cov* is only computed and stored partially. As such, our algorithm can handle instances containing up to one million points within reasonable time limits, for small but realistic values of p.

The remainder of this paper is organized as follows. Section 2 reviews the related literature. Section 3 presents the framework of the relaxation-based algorithm and the proposed improvements. Extensive computational experiments and results are provided in Section 4. Conclusions are drawn in Section 5.

2 Literature review

The PCP was introduced by Hakimi (1964), who presented and solved the absolute 1-center problem on a graph. In this first version (also known as *absolute p-center problem*), the center can be located either on the edges or on the vertices of an input graph. Later, Minieka (1970) extended the problem to the case with $p > 1$ and proposed a method to restrict the continuous set of candidate centers to a discrete set of points, without losing optimality. To solve the discretized problem, he proposed an iterative algorithm that solves an SCP at each iteration. A mathematical formulation for the case in which centers are restricted to be located on the vertices of a graph (the vertex p-center problem – PCP – that we study in this paper) was introduced by Daskin (1995). He solved the problem by means of a BS algorithm that, similarly to Minieka's approach, iteratively invokes an SCP. Later, Daskin (2000) adopted the same BS idea, but solved *maximum covering problems* (MCPs) instead of SCPs. Elloumi et al. (2004) proposed a new formulation for the PCP whose continuous relaxation value strictly dominates the one provided by Daskin's model.

Good lower bounds for the PCP can be obtained by using Algorithm 1 and by setting it to obtain at each iteration a relaxed continuous solution at the subproblems instead of an optimal solution (line 6). Such approach was adopted by Elloumi et al. (2004) and Ilhan et al. (2002) to obtain initial bounds for their algorithms. The former then performed a BS within the obtained bounds, while the latter performed a sequential search starting from the obtained lower bound.

In contrast with the previous approaches, Caruso et al. (2003) proposed exact and heuristic algorithms based on the idea of strong and weak dominance rules (i.e., rules that guarantee or not the conservation of optimal solutions). Chen and Handler (1987) and Chen and Chen (2009) presented relaxation-based iterative algorithms that consider only subsets of clients. If the largest dissimilarity between a client not in the subset and its closest open center in the solution of the relaxed problem is lower than or equal to the solution value, then this solution is also feasible (and hence optimal) for the original instance. If not, the subset is updated by adding a constant number of clients. Similarly, Aloise and Contardo (2018) solved the minimax diameter clustering problem considering only a sample set of points and increasing it iteratively by means of tailored procedures which led to the optimal solution of very large instances.

The idea of solving reduced subproblems was also adopted by Irawan et al. (2015), but by aggregating clients. The aggregated instances were solved heuristically, and results for instances containing up to 71,009 nodes were reported. For an overview of aggregation methods for location models we refer the interested reader to the survey by Francis et al. (2009).

To the best of our knowledge the most recent exact approach for solving the PCP was presented by Calik and Tansel (2013). The authors proposed a set covering based formulation that contains a block of covering constraints for every distinct value of the dissimilarity matrix. Based on this formulation, they developed a search algorithm that solves a restricted model containing only two blocks of covering constraints. Differently from the BS algorithm, and depending on the solution value, both upper and lower bounds may be updated at the same iteration. However, as reported by the authors, on large-size instances solving classical SCPs at each iteration performs better than using the model with the two blocks of constraints.

As recent publications on PCP variants, we cite the works of Lu and Sheu (2013) on the robust PCP, Chen and Chen (2013) on the PCP with multiple coverage, Kramer et al. (2018) on the capacitated PCP, Martínez-Merino et al. (2017) on the probabilistic PCP, Bai et al. (2017) on the connected PCP, and Callaghan et al. (2017) on the PCP in the plane.

3 A row generation algorithm

To solve the PCP we propose a row generation algorithm that is inspired on the observation that, for a given radius, covering a subset of points is computationally easier than covering all the points in the dataset (Chen and Handler, 1987; Chen and Chen, 2009). More formally, let us define, for a given subset $U \subseteq N$, $\texttt{PCP}(M, p, U)$ as the problem of finding a subset $V \subset M$ of cardinality p such that the maximum dissimilarity between a point (client) in U and its closest point (center) in V is minimized. Let r^* and V^* be the optimal radius, i.e., the optimal dissimilarity value, and an optimal set of centers, respectively. We have that

$$r^* \leq z^*, \tag{5}$$

where z^* is the optimal value of the original PCP (and r^* is thus a lower bound). If the maximum dissimilarity between any client in N and its closest center in V^* is lower than or equal to r^*, then V^* is also an optimal solution to the original problem, and thus $z^* = r^*$. Otherwise, subset U should be updated to include nodes from $N \setminus U$. The algorithm consists in updating U, solving $\texttt{PCP}(M, p, U)$, and checking (e.g., by inspection) whether V^* is also optimal for the original PCP (see Algorithm 2). To ensure optimality, an exact procedure must be executed to solve the $\texttt{PCP}(M, p, U)$ subproblems.

A critical component of Algorithm 2 concerns the way subset U is updated. Chen and Handler (1987) updated it by adding a single node $k \in N \setminus U$ that maximized $\min\{d(k, j) : j \in V^*\}$ (i.e., the most dissimilar client to V^*). It must be noted that this strategy does not ensure that the lower bound r^* will be increased in the next iteration, due to the possible existence of multiple solutions having the same cost. This situation is illustrated in Figure 1, where the choice of updating U by adding the worst dissimilarity point not in U would lead to a new optimal solution of the same cost that that of the previous iteration. In fact, the addition to U of the farthest point at the right bottom of the figure would lead to a same radius solution with just a change (highlighted by an arrow) in the selected centers. The addition to U of the closer point at the top

Algorithm 2 Row generation algorithm for the PCP

Require: N, M, p
Ensure: An optimal set of centers V^* of cardinality p
$U \leftarrow$ `InitializeSubset(`N`)` ▷ e.g., random $p+1$ nodes from N
repeat
  $(r^*, V^*) \leftarrow$ `solvePCP(`M, p, U`)` ▷ e.g., using Algorithm 1
  $z \leftarrow \max\{\min\{d(i,j) : j \in V^*\} : i \in N \setminus U\}$
  if $z \leq r^*$ **then**
    $W \leftarrow \emptyset$ ▷ optimal solution found
  else
    $W' \leftarrow \{k \in N : \min\{d(k,j), j \in V^*\} > r^*\}$
    $W \leftarrow$ `SelectSubsetNodesFrom(`W'`)`
    $U \leftarrow U \cup \{W\}$
  end if
until $W = \emptyset$
return (r^*, V^*)

left part of the figure would lead instead to an increase in the radius. Chen and Chen (2009) attempted to circumvent this issue by adding a set $\{u_1, u_2, \ldots, u_\kappa\}$ of κ nodes at each iteration, in such a way that $\sum_{i=1}^{\kappa} \min\{d(u_i, j) : j \in V^*\}$ is maximized. The latter strategy improves the former one by avoiding several useless iterations (i.e., iterations that do not improve r^*). However, since κ is a fixed parameter, subset U tends to increase relatively fast, leading to memory issues when solving very large instances.

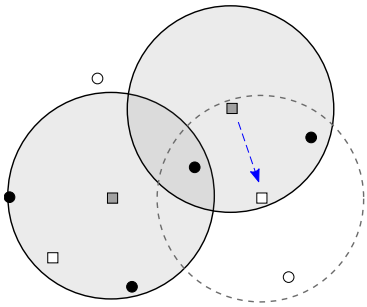

Figure 1: Illustration of a case in which the strategy of updating U by adding the farthest node (from $N \setminus U$) does not ensure a lower bound improvement when solving the updated problem. Circle nodes represent the set N, square nodes the set M, black circle nodes the current set U, and gray square nodes the current set V^*

Based on the aforementioned observations, the main contributions of our algorithm are the following:

1. At each iteration, it provides a pool $\mathcal{P}$ of optimal solutions for the subproblem `PCP`(M, p, U), instead of a single one as in the previous approaches;
2. The number of clients that it adds to U at each update is variable. This avoids the necessity of parameter tunning, and satisfies the improvement suggested by Chen and Chen (2009) that "*the best approach is a varying value of* κ";
3. It updates U by adding a small number of clients (from $N \setminus U$) that makes all the solutions in $\mathcal{P}$ infeasible. This slows down the increase of $|U|$ and reduces memory issues;
4. It does not require the computation of the full dissimilarity matrix d, and makes use of the dominance rules by Francis et al. (1992) to eliminate rows and columns. This makes the algorithm scalable for solving very large instances;
5. At the end of each iteration, after solving the relaxed problem, a completion heuristic is invoked with the aim of finding a solution of cost r^*, if any, that is feasible (and hence optimal) for the original instance.

These contributions are discussed more in details in the next subsections. A pseudo-code of the proposed approach is given in Algorithm 3.

Algorithm 3 Improved row generation algorithm RG_PCP(N, M, p)

Require: N, M, p
Ensure: An optimal set of centers V^* of cardinality p
  $W = U \leftarrow$ InitializeSubset(N) ▷ Subsection 3.1
  repeat
    $(r^*, V^*) \leftarrow$ SolvePCP(M, p, U) ▷ Subsection 3.2
    $z \leftarrow \max\{\min\{d(i,j) : j \in V^*\} : i \in N \setminus U\}$
    if $z \leq r^*$ **then**
      $W \leftarrow \emptyset$ ▷ optimal solution to PCP(M, p, N) has been found
    else
      $\mathcal{P} \leftarrow$ ComputePoolSolutions(N, U, V^*) ▷ Subsection 3.3
      $(V', W) \leftarrow$ ComplHeur&UpdSample$(N, M, p, U, \mathcal{P})$ ▷ Subsection 3.4
      if $W = \emptyset$ **then** ▷ optimal solution to PCP(M, p, N) has been found
        $V^* \leftarrow$ ExtractOptimalCenters(V')
      end if
    end if
  until $W = \emptyset$
  return (r^*, V^*)

3.1 Initialization of the subset U

The initial subset $U \subset N$ is obtained by a procedure that randomly selects a subset U' of $p+2$ nodes and solves the subproblem PCP(M, p, U') for a certain number of iterations. Denoting by r the solution cost of PCP(M, p, U') and by r' the maximum cost obtained from all iterations, the procedure is stopped as soon as a maximum of Δ consecutive iterations without increasing r' is reached. Note that r' is a lower bound to PCP(M, p, N). This procedure is similar to the one developed by Aloise and Contardo (2018) for a minimax diameter clustering problem.

3.2 Solving subproblem PCP(M, p, U)

The subproblem PCP(M, p, U) is solved by an algorithm based on both exponential search and BS. First of all, by using the current subset U as customer set, for a given radius r we execute the rules of Francis et al. (1992) to reduce the size of M, thus creating a reduced set $M(U, r)$ of candidate centers to cover the nodes in U with clusters of radii at most r. The exponential search is executed indexed by t, and initialized by $t = 0$. At iteration t, we let $u_t = (l-1) + 2^t$ and solve the problem SetCover$(U, M(U, u_t), p, u_t)$. This iterative process is performed until problem SetCover$(U, M(U, u_t), p, u_t)$ becomes feasible, thus providing an upper bound on the optimal value of PCP(M, p, U). At this point, we perform a BS algorithm in the range $[u_{t-1} + 1, u_t]$ (see Algorithm 1). At the end, the optimal value r^* and the optimal centers V^* are returned. This procedure avoids the quadratic complexity $O(nm)$ by solving SCPs that are usually several orders of magnitude smaller than the original problem.

3.3 Computing a pool of solutions

Once an optimal solution V^* of cost r^* is obtained for PCP(M, p, U), an alternative *maximum coverage problem* (MCP-a) is solved with the aim of obtaining a new solution of the same cost but with a certain diversification among the centers. The MCP-a is formally defined as follows. Let $U' = \{u \in N \setminus U : d(u, v) > r^*, \text{ for all } v \in V^*\}$ be the set of *uncovered* clients. Let $\mathcal{U}$ be a set of *optional* clients, initialized with β clients chosen at random from U'. Every client $u \in \mathcal{U}$ is associated with a profit p_u, initialized as

$$p_u = \begin{cases} p_u^- & \text{if } \min\{d(u, v) : v \in V^*\} \leq r^* \\ p_u^+ & \text{otherwise,} \end{cases} \tag{6}$$

with $p_u^+ > p_u^-$. In this way, clients not covered in the previous solution are preferable to be covered in the new attempt. Let also x_v be a binary variable taking value 1 if center $v \in M$ is selected, 0 otherwise, and y_u be a

binary variable stating whether client $u \in \mathcal{U}$ is covered or not. We recall that $ng(u) = \{v \in M : d(u,v) \leq r^*\}$ is the set of candidate centers that can cover u. The MCP-a can thus be modeled as:

$$\max \quad \sum_{u \in \mathcal{U}} p_u y_u \tag{7}$$

$$\sum_{v \in ng(u)} x_v \geq 1 \qquad u \in U, \tag{8}$$

$$\sum_{v \in ng(u)} x_v \geq y_u \qquad u \in \mathcal{U}, \tag{9}$$

$$\sum_{v \in M} x_v \leq p, \tag{10}$$

$$x_v \in \{0,1\} \qquad v \in M, \tag{11}$$

$$y_u \in \{0,1\} \qquad u \in \mathcal{U}. \tag{12}$$

The objective function (7) maximizes the total profit of the covered optional clients. Constraints (8) ensure that every client in U is covered, while constraints (9) state that the profit from client $u \in \mathcal{U}$ can be collected only if at least one of the centers in $ng(u)$ is selected. The total number of chosen centers is limited to p by constraint (10). Constraints (11) and (12) provide the binary conditions.

Each time that a new (optimal) solution to this problem is found, one may add a local branching constraint to cut this solution, update the profits $(p_u)_{u \in \mathcal{U}}$ and then execute the enlarged problem to eventually find an alternative solution to the problem. However, solving such ILP may become prohibitively too time consuming after a few iterations, especially towards the end of the algorithm when subsets U and $\mathcal{U}$ may contain a large number of points.

Let us recall however that the solution to subproblem $\texttt{PCP}(M,p,U)$ already provides at least one solution for the pool $\mathcal{P}$. Therefore, we have implemented an iterated local search (ILS, see Lourenço et al. 2010) algorithm that enlarges $\mathcal{P}$ by heuristically solving MCP-a multiple times and by adjusting the profits after each successful iteration. Moreover, to reduce the complexity of this ILS even further, we replaced the set M by the set $M(U,r^*)$ obtained from discarding dominated candidate centers from M.

The outline of the aforementioned ILS is presented in Algorithm 4. Let S and S_{best} be two sets of p centers of a generic MCP-a solution. The sets S and S_{best}, and the pool $\mathcal{P}$ are firstly initialized with the same set of centers of the last $\texttt{PCP}(M,p,U)$ solution (line 1). The set $\mathcal{U}$ is initialized with β random nodes from $N \setminus U$ (lines 2–3). At each iteration (lines 5–19) the incumbent solution S_{best} is modified by a perturbation mechanism (line 6), and the current solution S is possibly improved by a local search procedure (line 7). If S improves the incumbent solution (line 8), it is added to $\mathcal{P}$ (line 12) and the profits p_u, for all $u \in \mathcal{U}$, are updated according to Equation (6) (line 13). In case all nodes in $\mathcal{U}$ are covered (line 9), set $\mathcal{U}$ is augmented with β nodes selected at random from $N \setminus (U \cup \mathcal{U})$ and uncovered by all solutions of $\mathcal{P}$ (line 10). The ILS is run until η_{iter} successive iterations of perturbation and local search are performed without improvement on S_{best}, and returns $\mathcal{P}$ in output.

Both local search and perturbation phases are based on *exchange* moves that replaces a center from S with a center from $M(U,r^*) \setminus S$, satisfying constraints (8)–(10). The perturbation mechanism applies α random *exchange* moves, while the local search procedure repetitively performs the best *exchange* move in the neighborhood of S, until no improvement can be obtained.

3.4 Completion heuristic and update of sample U

Keeping the same notation adopted in Sections 3.2 and 3.3, let r^* denote the optimal cost of subproblem $\texttt{PCP}(M,p,U)$. After solving $\texttt{PCP}(M,p,U)$ (as described in Section 3.2) and computing the pool $\mathcal{P}$ of solutions (as described in Section 3.3), a completion heuristic is invoked with the aim of looking for a solution of cost r^* that is feasible/optimal for the original problem $\texttt{PCP}(M,p,N)$, or looking for a subset W of $N \setminus U$ whose

addition to U would necessarily entail an increase in the radius needed to cover all points, for each of the solutions in $\mathcal{P}$, using the current clustering.

Algorithm 4 Computing pool of solutions $\mathcal{P}$ – ILS algorithm

Require: N, U, V^*
Ensure: Set of solutions $\mathcal{P}$
$\mathcal{P} \leftarrow S_{\text{best}} \leftarrow S \leftarrow V^*$
`InitializeSet`$(\mathcal{U})$ ▷ β nodes randomly chosen from $N \setminus U$
`InitializeProfits`$(N, S, \mathcal{U})$ ▷ According to Equation (6)
$iter \leftarrow 1$
while $iter \leq \eta_{iter}$ **do**
  $S \leftarrow$ `Perturbation`(N, S_{best})
  $S \leftarrow$ `LocalSearch`(N, S)
  if $z(S) > z(S_{\text{best}})$ **then**
    if All nodes from $\mathcal{U}$ are covered by S **then**
      `UpdateSet`$(\mathcal{U}, S)$ ▷ Add β nodes randomly chosen from $N \setminus (U \cup \mathcal{U})$
    end if
    $\mathcal{P} = \mathcal{P} \cup S$
    `UpdateProfits`$(N, S, \mathcal{U})$ ▷ According to Equation (6)
    $S_{\text{best}} \leftarrow S$
    $iter \leftarrow 1$
  else
    $iter \leftarrow iter + 1$
  end if
end while
return $\mathcal{P}$

First of all, for every solution $s \in \mathcal{P}$, we let $V^*(s) = \{v_1^s, \ldots, v_p^s\}$ be the centers associated to solution s as found by the ILS procedure described in Section 3.3. Because every node in U may be covered by more than one center, we further force the assignment of every node $u \in U$ to its closest center. At the end of this procedure, each node $u \in U$ is assigned to one and only one center in $V^*(s)$. This induces p non-intersecting clusters $\{U_s^i : i = 1 \ldots p\}$. For each such cluster U_s^i, we further denote $M(s, i, r^*)$ the set of nodes in M that could be used as centers of the cluster U_s^i without exceeding the radius r^*. To alleviate the computational complexity of computing these sets (as it would involve the evaluation of $m - p$ nodes for each such pair (s, i)), we actually restrict this set to contain nodes in $M(U, r^*)$, which is the set of non-dominated centers previously computed. We call $M(s, i, r^*)$ as the set of *potential centers* of the i-th cluster of solution s. The computation of the potential centers of every cluster for every solution $s \in \mathcal{P}$ allows to encode in an implicit and efficient way many additional solutions to those of $\mathcal{P}$. This idea is explored to improve the convergence of the algorithm as described next.

Let $M(\mathcal{P}, r^*) = \cup\{M(s, i, r^*) : s \in \mathcal{P}, i = 1 \ldots p\}$ be the set of all potential centers from all solutions in $\mathcal{P}$ combined. For every point $u \in N \setminus U$, we let $\omega(u, \mathcal{P}, r^*) = \min\{d(u, v) : v \in M(\mathcal{P}, r^*)\}$ be the minimum dissimilarity of u with respect to any of the potential centers. We use this quantity to sort all points in $N \setminus U$ in non-increasing order of $\omega(\cdot, \mathcal{P}, r^*)$.

Now, for every solution $s \in \mathcal{P}$, we let $f(s)$ be an integer flag equal to 0 if the solution s is still possible to be enlarged further without the need of increasing the radius r^*, and equal to $i \in \{1 \ldots p\}$ if for the i-th cluster at solution s the set $M(s, i, r^*)$ becomes $\emptyset$. We also let, for every $s \in \mathcal{P}$ and for every cluster $i = 1 \ldots p$, $W(s, i)$ be the set of points whose insertion at cluster i in solution s would entail a shrinking of the set of potential centers $M(s, i, r^*)$. This set is initialized as $\emptyset$ for pair $(s, i) \in \mathcal{P} \times \{1 \ldots p\}$.

Starting from the first point u in $N \setminus U$, according to the ordering induced by $\omega(u, \mathcal{P}, r^*)$, we try to insert u, for every solution s such that $f(s) = 0$, in its closest cluster, that is in the cluster whose closest potential center is as close as possible. Let us say that the closest potential center to u lies in cluster i. The set of potential centers $M(s, i, r^*)$ is then updated by removing the potential centers whose dissimilarities to u exceed r^*. If the set of potential centers $M(s, i, r^*)$ is reduced, then we also add u to the set $W(s, i)$. In the eventuality that there is no such possible insertion into any cluster of s, the flag $f(s)$ is set to i, meaning that it is no longer possible to extend the solution s because there are no potential centers at cluster i that can make that possible, and the point u is also added to $W(s, i)$.

In the eventuality that all the solutions are no longer possible to be extended (meaning that $f(s) > 0$ for every $s \in \mathcal{P}$), we return $W = \cup\{W(s, f(s)) : s \in \mathcal{P}\}$. Otherwise, the method indeed has identified one solution $s \in \mathcal{P}$ that can be extended to a solution of the full problem, using a radius of r^*, and then $W = \emptyset$ is returned, along with a set of centers built from picking arbitrarily one center from each of the potential centers remaining, for each of the clusters. The pseudo-code in Algorithm 5 describes in more detail this procedure.

Algorithm 5 Completion heuristic

Require: $N, U, r^*, M(U, r^*), p, \mathcal{P}$
Ensure: Optimal set of centers V^*, or a set W used to enlarge the set U
$\{W(s,i) \leftarrow \emptyset : s \in \mathcal{P}, i = 1 \dots p\}$
$\{f(s) \leftarrow 0 : s \in \mathcal{P}\}$
Sort the nodes in $N \setminus U$ in non-increasing order of $\omega(\cdot, \mathcal{P}, r^*)$
for $u \in N \setminus U$ **do**
  $\mathcal{Q} \leftarrow \{s \in \mathcal{P} : f(s) = 0\}$
  for $s \in \mathcal{Q}$ **do**
    $i \leftarrow \arg\min\{\min\{d(u,v) : v \in M(s,i,r^*)\} : i = 1 \dots p\}$
    $R \leftarrow \{v \in M(s,i,r^*) : d(u,v) > r^*\}$
    if $R \neq \emptyset$ **then**
      $W(s,i) \leftarrow W(s,i) \cup \{u\}$
      $M(s,i,r^*) \leftarrow M(s,i,r^*) \setminus R$
      if $M(s,i,r^*) = \emptyset$ **then**
        $f(s) \leftarrow i$
      end if
    end if
  end for
  if $f(s) > 0$ for every $s \in \mathcal{P}$ **then**
    return $W \leftarrow \cup\{W(s, f(s)) : s \in \mathcal{P}\}$
  end if
end for
return some $s \in \mathcal{P}$ such that $f(s) = 0$

4 Computational experiments

The proposed row generation algorithm was evaluated by solving instances derived from the TSPLIB with $p \in \{2, 3, 5, 10, 15, 20, 25, 30\}$ and $M = N$. It was coded in *Julia* (Bezanson et al., 2017) with *JuMP* modeling language (Dunning et al., 2017), and executed on an Intel Xeon E5462 2.8 GHz with 16 GB of RAM. The mathematical models have been solved with *Gurobi 7.0.1* (Gurobi Optimization, 2017), using the default parameters and setting it to run on a single thread. From preliminary computational experiments, the following parameter values were adopted in our algorithms: $\eta_{iter} = 3$, $\Delta = 10$, $\beta = 5$, $p_u^- = 1$, $p_u^+ = 3$, and $\alpha = 3$. Finally, a time limit of 24 hours was imposed for each run. In total 360 instances have been considered, with n varying from 1,621 to 1,000,000.

The results that we obtained are reported in Tables 3–10 in the A, and summarized in Tables 1 and 2 below. The former aggregate the results by values of p, and the latter by values of n. Each row in the tables report the following aggregated information: column *#inst.* reports the number of instances aggregated; column *#optimal* reports the number of instances solved to proven optimality; column *gap(%)* gives the percentage gap computed as $100(UB - LB)/UB$, where LB and UB are the lower and upper bounds at the end of the execution of the algorithm, respectively; column *time(s)* gives the runtime in seconds for the full execution; and column $|U|$ reports the number of nodes in the sample set U at the end of the execution of the algorithm.

Table 1 shows a very good performance of the algorithm for solving instances with small values of p. For $p \in \{2, 3, 5, 10\}$, all but one of the instances could be solved to proven optimality within a reasonable CPU time (given the size of the instances considered). This highlights the efficiency of the algorithm for solving instances with $p \leq 10$. It can be noticed also that the performance of the algorithm drops as p increases. Table 2 shows also a good performance of the algorithm for solving instances with $|N| = |M| \leq 10^4$, where, for each line, at least 6 out 8 are solved to optimality within the time limit. In total, about 80% of the instances could be solved to proven optimality.

Table 1: Computational evaluation of the improved row generation algorithm – Results aggregated by value of p (in each row, n varies from 1,621 to 1,000,000)

p	#inst.	#optimal	gap(%)	time(s)*	$\|U\|$*
2	45	45	0.00	508.80	14.20
3	45	45	0.00	541.69	23.42
5	45	45	0.00	1,439.20	53.18
10	45	44	0.04	2,797.02	160.14
15	45	36	1.10	4,474.32	273.92
20	45	29	2.83	9,106.97	361.86
25	45	21	4.65	7,487.41	380.76
30	45	18	6.91	8,944.81	415.56
sum/avg.	360	283	1.94	3,457.68	165.95

*Considering only instances solved to proven optimality.

Figures 2–3 report the average CPU times required by the algorithm to solve the instances (separated by values of p) and to compute only the full dissimilarity matrix d (label *dmc*). From the left part of Figure 2, it can be observed that instances with $n < 10^4$ and small values of p can be solved to optimality in less than 15 seconds. From the right part of Figure 2, we can notice that the time required by the algorithm to find an optimal solution for instances with $n \geq 10^4$ and small values of p is usually lower than the time required to only compute the full dissimilarity matrix (instances `lrb744710` and `world-rand-1000000` require more than 24 hours for *dmc*). Even if CPU time was not an issue, memory could be. Indeed, to store a dissimilarity matrix for a problem containing 1M nodes, 3.63 TB of RAM would be necessary, far beyond the reach of personal computers or workstations of common use. Clearly, for these instances any algorithm requiring the explicit computation or storage of d is dominated by our row generation approach.

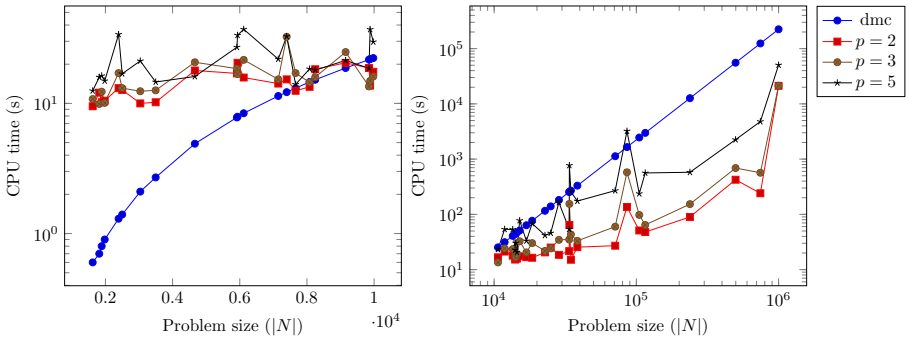

Figure 2: Computational times for instances with $n(= m) < 10^4$ (left), $n(= m) > 10^4$ (right), and $p \in \{2, 3, 5\}$

A somehow opposite behavior can be observed for instances with $p \in \{15, 20, 25, 30\}$. In these cases, regardless the size of the instances, the algorithm consumes more CPU time to solve the problems to optimality (Figure 3). This is justified because as p increases the number of degenerate solutions increases. This clearly compromises the performance of the algorithm, and suggests the necessity of an even more efficient method for computing the pool of solutions $\mathcal{P}$. Moreover, it suggests that the minimum cardinality of subset $U \subseteq N$ that ensures that the optimal cost of $\texttt{PCP}(M, p, U)$ is also optimal for $\texttt{PCP}(M, p, N)$ tends to be large on instances with large p. In particular, in the right part of Figure 3 we notice that several instances could not be solved within the time limit of 24 hours.

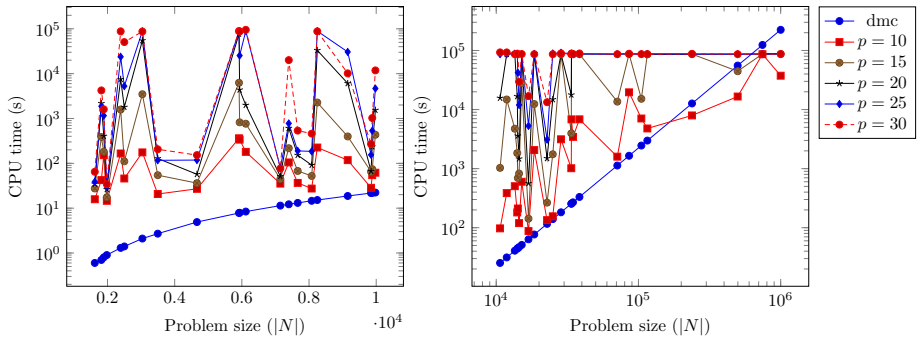

Figure 3: Computational times for instances with $n(=m) < 10^4$ (left), $n(=m) > 10^4$ (right), and $p \in \{10, 15, 20, 25, 30\}$

Table 2: Computational evaluation of the improved row generation algorithm – Results aggregated by value of n (in each row, p varies from 2 to 30)

instance name	n	#inst.	#optimal	gap(%)	time(s)*	$\|U\|$*
rw1621	1621	8	8	0.00	26.80	103.25
u1817	1817	8	8	0.00	1,092.79	238.88
rl1889	1889	8	8	0.00	441.93	191.75
mu1979	1979	8	8	0.00	19.83	57.63
pr2392	2392	8	7	0.90	4,725.61	230.86
d15112-2500	2500	8	8	0.00	7,236.10	221.75
pcb3038	3038	8	6	2.03	9,749.33	221.33
nu3496	3496	8	8	0.00	70.41	130.13
ca4663	4663	8	8	0.00	55.71	92.75
rl5915	5915	8	6	2.47	13,895.77	253.33
rl5934	5934	8	7	1.21	4,413.86	234.29
tz6117	6117	8	6	2.66	501.37	157.67
eg7146	7146	8	8	0.00	42.33	61.75
pla7397	7397	8	8	0.00	2,729.11	179.63
ym7663	7663	8	8	0.00	128.24	110.25
pm8079	8079	8	8	0.00	107.75	122.00
ei8246	8246	8	6	1.89	5,949.43	220.00
ar9152	9152	8	8	0.00	5,915.91	249.25
ja9847	9847	8	8	0.00	79.28	83.88
gr9882	9882	8	8	0.00	248.36	142.75
kz9976	9976	8	8	0.00	2,323.13	236.50
fi10639	10639	8	6	1.46	2,811.60	186.33
rl11849	11849	8	5	4.39	3,058.46	163.40
usa13509	13509	8	5	2.54	1,067.90	101.80
brd14051	14051	8	6	1.69	3,131.90	190.50
mo14185	14185	8	7	0.66	6,669.69	231.71
ho14473	14473	8	8	0.00	5,494.78	276.25
d15112	15112	8	5	3.51	5,724.98	186.00
it16862	16862	8	8	0.00	2,875.01	217.38
d18512	18512	8	5	3.64	2,915.80	194.60
vm22775	22775	8	8	0.00	2,299.19	230.13
sw24978	24978	8	6	1.97	2,806.27	176.83
fyg28534	28534	8	4	4.70	838.60	123.50
bm33708	33708	8	6	1.73	3,814.43	203.83
pla33810	33810	8	4	5.13	1,953.55	92.75
bby34656	34656	8	4	6.36	941.08	122.75
pba38478	38478	8	4	5.26	1,766.13	135.00
ch71009	71009	8	5	3.11	3,114.60	129.00
pla85900	85900	8	4	6.33	5,875.28	89.75
sra104815	104815	8	5	2.29	4,530.02	176.40
usa115475	115475	8	4	4.74	1,364.60	77.25
ara238025	238025	8	4	5.84	2,193.78	84.50
lra498378	498378	8	5	3.80	12,909.82	127.80
lrb744710	744710	8	3	5.94	1,863.27	51.00
world-rand-1000000	1000000	8	4	6.49	32,619.13	102.25
sum/avg.	-	360	283	1.94	3,457.68	165.95

*Considering only instances solved to proven optimality.

5 Conclusions

This paper dealt with the PCP, a variant of the facility location problem that consists in selecting p centers from a set M of candidates to service a set N of clients, aiming at the minimization of the maximum dissimilarity between a customer and its closest center. To solve the problem, we proposed a row generation algorithm that iteratively solves smaller subproblems by considering only a subset $U \subseteq N$ of clients. Such subset is updated at the end of each iteration by adding a variable number of points and taking into account multiple equivalent solutions obtained from the last solved iteration. Moreover, the proposed algorithm does not rely on the pre-computation or storage of the full dissimilarity matrix involving clients and candidate facilities and use a number of efficient heuristic procedures. These ideas make the algorithm scalable for solving very large instances.

Extensive computational experiments have been performed on instances containing up to one million clients (and centers) and confirmed the scalability of the algorithm. Most of the instances with small values of p could be solved to optimality, and those with $n \geq 10^4$ could be solved within a time even shorter than that required to compute the full dissimilarity matrix. Instances with large values of p could not be solved as efficiently as those with small values. Therefore, future researches could be carried out for the development of better methods to update the subset U with the aim of reducing the number of degenerate iterations, especially on instances with large values of p.

A Detailed computational results

Tables 3–10 report the results obtained by our algorithm for each individual instance. Each table report the results for a given value of p. Columns LB and UB report the lower and upper bounds at the end of the execution of the algorithm, column *gap(%)* gives the percentage gap computed as $100(UB - LB)/UB$, *time(s)* gives the runtime (in seconds) for the full execution, and column $|U|$ reports the number of nodes in the sample set U at the end of the execution of the algorithm. Summary results are given at the bottom of each table.

Table 3: Results for $p = 2$

Instance	LB	UB	gap(%)	time(s)	$\|U\|$
rw1621	714	714	0.0	9.5	10
u1817	1061	1061	0.0	12.1	15
rl1889	6931	6931	0.0	10.2	14
mu1979	3284	3284	0.0	10.5	8
pr2392	6060	6060	0.0	13.1	17
d15112-2500	9234	9234	0.0	12.7	15
pcb3038	1734	1734	0.0	10.0	12
nu3496	2140	2140	0.0	10.2	9
ca4663	25781	25781	0.0	17.9	13
rl5915	7385	7385	0.0	17.0	12
rl5934	7004	7004	0.0	20.5	19
tz6117	4582	4582	0.0	15.8	16
eg7146	4643	4643	0.0	14.2	12
pla7397	310664	310664	0.0	15.3	12
ym7663	3807	3807	0.0	12.5	9
pm8079	1686	1686	0.0	13.4	11
ei8246	1813	1813	0.0	18.3	12
ar9152	9725	9725	0.0	20.5	12
ja9847	10099	10099	0.0	18.7	6
gr9882	3757	3757	0.0	13.7	9
kz9976	11325	11325	0.0	17.5	19
fi10639	4922	4922	0.0	16.7	14
rl11849	7298	7298	0.0	21.2	13
usa13509	175750	175750	0.0	17.9	10
brd14051	2970	2970	0.0	15.2	11
mo14185	3698	3698	0.0	16.2	12
ho14473	1834	1834	0.0	15.8	13
d15112	9406	9406	0.0	17.4	13
it16862	4677	4677	0.0	16.8	10
d18512	3301	3301	0.0	16.4	14
vm22775	4266	4266	0.0	20.5	14
sw24978	4960	4960	0.0	25.1	15
fyg28534	448	448	0.0	18.5	17
bm33708	5167	5167	0.0	21.6	12
pla33810	330462	330462	0.0	64.3	16
bby34656	474	474	0.0	15.1	9
pba38478	469	469	0.0	25.5	19
ch71009	19988	19988	0.0	27.1	8
pla85900	436008	436008	0.0	136.1	18
sra104815	908	908	0.0	51.1	17
usa115475	17745	17745	0.0	47.9	12
ara238025	1484	1484	0.0	89.8	15
lra498378	5888	5888	0.0	423.7	18
lrb744710	1930	1930	0.0	241.0	12
world-rand-1000000	9636738	9636738	0.0	21231.3	65
Number of instances solved to proven optimality					45/45
Average time(s) of instances solved to proven optimality					508.8
Average gap of instances not solved to proven optimality					–

Table 4: Results for $p = 3$

Instance	LB	UB	gap(%)	time(s)	$\|U\|$
rw1621	624	624	0.0	10.8	17
u1817	895	895	0.0	9.9	20
rl1889	6066	6066	0.0	12.3	22
mu1979	2326	2326	0.0	10.1	13
pr2392	5413	5413	0.0	17.1	37
d15112-2500	7952	7952	0.0	13.1	23
pcb3038	1519	1519	0.0	12.4	26
nu3496	1513	1513	0.0	12.6	18
ca4663	22680	22680	0.0	20.7	16
rl5915	6377	6377	0.0	18.5	30
rl5934	6005	6005	0.0	16.8	27
tz6117	4171	4171	0.0	21.6	29
eg7146	3702	3702	0.0	15.3	13
pla7397	279243	279243	0.0	32.5	26
ym7663	3107	3107	0.0	17.1	12
pm8079	1382	1382	0.0	14.5	16
ei8246	1522	1522	0.0	15.9	19
ar9152	8196	8196	0.0	24.8	18
ja9847	7872	7872	0.0	13.5	10
gr9882	3170	3170	0.0	14.9	12
kz9976	8244	8244	0.0	16.1	17
fi10639	3742	3742	0.0	13.5	18
rl11849	6452	6452	0.0	24.2	28
usa13509	134489	134489	0.0	23.5	16
brd14051	2426	2426	0.0	17.9	21
mo14185	2992	2992	0.0	16.7	14
ho14473	1534	1534	0.0	19.0	15
d15112	8154	8154	0.0	32.6	29
it16862	3724	3724	0.0	20.5	14
d18512	2914	2914	0.0	30.2	32
vm22775	3135	3135	0.0	21.8	22
sw24978	4120	4120	0.0	24.2	19
fyg28534	400	400	0.0	34.6	36
bm33708	4213	4213	0.0	35.1	20
pla33810	302161	302161	0.0	154.7	37
bby34656	420	420	0.0	43.3	38
pba38478	413	413	0.0	33.2	26
ch71009	13292	13292	0.0	59.9	17
pla85900	399677	399677	0.0	577.3	42
sra104815	688	688	0.0	97.7	33
usa115475	13530	13530	0.0	64.3	19
ara238025	1166	1166	0.0	153.3	19
lra498378	4995	4995	0.0	689.4	31
lrb744710	1702	1702	0.0	567.1	24
world-rand-1000000	8304744	8304744	0.0	21281.4	63
Number of instances solved to proven optimality					45/45
Average time(s) of instances solved to proven optimality					541.69
Average gap of instances not solved to proven optimality					–

Table 5: Results for $p = 5$

Instance	LB	UB	gap(%)	time(s)	$\|U\|$
rw1621	414	414	0.0	12.5	28
u1817	715	715	0.0	15.9	66
rl1889	4792	4792	0.0	16.3	45
mu1979	1877	1877	0.0	14.9	27
pr2392	3827	3827	0.0	33.9	78
d15112-2500	5856	5856	0.0	16.9	47
pcb3038	1064	1064	0.0	21.2	56
nu3496	1123	1123	0.0	14.6	32
ca4663	16837	16837	0.0	16.1	23
rl5915	4554	4554	0.0	27.0	49
rl5934	4792	4792	0.0	33.4	48
tz6117	2918	2918	0.0	37.1	62
eg7146	2590	2590	0.0	21.9	31
pla7397	174542	174542	0.0	32.7	37
ym7663	2043	2043	0.0	14.0	20
pm8079	938	938	0.0	18.4	31
ei8246	1042	1042	0.0	18.0	37
ar9152	6752	6752	0.0	21.4	32
ja9847	4503	4503	0.0	18.6	17
gr9882	2151	2151	0.0	37.0	45
kz9976	5995	5995	0.0	29.7	39
fi10639	2739	2739	0.0	23.3	39
rl11849	4873	4873	0.0	53.7	62
usa13509	103671	103671	0.0	53.4	37
brd14051	1822	1822	0.0	22.7	29
mo14185	2143	2143	0.0	30.3	36
ho14473	1112	1112	0.0	20.7	28
d15112	5890	5890	0.0	77.4	66
it16862	2811	2811	0.0	33.1	27
d18512	2073	2073	0.0	68.4	66
vm22775	2269	2269	0.0	41.7	37
sw24978	3022	3022	0.0	45.9	38
fyg28534	261	261	0.0	164.0	107
bm33708	2747	2747	0.0	53.8	39
pla33810	203597	203597	0.0	769.5	75
bby34656	286	286	0.0	256.7	115
pba38478	313	313	0.0	174.1	93
ch71009	10944	10944	0.0	269.3	57
pla85900	269544	269544	0.0	3212.0	96
sra104815	508	508	0.0	237.2	65
usa115475	10414	10414	0.0	562.2	66
ara238025	855	855	0.0	578.2	60
lra498378	3259	3259	0.0	2250.3	59
lrb744710	1170	1170	0.0	4781.7	117
world-rand-1000000	5992676	5992676	0.0	50512.9	129
Number of instances solved to proven optimality					45/45
Average time(s) of instances solved to proven optimality					1439.2
Average gap of instances not solved to proven optimality					–

Table 6: Results for $p = 10$

Instance	LB	UB	gap(%)	time(s)	$\|U\|$
rw1621	285	285	0.0	15.8	74
u1817	458	458	0.0	42.3	137
rl1889	3101	3101	0.0	151.8	170
mu1979	1161	1161	0.0	14.6	47
pr2392	2581	2581	0.0	166.1	187
d15112-2500	3705	3705	0.0	45.9	109
pcb3038	729	729	0.0	175.4	206
nu3496	757	757	0.0	20.7	78
ca4663	10499	10499	0.0	27.0	61
rl5915	3137	3137	0.0	361.8	228
rl5934	3092	3092	0.0	337.5	197
tz6117	1902	1902	0.0	180.3	175
eg7146	1826	1826	0.0	35.3	58
pla7397	121968	121968	0.0	104.8	82
ym7663	1447	1447	0.0	36.6	63
pm8079	653	653	0.0	27.4	68
ei8246	722	722	0.0	225.2	189
ar9152	4273	4273	0.0	117.7	111
ja9847	2766	2766	0.0	28.5	40
gr9882	1372	1372	0.0	54.4	90
kz9976	4155	4155	0.0	61.8	73
fi10639	1855	1855	0.0	97.4	108
rl11849	3164	3164	0.0	385.0	191
usa13509	67075	67075	0.0	503.8	130
brd14051	1265	1265	0.0	181.2	129
mo14185	1405	1405	0.0	211.6	125
ho14473	738	738	0.0	119.5	115
d15112	3785	3785	0.0	599.7	213
it16862	1574	1574	0.0	87.8	70
d18512	1340	1340	0.0	2060.1	318
vm22775	1315	1315	0.0	134.3	99
sw24978	2061	2061	0.0	155.9	108
fyg28534	176	176	0.0	3137.3	334
bm33708	1907	1907	0.0	1015.4	203
pla33810	136517	136517	0.0	6825.7	243
bby34656	192	192	0.0	3449.2	329
pba38478	207	207	0.0	6831.7	402
ch71009	7183	7183	0.0	1595.2	159
pla85900	180497	180497	0.0	19575.7	203
sra104815	354	354	0.0	7059.5	332
usa115475	6736	6736	0.0	4784.0	212
ara238025	552	552	0.0	7953.8	244
lra498378	2232	2232	0.0	16623.1	184
lrb744710	795	810	1.9	time limit	345
world-rand-1000000	3967927	3967927	0.0	37450.9	152
Number of instances solved to proven optimality					44/45
Average time(s) of instances solved to proven optimality					2797.02
Average gap of instances not solved to proven optimality					1.9%

Table 7: Results for $p = 15$

Instance	LB	UB	gap(%)	time(s)	$\|U\|$
rw1621	217	217	0.0	27.5	129
u1817	359	359	0.0	386.2	300
rl1889	2384	2384	0.0	181.6	235
mu1979	868	868	0.0	17.5	54
pr2392	2039	2039	0.0	1581.3	334
d15112-2500	2972	2972	0.0	110.7	181
pcb3038	578	578	0.0	3474.9	398
nu3496	604	604	0.0	54.4	154
ca4663	8296	8296	0.0	36.2	80
rl5915	2441	2441	0.0	6266.2	471
rl5934	2393	2393	0.0	821.2	308
tz6117	1528	1528	0.0	764.7	308
eg7146	1308	1308	0.0	44.7	62
pla7397	95271	95271	0.0	217.5	183
ym7663	1054	1054	0.0	67.6	106
pm8079	517	517	0.0	52.3	107
ei8246	579	579	0.0	2268.7	393
ar9152	3250	3250	0.0	397.7	229
ja9847	1990	1990	0.0	65.1	87
gr9882	1038	1038	0.0	74.4	110
kz9976	3261	3261	0.0	433.1	227
fi10639	1461	1461	0.0	1030.1	324
rl11849	2462	2462	0.0	14808.2	523
usa13509	52178	52178	0.0	4740.9	316
brd14051	966	966	0.0	1832.5	367
mo14185	1118	1118	0.0	692.6	257
ho14473	585	585	0.0	826.1	300
d15112	3037	3037	0.0	27897.8	609
it16862	1237	1237	0.0	142.1	125
d18512	1075	1075	0.0	12403.9	543
vm22775	1099	1099	0.0	265.6	156
sw24978	1637	1637	0.0	1751.5	311
fyg28534	142	151	6.0	time limit	815
bm33708	1475	1475	0.0	3952.0	388
pla33810	110019	126933	13.3	time limit	509
bby34656	155	164	5.5	time limit	728
pba38478	160	163	1.8	time limit	692
ch71009	5664	5664	0.0	13621.5	404
pla85900	142938	159484	10.4	time limit	363
sra104815	277	277	0.0	15204.6	435
usa115475	5288	5419	2.4	time limit	571
ara238025	441	458	3.7	time limit	581
lra498378	1755	1755	0.0	44562.6	347
lrb744710	626	653	4.1	time limit	336
world-rand-1000000	3156009	3237583	2.5	time limit	209
Number of instances solved to proven optimality					36/45
Average time(s) of instances solved to proven optimality					4474.32
Average gap of instances not solved to proven optimality					5.52%

Table 8: Results for $p = 20$

Instance	LB	UB	gap(%)	time(s)	$\|U\|$
rw1621	186	186	0.0	34.8	166
u1817	309	309	0.0	2171.8	430
rl1889	2089	2089	0.0	405.0	301
mu1979	751	751	0.0	26.2	87
pr2392	1736	1736	0.0	7500.1	442
d15112-2500	2573	2573	0.0	1795.9	371
pcb3038	493	493	0.0	54802.1	630
nu3496	519	519	0.0	128.3	235
ca4663	7024	7024	0.0	57.0	124
rl5915	2083	2083	0.0	76684.1	730
rl5934	2100	2100	0.0	4367.2	403
tz6117	1278	1278	0.0	1988.7	356
eg7146	972	972	0.0	51.3	89
pla7397	78817	78817	0.0	608.2	292
ym7663	899	899	0.0	152.1	162
pm8079	430	430	0.0	91.4	153
ei8246	497	497	0.0	33150.5	670
ar9152	2695	2695	0.0	6067.8	482
ja9847	1724	1724	0.0	66.7	87
gr9882	877	877	0.0	235.6	204
kz9976	2790	2790	0.0	1540.2	338
fi10639	1245	1245	0.0	15688.6	615
rl11849	2119	2273	6.8	time limit	649
usa13509	44740	46719	4.2	time limit	633
brd14051	803	803	0.0	16721.9	586
mo14185	919	919	0.0	3597.1	437
ho14473	497	497	0.0	1470.2	376
d15112	2581	2717	5.0	time limit	669
it16862	1059	1059	0.0	563.6	246
d18512	912	969	5.9	time limit	616
vm22775	932	932	0.0	1491.9	351
sw24978	1421	1421	0.0	14835.0	570
fyg28534	118	130	9.2	time limit	567
bm33708	1257	1257	0.0	17808.7	561
pla33810	91302	106076	13.9	time limit	345
bby34656	128	138	7.2	time limit	570
pba38478	136	148	8.1	time limit	569
ch71009	4798	5250	8.6	time limit	650
pla85900	119643	136984	12.7	time limit	342
sra104815	232	236	1.7	time limit	832
usa115475	4453	4747	6.2	time limit	509
ara238025	372	407	8.6	time limit	563
lra498378	1442	1573	8.3	time limit	440
lrb744710	524	580	9.7	time limit	358
world-rand-1000000	2611926	2936496	11.1	time limit	210
Number of instances solved to proven optimality					29/45
Average time(s) of instances solved to proven optimality					9106.97
Average gap of instances not solved to proven optimality					7.95%

Table 9: Results for $p = 25$

Instance	LB	UB	gap(%)	time(s)	$\|U\|$
rw1621	166	166	0.0	38.7	178
u1817	272	272	0.0	1875.8	442
rl1889	1866	1866	0.0	1154.8	372
mu1979	639	639	0.0	30.4	95
pr2392	1520	1520	0.0	23767.7	521
d15112-2500	2243	2243	0.0	5266.2	459
pcb3038	433	470	7.9	time limit	580
nu3496	441	441	0.0	116.8	236
ca4663	5966	5966	0.0	118.2	193
rl5915	1823	1916	4.9	time limit	626
rl5934	1850	1850	0.0	25300.4	638
tz6117	1152	1258	8.4	time limit	754
eg7146	855	855	0.0	79.1	116
pla7397	69508	69508	0.0	776.8	367
ym7663	785	785	0.0	189.4	205
pm8079	367	367	0.0	184.8	230
ei8246	429	461	6.9	time limit	641
ar9152	2355	2355	0.0	30531.7	622
ja9847	1362	1362	0.0	154.8	170
gr9882	772	772	0.0	532.3	301
kz9976	2479	2479	0.0	4666.4	498
fi10639	1103	1173	6.0	time limit	738
rl11849	1838	2099	12.4	time limit	542
usa13509	38150	40578	6.0	time limit	636
brd14051	703	737	4.6	time limit	747
mo14185	818	818	0.0	42123.3	741
ho14473	436	436	0.0	11941.1	625
d15112	2233	2447	8.7	time limit	534
it16862	951	951	0.0	5284.4	506
d18512	795	881	9.8	time limit	600
vm22775	799	799	0.0	3102.5	481
sw24978	1233	1285	4.0	time limit	722
fyg28534	102	112	8.9	time limit	509
bm33708	1106	1140	3.0	time limit	737
pla33810	81800	88861	7.9	time limit	576
bby34656	112	128	12.5	time limit	563
pba38478	117	134	12.7	time limit	541
ch71009	4145	4321	4.1	time limit	602
pla85900	107527	124693	13.8	time limit	425
sra104815	206	216	4.6	time limit	757
usa115475	3808	4453	14.5	time limit	474
ara238025	319	357	10.6	time limit	570
lra498378	1228	1339	8.3	time limit	418
lrb744710	454	533	14.8	time limit	370
world-rand-1000000	2216367	2573650	13.9	time limit	222
Number of instances solved to proven optimality					21/45
Average time(s) of instances solved to proven optimality					7487.41
Average gap of instances not solved to proven optimality					8.72%

Table 10: Results for $p = 30$

Instance	LB	UB	gap(%)	time(s)	$\|U\|$
rw1621	147	147	0.0	64.8	224
u1817	241	241	0.0	4228.3	501
rl1889	1657	1657	0.0	1603.4	375
mu1979	552	552	0.0	34.4	130
pr2392	1379	1471	6.3	time limit	556
d15112-2500	2029	2029	0.0	50627.4	569
pcb3038	386	412	6.3	time limit	532
nu3496	396	396	0.0	205.7	279
ca4663	5361	5361	0.0	152.6	232
rl5915	1624	1853	12.4	time limit	537
rl5934	1658	1812	8.5	time limit	643
tz6117	1025	1142	10.2	time limit	648
eg7146	744	744	0.0	76.8	113
pla7397	63770	63770	0.0	20045.1	438
ym7663	715	715	0.0	536.6	305
pm8079	330	330	0.0	459.8	360
ei8246	386	412	6.3	time limit	615
ar9152	2080	2080	0.0	10145.7	488
ja9847	1220	1220	0.0	268.3	254
gr9882	677	677	0.0	1024.6	371
kz9976	2230	2230	0.0	11820.2	681
fi10639	974	1017	4.2	time limit	600
rl11849	1641	1855	11.5	time limit	505
usa13509	34471	37306	7.6	time limit	485
brd14051	620	668	7.2	time limit	613
mo14185	732	767	4.6	time limit	730
ho14473	396	396	0.0	29545.8	738
d15112	2009	2254	10.9	time limit	541
it16862	854	854	0.0	16851.8	741
d18512	709	786	9.8	time limit	534
vm22775	696	696	0.0	13315.2	681
sw24978	1096	1215	9.8	time limit	558
fyg28534	92	108	14.8	time limit	537
bm33708	968	1065	9.1	time limit	708
pla33810	71480	83187	14.1	time limit	390
bby34656	100	133	24.8	time limit	470
pba38478	105	125	16.0	time limit	497
ch71009	3724	4098	9.1	time limit	512
pla85900	93871	114244	17.8	time limit	375
sra104815	186	206	9.7	time limit	784
usa115475	3391	3874	12.5	time limit	462
ara238025	288	368	21.7	time limit	554
lra498378	1073	1192	10.0	time limit	432
lrb744710	403	475	15.2	time limit	374
world-rand-1000000	2009692	2523416	20.4	time limit	240
Number of instances solved to proven optimality					18/45
Average time(s) of instances solved to proven optimality					8944.81
Average gap of instances not solved to proven optimality					11.51%

References

Aloise, D. and Contardo, C. (2018). A sampling-based exact algorithm for the solution of the minimax diameter clustering problem. Journal of Global Optimization. Forthcoming.

Bai, C., Kang, L., and Shan, E. (2017). The connected p-center problem on cactus graphs. Theoretical Computer Science. Forthcoming.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. SIAM Review, 59(1):65–98.

Calik, H. and Tansel, B. C. (2013). Double bound method for solving the p-center location problem. Computers & Operations Research, 40(12):2991–2999.

Callaghan, B., Salhi, S., and Nagy, G. (2017). Speeding up the optimal method of Drezner for the p-centre problem in the plane. European Journal of Operational Research, 257(3):722–734.

Caruso, C., Colorni, A., and Aloi, L. (2003). Dominant, an algorithm for the p-center problem. European Journal of Operational Research, 149(1):53–64.

Chen, D. and Chen, R. (2009). New relaxation-based algorithms for the optimal solution of the continuous and discrete p-center problems. Computers & Operations Research, 36(5):1646–1655.

Chen, D. and Chen, R. (2013). Optimal algorithms for the α-neighbor p-center problem. European Journal of Operational Research, 225(1):36–43.

Chen, R. and Handler, G. Y. (1987). Relaxation method for the solution of the minimax location-allocation problem in euclidean space. Naval Research Logistics, 34(6):775–788.

Chu, S. C. and Chu, L. (2000). A modeling framework for hospital location and service allocation. International transactions in operational research, 7(6):539–568.

Daskin, M. (1995). Center problems. In Daskin, M., editor, Network and Discrete Location: Models, Algorithms, and Applications, pages 154–197. Wiley, New York.

Daskin, M. (2000). A new approach to solving the vertex p-center problem to optimality: Algorithm and computational results. Communications of the Operations Research Society of Japan, 45(9):428–436.

Dunning, I., Huchette, J., and Lubin, M. (2017). JuMP: A modeling language for mathematical optimization. SIAM Review, 59(2):295–320.

Elloumi, S., Labbé, M., and Pochet, Y. (2004). A new formulation and resolution method for the p-center problem. INFORMS Journal on Computing, 16(1):84–94.

Francis, R. L., Lowe, T. J., Rayco, M. B., and Tamir, A. (2009). Aggregation error for location models: survey and analysis. Annals of Operations Research, 167(1):171–208.

Francis, R. L., McGinnis, L. F., and White, J. A. (1992). Facility layout and location: an analytical approach. Pearson College Division.

Grangier, P., Gendreau, M., Lehuédé, F., and Rousseau, L.-M. (2016). An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. European Journal of Operational Research, 254(1):80–91.

Gurobi Optimization, Inc. (2017). Gurobi optimizer reference manual.

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

Hansen, P., Brimberg, J., Urošević, D., and Mladenović, N. (2009). Solving large p-median clustering problems by primal–dual variable neighborhood search. Data Mining and Knowledge Discovery, 19(3):351–375.

Ilhan, T., Özsoy, F. A., and Pınar, M. Ç. (2002). An efficient exact algorithm for the vertex p-center problem and computational experiments for different set covering subproblems. Technical report, Optimization-Online. Available at http://www.optimization-online.org/DB_HTML/2002/12/588.html.

Irawan, C. A., Salhi, S., and Drezner, Z. (2015). Hybrid meta-heuristics with VNS and exact methods: Application to large unconditional and conditional vertex p-centre problems. Journal of Heuristics, 22(4):507–537.

Kramer, R., Iori, M., and Vidal, T. (2018). Mathematical models and search algorithms for the capacitated p-center problem. Technical report. Available at https://arxiv.org/abs/1803.04865.

Labbé, M. and Laporte, G. (1986). Maximizing user convenience and postal service efficiency in post box location. Belgian Journal of Operations Research, Statistics and Computer Science, 26:21–35.

Lourenço, H., Martin, O., and Stützle, T. (2010). Iterated local search: Framework and applications. In Gendreau, M. and Potvin, J.-Y., editors, Handbook of Metaheuristics, International Series in Operations Research and Management Science, chapter 12, pages 362–397. Springer, 2 edition.

Lu, C.-C. and Sheu, J.-B. (2013). Robust vertex p-center model for locating urgent relief distribution centers. Computers & Operations Research, 40(8):2128–2137.

Martínez-Merino, L. I., Albareda-Sambola, M., and Rodríguez-Chía, A. M. (2017). The probabilistic p-center problem: Planning service for potential customers. European Journal of Operational Research, 262(2):509–520.

Minieka, E. (1970). The m-center problem. SIAM Review, 12(1):138–139.

Plane, D. R. and Hendrick, T. E. (1977). Mathematical programming and the location of fire companies for the Denver fire department. Operations Research, 25(4):563–578.