ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computers & Operations Research 33 (2006) 1420–1436

computers & operations research

www.elsevier.com/locate/cor

An exact algorithm for the capacitated vertex p-center problem

F. Aykut Özsoy[a], Mustafa Ç. Pınar[b,*]

[a] *Department of Industrial Engineering, Bilkent University, 06800 Ankara, Turkey*
[b] *Department of Industrial Engineering, Bilkent University, 06800 Ankara, Turkey*

Abstract

We develop a simple and practical exact algorithm for the problem of locating p facilities and assigning clients to them within capacity restrictions in order to minimize the maximum distance between a client and the facility to which it is assigned (capacitated p-center). The algorithm iteratively sets a maximum distance value within which it tries to assign all clients, and thus solves bin-packing or capacitated concentrator location subproblems using off-the-shelf optimization software. Computational experiments yield promising results.

Keywords: Integer programming; Capacitated p-center problem; Facility location

1. Introduction

In this paper the capacitated vertex p-center problem is investigated. An exact algorithm developed by Ilhan and Pınar [1] for the basic vertex p-center problem is modified and extended to solve the capacitated version to optimality. Computational experiments are carried out and promising results are reported.

The basic p-center problem consists of locating p facilities and assigning clients to them so as to minimize the maximum distance between a client and the facility it is assigned to. The problem is known to be NP-hard [2]. Recent articles on the basic p-center problem include Mladenović et al. [3], Elloumi et al. [4] and Daskin [5]. For a detailed exposition of the p-center problem and available solution methodology, the reader is directed to Chapter 5 of the textbook [6].

In the capacitated version of the p-center problem, each client is labelled with some quantity of demand, and assignment of clients to facilities is constrained with capacity restrictions of facilities (i.e., the total

* Corresponding author.
E-mail addresses: aykut@bilkent.edu.tr (F. Aykut Özsoy), mustafap@bilkent.edu.tr (M.Ç. Pınar).

0305-0548/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2004.09.035

demands of clients assigned to a certain facility cannot exceed the facility's capacity). Namely, the capacitated p-center problem can be articulated as locating p capacitated facilities on a network and assigning clients to them within capacity restrictions so as to minimize maximum distance between a client and the facility it is assigned to.

Typical applications of the problem are locating fire stations or health centers where stations or centers have capacities of service. Clients can be labelled with demand values associated with their populations or risks of casualties in case of an emergency. Obviously, an ordinary warehouse and a hypermarket (or a factory) would claim different amounts of resources in such a situation. Demand values can be interpreted as a measure of this difference among clients.

Let $W = \{w_1, w_2, \ldots, w_m\}$ be the set of all possible locations for facilities with $|W| = M$, $V = \{v_1, v_2, \ldots, v_n\}$ be the set of all clients with $|V| = N$. The distance for each facility pair (w_i, v_j) is given as d_{ij}. We assume in this paper that $W \cup V$ is the vertex (node) set of a complete graph, and distances d_{ij} represent the length of the shortest path between vertices i and j (in the test problems of Section 4, we deal with a single vertex set, i.e., with the special case where $W = V$). Let h_i be demand associated with client i and assume each facility site j has a service capacity of Q_j. This means, if we locate a facility on site j, the total of the demands assigned to that facility cannot exceed Q_j.

Now, we present a linear mixed-integer programming formulation of the problem below.

Problem CPC:

$$\text{Minimize} \quad z$$

$$\text{subject to} \quad \sum_{j \in W} x_{ij} = 1, \quad \forall i \in V, \tag{1}$$

$$x_{ij} \leqslant y_j, \quad \forall i \in V, j \in W, \tag{2}$$

$$\sum_{j \in W} y_j \leqslant p, \tag{3}$$

$$\sum_{j \in W} d_{ij} x_{ij} \leqslant z, \quad \forall i \in V, \tag{4}$$

$$\sum_{i \in V} h_i x_{ij} \leqslant Q_j, \quad \forall j \in W, \tag{5}$$

$$x_{ij}, y_j \in \{0, 1\}, \quad \forall i \in V, \forall j \in W, \tag{6}$$

where x_{ij} assumes value one if client i is assigned to facility site j, and value zero otherwise. The binary variable y_j assumes value one if facility site j is open. Constraints (1) express the requirements that all clients must be assigned to some facility site. Constraints (2) prevent a client from an assignment to a facility site which is not open. In total at most p facility sites are to be opened, a requirement which is modeled by constraint (3). Capacity restrictions of the facilities are incorporated into the model by constraints (5) (one can obtain the basic p-center formulation by excluding constraints (5) from the above model).

The rest of this paper is organized as follows. Section 2 presents a brief literature survey on the capacitated p-center problem. Section 3 gives a detailed description of the proposed algorithm. Section 4 contains computational results obtained with the algorithm. Concluding remarks are given in Section 5.

2. Background

The capacitated vertex p-center problem has not been thoroughly investigated yet. Among the few papers that covered this problem, the one by Bar-Ilan et al. [7] is the first to develop a polynomial time approximation scheme for the special case where $h_i = 1\ \forall i \in V$. This procedure has an approximation factor of 10. In this algorithm, a subgraph is generated in each iteration by choosing edges with smaller distance value than a specified radius. Following this, in each iteration, a maximal independent set is extracted from the subgraph. This maximal independent set is used to obtain a set of centers and assignment of nodes to these centers. The algorithm searches the smallest radius that yields a set of centers with cardinality at most p.

Khuller and Sussmann [8] improve this algorithm and draw its approximation factor down to 6 by proposing a specially tailored method for finding the maximal independent sets in the subgraph and by changing the way nodes in the maximal independent set are handled. They assume all clients have unit demand (i.e., $h_i = 1\ \forall i \in V$) and cover a variant where locating multiple centers on a node is possible. They name this variant "the capacitated multi p-center problem".

Besides these heuristic algorithms, a polynomial exact algorithm for tree networks is developed by Jaeger and Goldberg [9]. Jaeger and Goldberg also considers the case where multiple centers can be located on a node and $h_i = 1\ \forall i \in V$. The method described in the article solves set-covering subproblems on trees with a polynomial algorithm.

We are also aware of a recent study by Pallottino et al. [10] on the application of local search heuristics to the problem. Their algorithm iteratively carries out multiple exchange of clients between facilities, and then performs local optimization for locations of facilities.

The problem we define in this paper differs from the aforementioned articles' problems in two ways: (1) we do not have the requirement that all h_i's have to be equal to 1; (2) our facilities do not have to be equally capacitated. We can solve the problem for any demand values associated with the clients and any capacity values associated with the facilities.

3. The proposed algorithm for solving CPC

The algorithm we propose for solving CPC is an extension of an exact algorithm developed for the vertex p-center problem by Ilhan and Pınar [1]. This algorithm relies on solving a series of set covering problems using an off-the-shelf IP solver while carrying out an iterative search over the coverage distances. At each iteration, it sets a threshold distance as a radius to see whether it is possible to cover all clients with p or less facilities within this radius (i.e., it uses this radius as the coverage distance of a set-covering problem), and updates lower and upper bounds on the optimal radius in the light of this information.

The idea underlying the solution procedure is roughly to proceed as follows:

Step 1: Select initial lower and upper bounds on the optimal objective function value, set the coverage distance to the average of the lower and upper bounds.

Step 2: Solve an appropriate set-covering subproblem with the coverage distance found.

Step 3: If it turns out that it is possible to cover all clients with at most p facilities, reset the upper bound to the coverage distance that was just used.

Step 4: If it is not possible, reset the lower bound to the coverage distance just used.

Step 5: If the lower and upper bounds are equal, stop; otherwise set the coverage distance as the average of lower and upper bounds, and go to Step 2.

The set-covering feasibility subproblem used in this algorithm is not suitable for solving the capacitated *p*-center problem. For this reason, two new subproblems are developed here for solving CPC. Both of these subproblems have the objective of minimizing the number of facilities to be located (reasons for choosing this objective will be clarified later). Now, in Section 3.1 we describe the subproblems developed and Section 3.2 includes detailed descriptions of the algorithm.

3.1. Subproblems

We develop two subproblems which give an answer as to whether we can cover all clients within a certain radius ε with at most p capacitated facilities. We considered the objective of minimizing the number of facilities to be located while covering all clients. The reason for choosing this objective function for our subproblems is that this objective together with assignment features leads us to two well-known problems from combinatorial optimization literature as subproblems; *the capacitated concentrator location problem* and *the bin-packing problem*. The subproblems and their relations with these problems are explained in this section.

The assignment variables (x_{ij}'s) and location variables (y_j's) to be used in our subproblems are defined as follows:

$$x_{ij} = \begin{cases} 1 & \text{if client at node } i \text{ is assigned to facility located at node } j, \\ 0 & \text{otherwise,} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if a facility is located at node } j, \\ 0 & \text{otherwise.} \end{cases}$$

With these variables the subproblem SP_1 with respect to a radius value ε is formulated as follows:

Subproblem $SP_1(\varepsilon)$:

$$\min \quad \sum_{j=1}^{n} y_j$$

$$\text{s. t.} \quad \sum_{j \in W:\ d_{ij} \leqslant \varepsilon} x_{ij} = 1 \ \forall i \in V, \tag{7}$$

$$\sum_{i \in V:\ d_{ij} \leqslant \varepsilon} h_i x_{ij} \leqslant Q_j \ \forall j \in W, \tag{8}$$

$$x_{ij} \leqslant y_j \ \{(i, j) \in V \times W : d_{ij} \leqslant \varepsilon\}, \tag{9}$$

$$x_{ij}, y_j \in \{0, 1\} \ \{(i, j) \in V \times W : d_{ij} \leqslant \varepsilon\}.$$

Constraints (7) ensure that each client is covered by one facility whose distance to the client is at most ε. Constraints (8) stand for capacity restrictions of facilities. And constraints (9) prevent a client to be assigned to a node at which no facility is located.

This subproblem is a variant of '*the capacitated concentrator location problem*' from discrete location literature (e.g. see [11]). Capacitated concentrator location problem can be described as follows: given

a set of possible sites W for facilities of fixed capacity Q, we try to locate facilities at a subset of sites in W and connect n clients from V to these facilities, where client i uses h_i units of a facility's capacity, in such a way that each client is connected to exactly one facility, the facility capacity is not exceeded and total cost of assigning clients to facilities and locating facilities is minimized. Assigning client i to a facility on site j has a cost of c_{ij} units and locating (setting-up) a facility on site j has a cost of S_j units. The objective function of the problem is clearly

$$\sum_{i\in V}\sum_{j\in W} c_{ij}x_{ij} + \sum_{j\in W} S_j y_j,$$

whereas the objective function of SP_1 merely minimizes the number of facilities to be located (i.e., $c_{ij} = 0\ \forall i \in V,\ j \in W$ and $S_j = 1\ \forall j \in W$ parameter setting is used in SP_1). As to constraints, there are two differences between SP_1 and the concentrator location problem: first, in SP_1 facilities located on different nodes assume different capacities whereas in the capacitated concentrator location problem all facilities located are equally capacitated; and secondly, in SP_1 we have the set-covering conditions under the summation operations of constraints (7) and (8) (i.e., the condition $\{j \in W:\ d_{ij} \leqslant \varepsilon\}$ in (7) and $\{i \in V:\ d_{ij} \leqslant \varepsilon\}$ in (8)) unlike the capacitated concentrator location problem. These conditions stand for the requirement that each client has to be assigned to a facility whose distance to the client is at most ε. However, in the capacitated concentrator location problem there is no restriction as to the assignments of clients to facilities, that is, any client can be assigned to a facility located on any node. Thus if we replace the aforementioned condition in (7) by $\{\forall j \in W\}$, the one in (8) by $\{\forall i \in V\}$ and Q_j's by a fixed capacity value of Q, constraints of SP_1 exactly constitute the constraints of capacitated concentrator location problem.

An alternative subproblem can be formulated by replacing constraints (8) and (9) in SP_1 by one single set of constraints. Intuitively, it is easy to see that the constraint set

$$\sum_{i\in V:\ d_{ij}\leqslant\varepsilon} h_i x_{i,j} \leqslant Q_j y_j, \quad \forall j \in W \tag{10}$$

can account for these two sets of inequalities. By introducing (10), we can formulate an alternative subproblem SP_2 which defines the same feasible region as SP_1.

Subproblem $SP_2(\varepsilon)$:

$$\min \quad \sum_{j=1}^{n} y_j$$

$$\text{s. t} \quad \sum_{j\in W:\ d_{ij}\leqslant\varepsilon} x_{ij} = 1, \quad \forall i \in V, \tag{11}$$

$$\sum_{i\in V:\ d_{ij}\leqslant\varepsilon} h_i x_{ij} \leqslant Q_j y_j, \quad \forall j \in W \tag{12}$$

$$x_{ij}, y_j \in \{0, 1\}\ \{(i, j) \in V \times W : d_{ij} \leqslant \varepsilon\}.$$

This subproblem resembles '*the bin-packing problem*' (e.g. see [12]). The bin-packing problem consists of assigning each of n items with weight $h_i\ \forall i \in V$ to one of n identical bins of capacity Q so that the

total weight of the items in each bin does not exceed Q, and the number of bins used is minimized. In a location context, the term 'client' replaces 'item' and 'facility' takes place of 'bin'. The only differences between SP_2 and bin-packing arise in the constraints. The two differences between constraints of SP_1 and capacitated concentrator location problem exactly apply to SP_2 and bin-packing, too.

Note that, the number of assignment variables in both subproblems is dependent on the radius value ε. Since we cannot assign a client i to a facility on node j whose distance to i is more than ε, variable x_{ij}'s where $d_{ij} > \varepsilon$ are not included in our models. Thus, number of assignment variables in both of our subproblems is equal to cardinality of the set $\{(i, j) : d_{ij} \leqslant \varepsilon\}$.

Equivalence of SP_1 *and* SP_2: It is easy to see that $SP_1(\varepsilon)$ and $SP_2(\varepsilon)$ are equivalent in the sense that they have identical sets of feasible solutions, and their optimal values are equal. This property can also be established without using the integrality restrictions of x_{ij} variables. This means that the property also holds when x_{ij} variables are allowed to take on continuous values between 0 and 1 in both subproblems. However, relaxing y_j variables disturbs this equivalence. For this reason, the LP relaxations of the two subproblems are not equivalent to each other, as will be seen later. The essence of this property lies in identical paths that the algorithm follows when we employ SP_1 or SP_2.

3.2. Algorithm

In [7], the roughly mentioned procedure in Section 3 for solving the basic p-center problem is split into two phases to speed up the procedure. In Phase I, LP relaxations of the subproblems are solved to carry out a binary search over the distance values in order to provide a suitable starting point for the search in Phase II. The second phase iteratively increases coverage radii, starting from the radius value provided by Phase I until the optimal radius, for which it is possible to cover all clients by at most p facilities, is found.

Now, the step-by-step flow of the algorithm that employs subproblem SP_k ($k = 1$ or 2) is given. Note that, we denote the LP relaxation of SP_1 as $SP_1^{LR}(.)$.

Phase I

Step 1: Find the minimum, l, and maximum, u, of weights of all edges,
$l = \min\{d_{ij} : \forall i \in V, \forall j \in W\}$,
$u = \max\{d_{ij} : \forall i \in V, \forall j \in W\}$.
Step 2: Calculate $dif = \lfloor (u - l)/2 \rfloor$, $\varepsilon = l + dif$.
Step 3: Solve $SP_1^{LR}(\varepsilon)$.

Step 3.1: If optimal objective is greater than p, then set $l = \varepsilon$.
Step 3.2: Else set $u = \varepsilon$.

Step 4: Calculate $(u - l)$.

Step 4.1: If $(u - l) \leqslant 1$ then go to Step 5.
Step 4.2: Else go to Step 2.

Step 5: If optimal objective of previous subproblem was greater than p, then set $\varepsilon = u$, else set $\varepsilon = l$.

Table 1
Phase I bounds of the algorithm with LP relaxations of SP_1 and SP_2

File no.	Size	p	SP_1		SP_2	
			Phase I obj.	Iter. no.	Phase I obj.	Iter. no.
1	50	5	28	7	1	6
2	50	5	31	7	1	7
3	50	5	25	7	2	6
4	50	5	31	7	1	6
5	50	5	27	7	1	6
6	50	5	27	7	1	6
7	50	5	30	7	1	6
8	50	5	29	7	1	7
9	50	5	27	7	1	6
10	50	5	29	7	1	6
11	100	10	18	7	0	6
12	100	10	19	7	0	6
13	100	10	19	7	1	7
14	100	10	20	7	1	6
15	100	10	19	7	1	7
16	100	10	18	7	1	6
17	100	10	20	7	0	7
18	100	10	19	7	1	6
19	100	10	20	7	0	7
20	100	10	18	7	1	6

Phase II
Step 6: Create $SP_k(\varepsilon)$ and solve.

Step 6.1: If optimal objective is greater than p, then increase the value of ε:
$\varepsilon' = \min\{d_{ij} : d_{ij} > \varepsilon, \forall i \in V, \forall j \in W\}$ then set $\varepsilon = \varepsilon'$ and go to Step 6.
Step 6.2: Else stop.

Note that, in the algorithm, we use LP relaxation of SP_1 (which we denote by $SP_1^{LR}(.)$) in the first phase no matter which subproblem is utilized in Phase II. The reason for this choice will be explained now.

It is easy to see that the success of the bound obtained from first phase of the algorithm is directly dependent on the tightness of LP relaxation of the subproblem. For instance, linear relaxation of SP_2 gives loose bounds, which makes usage of the linear relaxation of SP_2 in Phase I inconvenient. Although computational experiments are explained in detail in Section 4, we will present a preliminary result here to display the poor Phase I bounds obtained with SP_2. Table 1 compares Phase I objective values obtained by using LP relaxations of SP_1 and SP_2. In the table, "*file no.*" represents the number of the "*capacitated p-median*" instance in OR-Lib, "*size*" refers to cardinality of V (recall that we take $V = W$ in our computational experiments) and column labelled "p" gives the value of the parameter p. Under headings "SP_1" and "SP_2", we give results corresponding to Phase I of the algorithm while utilizing SP_1 and SP_2, respectively. Two columns under SP_1 and SP_2 display objective value of Phase I and number of iterations carried out in Phase I.

The reason for the poor objective values provided by Phase I of algorithm with SP_2 is the loose bounds that LP relaxations of SP_2 give. For example in instance 1 of Table 1, the optimal objective value for LP relaxation of SP_2 turns out to be less than or equal to p for any radius value that is tried. This causes the upper bound u in Phase I to be pulled downwards until it is equal to the initial lower bound $l = \min\{d_{ij} : i \in V,\ j \in W\}$), which yields 1 as the Phase I bound. However, the optimal radius for this instance is 29. This means that in Phase II, the algorithm will be solving different SP_2 instances in IP forms for increasing radius values from 1 until 29. These findings about SP_2 on instance 1 exactly apply to other instances of Table 1, too. This is an important handicap for the efficiency of the algorithm. On the other hand, we can observe that the LP relaxation of SP_1 yields quite successful objective values in Phase I when compared with those of SP_2. Hence, we always use the LP relaxation of SP_1 (named as $SP_1^{LR}(.)$) in the first phase, even when we employ SP_2 as the subproblem of Phase II.

This common usage of SP_1^{LR} in Phase I provides that we will have the same record of lower (l) and upper (u) bound updates in the first phase of the algorithm regardless of the subproblem we use in Phase II. This means, SP_1 and SP_2 will start their searches in Phase II from exactly the same radius value. Due to the equivalence of the two subproblems (aforementioned in Section 3.2), they will also exhibit the same course of search in Phase II (i.e., both of the subproblems will bring along the same optimal values for any of the radius values). In other words, SP_1 and SP_2 will terminate Phase II (and thus, the whole algorithm) in the same number of iterations. Indeed the only difference between them will be the cpu times they spend until termination of the algorithm.

3.3. A modification of Phase II

As can be seen in the computational results of Section 4, for some instances SP_1 and SP_2 display long cpu times. This is primarily due to difficulty in solving IP forms of subproblems for those instances. We now propose a modification of the given algorithm to decrease cpu time of Phase II. For this purpose we relax integrality restrictions of assignment variables (x_{ij}'s) to obtain the subproblems referred to as $SP_1^R(\varepsilon)$ and $SP_2^R(\varepsilon)$, respectively.

Mainly, we split Phase II into two subsequent sub-phases, which we call Phase IIa and Phase IIb. In Phase IIa, we solve the MIP versions of the subproblems (i.e., $SP_1^R(.)$ or $SP_2^R(.)$) for increasing distance values as coverage radii. Once the subproblem used yields an optimal objective value which is at most p for a radius value, the algorithm terminates Phase IIa. Phase IIb starts from the terminating radius value of Phase IIa, and solves IP versions of the subproblems (i.e., $SP_1(.)$ or $SP_2(.)$) for increasing radius values. Phase IIb terminates when the optimal radius (i.e., the radius that makes the optimal objective value of the subproblem at most p) is reached.

The idea behind this modification is speeding up Phase II by solving some of the Phase II subproblems in a relaxed form. This way of thinking leads us to solve one extra subproblem in total, but fewer subproblems solved as IPs. So, by this modification we may expect to shorten the cpu time of the algorithm. Now follows the formal representation of Phase II after the proposed modification (for SP_k where $k = 1$ or 2):

Modified Phase II

Phase IIa

Step 6: Create $SP_k^R(\varepsilon)$ and solve.

Step 6.1: If optimal objective is greater than p, then increase the value of ε:
$\varepsilon' = \min\{d_{ij} : d_{ij} > \varepsilon, \forall i \in V, \forall j \in W\}$ then set $\varepsilon = \varepsilon'$ and go to Step 6.

Step 6.2: Else go to Step 7.

Phase IIb
Step 7: Create $SP_k(\varepsilon)$ and solve.

Step 7.1: If optimal objective is greater than p, then increase the value of ε:

$$\varepsilon' = \min\{d_{ij} : d_{ij} > \varepsilon, \forall i \in V, \forall j \in W\}$$ then set $\varepsilon = \varepsilon'$ and go to Step 7.

Step 7.2: Else stop.

Since one of the radius values is commonly used in both of Phase IIa and Phase IIb, this modification increases the number of subproblems solved in Phase II by 1. However, computational results in Section 4 show that this modified version of the algorithm yields shorter cpu times than its original version given in Section 3.2.

Note that, since the equivalence between the subproblems (which is stated in Section 3.2) also holds when x_{ij}'s are relaxed (i.e., $SP_1^R(.)$ and $SP_2^R(.)$ are equivalent), $SP_1^R(.)$ and $SP_2^R(.)$ bring along the same number of Phase II iterations (i.e. the same number of subproblems solved). That is, all results but the cpu times will be the same for SP_1 and SP_2 employed in modified algorithm, just as SP_1 and SP_2 in the original algorithm.

4. Computational results

In this section, we report the computational results obtained with the algorithm using the two subproblems. The experiments are carried out on three data sets using CPLEX 7.0 linear and mixed integer program solvers to solve the subproblems. All programs are coded in C and run on Sun UltraSparc Workstation running Solaris.

The first data set we use includes the *capacitated p-median* problems from OR-Lib [13]. In OR-Lib sizes of the instances (i.e., the cardinality of V) are 50 and 100. This implies that we are dealing with IP problems with 2550 and 10,100 binary variables, respectively. Coordinates of nodes on Euclidean coordinate plane are given. Distance matrices are constructed by finding the Euclidean distances between every pairs of nodes and rounding these values to the nearest integers.

The second data set we use comes from a capacitated p-median article of Lorena and Senne [14]. The sizes of the instances range from 100 to 402. Coordinates of nodes on Euclidean plane are given. Distances between the nodes are calculated by finding the Euclidean distances between each pair of nodes. In this data set, the distance values are not rounded to the nearest integers to be able to see the performance of the algorithm on continuous distance values.

The third data set we use comes from Pallottino et al.'s article [10]. The sizes of the instances are 100 and 150. The distance values given in this data set are integral values.

We present the results in eight tables: first four, Tables 2–5, display the results for the first data set. The first two of the tables present the results obtained by SP_1, the third and the fourth tables show the analogous results obtained by SP_2 in the experiments carried out on OR-Lib instances. Only SP_1 is used in the algorithm during our experiments on the second and third data sets. Experiments with SP_2 are not carried out in these data sets due to practical time limitations. The experiments on the first data set carried

Table 2
Computational results on OR-Lib problems using SP_1 ($1 \leqslant h_i \leqslant 20\ \forall i \in V$, $Q_j = 120\ \forall j \in W$)

File no.	Size	p	Phase I			Compl. Algorithm				CPLEX	
			Obj.	Iter. no.	cpu t.	Obj.	Iter. no.	cpu t.	Devia (%)	cpu t.	% Impv.
1	50	5	28	7	1.97	29	9	2.29	3.45	271.23	99.15
2	50	5	31	7	2.42	33	10	20.6	6.06	10237.82	99.79
3	50	5	25	7	2.17	26	9	3.89	3.85	729.18	99.47
4	50	5	31	7	2.47	32	9	5.55	3.13	662.27	99.16
5	50	5	27	7	2.27	29	10	9.97	6.90	582.24	98.29
6	50	5	27	7	2.13	31	12	35.74	12.90	878.81	95.93
7	50	5	30	7	2.08	30	8	5.69	0.00	1034.53	99.45
8	50	5	29	7	2.45	31	10	20.48	6.45	1069.14	98.08
9	50	5	27	7	2.44	28	9	6.2	3.57	727.83	99.15
10	50	5	29	7	2.2	32	11	59.25	9.38	2956.88	98.00
11	100	10	18	7	19.65	19	9	45.74	5.26	—	—
12	100	10	19	7	17.59	20	9	53.31	5.00	—	—
13	100	10	19	7	25.32	20	9	49.85	5.00	—	—
14	100	10	20	7	22.19	20	8	32.79	0.00	—	—
15	100	10	19	7	19.26	21	10	189.19	9.52	—	—
16	100	10	18	7	21.59	20	10	2147.98	10.00	—	—
17	100	10	20	7	18.19	22	10	16664.12	9.09	—	—
18	100	10	19	7	15.72	21	10	328.08	9.52	—	—
19	100	10	20	7	26.53	21	9	292.67	4.76	—	—
20	100	10	18	7	18.99	21	10	20187.72	10.00	—	—
			Total cpu time		227.63	Total cpu time		40161.11			

out using SP_1 and SP_2 give a general idea about performances of SP_1 and SP_2. Thus, cpu-time that the algorithm would have given with SP_2 can be inferred from the results of SP_1 for the second and third data sets. Tables 6 and 7 display the results for the second data set, and Tables 8 and 9 display the results for the third data set.

In all tables, "*file no*." represents the identification of data file in the corresponding data set, "*size*" refers to the size of the networks (i.e., number of nodes in the network) and column "*p*" stands for value of the parameter *p*. Under heading "*Phase I*", "*obj*" is the objective function value obtained (i.e., bound passed to Phase II), "*iter. no.*" is number of iterations carried out and "*cpu t.*" is the amount of cpu time in seconds. Heading "*Compl. Algorithm*" includes analogous results for the complete algorithm (Phases I and II together). We have calculated total cpu time spent by the algorithm for all of the instances in each table. In Tables 2 and 4, we give percent deviations of bounds obtained in Phase I from optimal value. These percent deviation figures are calculated by dividing the difference between first and second phase objective function values by second phase objective and multiplying by 100. These tables also involve two more columns under heading "*CPLEX*". Column "*cpu t.*" gives cpu time CPLEX spends for solving CPC formulations of the instances. Column "*%Impv.*" displays the percent improvements of cpu time of the algorithm over cpu time of CPLEX. Percent improvements in this column are calculated by subtracting cpu time of complete algorithm from cpu time of CPLEX and dividing this difference by cpu time of CPLEX. This ratio is then multiplied by 100 to obtain the percentage improvement. In Tables 3

Table 3
Computational results obtained with modified Phase II on OR-Lib problems using SP_1^R ($1 \leqslant h_i \leqslant 20\ \forall i \in V$, $Q_j = 120\ \forall j \in W$)

File no.	Size	p	Phase I			Compl. algorithm			Cpu time impv. (%)	
			Obj.	Iter. no.	cpu t.	Obj.	Iter. no.	cpu t.	Over CPLEX	Over algo.
1	50	5	28	7	1.94	29	9	2.29	99.15	0.00
2	50	5	31	7	2.34	33	10	11.8	99.88	42.72
3	50	5	25	7	2.07	26	9	4.06	99.44	−4.37
4	50	5	31	7	2.45	32	9	5.49	99.17	1.08
5	50	5	27	7	2.17	29	10	6.91	98.81	30.69
6	50	5	27	7	2.04	31	12	22.36	97.46	37.44
7	50	5	30	7	2.04	30	8	5.89	99.46	−3.51
8	50	5	29	7	2.41	31	10	12.39	98.84	39.50
9	50	5	27	7	2.34	28	9	6.51	99.11	−5.00
10	50	5	29	7	2.18	32	11	30.55	98.97	48.44
11	100	10	18	7	19.46	19	9	33.95	—	25.78
12	100	10	19	7	16.95	20	9	45.27	—	15.08
13	100	10	19	7	24.9	20	9	38.57	—	22.63
14	100	10	20	7	21.94	20	8	36.01	—	−9.82
15	100	10	19	7	18.97	21	10	125.36	—	33.74
16	100	10	18	7	21.12	20	10	373.6	—	82.61
17	100	10	20	7	18.01	22	10	16180.69	—	2.90
18	100	10	19	7	15.79	21	10	183.23	—	44.15
19	100	10	20	7	26.69	21	9	260.7	—	10.92
20	100	10	18	7	18.82	21	11	690.04	—	96.58
			Total cpu time		224.63	Total cpu time		18075.67		

and 5, solution time improvements of modified Phase II over CPLEX and original algorithm are given, each in one column under heading "*Cpu time Impv.* (%)". These improvement figures are calculated by subtracting cpu time of modified algorithm from cpu time of CPLEX (respectively, cpu time of original algorithm) and dividing by cpu time of CPLEX (respectively, cpu time of original algorithm). This ratio is then multiplied by 100 to obtain the percentage value. If this value turns out to be negative, then this means modification increased solution time of the instance under consideration. Cpu time improvements over CPLEX are calculated only in the first four tables, because, the instances of the second and third data sets are not tractable by CPLEX.

Before starting with the analysis of tables, we should make some explanations regarding the system we have made experiments on. It was a multi-user system and cpu time that the server is employed changes from time to time depending on the traffic and priority assignments to tasks. Trials we have made showed that this difference in solution time is always within 1.5 s for individual instances regardless of type of linear model we solve (an IP, an MIP or and LP). Since we have made no modification on Phase I and we always use the same subproblem in Phase I (i.e., SP_1^{LR}), we expect that Phase I spends equal amounts of cpu time for each instance within first data set regardless of the subproblem we employ in Phase II. However, due to explained unavoidable conditions, we observed different solution times for each instance of first data set in the tables corresponding to SP_1 and SP_2. For example, total cpu time measures of Phase I in the first four tables are different from each other. The difference between the lowest and highest is

Table 4
Computational results on OR-Lib problems with the algorithm using SP_2 ($1 \leqslant h_i \leqslant 20\ \forall i \in V$, $Q_j = 120\ \forall j \in W$)

File no.	Size	p	Phase I			Compl. algorithm				CPLEX	
			Obj.	Iter. no.	cpu t.	obj.	Iter. no.	cpu t.	Devia (%)	cpu t.	% Impv.
1	50	5	28	7	2.01	29	9	11.64	3.45	271.23	95.71
2	50	5	31	7	2.36	33	10	24.9	6.06	10237.82	99.76
3	50	5	25	7	2.15	26	9	6.52	3.85	729.18	99.11
4	50	5	31	7	2.42	32	9	3.34	3.13	662.27	99.50
5	50	5	27	7	2.14	29	10	43.53	6.90	582.24	92.52
6	50	5	27	7	2.16	31	12	122.2	12.90	878.81	86.09
7	50	5	30	7	2.16	30	8	3.92	0.00	1034.53	99.62
8	50	5	29	7	2.44	31	10	13.04	6.45	1069.14	98.78
9	50	5	27	7	2.37	28	9	20.96	3.57	727.83	97.12
10	50	5	29	7	2.21	32	11	16.42	9.38	2956.88	99.44
11	100	10	18	7	20.08	19	9	103.06	5.26	—	—
12	100	10	19	7	18.12	20	9	117.45	5.00	—	—
13	100	10	19	7	25	20	9	120.65	5.00	—	—
14	100	10	20	7	22.67	20	8	513.85	0.00	—	—
15	100	10	19	7	19.29	21	10	1715.75	9.52	—	—
16	100	10	18	7	21.38	20	10	148.5	10.00	—	—
17	100	10	20	7	18.64	22	10	3852.08	9.09	—	—
18	100	10	19	7	15.88	21	10	6264.69	9.52	—	—
19	100	10	20	7	26.86	21	9	266.31	4.76	—	—
20	100	10	18	7	19.93	21	10	22342.75	10.00	—	—
			Total cpu time		230.27	Total cpu time		35711.56			

7.29 s (224.63 s in Table 3 and 231.92 s in Table 5), averaging 0.36 s for individual instances. This figure is negligible when considered within cpu times of the complete algorithm displayed in these tables. Having given this explanation on Phase I cpu times, from here on in our analysis we give comments on cpu times of complete algorithm, and take obtained solution time data as given without paying attention to possible fluctuations.

First, we analyze results of experiments on the first data set (Tables 2–5). Sizes of first 10 instances in OR-Lib are 50. The last 10 instances have size 100. We could solve CPC formulations of first ten instances with CPLEX in reasonable amounts of time. However, we could not solve any of the last ten instances within 12 h of clock time. For this reason, we present CPLEX cpu times and calculate improvements of algorithm over CPLEX for only first ten instances.

While this paper was under preparation, we became aware of the paper by Pallottino et al. [10] where the same OR-Lib instances were used to test a local search based heuristic algorithm. In this reference, the authors also try to solve the same test problems to optimality using the CPLEX 7.0 IP solver. They report that CPLEX fails to report an optimal solution within 15 h of computing time for instances No. 15, 18 and 20. We solved all these problems as well as the remaining ones to optimality. Our algorithm is the first, to the best of our knowledge, to obtain provably optimal solutions to these instances, in reasonable time.

Table 5
Computational results obtained with modified Phase II on OR-Lib problems using SP_2^R

File no.	Size	p	Phase I			Compl. Algorithm			Cpu time Impv.(%)	
			obj.	Iter. no.	cpu t.	obj.	Iter. no.	cpu t.	Over CPLEX	Over Algo.
1	50	5	28	7	1.99	29	9	7.19	97.35	38.23
2	50	5	31	7	2.41	33	10	11.17	99.89	55.14
3	50	5	25	7	2.16	26	9	4.15	99.43	36.35
4	50	5	31	7	2.55	32	9	3.82	99.42	−14.37
5	50	5	27	7	2.29	29	10	11.52	98.02	73.54
6	50	5	27	7	2.14	31	12	36.95	95.80	69.76
7	50	5	30	7	2.16	30	8	4.66	99.55	−18.88
8	50	5	29	7	2.44	31	10	10.88	98.98	16.56
9	50	5	27	7	2.43	28	9	23.27	96.80	−11.02
10	50	5	29	7	2.21	32	11	10.6	99.64	35.44
11	100	10	18	7	20.61	19	9	57.51	—	44.20
12	100	10	19	7	17.79	20	9	90.9	—	22.61
13	100	10	19	7	25.75	20	9	70.1	—	41.90
14	100	10	20	7	22.38	20	8	546.91	—	−6.43
15	100	10	19	7	19.3	21	10	330.31	—	80.75
16	100	10	18	7	21.58	20	10	121.44	—	18.22
17	100	10	20	7	18.68	22	10	2992.58	—	22.31
18	100	10	19	7	16.07	21	10	4534.19	—	27.62
19	100	10	20	7	27.49	21	9	187.18	—	29.71
20	100	10	18	7	19.49	21	11	2528.67	—	88.68
			Total cpu time		231.92	Total cpu time		11584.00		

Table 6
Computational results obtained with the algorithm on the second data set instances using SP_1

File no.	size	p	Phase I			Compl. algorithm			
			obj.	Iter. no.	cpu t.	obj.	Phase II iter. no.	cpu t.	Status
SJC1.dat	100	10	315.805328	12	26.82	364.72592	141	241980.88	Optimal
SJC2.dat	200	15	302.03476	11	235.52	304.13812	32	9075.2	Optimal
SJC3a.dat	300	25	274.016418	12	1076.54	278.95877	60	123559.82	Optimal
SJC3b.dat	300	30	244.002045	12	980.44	253.71243	104	41716.12	Optimal
SJC4a.dat	402	30	275.045441	12	3461.32	277.87228	51	512489.16	Lower bnd.
SJC4b.dat	402	40	236.072021	12	3293.28	239.38463	69	361921.28	Optimal

Tables 2 and 4 show that the cpu times that our algorithm exhibits in the first ten instances are quite low when compared with the cpu times of CPLEX. Improvement of the algorithm over the cpu time of CPLEX for each instance is given in right-most column of these tables. All of these improvement figures in both of the tables are higher than 95% and most of them are around 99%. They clearly show that our algorithm yields by far shorter solution times than CPLEX for both subproblems. All instances except

Table 7
Computational results obtained with modified Phase II on the second data set instances using SP_1^R

File no.	Size	p	Phase I			Compl. algorithm				
			Obj.	Iter. no.	cpu t.	Obj.	Ph. IIa Iter. no.	Ph. IIb Iter. no.	cpu t.	Status
SJC1.dat	100	10	315.80533	12	26.91	364.72592	102	40	196453.48	Opt.
SJC2.dat	200	15	302.03476	11	240.88	304.13812	32	1	2522.48	Opt.
SJC3a.dat	300	25	274.01642	12	1050.16	278.95877	56	5	37809.72	Opt.
SJC3b.dat	300	30	244.00205	12	1030.72	253.71243	91	14	13534.56	Opt.
SJC4a.dat	402	30	275.04544	12	3354.46	283.21899	134	13	974548.81	Lower b.
SJC4b.dat	402	40	236.07202	12	3372.53	239.38463	66	4	221112.05	opt.

Table 8
Computational results obtained with the algorithm on the third data set instances using SP_1

File no.	Size	p	Phase I			Compl. algorithm			
			Obj.	Iter. no.	cpu t.	Obj.	Phase II iter. no.	cpu t.	Status
G1.txt	100	5	92	8	53.12	94	3	1237.43	Optimal
G2.txt	100	5	92	8	47.13	94	3	1362.37	Optimal
G3.txt	100	10	58	7	36.57	83	6	4511.32	Optimal
G4.txt	100	10	58	7	38.09	84	7	37972.31	Optimal
G5.txt	150	10	93	8	139.26	95	3	11318.17	Optimal
G6.txt	150	10	93	8	156.82	96	4	60668.63	Optimal
G7.txt	150	15	85	8	119.13	89	5	21193.79	Optimal
G8.txt	150	15	85	8	146.99	89	5	52427.16	Optimal

Table 9
Computational results obtained with modified Phase II on the third data set instances using SP_1^R

File no.	Size	p	Phase I			Compl. Algorithm				
			Obj.	Iter. no.	cpu t.	Obj.	Ph. IIa Iter. no.	Ph. IIb Iter. no.	cpu t.	Status
G1.txt	100	5	92	8	54.62	94	3	1	322.6	Optimal
G2.txt	100	5	92	8	47.80	94	3	1	419.18	Optimal
G3.txt	100	10	58	7	37.65	83	6	1	1392.45	Optimal
G4.txt	100	10	58	7	39.28	84	7	1	3562.89	Optimal
G5.txt	150	10	93	8	148.98	95	3	1	8072.89	Optimal
G6.txt	150	10	93	8	164.46	96	4	1	10288.35	Optimal
G7.txt	150	15	85	8	125.06	89	5	1	10422.89	Optimal
G8.txt	150	15	85	8	149.68	89	5	1	18577.65	Optimal

three (files 16, 17 and 20) are solved within 5.5 min when the algorithm employs SP_1. As for SP_2, we see in Table 4 that files 15, 17, 18 and 20 could not be solved very efficiently.

In Tables 2 and 4 we can also see that SP_1 performs better than SP_2 in 14 of the 20 instances. However, the total of the cpu times for 20 instances display that SP_2 (35711.56 s) took shorter time than SP_1 (40161.11 s). This difference in the total solution times of the two subproblems is caused mainly by instances 16 and 17. SP_2 could solve instance 16 in 148.5 s and instance 17 in 3852.08 s. These solution times are quite small compared to the figures yielded by SP_1, 2147.98 s for instance 16 and 16664.12 s for instance 17. Although SP_1 gave better solution times for most of the instances, since its solution times for the mentioned two instances turned out to be far higher than those of SP_2, total cpu time measure tells us SP_2 worked faster on first data set. However, if we exclude the two instances from the total cpu time, SP_1 beats SP_2 by 21349.01 to 31710.98 s. From these results we can say that SP_1 works faster for most of first data set instances, however, for some instances it may give high run times.

Comparing Tables 2 and 3 we can assess the performance improvement of modification when SP_1 is employed. In these two tables, in 4 of the instances (files 3, 7, 9 and 14), solution times of the modified algorithm turned out to be slightly longer than those of the original algorithm. For the rest of the instances, modified Phase II brought improvements that are displayed on the last column of Table 3. These improvement figures clearly show that modification reduces the solution time of the algorithm considerably. Total cpu time of the algorithm decreased from 40161.11 to 18075.67 s by the modification, which is another sign of effectiveness of the modification in increasing the efficiency of the algorithm.

Similarly, from Tables 4 and 5 we see that four instances (files 4, 7, 9, 14) could be solved more efficiently by the original algorithm using SP_2. However, total cpu time is reduced from 35711.56 to 11,584 s by the modification. Also improvement figures once more clearly show that modification on Phase II proves beneficial in increasing the efficiency of the algorithm on these instances.

If solution time of the original algorithm is already not high or number of Phase II iterations carried out is low (e.g. one), then cpu time improvement obtained with the algorithm may turn out to be low, even negative. This is because our modification works by replacing some of IPs by MIPs in Phase II. At least one IP remains in modified Phase II. If replaced IPs are already easy problems, then our modification does not shorten solution time since we increase the total number of subproblems in second phase. Our modification yields good results when replaced IPs are hard, bottleneck problems and their MIP counterparts are more easily solvable. If our modification fails to by-pass the bottleneck problems, that is, if the IPs that remain to be solved in modified Phase II are hard problems, then our modification brings negligible or no improvement.

When we compare performances of SP_1 and SP_2 in modified Phase II (Tables 3 and 5), we see that total cpu time for SP_1 is 18075.67 s, where the same figure for SP_2 is 11584 s. However, a closer look at the tables reveals that instance 17 turns out to be pathologic for SP_1 (solved in 16180.69 s) and its solution time constitutes most of the total cpu time. If we exclude this instance from total cpu time, we obtain a total duration of 1894.98 s for SP_1. The same total for SP_2 is 8591.42 s, which is much higher than cpu time corresponding to SP_1. Similarly, results of modified algorithm tell that SP_1 works faster for most of the instances from OR-Lib, however, it may fail to give quick results for some of the instances. SP_2 may prove faster than SP_1 for cases where SP_1 took long time to solve.

In Tables 6–9 we summarize our computational experience with problems that were used by Pallottino et al. [10] to test their local search heuristic algorithms. These problems are usually much harder to solve to optimality compared to the OR-Lib problems. Nevertheless, we were able to obtain optimal solutions to most of these problems (with the exception of SJC4a, where we use the naming convention

of Pallottino et al. [10] for test problems) albeit after very long computing times in certain cases. However, it is inconceivable to obtain optimal solutions to these problems by an off-the-shelf solver such as CPLEX even within such computing times. Furthermore, the solutions reported in [10] for the same test problems were found to be never optimal. In fact, in most cases their solution were far from the optimal value. Our contribution on the second and third set of test problems is to be able to solve, for the first time to the best of our knowledge, to optimality these very difficult instances of capacitated *p*-center problems. In the only instance SJC4a where we were not able to find the optimal solution, our algorithm computed a lower bound equal to 283.21899 after approximately 10 days of computing time whereas the upper bound computed by the Pallottino et al. heuristic is equal to 317.65 computed in approximately 10 min of CPU time. In the second set of test problems, the considerable increase in computing time can be attributed to the fact that the distance matrices of these test problems contain very closely clustered distance values which causes the number of Phase II iterations to increase significantly. This difficulty is not as pronounced in the third set of test problems, and we obtain the optimal solutions in Tables 8 and 9 in more reasonable computing times. We observe in the second and third set of problems that again the modification of Phase II helps decrease run times of the algorithms. Hence, in all cases it is advisable to adopt the modified Phase II algorithm. An exception occurred in the second set of test problems, in problem SJC4a where the algorithm with modified Phase II took longer run time to reach a lower bound than the version without the Phase II modification. The reason for this discrepancy is the following. In carrying out the tests on the second and third set of data, we enforced an upper bound of 24 h of CPU time (86,400 s) per subproblem. When this bound was reached for any subproblem, the algorithm terminated with a lower bound, which occurred only for SJC4a in our experiments. However, this bound was reached earlier in our run reported in Table 6 compared to the run in Table 7. Therefore, the algorithm with the modified Phase II carried many more iterations (147 versus 51) than the algorithm without the modification. Hence, the longer computing time.

5. Concluding remarks

In this paper, we propose an exact and practical algorithm for solving the capacitated *p*-center problem. To the best of our knowledge, it is the first algorithm that optimally solves the problem in general graphs. We obtained excellent improvements over cpu times exhibited by CPLEX. However, we should state that the algorithm we present here has still room for further improvements. Bottleneck of the algorithm is the subproblems of Phase II. Thus, by decreasing the number of subproblems solved in Phase II and by solving the subproblems more efficiently, we can reach smaller running time figures.

Tighter bounds from LP relaxation of $SP_1(.)$(i.e., $SP_1^{LR}(.)$) give a Phase I bound which is closer to optimal radius value. This decreases the number of Phase II iterations. Obtaining tight bounds from $SP_1^{LR}(.)$ may be possible by adding some valid inequalities or cuts to its formulation.

More efficient algorithms that exploit the special structures of SP_1 (the capacitated concentrator location problem) and SP_2 (the bin-packing problem) can be applied to the subproblems and shorter running times can be obtained. Indeed, several algorithms for each of these problems exist in the literature. However, existing algorithms have to be modified to account for the differences of our subproblems from these standard problems.

Acknowledgements

The authors thank Hande Yaman and Oya Karaşan for discussions on the subject of the paper, and an anonymous referee for useful suggestions.

References

[1] Ilhan T, Pınar MÇ. An efficient exact algorithm for the vertex p-center problem, http://www.optimization-online.org/DB-HTML/2001/09/376.html, 2001.
[2] Kariv O, Hakimi SL. An algorithmic approach to network location problems part I: the p-centers. SIAM Journal of Applied Mathematics 1979;37:513–38.
[3] Mladenović N, Labbé M, Hansen P. Solving the p-center problem with tabu search and variable neighborhood search. Networks 2003;42:48–64.
[4] Elloumi S, Labbé M, Pochet Y. A new formulation and resolution method for the p-center problem. INFORMS Journal on Computing, 2001, to appear.
[5] Daskin MS. A new approach to solving the vertex p-center problem to optimality: algorithm and computational results. Communications of the Operations Research Society of Japan 2000;45:428–36.
[6] Daskin MS. Network and discrete location. New York: Wiley; 1995.
[7] Bar-Ilan J, Kortsarz G, Peleg D. How to allocate network centers. Journal of Algorithms 1993;15:385–415.
[8] Khuller S, Sussmann YJ. The capacitated K-center problem. SIAM Journal on Discrete Mathematics 2000;13:403–18.
[9] Jaeger M, Goldberg J. A polynomial algorithm for the equal capacity p-center problem on trees. Transportation Science 1994;28:167–75.
[10] Pallottino S, Scappara MP, Scutellà MG. Large scale local search heuristics for the capacitated vertex p-center problem. Networks 2002;43(4):241–55.
[11] Deng Q, Simchi-Levi D. Valid inequalities, facets and computational experience for the capacitated concentrator location problem. Rsearch report, Department of Industrial Engineering and Operations Research, Columbia University, New York, 1993.
[12] Martello S, Toth P. Knapsack problems: algorithms and implementations. New York: Wiley; 1990.
[13] Beasley JE. A note on solving large p-median problems. European Journal Operational Research 1985;21:270–3.
[14] Lorena LAN, Senne ELF. A column generation approach to capacitated p-median problems. Computers and Operations Research 2004;31(6):863–76.