An Exact Algorithm for the Capacitated Vertex p-Center Problem

F. Aykut Özsoy*

Mustafa Ç. Pınar †

September 9, 2003

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. Experiments on OR-Lib instances yield promising results.

Key words. 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 [7] for the basic vertex p-center problem is modified and extended to solve the capacitated version to optimality. Computational experiments are carried out on instances from OR-Lib [2] 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 [9]. Recent articles on the basic p-center problem include Mladenović, Labbé and Hansen [12], Elloumi, Labbé and Pochet [6] and Daskin [4]. For a detailed exposition

*E-mail: *aykut@bilkent.edu.tr* Department of Industrial Engineering, Bilkent University, 06800 Ankara, Turkey.

†Corresponding author. E-mail: *mustafap@bilkent.edu.tr*. Department of Industrial Engineering, Bilkent University, 06800 Ankara, Turkey.

of the p-center problem and available solution methodology, the reader is directed to Chapter 5 of the textbook [3].

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 5, we deal with a single vertex set, i.e., with the special case where $W = V$).

An integer programming formulation for the basic p-center problem is the following (see, e.g., [3]):

$$
\begin{aligned}
&\text{Minimize} && z && && \\
&\text{subject to} && && && \\
&\sum_j x_{ij} && = && 1 \quad \forall i \in V && (1.1)\\
&x_{ij} && \leq && y_j \quad \forall i \in V, j \in W && (1.2)\\
&\sum_{j \in W} y_j && \leq && p && (1.3)\\
&\sum_{j \in W} d_{ij} x_{ij} && \leq && z && (1.4)\\
&x_{ij}, y_j && \in && \{0,1\} \quad \forall i \in V, \forall j \in W && (1.5)
\end{aligned}
$$

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 (1.1) express the requirements that all clients must be assigned to some facility site. Constraints (1.2) 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 (1.3).

In the basic p-center problem, we assume that the facilities have infinite capacities and thus assignment of clients to facilities can be done without considering quantities of demands clients have. In the capacitated version of the problem we are leaving aside this assumption. In this version each client is labelled with some quantity of demand, and assignment of clients to facilities is constrained with capacity restrictions of facilities. In other words, the total of demands of clients assigned to a certain facility can not 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.

Formulation of the capacitated p-center problem (CPC) is very similar to the p-center formulation. The only difference arises in the necessity for considering the capacity restrictions of the facilities. 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 can not exceed Q_j. We add one more set of inequalities (constraints (1.10) which account for capacity restrictions of facilities) to the p-center formulation to obtain the following IP model of the capacitated p-center problem:

Problem CPC

$$\begin{aligned}
&\text{Minimize} && z && && \\
&\text{subject to} && && && \\
&\sum_{j\in W} x_{ij} && = && 1 \quad \forall i \in V && (1.6)\\
&x_{ij} && \leq && y_j \quad \forall i \in V, j \in W && (1.7)\\
&\sum_{j\in W} y_j && \leq && p && (1.8)\\
&\sum_{j\in W} d_{ij}x_{ij} && \leq && z \quad \forall i \in V && (1.9)\\
&\sum_{i\in V} h_i x_{ij} && \leq && Q_j \quad \forall j \in W && (1.10)\\
&x_{ij}, y_j && \in && \{0,1\} \quad \forall i \in V, \forall j \in W. && (1.11)
\end{aligned}$$

The rest of this paper is organized as follows. Section 2 presents a brief literature survey on the capacitated p-center problem. In Section 3 we lay the grounds for the CPC algorithm. Section 4 gives a detailed description of the proposed algorithm. Section 5 contains computational results obtained with the algorithm. Concluding remarks are given in Section 6.

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, Kortsarz and Peleg [1] 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 [10] 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 [8]. 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, Scappara and Scutellà [13] 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 An Algorithm for the Basic p-Center Problem

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 [7]. 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 feasibility problem, which is going to be explained in the next paragraph), and updates lower and upper bounds on the optimal radius in the light of this information.

The set-covering feasibility problem that is used ('feasibility subproblem' from here on) aims to find whether it is possible to cover all clients within a coverage distance with at most p facilities. In other words, this subproblem is concerned with whether the following set of inequalities defined with respect to coverage distance ε is feasible.

$$\begin{aligned} \sum_{j \in W: d_{ij} \leq \varepsilon} y_j &\geq 1 \quad \forall i \in V \\ \sum_{j} y_j &\leq p \\ y_j &\in \{0,1\} \quad \forall j \in W \end{aligned}$$

The idea underlying the solution procedure is roughly to proceed as follows: select initial lower and upper bounds on the optimal objective function value; solve the feasibility subproblem by using the average of the lower and upper bounds as the coverage distance; if the subproblem turns out to be feasible, reset the upper bound to the coverage distance that was just used; if not, reset the lower bound to the coverage distance just used. If the lower and upper bounds are equal, stop; otherwise set the coverage distance as the average of lower and upper bounds, and continue the process.

To avoid solving successive IPs of set-covering subproblems, the algorithm is designed as a two-phase variant of the above procedure to speed up the process. In the first phase, a binary search for a suitable lower bound on the optimal value is carried out by solving LP relaxations of the subproblems. In the second phase subproblems are solved as IPs using availabe state-of-the-art general purpose IP software (e.g., CPLEX), for increasing coverage distances starting from the bound provided by Phase I. The first coverage distance encountered in this process within which all clients can be covered with at most p facilities is the optimal objective value of the p-center problem.

The step by step flow of the algorithm is as follows[1]:

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 the LP relaxation of the feasibility subproblem with coverage distance ε.

Step 3.1. If problem is infeasible with respect to this coverage distance, then set $l = \varepsilon$.

Step 3.2. else set $u = \varepsilon$.

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

Step 4.1. If $(u - l) \leq 1$ then go to Step 5.

Step 4.2. else go to Step 2.

Step 5. If previous subproblem was infeasible, then set $\varepsilon = u$, else set $\varepsilon = l$.

Phase II

Step 6. Create the feasibility subproblem (in IP form) by setting coverage distance to ε and solve.

[1] This algorithm is for integer data set.

Step 6.1. If feasibility subproblem is infeasible, 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.

4 The Proposed Algorithm for Solving CPC

As can be seen from the IP formulation of CPC, the capacitated version of the vertex p-center problem is the same as the uncapacitated version except that its feasible region is constrained with one more set of inequalities (i.e., constraints (1.10)). Although these new sets of constraints introduce some difficulties in the solution process, they do not imply a change in the main structure of the problem. That is, the problem still has a min-max type objective function. The objective function and constraints (1.9) in the formulation dictate that optimal objective value is going to be equal to an entry of the distance matrix, just as it is in the uncapacitated case. Since the algorithm of the previous section carries out a search over distance values, that search procedure can also be used for solving the capacitated version of the problem. The main body of the algorithm (i.e., the search procedure) needs no modification. The required modification is mainly the necessity for appending capacity restrictions to the subproblems. However, adding capacity restrictions to the subproblems is not straightforward computationally as will be explained now.

While solving the p-center problem, in each iteration of the algorithm, we are checking whether it is feasible to cover all clients within a specific radius value ε by using at most p facilities. This feasibility check is carried out by means of a set-covering subproblem as was explained in the previous section. The critical information that is obtained by the set-covering subproblem is the locations of the facilities. Assignment of clients to facilities is a trivial task since we are sure that every client has at least one facility located within its ε neighborhood. Each client can be assigned to any one of the facilities located within ε neighborhood. Thus, there is no need to use assignment variables in the subproblem of the previous section's basic p-center algorithm.

In context of capacitated p-center problem, however, information as to where to locate the facilities is not sufficient alone to conclude whether a radius value is feasible. Because, assignment of clients to facilities is not trivial as it is in the uncapacitated case. That is, for a set of facility sites which can cover all clients within ε radius, due to capacity restrictions of facilities we can not be sure as to whether a feasible assignment exists or not. In other words, covering all clients with at most p facilities does not guarantee that clients can be assigned to facilities within capacity restrictions. The total of demands of customers who are assigned to a certain facility may exceed its capacity, which implies infeasibility of the assignment under consideration. For this reason, in the capacitated context, assignment variables have to be included in subproblems along with

location variables to account for the non-trivial assignment of clients to facilities.

Unlike the feasibility subproblem of the previous section, subproblems that are going to be developed here for solving CPC have the objective of minimizing the number of facilities to be located (reasons for choosing this objective will be clarified later). In Section 4.1 we explain these subproblems in detail. Some changes in the algorithm are necessary owing to special structures of subproblems that will be used. These changes are given in Section 4.2. Finally, in Section 4.3, a modification of Phase II is proposed to speed up the procedure.

4.1 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. As we have discussed above, both subproblems have assignment variables. 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 the following paragraphs.

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 $SP_1(\varepsilon)$:

$$
\begin{aligned}
& min && \sum_{j=1}^{n} y_j \\
& subject\ to \\
& \sum_{j \in W:\ d_{ij} \le \varepsilon} x_{ij} && = \quad 1 \quad \forall i \in V && (4.1) \\
& \sum_{i \in V:\ d_{ij} \le \varepsilon} h_i x_{ij} && \le \quad Q_j \quad \forall j \in W && (4.2) \\
& x_{ij} && \le \quad y_j \quad \{(i,j) \in V \times W : d_{ij} \le \varepsilon\} && (4.3) \\
& x_{ij}, y_j && \in \quad \{0,1\} \quad \{(i,j) \in V \times W : d_{ij} \le \varepsilon\}
\end{aligned}
$$

Constraints (4.1) ensure that each client is covered by one facility whose distance to the client is at most ε. Constraints (4.2) stand for capacity restrictions of facilities. And constraints (4.3) 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 [5]). Capacitated concentrator location problem can be described as follows: given 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 (4.1) and (4.2) (i.e., the condition $\{j \in W :\ d_{ij} \le \varepsilon\}$ in (4.1) and $\{i \in V :\ d_{ij} \le \varepsilon\}$ in (4.2)) 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 (4.1) by $\{\forall j \in W\}$, the one in (4.2) 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 (4.2) and (4.3) in SP_1 by one single set of constraints. Intuitively, it is easy to see that the constraint set

$$\sum_{i \in V:\ d_{ij} \leq \varepsilon} h_i x_{i,j} \quad \leq \quad Q_j y_j \quad \forall j \in W \tag{4.4}$$

can account for these two sets of inequalities. By introducing (4.4), we can formulate an alternative subproblem SP_2 which defines the same feasible region as SP_1 (this result will be proved later).

Subproblem $SP_2(\varepsilon)$:

$$\begin{aligned} & min && \sum_{j=1}^{n} y_j \\ & subject\ to \\ & \sum_{j \in W:\ d_{ij} \leq \varepsilon} x_{ij} && = \quad 1 \quad \forall i \in V && (4.5) \\ & \sum_{i \in V:\ d_{ij} \leq \varepsilon} h_i x_{ij} && \leq \quad Q_j y_j \quad \forall j \in W && (4.6) \\ & x_{ij}, y_j && \in \quad \{0,1\} \quad \{(i,j) \in V \times W : d_{ij} \leq \varepsilon\} \end{aligned}$$

This subproblem resembles *'the bin-packing problem'* (e.g., see [11]). 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 can not 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 $\{(i,j) : d_{ij} \leq \varepsilon\}$.

Now we establish equivalence of $SP_1(\varepsilon)$ and $SP_2(\varepsilon)$.

Proposition 4.1. *If one of $SP_1(\varepsilon)$ and $SP_2(\varepsilon)$ is infeasible then the other is infeasible, too. If one of $SP_1(\varepsilon)$ and $SP_2(\varepsilon)$ has a feasible solution then the other also does, and their optimal objective values are equal to each other.*

Proof: Let $\mathcal{N}$ denote the number of assignment variables (i.e., x_{ij}'s) in subproblems $SP_1(\varepsilon)$ and $SP_2(\varepsilon)$ constructed for an instance with distance matrix D. We will show that if there exists a feasible point that is feasible for one of SP_1 and SP_2, it must also be feasible for the other, too. Let $(\mathbf{x}^*, \mathbf{y}^*) \in \{0,1\}^{\mathcal{N}+M}$ be a feasible point for SP_2. Since $(\mathbf{x}^*, \mathbf{y}^*)$ is feasible for SP_2, $\mathbf{x}^*$ satisfies 4.1 in SP_1. Now let $S_0 = \{j : y_j^* = 0\}$ and $S_1 = \{j : y_j^* = 1\}$ (note that $S_0 \cup S_1 = W$).

- For each $\tilde{j} \in S_0$, we have $x^*_{i\tilde{j}} = 0 \; \forall i \in \{i : d_{i\tilde{j}} \leq \varepsilon\}$ by constraints 4.6. This means, for each $\tilde{j} \in S_0$, $y^*_{\tilde{j}}$ and $x^*_{\tilde{i}\tilde{j}}$ $(\forall \tilde{i} \in \{i : d_{i\tilde{j}} \leq \varepsilon\})$ satisfy constraints 4.2 and 4.3 in SP_1.
- For each $\tilde{j} \in S_1$, constraints 4.6 in SP_2 turn equivalent to 4.2 in SP_1. Also we can observe that for $\tilde{j} \in S_1$, constraints 4.3 in SP_1 become redundant. Thus, for each $\tilde{j} \in S_1$, $y^*_{\tilde{j}}$ and $x^*_{i\tilde{j}}$ $(\forall \tilde{i} \in \{i : d_{i\tilde{j}} \leq \varepsilon\})$ satisfy constraints 4.2 and 4.3 in SP_1.

This means $(\mathbf{x}^*, \mathbf{y}^*)$ is also feasible for SP_1. Now assume $(\mathbf{x}^*, \mathbf{y}^*) \in \{0,1\}^{\mathcal{N}+M}$ is feasible for SP_1. Similarly, since $(\mathbf{x}^*, \mathbf{y}^*)$ is feasible for SP_1, $\mathbf{x}^*$ satisfies 4.5. Define S_0 and S_1 as above.

- For each $\tilde{j} \in S_0$, we must have $x^*_{i\tilde{j}} = 0 \; \forall \tilde{i} \in \{i : d_{i\tilde{j}} \leq \varepsilon\}$ by constraints 4.3. This means defined $y_{\tilde{j}}$ and $x_{\tilde{i}\tilde{j}}$ are feasible for SP_2.
- For each $\tilde{j} \in S_1$, $x_{\tilde{i}\tilde{j}}$ (where $\tilde{i} \in \{i : di\tilde{j} \leq \varepsilon\}$) satisfy 4.6 in SP_2. Thus, defined $y_{\tilde{j}}$'s and $x_{\tilde{i}\tilde{j}}$'s are also feasible for SP_2.

This shows that a feasible point for SP_1 must also be feasible for SP_2 and vice versa. This establishes that if one of the problems is infeasible, then the other must also be infeasible (first part of result). Moreover, if these problems are feasible, they have the same set of feasible points. Since they have the same objective function, they yield the same optimal objective value (second part of result). □

Note that, in the proof of Proposition 4.1 we do not use the integrality restrictions of x_{ij} variables. This means this proof is also applicable to the case where x_{ij} variables are allowed to take on continuous values between 0 and 1 in both subproblems. Thus, if we replace '$x_{ij} \in \{0,1\} \; \{(i,j) \in V \times W : d_{ij} \leq \varepsilon\}$' with '$0 \leq x_{ij} \leq 1 \; \{(i,j) \in V \times W : d_{ij} \leq \varepsilon\}$' in both subproblems, we can still claim that Proposition 4.1 holds. However, relaxing y_j variables disturbs this equivalence. For this reason, LP relaxations of the two subproblems are not equivalent to each other, as will be seen later.

The essence of this proposition lies in identical paths that Phase II follows when we employ SP_1 or SP_2. For a specific radius value, the two subproblems return the same answer. Thus, they carry out the same number of iterations until termination provided that they start Phase II from the same radius value.

4.2 Algorithm

The main modification we propose in the basic p-center algorithm is the development of new subproblems. However, these new subproblems necessitate some minor modifications in some steps of the algorithm. In this section, we explain these required minor modifications and finally present the statement of the algorithm after necessary changes are made.

The basic p-center algorithm of Section 3 uses feasibility type subproblems and iteratively checks whether subproblems constructed for different radius values have feasible points. However, our subproblems have the objective of minimizing the number of facilities to be located. If optimal objective of our subproblems turn out to be more than p, this means it is not possible to find a set of at most p facility sites which can cover all clients within ε radius without exceeding capacity restrictions. Conversely, if optimal objective of our subproblems is at most p, this means it is possible to find such a set of at most p sites. Impossibility of finding such a set is called 'infeasibility' in terms of Section 3. That is, infeasibility of feasibility subproblem has a similar interpretation with optimal objectives of our subproblems being strictly greater than p. Our first minor modification in the algorithm stems from the necessity for different articulations of this 'impossibility' due to usage of different subproblems. Using SP_1 and SP_2 requires replacement of the statements 'if problem is infeasible' with 'if objective is more than p' in steps 3.1, 5 and 6.1 of the algorithm.

Our second minor modification is about subproblems of Phase I. Recall that the algorithm solves LP relaxations of the subproblems in the first phase. The success of the bound obtained from first phase of the algorithm is directly dependent on the tightness of LP relaxation of the subproblem. This dependence is going to be explained now.

Let $OPT(X)$ be the optimal objective value of a linear optimization model X, where X may be an LP, IP or an MIP. Suppose we are using subproblem SP_k (k=1 or 2) in the algorithm. Define SP_k^{LR} as LP relaxation of SP_k. Suppose ε_I^* is the radius value that is passed from Phase I to Phase II. Let ε^* be the minimum value in distance matrix such that $OPT(SP_k(\varepsilon^*)) \leq p$ (i.e., ε^* is the optimal radius we are looking for). It is easy to see that $\forall\, \varepsilon < \varepsilon^*$, $OPT(SP_k(\varepsilon)) > p$ and conversely, $\forall\, \varepsilon \geq \varepsilon^*$, $OPT(SP_k(\varepsilon)) \leq p$. Note that $\varepsilon_I^* \leq \varepsilon^*$. The success of the algorithm directly depends on proximity of ε_I^* to ε^*. The number of distinct d_{ij} values between ε_I^* and ε^* gives number of Phase II iterations that needs to be carried out. Tightness of LP relaxation is the most important factor that determines the gap between these two values. To clarify this further, now assume we are trying an ε value that is slightly smaller than ε^* in Phase I. We know $OPT(SP_k(\varepsilon)) > p$. If $OPT(SP_k^{LR}(\varepsilon)) \leq p$, Phase I of the algorithm will iteratively try smaller values of ε until it obtains $OPT(SP_k^{LR}(\varepsilon)) > p$. If the gap between $OPT(SP_k^{LR}(\varepsilon))$ and $OPT(SP_k(\varepsilon))$ is large (if LP relaxation of $SP_k(\varepsilon)$ gives a loose bound), it will need to try smaller and smaller values of ε to terminate Phase I (ε_I^* will fall far away from ε^*). However, if that gap is small (if LP relaxation of $SP_k(\varepsilon)$ gives a tight bound), we will not need to decrease ε too much to terminate Phase I and thus obtain a nice ε_I^* bound to start search in Phase II.

This case is encountered in computational experiments carried out with LP relaxation of SP_2. Although computational experiments are explained in detail in Section 5, we will present a preliminary result here to display the poor Phase I bounds obtained with SP_2. Table 1 on page 12 compares Phase I objective values (i.e., ε_I^*) obtained by using LP relaxations of SP_1 and SP_2. In

			SP_1		SP_2	
file no	size	p	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

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

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 drawn downwards until it is equal to the initial lower bound $l = \min\{d_{ij} : i \in V, j \in W\}$), which yields $\varepsilon_I^* = 1$. However, the optimal radius ε^* 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 following 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.

Subproblem $SP_1^{LR}(\varepsilon)$:

$$
\begin{aligned}
& min && \sum_{j=1}^{n} y_j \\
& subject\ to \\
& \sum_{j \in W:\ d_{ij} \le \varepsilon} x_{ij} && = 1 \quad \forall i \in V && (4.7) \\
& \sum_{i \in V:\ d_{ij} \le \varepsilon} h_i x_{ij} && \le Q \quad \forall j \in W && (4.8) \\
& x_{ij} && \le y_j \quad \{(i,j) \in V \times W : d_{ij} \le \varepsilon\} && (4.9) \\
& 0 \le \quad x_{ij} && \le 1 \quad \{(i,j) \in V \times W : d_{ij} \le \varepsilon\} \\
& 0 \le \quad y_j && \le 1 \quad \forall j \in W
\end{aligned}
$$

Now we can state the proposed algorithm as can be applied to the capacitated p-center problem. The step-by-step flow of the algorithm that employs subproblem SP_k ($k = 1$ or 2) is as follows:

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 more than p, then set $l = \varepsilon$.

Step 3.2. else set $u = \varepsilon$.

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

Step 4.1. If $(u - l) \le 1$ then go to Step 5.

Step 4.2. else go to Step 2.

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

Phase II

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

Step 6.1. If optimal objective is more 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, LP relaxation of SP_1 is used in the first phase no matter which subproblem is utilized in 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 upates 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. Since they will exhibit the same course of search in second phase by Proposition 4.1, SP_1 and SP_2 will terminate the 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.

4.3 A Modification of Phase II

As can be seen in the computational results of Section 5, 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 following subproblems:

Subproblem $SP_1^R(\varepsilon)$:

$$
\begin{aligned}
& min && \sum_{j=1}^{n} y_j && \\
& subject\ to && && \\
& \sum_{j \in W:\ d_{ij} \le \varepsilon} x_{ij} && = 1 \quad \forall i \in V && (4.10)\\
& \sum_{i \in V:\ d_{ij} \le \varepsilon} h_i x_{ij} && \le Q_j \quad \forall j \in W && (4.11)\\
& x_{ij} && \le y_j \quad \{(i,j) \in V \times W : d_{ij} \le \varepsilon\} && (4.12)\\
& y_j && \in \{0,1\} \quad \forall j \in W && (4.13)\\
& 0 \le\ x_{ij} && \le 1 \quad \{(i,j) \in V \times W : d_{ij} \le \varepsilon\} &&
\end{aligned}
$$

Subproblem $SP_2^R(\varepsilon)$:

$$
\begin{aligned}
& min && \sum_{j=1}^{n} y_j && (4.14)\\
& subject\ to && && \\
& \sum_{j \in W:\ d_{ij} \le \varepsilon} x_{ij} && = 1 \quad \forall i \in V && (4.15)\\
& \sum_{i \in V:\ d_{ij} \le \varepsilon} h_i x_{ij} && \le Q_j y_j \quad \forall j \in W && (4.16)\\
& y_j && \in \{0,1\} \quad \forall j \in W && (4.17)\\
& 0 \le\ x_{ij} && \le 1 \quad \{(i,j) \in V \times W : d_{ij} \le \varepsilon\} &&
\end{aligned}
$$

In second phase of the algorithm, before solving the subproblem in IP form, we are going to try the above MIP versions of the subproblems. If their optimal objective turn out to be more than

p, we do not need to solve the subproblem in IP form. This means we find out whether a radius value is infeasible in a quicker manner. If the optimal objective value of $SP_{\cdot}^{R}(\varepsilon)$ turns out to be less than or equal to p, we will solve the subproblem in IP form and continue as in the previous reasoning. This way of thinking leads us to a larger number of subproblems solved in total but fewer subproblems solved as IPs. So, we may expect to shorten the cpu time of the algorithm by this modification. Now follows the formal representation of Phase II after the proposed modification (for SP_k where k=1 or 2):

Modified Phase II

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

Step 6.1. If optimal objective is more 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.

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

Step 7.1. If optimal objective is more 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 7.2. else Stop.

It is easy to see that this modification increases the number of subproblems solved in Phase II by at least 1. However, computational results in Section 5 show that this modified version of the algorithm yields shorter cpu times than its original version given on page 13. Indeed, the increase in number of subproblems solved is realized as one most of the time. We illustrate this in Table 2 where we give the number of subproblems solved in original and modified versions of Phase II for each instance of OR-Lib. Since the original Phase II of the algorithm solves one subproblem in each iteration, we can take the number of subproblems solved in original Phase II as the number of Phase II iterations carried out. In this table, we see that in only one of twenty instances (File 9) the number of subproblems solved in modified Phase II exceeds number of Phase II iterations by more than one.

Note that, since Proposition 4.1 also holds when x_{ij}'s are relaxed (i.e., $SP_1^R(.)$ and $SP_2^R(.)$ are equivalent by Proposition 4.1), $SP_1^R(.)$ and $SP_2^R(.)$ will bring along the same number of Phase II iterations and the same number of subproblems solved in Phase II. 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.

File No.	No. of Phase II Subproblems Solved	No. of Modified Phase II Subproblems Solved
1	2	3
2	3	4
3	2	3
4	2	3
5	3	4
6	5	6
7	1	2
8	3	4
9	2	4
10	4	5
11	2	3
12	2	3
13	2	3
14	1	2
15	3	4
16	3	4
17	3	4
18	3	4
19	2	3
20	4	5

Table 2: Numbers of subproblems solved for original Phase II and modified Phase II.

5 Computational Results

In this section we report the computational results obtained with the algorithm using the two subproblems. The experiments are carried out on OR-Lib [2] *capacitated p-median* problems 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.

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 10100 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. We have made experiments on two data sets both of which depend on these distance matrices. First data set uses the demand and capacity data given in OR-Lib: demands of clients are integers between 1 and 20 (i.e., $1 \leq h_i \leq 20 \; \forall i \in V$) and capacity of each facility is equal to 120.

As we have mentioned before, we cover a generalized version of the problem in this study. All papers referenced in Section 2 cover the case where facilities are equally capacitated and clients have equal (or unit) demand (i.e., a limit is put on the number of clients a facility can serve). To be able to try our algorithm also in this setting, we obtained our second data set by just changing demands of clients to unity and capacities of facilities to 12 in each instance of OR-Lib (i.e., $h_i = 1 \; \forall i \in V \;\; Q_j = 12 \; \forall j \in W$).

We present the results in eight tables: first four, Tables 3 - 6, display the results for the first data set, and last four, Tables 7 - 10, present results corresponding to the second data set. Tables 3 and 4 involve results regarding usage of SP_1 in the algorithm when it is applied to first data set instances with original and modified Phase II's, respectively. Tables 5 and 6 display analogous results for SP_2. Tables 7 and 8 present results of experiments on the second data set while employing SP_1 in the algorithm with original and modified Phase II's, respectively. Tables 9 and 10 give analogous results for SP_2. In all tables, *"file no."* represents the identification of data file in OR-Lib, *"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, ε_I^*), *"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 (Phase I and Phase II together). We have calculated total cpu time spent by the algorithm for all of the instances in each table. In Tables 3, 5, 7 and 9 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 4, 6, 8 and 10 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 divided 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.

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 seconds 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 and equal amounts of cpu time for each instance within second data set. However, due to explained unavoidable conditions, we observed different solution times for each instance in each table within first and second data sets. For example, total cpu time measures of Phase I in first four tables are different from each other. The difference between the lowest and highest is 7.29 seconds (224.63 seconds in Table 4 and 231.92 seconds in Table 6), averaging 0.36 seconds for individual instances. This figure is negligible when considered within cpu times of the complete algorithm displayed in these tables. Similar results can also be seen in Tables 7 - 10. 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 3 - 6). 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 hours 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. [13] 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 hours 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.

Tables 3 and 5 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 three (files 16, 17 and 20) are solved within 5.5 minutes when the algorithm employs SP_1. As for SP_2, we see in Table 5 that files 15, 17, 18 and 20 could not be solved very efficiently.

In Tables 3 and 5 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 seconds) took shorter time than SP_1 (40161.11 seconds). 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 seconds and instance 17 in 3852.08 seconds. These solution times are quite small compared to the figures yielded by SP_1, 2147.98 seconds for instance 16 and 16664.12 seconds 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 seconds to 31710.98 seconds. 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 very high solution times.

Comparing Tables 3 and 4 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 more 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 4. 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 seconds to 18075.67 seconds by the modification, which is another sign of effectiveness of the modification in increasing the efficiency of the algorithm.

Similarly, from Tables 5 and 6 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 seconds to 11584 seconds 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 bacause 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 8 and 10), we see that total cpu time for SP_1 is 18075.67 seconds, where the same figure for SP_2 is 11584 seconds. However, a closer look at the tables reveals that instance 17 turns out to be pathologic for SP_1 (solved in 16180.69 seconds) 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 seconds for SP_1. The same total for SP_2 is 8591.42 seconds, 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 ORLib, 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.

Now we examine the results of experiments on instances from second data set (Tables 7 - 10). Similarly we could not solve the last ten instances of the second data set within 8 hours of time limit. In these results, we see that Phase I objective values provide better approximations to optimal values. In 8 of 20 instances, Phase I could find the optimal values and algorithm terminated with only one Phase II iteration. In the majority of remaining cases, we observe that number of Phase II iterations carried out decreased compared to the first four tables.

In the first data set instances, we observe that SP_1 yields much better solution times than SP_2. Total cpu time yielded by the algorithm using SP_1 in Table 7 is 387.22 seconds. This figure is 4761.92 seconds for SP_2. Only instance 15 could be solved in 4079.38 seconds by the algorithm using SP_2. Even if we take this instance as an outlier and compare the total cpu time for the rest of the instances, we see that SP_1 is still much faster than SP_2. This result also applies when we use modified Phase II, that is, SP_1 is again much more efficient than SP_2 in modified Phase II.

Author observation is that the modification does not bring important improvements in cpu times on the second data set since the algorithm itself worked quite efficiently on these instances. A reduction from 387.22 to 373.83 seconds in total cpu time is exhibited when the algorithm employs SP_1. Similarly, when we use SP_2, total solution time fell only from 4761.92 seconds to 4656.61 seconds.

6 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 useful discussions on the subject of the paper.

References

[1] J. Bar-Ilan, G. Kortsarz and D. Peleg (1993), "How to allocate network centers", *J. Algorithms* 15, 385–415.

[2] J.E. Beasley (1985), "A note on solving large p-median problems", *European J. Oper. Res.* 21, 270–273.

[3] M. S. Daskin (1995), *Network and Discrete Location*, Wiley, New York.

[4] M. S. Daskin (2000), "A new approach to solving the certex p-center problem to optimality: algorithm and computational results", *Communications of the Operations Research Society of Japan* 45, 428–436.

[5] Q. Deng, D. Simchi-Levi (1993), "Valid Inequalities, facets and computational experience for the capacitated concentrator location problem", Research Report, Department of Industrial Engineering and Operations Research, Columbia University, New York.

[6] S. Elloumi, M. Labbé, Y. Pochet, "A New Formulation and Resolution Method for the p-Center Problem", to appear in *INFORMS Journal on Computing*.

[7] T. Ilhan, M. Ç. Pınar (2001), "An Efficient Exact Algorithm for the Vertex p-Center Problem", http://www.optimization-online.org/DB-HTML/2001/09/376.html.

[8] M. Jaeger, J. Goldberg (1994), "A Polynomial Algorithm for the Equal-Capacity p-Center Problem on Trees", *Transportation Science* 28, 167–175.

[9] O. Kariv and S.L. Hakimi (1979), "An algorithmic approach to network location problems Part I: The p-centers", *SIAM J. Appl. Math.* 37, 513–538.

[10] S. Khuller, Y. J. Sussmann (2000), "The capacitated K-Center Problem", *SIAM J. Discrete Math.* 13, 403–418.

[11] S. Martello, P. Toth (1990), *Knapsack Problems: Algoritms and Computer Implementations*, Wiley, New York.

[12] N. Mladenović, M. Labbé and P. Hansen (2003), "Solving the p-Center Problem with Tabu Search and Variable Neighborhood Search", *Networks* 42, 48–64.

[13] S. Pallottino, M. P. Scappara and M. G. Scutellà (2002), "Large Scale Local Search Heuristics for the Capacitated Vertex p-Center Problem", Dipartimento di Informatica, Università di Pisa, TR-02-16, submitted to *Networks*.

			Phase I			Compl. Algorithm				CPLEX	
file no	size	p	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			

Table 3: Computational results on OR-Lib problems using SP_1 ($1 \leq h_i \leq 20\ \forall i \in V$, $Q_j = 120\ \forall j \in W$).

			Phase I			Compl. Algorithm			Cpu time Impv.(%)	
file no	size	p	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		

Table 4: Computational results obtained with modified Phase II on OR-Lib problems using SP_1^R ($1 \leq h_i \leq 20\ \forall i \in V$, $Q_j = 120\ \forall j \in W$).

			Phase I			Compl. Algorithm				CPLEX	
file no	size	p	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			

Table 5: Computational results on OR-Lib problems with the algorithm using SP_2 ($1 \leq h_i \leq 20\ \forall i \in V$, $Q_j = 120\ \forall j \in W$).

			Phase I			Compl. Algorithm			Cpu time Impv.(%)	
file no	size	p	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 modified Phase II on OR-Lib problems using SP_2^R ($1 \leq h_i \leq 20\ \forall i \in V$, $Q_j = 120\ \forall j \in W$).

			Phase I			Compl. Algorithm				CPLEX	
file no	size	p	obj.	iter no.	cpu t.	obj.	iter no.	cpu t.	devia(%)	cpu t.	% Impv.
1	50	5	28	7	1,88	29	9	2,31	3.45	137.66	98.32
2	50	5	31	7	2,57	33	10	17,74	6.06	554.66	96.80
3	50	5	25	7	2,01	26	9	2,36	3.85	490.93	99.52
4	50	5	31	7	2,29	32	9	10,44	3.12	497.48	97.90
5	50	5	27	7	2,14	27	8	2,34	0.00	268.81	99.13
6	50	5	27	7	2,28	29	10	3,18	6.90	294.78	98.92
7	50	5	30	7	2,08	30	8	2,54	0.00	370.76	99.31
8	50	5	29	7	2,34	30	9	7,18	3.33	612.26	98.83
9	50	5	27	7	2,61	27	8	2,74	0.00	241.63	98.87
10	50	5	29	7	2,18	29	8	2,30	0.00	318.11	99.28
11	100	10	18	7	19,42	19	9	21,49	5.26	-	-
12	100	10	19	7	20,87	20	9	29,07	5.00	-	-
13	100	10	19	7	28,17	20	9	44,65	5.00	-	-
14	100	10	20	7	21,67	20	8	23,41	0.00	-	-
15	100	10	19	7	20,40	20	9	31,60	5.00	-	-
16	100	10	18	7	20,15	19	9	40,43	5.26	-	-
17	100	10	20	7	16,86	20	8	40,97	0.00	-	-
18	100	10	19	7	20,35	20	9	39,46	5.00	-	-
19	100	10	20	7	25,59	20	8	37,49	0.00	-	-
20	100	10	18	7	18,91	18	8	25,52	0.00	-	-
						Total cpu time		387.22			

Table 7: Computational results on OR-Lib problems using SP_1 ($h_i = 1\ \forall i \in V$, $Q_j = 12\ \forall j \in W$).

			Phase I			Compl. Algorithm			Cpu time Impv.(%)	
file no	size	p	obj.	iter no.	cpu t.	obj.	iter no.	cpu t.	Over CPLEX	Over Algo.
1	50	5	28	7	1,94	29	9	2,47	98.21	-6.93
2	50	5	31	7	2,52	33	10	7,48	98.65	57.84
3	50	5	25	7	2,04	26	9	2,59	99.47	-9.75
4	50	5	31	7	2,33	32	9	9,91	98.01	5.08
5	50	5	27	7	2,18	27	8	2,59	99.04	-10.68
6	50	5	27	7	2,33	29	10	3,50	98.81	-10.06
7	50	5	30	7	2,09	30	8	2,95	99.20	-16.14
8	50	5	29	7	2,30	30	9	8,02	98.69	-11.70
9	50	5	27	7	2,54	27	8	2,79	98.85	-1.82
10	50	5	29	7	2,26	29	8	2,51	99.21	-9.13
11	100	10	18	7	18,35	19	9	21,70	-	-0.98
12	100	10	19	7	20,11	20	9	25,16	-	13.45
13	100	10	19	7	27,25	20	9	43,25	-	3.14
14	100	10	20	7	20,88	20	8	26,13	-	-11.62
15	100	10	19	7	19,76	20	9	32,10	-	-1.58
16	100	10	18	7	19,60	19	9	39,41	-	2.52
17	100	10	20	7	16,48	20	8	39,64	-	3.25
18	100	10	19	7	18,74	20	9	36,21	-	8.24
19	100	10	20	7	24,56	20	8	40,34	-	-7.60
20	100	10	18	7	17,94	18	8	25,08	-	1.72
						Total cpu time		373.83		

Table 8: Computational results obtained with modified Phase II on OR-Lib problems using SP_1^R ($h_i = 1\ \forall i \in V$, $Q_j = 12\ \forall j \in W$).

			Phase I			Compl. Algorithm				CPLEX	
file no	size	p	obj.	iter no.	cpu t.	obj.	iter no.	cpu t.	devia(%)	cpu t.	% Impv.
1	50	5	28	7	1,61	29	9	13,00	3.45	137.66	90.56
2	50	5	31	7	2,02	33	10	6,99	6.06	554.66	98.74
3	50	5	25	7	1,67	26	9	4,56	3.85	490.93	99.07
4	50	5	31	7	1,73	32	9	2,74	3.12	497.48	99.45
5	50	5	27	7	1,88	27	8	3,16	0.00	268.81	98.82
6	50	5	27	7	1,81	29	10	13,55	6.90	294.78	95.40
7	50	5	30	7	1,75	30	8	4,96	0.00	370.76	98.66
8	50	5	29	7	1,77	30	9	16,29	3.33	612.26	97.34
9	50	5	27	7	1,68	27	8	2,81	0.00	241.63	98.84
10	50	5	29	7	1,76	29	8	6,97	0.00	318.11	97.81
11	100	10	18	7	10,46	19	9	60,48	5.26	-	-
12	100	10	19	7	13,86	20	9	54,10	5.00	-	-
13	100	10	19	7	12,34	20	9	74,33	5.00	-	-
14	100	10	20	7	13,16	20	8	154,49	0.00	-	-
15	100	10	19	7	12,45	20	9	4079,38	5.00	-	-
16	100	10	18	7	14,91	19	9	34,31	5.26	-	-
17	100	10	20	7	10,70	20	8	44,50	0.00	-	-
18	100	10	19	7	10,03	20	9	117,03	5.00	-	-
19	100	10	20	7	13,60	20	8	36,26	0.00	-	-
20	100	10	18	7	10,18	18	8	32,01	0.00	-	-
						Total cpu time		4761.92			

Table 9: Computational results on OR-Lib problems using SP_2 ($h_i = 1\ \forall i \in V$, $Q_j = 12\ \forall j \in W$).

			Phase I			Compl. Algorithm			Cpu time Impv.(%)	
file no	size	p	obj.	iter no.	cpu t.	obj.	iter no.	cpu t.	Over CPLEX	Over Algo.
1	50	5	28	7	1,88	29	9	13,10	90.48	-0.77
2	50	5	31	7	2,52	33	10	5,45	99.02	22.03
3	50	5	25	7	1,96	26	9	3,89	99.21	14.69
4	50	5	31	7	2,24	32	9	3,41	99.31	-24.45
5	50	5	27	7	2,09	27	8	3,95	96.33	-25.00
6	50	5	27	7	2,21	29	10	9,86	96.66	27.23
7	50	5	30	7	2,06	30	8	6,01	98.38	-21.17
8	50	5	29	7	2,28	30	9	15,48	97.47	4.97
9	50	5	27	7	2,44	27	8	4,00	98.34	-42.35
10	50	5	29	7	2,13	29	8	8,24	97.41	-18.22
11	100	10	18	7	18,18	19	9	36,63	-	39.43
12	100	10	19	7	20,41	20	9	46,94	-	13.23
13	100	10	19	7	26,43	20	9	85,08	-	-14.46
14	100	10	20	7	20,40	20	8	174,62	-	-13.03
15	100	10	19	7	19,07	20	9	3978,31	-	2.48
16	100	10	18	7	18,84	19	9	29,00	-	15.48
17	100	10	20	7	15,71	20	8	52,82	-	-18.70
18	100	10	19	7	18,42	20	9	76,62	-	34.53
19	100	10	20	7	23,94	20	8	60,05	-	-65.61
20	100	10	18	7	17,03	18	8	43,15	-	-34.80
						Total cpu time		4656.61		

Table 10: Computational results with modified Phase II on OR-Lib problems using SP_2^R ($h_i = 1\ \forall i \in V$, $Q_j = 12\ \forall j \in W$).