Journal of the Operational Research Society (2002) 53, 1357–1366 0160-5682/02 $15.00
www.palgrave-journals.com/jors

A new algorithm for the undesirable 1-center problem on networks

M Colebrook*, J Gutiérrez, S Alonso and J Sicilia
University of La Laguna, Tenerife, Spain

Recent papers have developed efficient algorithms for the location of an undesirable (obnoxious) 1-center on general networks with n nodes and m edges. Even though the theoretical complexity of these algorithms is fine, the computational time required to get the solution can be diminished using a different model formulation and slightly improving the upper bounds. Thus, we present a new O(mn) algorithm, which is more straightforward and computationally faster than the previous ones. Computing time results comparing the former approaches with the proposed algorithm are supplied.
Journal of the Operational Research Society (2002) **53,** 1357–1366. doi:10.1057/palgrave.jors.2601468

Keywords: location; networks and graphs; undesirable (noxious/obnoxious) facility

Introduction

Network location problems deal with finding the right position where one or more facilities should be placed, in order to optimize a certain objective function that is related to the distance from the facility to the demand points (customers). Usually, the facilities to be located are *desirable*, that is, potential customers (nodes) try to attract them as closely as possible. For example, services such as police/fire stations, hospitals, schools, or even shopping centers are typical desirable facilities.

Hakimi[1] introduced the network location analysis, addressing the center problem (minimize the farthest distance) and the median problem (minimize the sum of distances). Later on, several authors have studied thoroughly these problems and they have proposed polynomial algorithms to solve them (see Minieka[2] and Kariv and Hakimi[3,4]).

However, sometimes the facilities can be considered undesirable for the surrounding population, such as nuclear reactors, military installations, polluting plants, prisons, correctional centers, and garbage dump sites. Erkut and Neuman[5] distinguish between *noxious* (harmful, lethal) and *obnoxious* (annoying, unbearable) facilities. For the sake of clearness, we call them *undesirable*.

Even though location theory begins in the 17th century, location problems involving undesirable facilities have only been discussed since the early 1970s. This is due to the fact that undesirable facilities are the consequence of technology and industrialization. In this sense, nuclear reactors, power plants, dump sites, and huge airports are all contemporary problems, whereas there have been desirable facilities, such as police stations, hospitals, schools, and warehouses, for centuries.

There are not many papers devoted to undesirable location on networks. Church and Garfinkel[6] studied the *one-facility maximum median* (*maxian*) problem, providing an O($mn \log n$) algorithm. This was improved by Tamir[7] who briefly suggested an O(mn) procedure. Minieka[8] also proposed the *anticenter* (maxmax) and the *antimedian* (maxsum) procedures.

According to Erkut and Neuman[5] and Cappanera,[9] there was no paper regarding the location of one undesirable center (*maximin*) in the location literature thus far. The first O(mn) algorithm for the 1-maximin problem was briefly suggested by Tamir[10] using Megiddo[11] and Dyer.[12] In the particular cases in which the underlying graph is a path, a star, or a tree, Burkard *et al*[13] have developed algorithms that improve those given by Tamir.[10] Lately, Melachrinoudis and Zhang[14] have proposed another O(mn) procedure based on upper bounds and on a minor modification to Dyer.[12] The most recent paper regarding this problem is written by Berman and Drezner,[15] who gave a linear programming approach in O(mn) time. The algorithm we present computationally improves these former approaches.

The main purpose of this paper is twofold. First, we tighten the upper bounds already proposed,[14] reducing even more both the number of edges to be processed and, on each edge, the number of operations to get the optimal point. Secondly, we put forward a new algorithm in O(mn) time for the undesirable 1-center on networks. This new approach relies on the intersection of the distance function lines with opposite sign slopes, and avoids the matching of superfluous lines.[14] Even though the theoretical complexity is identical to the approaches formerly reported, the computing times of

**Correspondence: M Colebrook, Departamento de Estadística, I.O. y Computación, Facultad de Matemáticas, av. Astrofisico F. Sánchez, 38271-La Laguna, Tenerife, Spain.*
E-mail: mcolesan@ull.es

the new algorithm are normally smaller. This fact becomes quite outstanding when we want to test the problem several times in a sensitivity analysis. Likewise, some harder problems, such as multicriteria network location problems, require computing the solutions for each single criterion to get the set of local non-dominated points.

The rest of the paper is structured as follows. First, we present the basic notation and the formulation of the undesirable 1-center problem, as well as the analysis of the unweighted case. The next section states new properties for the weighted undesirable 1-center problem. In the following section the latest approaches to this problem are analysed, along with the new tightened upper bounds. Hence, we demonstrate that by reformulating the maximin problem in an easier way we can greatly improve the computational complexity. Finally, several graphics and tables are presented comparing the new algorithm with the two latest approaches. In the last section, we summarize the paper and some future research issues are presented.

Notation and model formulation

Let $N = (V, E)$ be a simple (no loops or multiple edges) undirected and connected network, $V = \{v_1, v_2, \ldots, v_n\}$ being the set of nodes, and $E = \{(v_s, v_t): v_s, v_t \in V\}$ the set of edges, with $|E| = m$. On each node v_i, we set a positive weight (demand) w_i as follows:

$$w: V \to \mathbb{R}^+$$
$$v_i \in V \to w(v_i) = w_i > 0$$

The lower the node weight, the farther the undesirable facility is located from that node. Also, each edge $e = (v_s, v_t)$ is labeled with a positive length (travel cost) l_e. So, we have a length function:

$$l: E \to \mathbb{R}^+$$
$$e = (v_s, v_t) \in E \to l(e) = l_e > 0$$

Thus, a point $x \in e$ ranges in the interval $[0, l_e]$.

For each pair of nodes v_i, $v_j \in V$ we define the *distance* between two nodes $d(v_i, v_j)$ as the length of the shortest path between v_i and v_j.

Given any edge $e = (v_s, v_t) \in E$, $v_i \in V$ and an inner point $x \in e$, we define the distance between x and a node v_i as $d(x, v_i) = \min\{x + d(v_s, v_i), l_e - x + d(v_t, v_i)\}$.

The point where $d(x, v_i)$ attains its equilibrium (ie $x + d(v_s, v_i) = l_e - x + d(v_t, v_i)$) is called a *bottleneck point*:

$$b_i = \frac{d(v_t, v_i) + l_e - d(v_s, v_i)}{2} \tag{1}$$

When b_i is located inside e, then $d(x, v_i)$ resembles Figure 1c. Otherwise, the bottleneck point is located over one of the two ending nodes.

Now we are ready to formulate the undesirable 1-center (maximin) problem on networks. Given any point $x \in N$ we

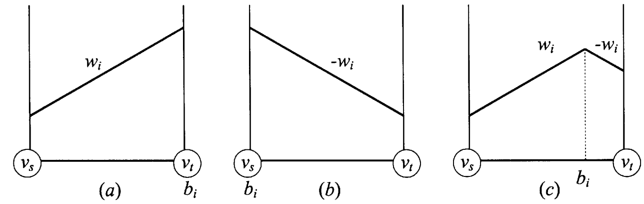

Figure 1 The three possible plots of $d(x, v_i)$.

define $f(x) = \min_{v_i \in V} w_i d(x, v_i)$. Then, the problem consists of calculating

$$\max_{x \in N} \min_{v_i \in V} w_i d(x, v_i) = \max_{x \in N} f(x) \tag{2}$$

and a point $x_N \in N$ is an undesirable 1-center point iff $f(x_N) = \max_{x \in N} f(x)$. This problem is the opposite to the 1-center problem (minimax), so it could be called the *anti-center*. Unfortunately, this term has already been coined by Minieka[8] to define the *maxmax* problem. We instead propose the term 1-*uncenter* (undesirable center) to define the optimal location point.

If there is at least one vertex v_i such that $w_i = 0$, then $f(x) = 0, \forall x \in N$ and obviously any point on network N would be a 1-uncenter. Therefore, we consider only $w_i > 0, \forall v_i \in V$.

Several interesting properties arise for this problem, all stated and proved in Melachrinoudis and Zhang[14] and in Berman and Drezner.[15]

Property 1 For any edge $e = (v_s, v_t) \in E, x \in e$, the objective function $f(x)$, is continuous, piecewise linear and concave in the interval $[0, l_e]$, consisting of at most $2n$ strictly monotonic line segments. The value of the objective function is zero at the ends of the edge (see Figure 2).

Let x_e be the point in edge $e = (v_s, v_t) \in E$ such that $f(x_e) = \max_{x \in e} f(x)$. This point x_e is called a *local 1-uncenter* on edge e.

Property 2 A unique local 1-uncenter x_e location exists on each edge e. Consequently, there are at most m 1-uncenter locations on a network.

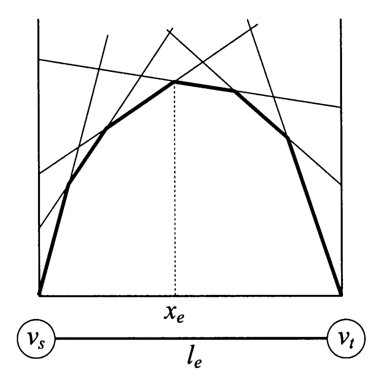

Figure 2 Objective function $f(x)$, which is actually the lower envelope of all distance functions.

We now begin discussing in brief the unweighted case for its simplicity, and later we will analyse the weight 1-uncenter problem.

When all the node weights are equal, $\forall v_i \in V, w_i = w$, the local 1-uncenter x_e is sited at the central point of edge e. Therefore, the unweighted 1-uncenter x_N is located in the middle of the longest edge(s).[14,15] This is done in O(m) time.

New properties for the weighted 1-center problem

The previous properties allow us to reformulate the 1-uncenter problem over each edge $e = (v_s, v_t) \in E$ as follows: $x_N \in N$ is a 1-uncenter point iff $f(x_N) = \max_{e \in E} f(x_e)$.

Since the local 1-uncenter point is the maximum value of the concave objective function $f(x)$, it should be located at the intersection of two distance functions lines with opposite sign slopes. Our goal is to find these two lines and the intersection point between them.

The bottleneck point (1) can give us an idea about whether the distance function line is increasing or decreasing. Thus, given $e = (v_s, v_t) \in E$ and for all $v_i \in V$ we can get these relationships:

$$\begin{aligned} b_i > 0 \Leftrightarrow & \text{ distance function line of vertex} \\ & v_i \text{ is increasing to the left of } b_i. \\ b_i < l_e \Leftrightarrow & \text{ distance function line of vertex} \\ & v_i \text{ is decreasing to the right of } b_i. \end{aligned} \tag{3}$$

Replace b_i in expression (3), and let $d_i = d(v_s, v_i) - d(v_t, v_i)$. Then:

$$\begin{aligned} d_i < l_e &\Leftrightarrow \text{increasing distance function line.} \\ -d_i < l_e &\Leftrightarrow \text{decreasing distance function line.} \end{aligned} \tag{4}$$

We divide the set of nodes V into two sets, depending on whether the distance function increases or decreases from v_s:

$L = \{v_k \in V: d_k < l_e\}$: nodes whose $d(x, v_k)$ is increasing from the left-end node v_s (Figure 1a,c).
$R = \{v_k \in V: -d_k < l_e\}$: nodes whose $d(x, v_k)$ is increasing from the right-end node v_t (Figure 1b,c).

A node v_k may belong to both sets, and hence, $|L| + |R| \leqslant 2n$. For any node $v_i \in V$, we now define the functions $F_i^L(x)$ and $F_i^R(x)$ as:

$$F_i^L(x) = w_i(x + d(v_s, v_i))$$
$$F_i^R(x) = w_i(l_e - x + d(v_t, v_i))$$

For any pair of nodes $v_i \in L$, $v_j \in R$ we also define

$$X(v_i, v_j) = \frac{w_j(l_e + d(v_t, v_j)) - w_i d(v_s, v_i)}{w_i + w_j}$$

which computes the intersection point between two distance function lines with opposite sign slopes, that is, the point x where both $F_i^L(x)$ and $F_j^R(x)$ are equal. For the special case where $v_i = v_j$, we get the bottleneck point b_i.

Note that our goal is to find the two distance function lines (with opposite sign slopes) that cross at the maximum value of the objective function. Since there are at most n distance function lines in sets L and R, there are at most n^2 possible intersection points. Let P_e be the set containing such intersection points for a given edge $e \in E$:

$$P_e = \{X(v_i, v_j): \forall v_i \in L, \forall v_j \in R\}, \qquad |P_e| \leqslant n^2$$

and let P_N be the set obtained joining, for each edge, all the points belonging to P_e, that is

$$P_N = \bigcup_{e \in E} P_e, \qquad |P_N| \leqslant mn^2$$

Hooker *et al*[16] defined the *arc bottleneck point set* $B_A = \{b_i: v_i \in V\}$, and the *center bottleneck point set* B_C. This set B_C contains points $x \in e$ such that, for any two distinct nodes $v_i, v_j \in V$, $w_i d(x, v_i) = w_j d(x, v_j)$, and besides, $d(x, v_i)$ and $d(x, v_j)$ do not both decrease when x is perturbed slightly in either direction. Obviously, $B_A \subset P_e$ and $B_C \subset P_e$.

Let $v_i \in L$ and $v_j \in R$. If $v_i = v_j$, then $X(v_i, v_i) = b_i \in B_A$. On the other hand, if $v_i \neq v_j$ then $X(v_i, v_j) \in B_C$. Hence, $P_e = B_A \cup B_C$.

Melachrinoudis and Zhang[14] stated that the finite dominating set (FDS) for the 1-maximin problem on networks with positive weights is $V \cup B_A \cup B_C$ (this result is also described more generally in Hooker *et al*[16]). Nevertheless, this is rather mistaken, and needs to be fixed. The following result determines the correct FDS.

Lemma 1 *The finite dominating set for the weighted 1-uncenter problem on networks is* $\mathrm{P_N}$.

Proof. According to Property 1, the value of the objective function is zero at the ends of the edges, so the maximum can never be at those points. On the other hand, this maximum value is unique on each edge (Property 2), and must be attained at the crossing point of two distance function lines with opposite sign slopes. These points are in P_e. Therefore, the FDS for the weighted network 1-uncenter problem is P_N. □

Taking into account these last results, we can get a new formulation for the 1-uncenter problem (2) as follows.

Given $e = (v_s, v_t) \in E$, let $F(x) = \{F_i^L(x): \forall v_i \in L\}$ (or $F(x) = \{F_i^R(x): \forall v_i \in R\}$) be the set of left (right) weighted distance functions on edge e. We define the point z_e on edge e such that $F(z_e) = \min_{x \in P_e} F(x)$.

Lemma 2 *The local 1-uncenter point* $\mathrm{x_e}$ *in edge* e *is* $\mathrm{z_e}$.

Proof. Properties 1 and 2 state that $f(x)$ is a concave function and has a unique maximum x_e. This point is obtained intersecting one increasing line $F_i^L(x)$ with a decreasing line $F_j^R(x)$. Therefore x_e must belong to set P_e.

Now we show that $x_e = z_e$. By the definition of z_e, we always have $F_i^L(x_e) \geqslant F_i^L(z_e)$. If $x_e \neq z_e$, and since all weights w_i must be positive, the line segments of function

$f(x)$ have non-zero slope, and thus $F_i^L(x_e) \neq F_i^L(z_e)$. Hence, we have $F_i^L(x_e) > F_i^L(z_e)$, which means that x_e would not be a local 1-uncenter point, and the result follows. □

Recall from (2) that our goal is to find a point on the network that maximizes the minimum distance from that point to the closest one. Then, denoting F_e as the value $F(x_e) = F(z_e)$, the original problem is equivalent to the next one.

Theorem 1 *The 1-uncenter problem on networks can be expressed as*

$$\max_{e \in E} \min_{x \in P_e} F(x)$$

and a point $x_N \in N$ *is an 1-uncenter point iff*

$$F(x_N) = \max_{e \in E} F_e.$$

Proof. According to Lemma 2, on each edge e the value of $\max_{e \in E} f(x_e)$ is F_e. Hence, the optimum value x_N on network N is the maximum of all F_e. That is, $\max_{e \in E} \min_{x \in P_e} F(x)$. □

Taking into consideration the previous result, the initial continuous 1-uncenter problem (2) on networks becomes a discrete problem. Finally we remark that, despite the size of set P_N being at most mn^2, the 1-uncenter point can be found on a network in O(mn) time. This result is proved in a subsequent section, where the new algorithm is presented. Previous to this, we briefly comment on the latest approaches and bounds cited in the literature, along with the new bounds that we propose.

Latest approaches and new bounds

As we mentioned in the introduction, few papers have been devoted to the 1-uncenter problem on networks thus far. One of the latest algorithms in O(mn) time has been presented by Melachrinoudis and Zhang.[14]

Their approach relies on three upper bounds that significantly reduce the number of edges and, over each edge, the number of distance function lines. Given an edge $e = (v_s, v_t) \in E$, the first upper bound is defined as $x_{UB1} = X(v_s, v_t)$ and $F_{UB1} = F_s^L(x_{UB1}) = F_t^R(x_{UB1})$ (Figure 3).

This bound cannot be improved. Nevertheless, the next two bounds can be tightened. Let

$$\begin{aligned} v_g \in V &: F_g^L(0) = \min_{\substack{v_k \in V \\ v_k \neq v_s}} F_k^L(0), \\ v_h \in V &: F_h^R(l_e) = \min_{\substack{v_k \in V \\ v_k \neq v_t}} F_k^R(l_e) \end{aligned} \tag{5}$$

be the nodes at which the distance functions attain their minimum value on each side. Ties are broken taking the node with the smallest weight w. The second upper bound is $x_{gh} = X(v_g, v_h)$ and $F_{gh} = F_g^L(x_{gh}) = F_h^R(x_{gh})$.

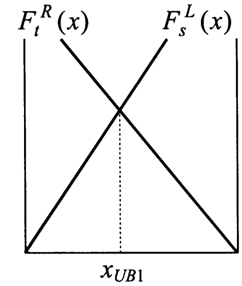

Figure 3 F_{UB1}, the first upper bound.

However, upper bound F_{gh} may be slightly improved in two special cases (Figure 4). So, we introduce a new point z and its ordinate, which are defined by:

$$(z, F_z) = \begin{cases} (X(v_s, v_h), F_s^L(X(v_s, v_h))) & \text{if } F_s^L(x_{gh}) \leqslant F_{gh} \text{ (Figure 4a)} \\ (X(v_g, v_t), F_t^R(X(v_g, v_t))) & \text{if } F_t^R(x_{gh}) \leqslant F_{gh} \text{ (Figure 4b)} \\ (0, \infty) & \text{otherwise.} \end{cases} \tag{6}$$

Then, we propose the new bound $F_{UB2} = \min\{F_{gh}, F_z, F_{UB1}\}$, and hence, x_{UB2} is equal to x_{gh}, z, or x_{UB1}.

Any distance function line over F_{UB2} is redundant and, therefore, can be completely removed. Despite the fact that the upper bound F_{gh} has been tightened to F_{UB2}, the proof in Melachrinoudis and Zhang[14] is valid for this result as well.

Likewise, the third upper bound is defined considering

$$\begin{aligned} v_p \in V &: F_p^L(l_e) = \min_{\substack{v_k \in V \\ v_k \neq v_s}} F_k^L(l_e), \\ v_q \in V &: F_q^R(0) = \min_{\substack{v_k \in V \\ v_k \neq v_t}} F_k^R(0) \end{aligned} \tag{7}$$

with $x_{pq} = X(v_p, v_q)$ and $F_{pq} = F_p^L(x_{pq}) = F_q^R(x_{pq})$.

This bound F_{pq} can also be improved by establishing a new point y and its ordinate, which are defined by:

$$(y, F_y) = \begin{cases} (X(v_s, v_q), F_s^L(X(v_s, v_q))) & \text{if } F_s^L(x_{pq}) \leqslant F_{pq} \\ (X(v_p, v_t), F_t^R(X(v_p, v_t))) & \text{if } F_t^R(x_{pq}) \leqslant F_{pq} \\ (0, \infty) & \text{otherwise.} \end{cases} \tag{8}$$

Then, we propose the new bound $F_{UB3} = \min\{F_{pq}, F_y, F_{UB1}\}$ and x_{UB3} is updated accordingly to x_{pq}, y, or x_{UB1}.

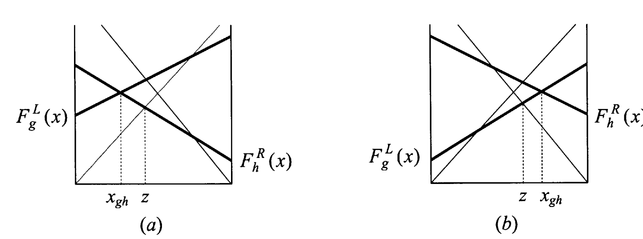

Figure 4 Tighter bounds. The value of F_z is better than F_{gh}.

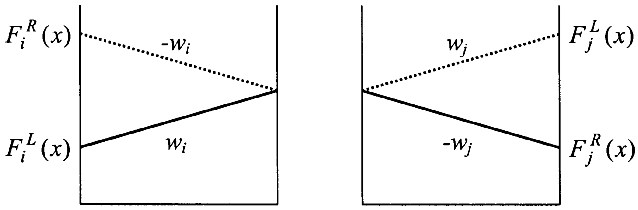

Figure 5 Superfluous lines plotted as dotted lines.

Before presenting the new algorithm which makes use of bounds (6) and (8), we outline the rest of Melachrinoudis and Zhang's algorithm.[14]

Once all the lines above $\min\{F_{UB1}, F_{gh}\}$ are deleted, the remaining lines are compared pairwise. For each pair of lines, either the intersection point is calculated or one of them is deleted (dominated). Then, the median value of the intersection points is projected on the maximin function (lowest lines). If the right and left gradients have opposite sign slopes, the maximin point is found. Otherwise, the gradients are used to delete a quarter of the paired lines. In the worst case, the procedure keeps on until two lines remain only.

The main disadvantage of this pairing algorithm is the matching of superfluous distance function lines, that is, lines that do not actually exist (Figure 5). These lines load the algorithm with useless computational effort and, therefore, they need to be excluded.

On the other hand, the most recent contribution to the 1-uncenter problem is due to Berman and Drezner,[15] who presented a brief paper on the location of an obnoxious facility on a network. They addressed this problem from a linear programming viewpoint, making use of the algorithm given in Megiddo[11] to get an O(mn) time procedure. However, this approach is not very fast (computationally speaking) since every single edge has to be checked to find the optimal value. This fact is proved later in the computational experience section.

All the improvements discussed above, together with the new upper bounds, are shown in the next algorithm that we propose to solve the 1-uncenter problem.

The algorithm

The algorithm has two main parts: the first computes the three upper bounds; the second seeks for the best point in the set of remaining distance function lines. For the sake of comprehensibility, we first show the outlined algorithm and then we explain each block of code.

function *UnCenter*(Network N, Distance Matrix d)
{ // *Current best value on network N.*
$F_N \leftarrow 0$
// *Solution set.*
$S \leftarrow \emptyset$
for all edges $e = (v_s, v_t) \in E$ **do**
{ // *Compute the upper bounds.*
$x_{UB1} \leftarrow X(v_s, v_t)$
$F_{UB1} \leftarrow F_s^L(x_{UB1})$
if $F_N > F_{UB1}$ **then** continue to next edge
Compute *UB2* using (5) and (6)
if $F_N > F_{UB2}$ **then** continue to next edge
Compute *UB3* using (7) and (8)
if $F_N > F_{UB3}$ **then** continue to next edge
// *Set* (x_e, F_e) *to the best value found.*
if $F_{UB2} \leqslant F_{UB3}$ **then** $(x_e, F_e) \leftarrow (x_{UB2}, F_{UB2})$
else $(x_e, F_e) \leftarrow (x_{UB3}, F_{UB3})$
Create sets L and R using (4). All lines must be below F_{UB2}.
// *Continue till the new value* F_e *cannot improve the current* F_N,
// *or until one of the node sets becomes empty.*
while $F_e \geqslant F_N$ **and** $(L \neq \emptyset$ **or** $R \neq \emptyset)$ **do**
{ Pair all nodes in L against R, using a $\max\{|L|, |R|\}$ matching
Project the value x_e on the lower envelope using (9) to get v_a and v_b
$x_e \leftarrow X(v_a, v_b)$
$F_e \leftarrow F_a^L(x_e)$
Remove from L and R all lines above the new value F_e
}
if $F_e \geqslant F_N$ **then**
{ $F_N \leftarrow F_e$
Store the pair (x_e, e) in S
}
}
return (F_N, S)
}

The function *UnCenter* needs only two inputs: the network $N = (V, E)$ and the distance matrix d, which can be computed in $O(mn + n^2 \log n)$ time using Fredman and Tarjan.[17] The output is F_N and the set of points S where this value is attained.

The calculation of the first upper bound is easy. The second one is computed using expressions (5) and (6), whereas expressions (7) and (8) calculate the third upper bound.

Then, the pair (x_e, F_e) is set to the best upper bound. The purpose of the rest of the algorithm is to sharpen F_e until the optimal value is found.

Next, we divide set V into two sets L and R. The distance function lines belonging to these sets are then matched, so that the number of matchings must be equal to $\max\{|L|, |R|\}$. For example, let $L = \{v_1, v_3, v_4\}$ and $R = \{v_2, v_3, v_5, v_7, v_8\}$. Then, the specific matchings $(v_i \in L, v_j \in R)$ are (v_1, v_2), (v_3, v_3), (v_4, v_5), (v_1, v_7), and (v_3, v_8). In each pairing, the intersection point between the two lines and its related ordinate value are computed. Besides, any dominated line is immediately removed.

The value of x_e is projected on the objective function (lower envelope), and thus, we obtain a new value for (x_e, F_e). All lines above F_e are then deleted from L and R. The algorithm keeps going until either $F_e < F_N$; that is, this edge cannot improve the network optimum, or both L and R are empty.

The maximum matching assures a maximum of n paired lines, which is essential to delete as many lines as possible. The following lemma states this result.

Lemma 3 *In each iteration of the 'while' loop, at least* $(\max\{|L|, |R|\})/2$ *nodes from* L *and* R *are removed.*

Proof. For each of the paired lines (v_i, v_j), $v_i \in L$, $v_j \in R$, let $Q_e = \{X(v_i, v_j)\}$ such that $|Q_e| = \max\{|L|, |R|\}$, that is, Q_e contains all the intersection points of the line pairing. Let $F_e = \min_{\substack{x \in Q_e \\ v_i \in L}} F_i^L(x)$ and x_e be, respectively, the minimum value of all the paired lines and the point where this minimal value is attained.

The value F_e might be optimal. Obviously, all lines belonging to L and R are then deleted. Otherwise, let

$$v_a \in L\colon F_a^L(x_e) = \min_{v_k \in L} F_k^L(x_e), \qquad v_b \in R\colon F_b^R(x_e) = \min_{v_k \in R} F_k^R(x_e) \tag{9}$$

be the lowest lines (lower envelope) from L and R (ties are broken taking the lower weight w). Let $x_e = X(v_a, v_b)$ and $F_e = F_a^L(x_e)$. This F_e is a new upper bound. Besides, since $F_a^L(x_e)$ or $F_b^R(x_e)$ belong to the lower envelope, any line above F_e can be removed. Indeed, each pair of lines (v_i, v_j) has only one line under F_e; to be precise, either $F_i^L(x_e) < F_e$ or $F_j^R(x_e) < F_e$. Both lines v_i and v_j cannot be below F_e since that contradicts the fact that F_e is the minimal value. Then, in the worst case, one single node belonging to each pair (v_i, v_j) can be removed from L or R. Therefore, each removal process deletes at least $(|Q_e|)/2$ nodes (lines). □

Given the distance matrix, the following theorem proves that the overall complexity of the new 1-uncenter algorithm is O(mn).

Theorem 2 *The previous algorithm solves efficiently the weighted 1-uncenter problem in O*(mn) *time.*

Proof. The computation of the second and third upper bounds takes O(n) time. The size of L and R is, in the worst case, $n \geqslant \max\{|L|, |R|\}$ nodes. According to Lemma 3, each iteration of the 'while' loop deletes $(n/2)$ nodes. Therefore, the complexity of that loop is:

$$n + \frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{2^k} = n\left(\frac{2^k + 2^{k-1} + \cdots + 1}{2^k}\right) = \frac{n}{2^k}\sum_{i=0}^{k} 2^i = \frac{n}{2^k}(2^{k+1} - 1)$$

In the worst case, this loop keeps on until one single line remains in both L and R. Then $(n/2^k) = 2 \Rightarrow n = 2^{k+1}$, and consequently, $(n/2^k)(2^{k+1} - 1) = 2(n-1) < 2n \in O(n)$.

This process must be applied to all m edges. Thus, the overall complexity is O(mn). □

The time complexity given in Melachrinoudis and Zhang[14] was bounded by $4n$, and hence, this may explain why the new algorithm is much faster. Moreover, as you may have noticed, the 1-uncenter algorithm does not make use of the median algorithm. Next, we illustrate the proposed algorithm with a brief example.

An example

The network is depicted in Figure 6. It has $n = 8$ nodes and $m = 18$ edges. The weights (in bold) on the nodes randomly range from 1 to 9, whereas the lengths (in italics) randomly vary from 1 to 49. The trace of the algorithm is summarized in Table 1.

In the first iteration, the three upper bounds are computed. The best of them is UB3. Since R is empty, there is no line pairing. Thus, the first local 1-uncenter on edge (v_1, v_3) is located at $x_e = 12.6$, with $F_e = 21.6$. The solution set S and the value F_N are updated.

The best upper bound on edge (v_1, v_4) is (17,26). Again, there is no line pairing and, since $F_e = 26 > F_N$, the set S is updated. The next four edges cannot improve F_N.

Edge (v_2, v_3) updates the best network 1-uncenter value to $F_N = 31.5$. The next edge (v_2, v_4) leaves F_N and S unchanged, while in the iteration of edge (v_2, v_5) the algorithm steps to the following edge as soon as it checks that UB2 is worst than F_N.

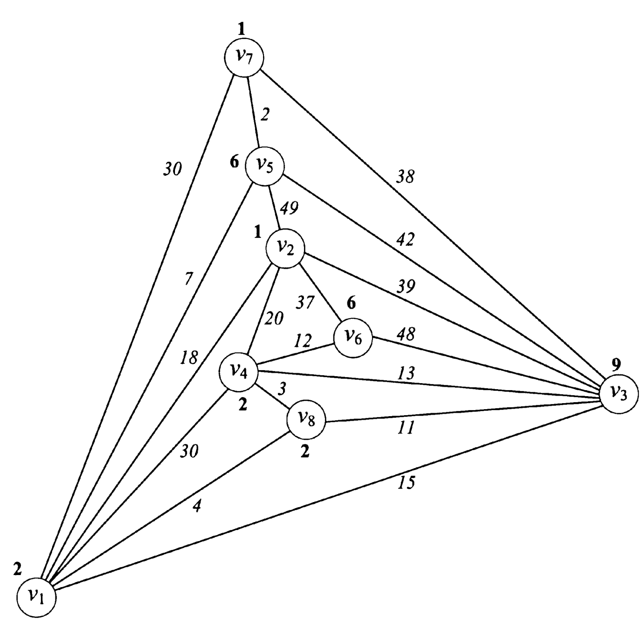

Figure 6 Planar network with $n = 8$ and $m = 18$.

Table 1 Trace of the 1-uncenter algorithm for the network of Figure 6

Edge	F_N	(x_{UB1}, F_{UB1})	(x_{UB2}, F_{UB2})	(x_{UB3}, F_{UB3})	(x_e, F_e)	L	R	S
$e_{13} = (v_1, v_3)$	0	(12.27, 24.54)	(12.27, 24.54)	(12.6, 21.6)	(12.6, 21.6)	$\{v_7\}$	∅	$\{(12.6, e_{13})\}$
$e_{14} = (v_1, v_4)$	21.6	(15, 30)	(15, 30)	(17, 26)	(17, 26)	$\{v_7\}$	∅	$\{(17, e_{14})\}$
$e_{15} = (v_1, v_5)$	26	(5.25, 10.5)	–	–	–	–	–	$\{(17, e_{14})\}$
$e_{17} = (v_1, v_7)$	26	(10, 20)	–	–	–	–	–	$\{(17, e_{14})\}$
$e_{18} = (v_1, v_8)$	26	(2, 4)	–	–	–	–	–	$\{(17, e_{14})\}$
$e_{21} = (v_2, v_1)$	26	(12, 12)	–	–	–	–	–	$\{(17, e_{14})\}$
$e_{23} = (v_2, v_3)$	26	(35.1, 35.1)	(33.33, 33.33)	(31.5, 31.5)	(31.5, 31.5)	∅	$\{v_7, v_8\}$	$\{(31.5, e_{23})\}$
$e_{24} = (v_2, v_4)$	31.5	(13.33, 13.33)	–	–	–	–	–	$\{(31.5, e_{23})\}$
$e_{25} = (v_2, v_5)$	31.5	(42, 42)	(25.5, 25.5)	–	–	–	–	$\{(31.5, e_{23})\}$
$e_{26} = (v_2, v_6)$	31.5	(31.71, 31.71)	(31.71, 31.71)	(31.71, 31.71)	(31.71, 31.71)	–	–	$\{(31.71, e_{26})\}$
$e_{34} = (v_3, v_4)$	31.71	(2.36, 21.27)	–	–	–	–	–	$\{(31.71, e_{26})\}$
$e_{35} = (v_3, v_5)$	31.71	(16.8, 151.2)	(7.33, 36.66)	(10, 34)	(10, 34)	$\{v_7, v_8\}$	$\{v_7\}$	$\{(10, e_{35})\}$
$e_{36} = (v_3, v_6)$	34	(19.2, 172.8)	(24.5, 71)	(26, 50)	(26, 50)	$\{v_2, v_7, v_8\}$	$\{v_2, v_4, v_7\}$	$\{(26, e_{36})\}$
$e_{37} = (v_3, v_7)$	50	(3.8, 34.2)	–	–	–	–	–	$\{(26, e_{36})\}$
$e_{38} = (v_3, v_8)$	50	(2, 18)	–	–	–	–	–	$\{(26, e_{36})\}$
$e_{46} = (v_4, v_6)$	50	(9, 18)	–	–	–	–	–	$\{(26, e_{36})\}$
$e_{48} = (v_4, v_8)$	50	(1.5, 3)	–	–	–	–	–	$\{(26, e_{36})\}$
$e_{57} = (v_5, v_7)$	50	(0.28, 1.71)	–	–	–	–	–	$\{(26, e_{36})\}$

The algorithm keeps on in the same way with edges (v_2, v_6) and (v_3, v_4), updating the network 1-uncenter value $F_N = 31.71$ and $S = \{(31.71, e_{26})\}$. The first lines paired arise in edge (v_3, v_5). The pairing is (v_7, v_7) and (v_8, v_7), which provides a new $(x_e, F_e) = (10, 34)$, and hence, a new F_N and S.

In the next edge (v_3, v_6) the line matching cannot improve $(x_e, F_e) = (26, 50)$. Given that no remaining edge provides a better value, $F_N = 50$ becomes the 1-uncenter value at $S = \{(26, e_{36})\}$.

Note that the algorithm processes only 6 out of 18 potential edges, with only 5 pairings. For the same example, the maximin algorithm[14] needs to process 7 edges, and computes 26 pairings. Even though these numbers may not seem important, they will be quite relevant when the network size gets bigger (both in nodes and edges), as shown in the next section.

Computational results

The computational results were developed using GNU g^{++} 2.95.2 programming language and LEDA (*Library of Efficient Datatypes and Algorithms*[18]), on a PC AMD K6-III 400 Mhz under Redhat Linux 6.1 (Cartman). The sources were built using the g^{++} compiler optimizing option '$-$O'.

The distance matrix was computed using an algorithm developed in LEDA, which is claimed to run in $O(mn + n^2 \log n)$ time.

For the sake of a homogeneous comparison with the algorithm reported by Melachrinoudis and Zhang,[14] we keep the same node weight range from 1 to 9, edge length ranges from 1 to 49, and the edge density $d = m/(n(n-1)/2)$ equal to 1/2, 1/4, 1/8, and 1/16. However, the sizes of the networks were too small for such a fast computer, since they provided computational times near to zero seconds. Thus, we decided to run the experiments from $n = 100$ up. The networks were created using the random graph generators provided by LEDA.

Previous to the comparison with the algorithm by Melachrinoudis and Zhang,[14] we present the results obtained for the comparison between the new algorithm and the linear programming approach proposed by Berman and Drezner.[15] For this task, we made use of the free linear solver *lp_solve*.[19] Since their method relies on an LP process over each and every edge, we decided to test the algorithms on low density networks. Thus, we created planar networks with $m = 3n - 6$ and $n = 100$ to 500, in steps of 25 nodes. Ten instances were generated for each value of n. Table 2 illustrates the average processed edges and the average computing time for the three experiments accomplished. The label 'B & D' stands for Berman and Drezner.

The first column in Table 2 shows the results for the original approach by Berman and Drezner.[15] These times are extremely high, since their method has to run over all existing edges. The next column shows the results for the

Table 2 Processed edges and computing times of Berman & Drezner's procedure and the new algorithm for planar networks ($m = 3n - 6$) with $n = 100$ to 500 nodes

	B & D		*B & D (with UBs)*		*New algorithm*		
n	*Proc. edges*	*Time (s)*	*Proc. edges*	*Time (s)*	*Proc. edges*	*Time (s)*	*Reduction (%)*
100	294	1.611	6	0.046	6	0.010	78
125	369	2.859	5	0.055	5	0.014	75
150	444	4.593	7	0.095	7	0.020	79
175	519	6.902	6	0.112	6	0.023	79
200	594	11.608	7	0.183	7	0.033	82
225	669	16.453	6	0.203	6	0.047	77
250	744	26.371	9	0.321	9	0.054	83
275	819	28.585	8	0.375	8	0.068	82
300	894	37.029	7	0.380	7	0.076	80
325	969	47.419	8	0.496	8	0.085	83
350	1044	57.553	8	0.570	8	0.101	82
375	1119	70.416	8	0.625	8	0.114	82
400	1194	82.602	7	0.678	7	0.134	80
425	1269	103.021	8	0.867	8	0.133	85
450	1344	110.540	8	0.758	8	0.130	83
475	1419	144.851	8	0.892	8	0.155	83
500	1494	169.766	7	0.864	7	0.159	82

same approach including the new upper bounds proposed in this paper. These bounds remarkably reduce the number of processed edges, and hence, the overall computing times. Finally, the third column presents the computing results of the new algorithm, which achieves faster computing times than the bounded version of Berman and Drezner. The time reduction percent between these two latter procedures is shown in the last column.

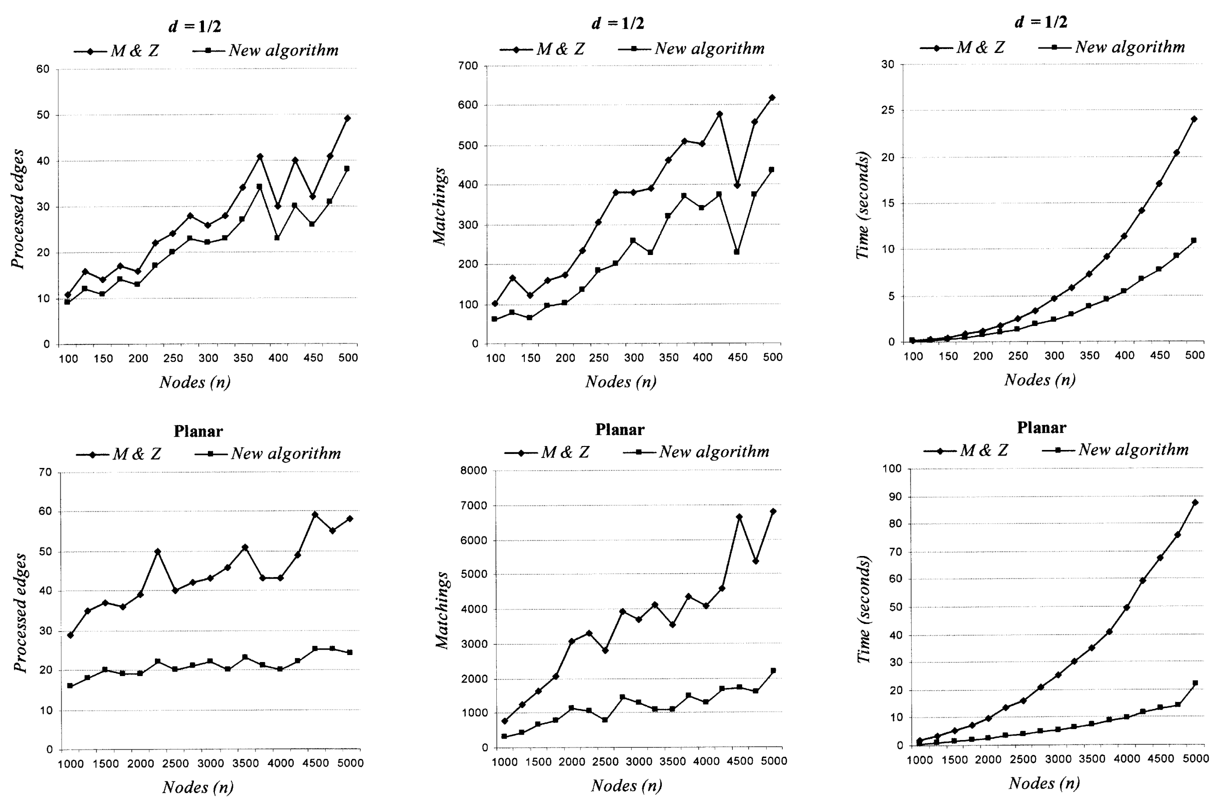

Figure 7 Processed edges, line pairings (matchings) and computing times for $d = 1/2$ and $n = 100$ to 500, and for planar networks ($m = 3n - 6$) with $n = 1000$ to 5000 nodes.

Table 3 Summary of the processed edges, line pairings (matchings), and computing times for $d = 1/2$, $1/4$, $1/8$, and $1/16$, and for planar networks ($m = 3n - 6$)

		Processed edges			*Matchings*			*Time*		
d	n	*M & Z*	*New algorithm*	*Reduction* (%)	*M & Z*	*New algorithm*	*Reduction* (%)	*M & Z*	*New algorithm*	*Reduction* (%)
1/2	100	11	9	18	103	61	41	0.144	0.092	36
	200	16	13	19	175	102	42	1.209	0.708	41
	300	26	22	15	382	259	32	4.595	2.399	48
	400	30	23	23	503	341	32	11.364	5.451	52
	500	49	38	22	620	434	30	24.042	10.813	55
1/4	100	7	7	0	60	33	45	0.070	0.046	34
	200	15	12	20	249	152	39	0.585	0.350	40
	300	16	14	13	203	135	33	2.154	1.148	47
	400	21	17	19	422	261	38	5.584	2.753	51
	500	25	21	16	579	397	31	11.955	5.389	55
1/8	125	13	9	31	172	75	56	0.061	0.044	28
	250	12	10	17	243	148	39	0.562	0.331	41
	375	17	14	18	492	296	40	2.215	1.130	49
	500	18	15	17	825	496	40	5.576	2.662	52
	625	18	15	17	775	433	44	12.208	5.276	57
	750	23	19	17	752	547	27	22.744	9.010	60
	875	26	22	15	847	578	32	39.812	14.669	63
	1000	29	22	24	1085	714	34	64.705	21.680	66
1/16	125	8	7	13	73	33	55	0.024	0.016	33
	250	10	8	20	160	93	42	0.238	0.138	42
	375	12	10	17	373	246	34	0.976	0.522	47
	500	13	11	15	424	280	34	2.560	1.270	50
	625	13	10	23	461	294	36	5.736	2.505	56
	750	14	11	21	530	355	33	10.644	4.359	59
	875	17	14	18	904	620	31	19.295	7.144	63
	1000	20	16	20	954	629	34	30.149	10.623	65
Planar	1000	29	16	45	774	304	61	1.846	0.670	64
	1500	37	20	46	1639	658	60	5.209	1.513	71
	2000	39	19	51	3071	1112	64	9.770	2.542	74
	2500	40	20	50	2813	7664	73	16.255	3.752	78
	3000	43	22	49	3675	1289	65	25.009	5.525	79
	3500	51	23	55	3516	1072	70	35.133	7.503	77
	4000	43	20	53	4079	1296	68	49.626	9.944	80
	4500	59	25	58	6655	1694	75	67.624	12.884	81
	5000	58	24	59	6809	2191	68	87.341	21.936	75

Regarding the comparison with Melachrinoudis and Zhang's procedure,[14] three kinds of experiments were performed. In the first one, n varies from 100 to 500 nodes in steps of 25, with d equal to 1/2, 1/4, 1/8, and 1/16. In the second, the number of nodes ranges from 525 to 1000 in steps of 25 nodes, with d equal to 1/8 and 1/16. In the last experiment, random planar ($m = 3n - 6$) networks were generated for $n = 1000$ to 5000, with a step of 250 nodes. In all cases, ten instances of each combination were run. The comparison is based on the average value of the processed edges, line pairings, and computing time. The label 'M & Z' stands for Melachrinoudis and Zhang.

Figure 7 shows the processed edges, line pairings, and computing times for $d = 1/2$. Due to the tighter bounds, there are fewer edges processed by the 1-uncenter algorithm than by the maximin procedure. Besides, the number of paired lines is much less in our algorithm. Likewise, the 1-uncenter algorithm beats the maximin in all the computing time graphics. Finally, in Figure 7 we also describe the results for random planar networks. It seems that the 1-uncenter algorithm behaves even better than the maximin procedure when the number of edges m is $O(n)$. In this particular case, the gap between the two algorithms is quite large.

In Table 3 we show an overall summary of numerical results obtained for the different set of densities as well as for planar networks. In all cases, the number of edges processed by our algorithm, and the number of matchings (line crossings) is fewer than Melachrinoudis and Zhang,[14] gaining in some instances a reduction of over 50%. As a consequence of all this, the computing times of the new algorithm are better, achieving in some cases a reduction of 80%. Besides, the reduction augments as the number of nodes n increases.

Concluding remarks and further research

The location of an undesirable facility under the max-min criterion is addressed. As was stated in the introduction, there are only a few references to this problem in the literature. One of the latest proposes a $O(mn)$ time algorithm[14] based on three upper bounds and on a modified procedure.[12] However, we show that their upper bounds can be tightened, and that pairing superfluous lines is not needed. The other paper[15] approaches the problem in a linear programming way. Although it has the same theoretical complexity, its running times are extremely high, since the algorithm has to process every single edge.

Hence, using tighter bounds and eliminating the superfluous line pairing by means of a more convenient problem formulation, we propose a new $O(mn)$ time algorithm. Besides, the algorithm needs no median procedure. As a result of all this, the proposed algorithm is more straightforward and its running times are faster than the ones already reported.[14]

Further research is mainly focused in getting an improved version of the anti-cent-dian problem searching for new tighter bounds to reduce computing times. Another source of research could arise in the development of the multicriteria 1-uncenter problem, considering several weights on the nodes and several lengths on the edges.

Acknowledgements—The authors are grateful to the two anonymous reviewers for their valuable comments. This work has been supported by two research projects from the University of La Laguna, grant numbers 221 43/99 and 180221024.

References

1 Hakimi SL (1964). Optimum locations of switching centers and the absolute centers and medians of a graph. *Opns Res* **12**(3): 450–459.
2 Minieka E (1981). A polynomial time algorithm for finding the absolute center of a network. *Networks* **11**(4): 351–355.
3 Kariv O and Hakimi SL (1979). An algorithmic approach to network location problems. I: The p-centers. *SIAM J Appl Math* **37**(3): 513–538.
4 Kariv O and Hakimi SL (1979). An algorithmic approach to network location problems. II: The p-medians. *SIAM J Appl Math* **37**(3): 539–560.
5 Erkut E and Neuman S (1989). Analytical models for locating undesirable facilities. *Eur J Opl Res* **40**(3): 275–291.
6 Church RL and Garfinkel RS (1978). Locating an obnoxious facility on a network. *Transportation Sci* **12**(2): 107–118.
7 Tamir A (1991). Obnoxious facility location on graphs. *SIAM J Discrete Math* **4**(4): 550–567.
8 Minieka E (1983). Anticenters and antimedians of a network. *Networks* **13**(3): 359–364.
9 Cappanera P (1999). A survey on obnoxious facility location problems. Technical Report TR-99-11, Dipartimento di Informatica, Università di Pisa.
10 Tamir A (1988). Improved complexity bounds for center location problems on networks by using dynamic data structures. *SIAM J Discrete Math* **1**(3): 377–396.
11 Megiddo N (1982). Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J Computing* **12**(4): 759–776.
12 Dyer ME (1984). Linear time algorithms for two-and three-variable linear programs. *SIAM J Computing* **13**(1): 31–45.
13 Burkard RE, Dollani H, Lin Y and Rote G (2001). The obnoxious center problem on a tree. *SIAM J Discrete Math* **14**(4): 498–509.
14 Melachrinoudis E and Zhang FG (1999). An O(*mn*) algorithm for the 1-maximin problem on a network. *Comp Opns Res* **26**(9): 849–869.
15 Berman O and Drezner Z (2000). A note on the location of an obnoxious facility on a network. *Eur J Opl Res* **120**(1): 215–217.
16 Hooker JN, Garfinkel RS and Chen CK (1991). Finite dominating sets for network location problems. *Opns Res* **39**(1): 100–118.
17 Fredman ML and Tarjan RE (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *J ACM* **34**(3): 596–615.
18 Melhorn K and Näher S (1999). *The LEDA Platform of Combinatorial and Geometric Computing.* Cambridge University Press (http://www.mpi-sb.mpg.de/LEDA/).
19 Mixed Integer Linear Program solver (ftp://ftp.es.ele.tue.nl/pub/lp_solve).

Received October 2001;
accepted July 2002 after one revision