An Aggressive Reduction Scheme for the Simple Plant Location Problem

Adam N. Letchford* Sebastian J. Miller

Published in *European Journal of Operational Research*, May 2014

Abstract

Pisinger *et al.* introduced the concept of 'aggressive reduction' for large-scale combinatorial optimisation problems. The idea is to spend much time and effort in reducing the size of the instance, in the hope that the reduced instance will then be small enough to be solved by an exact algorithm.

We present an aggressive reduction scheme for the 'Simple Plant Location Problem', which is a classical problem arising in logistics. The scheme involves four different reduction rules, along with lower- and upper-bounding procedures. The scheme turns out to be particularly effective for instances in which the facilities and clients correspond to points on the Euclidean plane.

Keywords: Facility location, combinatorial optimization.

1 Introduction

The *Simple Plant Location Problem* (SPLP), sometimes called the *Uncapacitated Facility Location Problem* or *Uncapacitated Warehouse Location Problem*, is a fundamental and much-studied problem in the Operational Research literature. A formal definition is as follows. We are given a set I of facilities and a set J of clients. For any $i \in I$, the fixed cost of opening facility i is f_i. For any $i \in I$ and any $j \in J$, the cost of serving client j from facility i is c_{ij}. The goal is to decide which facilities to open, and to assign each client to an open facility, at minimum cost.

An excellent survey of the early work on the SPLP is given by Krarup & Pruzan [17]. In that survey, it is also formally proven that the SPLP is NP-hard, by reduction from the Set Covering Problem. More recent surveys on theory, algorithms and applications include Cornuéjols *et al.* [9], Labbé *et al.* [19, 20] and Verter [28].

*Corresponding author. Department of Management Science, Lancaster University Management School, Lancaster LA1 4YX, United Kingdom. E-mail A.N.Letchford@lancaster.ac.uk

Some instances of the SPLP arising in practice have hundreds or even thousands of clients. Moreover, instances with large numbers of facilities and clients arise if one takes a *continuous* location problem and then 'discretises' it, by modelling continuous regions (approximately) as sets of discrete points. This led us, in our former paper [21], to devise fast heuristics and bounding procedures for large-scale instances. In this paper, we move on to consider how to solve such instances to proven optimality (or near-optimality).

Pisinger *et al.* [23] introduced the concept of 'aggressive reduction' for large-scale combinatorial optimisation problems. The idea is to spend much time and effort in reducing the size of the instance, using a suitable collection of variable-elimination tests. The hope is that the reduced instance will then be small enough to be solved by an exact algorithm.

In this paper, we present an aggressive reduction scheme for the SPLP, which uses four different reduction procedures. The scheme turns out to be particularly effective when the facilities and clients correspond to points on the Euclidean plane, and the cost of assigning a client to a facility is proportional to the distance between them. Indeed, for this case, we are able to solve instances that are significantly larger than those previously solved in the literature.

The structure of the paper is as follows. Section 2 is a brief literature review. In Section 3, we present two reduction procedures that are 'bound free', in the sense that no lower or upper bound is needed to apply them. In Section 4, we present some simple lower- and upper-bounding procedures, based on linear programming (LP). In Section 5, we present two more reduction procedures, that use the bounds in combination with LP duality. Extensive computational results are given in Section 6, and concluding remarks are given in Section 7.

We assume throughout the paper that the f_i and c_{ij} are positive integers. We also let m denote the number of facilities and n the number of clients.

2 Literature Review

In this section, we review the main papers on relaxations, lower bounds, reduction rules and exact algorithms for the SPLP. There are also many papers on heuristics, meta-heuristics and approximation algorithms for the SPLP, but we do not cover them, for the sake of brevity. Instead, we refer the reader to the surveys mentioned in the introduction.

2.1 Linear programming relaxation

It is possible to formulate the SPLP as a 0-1 LP in several ways. (See, e.g., [3, 8, 10, 17]. The most commonly used formulation, due to Balinski [3], is

the following:

$$\begin{aligned}
\min \quad & \textstyle\sum_{i\in I} f_i y_i + \sum_{i\in I}\sum_{j\in J} c_{ij} x_{ij} && && (1)\\
\text{s.t.} \quad & \textstyle\sum_{i\in I} x_{ij} = 1 && (\forall j \in J) && (2)\\
& y_i - x_{ij} \geq 0 && (\forall i \in I, j \in J) && (3)\\
& x_{ij} \in \{0,1\} && (\forall i \in I, j \in J) && (4)\\
& y_i \in \{0,1\} && (\forall i \in I). && (5)
\end{aligned}$$

Here, x_{ij} indicates whether client j is assigned to facility i, and y_i indicates whether facility i is opened. The constraints (2) and (3) will be called *assignment constraints* and *variable upper bounds* (VUBs), respectively.

The LP relaxation is obtained by replacing the constraints (4), (5) with lower and upper bounds of 0 and 1, respectively. A key feature of this relaxation is that it typically gives a very good lower bound, and is often even integral [1, 10, 22, 24]. On the other hand, the presence of the VUBs makes the LP highly degenerate. Specialised primal simplex methods have been devised to cope with VUBs [25, 26], but they are not entirely satisfactory. For an alternative formulation of the SPLP as a set covering problem, see [8].

2.2 Dual ascent and dual adjustment

In their seminal paper, Bilde & Krarup [7] proposed to compute a lower bound by solving the dual of the LP approximately. The dual can be written, after some simplification, in the following form:

$$\begin{aligned}
\max \quad & \textstyle\sum_{j\in J} v_j && && (6)\\
\text{s.t.} \quad & \textstyle\sum_{j\in J} w_{ij} \leq f_i && (\forall i \in I) && (7)\\
& v_j - w_{ij} \leq c_{ij} && (\forall i \in I, j \in J) && (8)\\
& v_j \geq 0 && (\forall j \in J) && (9)\\
& w_{ij} \geq 0 && (\forall i \in I, j \in J). && (10)
\end{aligned}$$

Here, the v_j and w_{ij} are the dual variables for the assignment constraints and VUBs, respectively. Now, observe that there always exists an optimal solution to the dual in which

$$w_{ij} = \max\{0, v_j - c_{ij}\} \qquad (\forall i \in I,\ j \in J). \tag{11}$$

This leads to the following so-called *condensed* dual:

$$\begin{aligned}
\max \quad & \textstyle\sum_{j\in J} v_j && && (12)\\
\text{s.t.} \quad & \textstyle\sum_{j\in J} \max\{0, v_j - c_{ij}\} \leq f_i && (\forall i \in I) && (13)\\
& v_j \geq 0 && (\forall j \in J).
\end{aligned}$$

Bilde and Krarup devised a fast heuristic, called *dual ascent*, for finding a good feasible solution to the condensed dual. The basic idea is to initialise the v_j at small values, and then repeatedly scan through the set of customers, increasing the dual values little by little until no more increase is possible.

In our own paper [21], we showed that dual ascent runs in $\mathcal{O}(m^2 n)$ time. We described an improved version, which is faster in practice but has the same worst-case running time, along with a modified version which runs in only $\mathcal{O}(mn \log m)$ time, yet still produces reasonably good lower bounds.

Erlenkotter [11] proposed an effective iterative method, called 'dual adjustment', for improving the dual solution generated by dual ascent. Several enhancements were also proposed by Körkel [16]. More recently, Hansen *et al.* [15] presented a variable neighborhood search (VNS) heuristic for the condensed dual. For the sake of brevity, we do not go into details.

2.3 Lagrangian relaxation

In [5, 13], it was proposed to solve the dual approximately using Lagrangian relaxation, rather than dual ascent/adjustment. The assignment constraints (2) are relaxed, using a vector $\lambda \in \mathbb{R}^n$ of Lagrangian multipliers. The relaxed problem is then to minimise the Lagrangian

$$F(x, y, \lambda) = \sum_{i \in I} f_i y_i + \sum_{i \in I} \sum_{j \in J} (c_{ij} - \lambda_j) x_{ij} + \sum_{j \in J} \lambda_j,$$

subject to (3)-(5). This relaxation can be solved quickly, by computing for each $i \in I$ the 'Lagrangian reduced cost':

$$r_i = f_i - \sum_{j \in J} \max \{0, \lambda_j - c_{ij}\}, \tag{14}$$

and then opening all facilities for which r_i is negative. The corresponding lower bound is:

$$\sum_{j \in J} \lambda_j - \sum_{i \in I} \max\{0, -r_i\}. \tag{15}$$

The problem of finding optimal Lagrangian multipliers, the so-called *Lagrangian* dual, takes the form:

$$\max_{\lambda \in \mathbb{R}^n} \min_{(x,y) \in \{0,1\}^{mn+m}} F(x, y, \lambda).$$

This can be solved approximately using, for example, the subgradient method.

Very recently, Beltran-Royo *et al.* [6] applied to the SPLP a method called *semi-Lagrangian* relaxation. It gives tighter lower bounds, but at the cost of an increased running time.

2.4 Problem reduction

By 'problem reduction', we mean permanently fixing variables to 0 or 1, without losing any optimal solutions.

Körkel [16] showed how to apply problem reduction to the SPLP, within a dual ascent or dual adjustment context. Let $\bar{v} \in \mathbb{R}^n$ denote a feasible solution to the condensed dual, let $\text{LB} = \sum_{j \in J} \bar{v}_j$ denote the corresponding lower bound, let UB be any upper bound, and, for all $i \in I$, define

$$s_i = f_i - \sum_{j \in J} \max\{0, \bar{v}_j - c_{ij}\}. \tag{16}$$

Then, just like r_i in the previous subsection, s_i can be viewed as an estimate of the reduced cost of y_i in the primal. So, for any $i \in I$ such that s_i exceeds UB − LB, the variable y_i can be permanently fixed to 0, along with x_{ij} for all $j \in J$. Also, for any $i \in I$ and $j \in J$ such that

$$s_i + \max\{0, c_{ij} - \bar{v}_j\} > \text{UB} - \text{LB},$$

the variable x_{ij} can be permanently fixed to 0.

Beasley [5] gave a slightly different problem reduction procedure, for use in a Lagrangian context. It uses the Lagrangian reduced costs r_i given in equation (14). Namely, if r_i is positive and LB $+r_i >$ UB for any i, then y_i can be permanently fixed to 0 and, if r_i is negative and LB $-r_i >$ UB for any i, then y_i can be permanently fixed to 1.

2.5 Exact algorithms

All of the exact algorithms of which we are aware are based on branch-and-bound. An early one of Efroymson & Ray [10] used a 0-1 LP formulation that has a weak lower bound, but can be solved by inspection. The one of Bilde & Krarup [7] derived lower bounds using dual ascent, and upper bounds by opening all facilities for which $s_i = 0$. The algorithm of Erlenkotter [11] is similar, except that dual adjustment was used to improve the lower bounds. Further improvements to this scheme were made in [16, 27]. These algorithms can solve instances with up to about 200 facilities and clients.

Beasley [5] derived lower bounds using Lagrangian relaxation, and upper bounds by opening all facilities for which $r_i \leq 0$. Galvão & Raggi [13] presented a three-phase exact algorithm, using dual ascent, then Lagrangian relaxation, then branch-and-bound. (This approach can in fact be viewed as a rudimentary aggressive reduction scheme.) These algorithms can solve instances with up to a couple of hundred facilities and clients.

More recently, Hansen *et al.* [15] presented a branch-and-bound algorithm based on both primal and dual VNS heuristics, and Beltran-Royo *et al.* [6] devised one based on semi-Lagrangian relaxation. These algorithms can solve some instances with thousands of facilities and clients, but some instances with a few hundred facilities and clients remain a challenge.

3 Two Bound-Free Reduction Procedures

In this section, we present our first two reduction procedures, which, as mentioned in the introduction, are 'bound-free'. We remark that all of our procedures store the assignment costs c_{ij} in a particular format. Specifically, we store the following for each client $j \in J$:

1. The 'degree' of the client, denoted by $d(j)$, which is the number of variables x_{ij} which have not yet been eliminated.
2. The set of the indices in I associated with those variables, which we denote by $I(j)$.
3. A list of the associated c_{ij} values.

Also, throughout the remainder of the paper, we let $\bar{d}$ denote the current maximum client degree, and $\sigma(d)$ denote the current sum of the degrees (which is equal to the number of x variables that still remain).

3.1 First bound-free reduction procedure

Our first bound-free reduction procedure is based on the following simple lemma:

Lemma 1 *For each client index $j \in J$, define*

$$\Delta_j = \min_{i \in I} \{c_{ij} + f_i\}.$$

One can then eliminate (i.e., permanently fix at zero) all variables x_{ij} for which $c_{ij} \geq \Delta_j$, without losing at least one optimal solution.

Proof. For a given $j \in J$, let $k \in I$ be such that $c_{kj} + f_k = \Delta_j$. Now, suppose a feasible solution has x_{ij} equal to 1 for some $i \in I \setminus \{k\}$ such that $c_{ij} \geq \Delta_j$. Then one can obtain another feasible solution with no larger cost, by changing the value of x_{ij} from 1 to 0, changing the value of x_{kj} from 0 to 1, and setting y_k to 1 (regardless of whether it was 0 or 1 before). □

The following lemma shows that one can apply this reduction rule efficiently:

Lemma 2 *One can eliminate variables in accordance with Lemma 1 in $\mathcal{O}(mn)$ time.*

Proof. For a given j, one can compute Δ_j in $\mathcal{O}(m)$ time. Once this is done, one can identify all $i \in I$ such that $c_{ij} \geq \Delta_j$ in $\mathcal{O}(m)$ time. Repeating this for all $j \in J$ gives the desired running time. □

Although this reduction procedure is very simple, it performs remarkably well when the facility costs are small relative to the assignment costs (see Section 6). It also has the nice feature that it processes each client independently, and therefore needs only $\mathcal{O}(m)$ space.

3.2 Second bound-free reduction procedure

Our second bound-free reduction procedure is based on the following theorem:

Theorem 1 *Let $j \in J$ and $i \in I(j)$ be fixed. Define the set:*

$$I^*(i,j) = \{p \in I(j) : \; c_{pj} \geq c_{ij}\}.$$

Also, for all $q \in J$, define:

$$c(i,j,q) = \min_{p \in I^*(i,j) \cap I(q)} \{c_{pq}\}.$$

(If $I^*(i,j) \cap I(q) = \emptyset$ for some q, then just set $c(i,j,q)$ to infinity.) Then, if there exists some facility $p \in I \setminus I^*(i,j)$ such that:

$$f_p \leq \sum_{q \in J:\; p \in I(q)} \max\{0, c(i,j,q) - c_{pq}\}, \tag{17}$$

then one can eliminate all variables x_{kj} for which $k \in I^*(i,j)$, without losing at least one optimal solution.

Proof. Suppose that, in an optimal solution, $x_{kj} = 1$ for some $k \in I^*(i,j)$. Then, all facilities in $I(j) \setminus I^*(i,j)$ must be closed, since, otherwise, client j could have been assigned to a nearer open facility. So, every client $q \in J$, including j itself, must be assigned to a nearest open facility in $I^*(i,j)$. The cost of assigning a client q to a facility in $I^*(i,j)$ is at least $c(i,j,q)$. Now, suppose that the specified facility p exists. Consider what would happen if the optimal solution was modified by opening facility p, and then reassigning to facility p every client for which $c(i,j,q) > c_{pq}$. The cost incurred by opening facility p would be f_p, whereas the total cost saved by the reassignment would be no smaller than the right-hand side of (17). Therefore, the new solution would be no more expensive than the original solution. Since the original solution was optimal by assumption, the new one must be too. Therefore there exists an optimal solution such that $x_{kj} = 0$ for all $k \in I^*(i,j)$. □

The following lemma gives a bound on the running time needed to put Theorem 1 into practice.

Lemma 3 *For a fixed $j \in J$, one can find in $\mathcal{O}(\sigma(d) \log m)$ time the location $i \in I(j)$ that is closest to j, among all locations for which a facility p satisfying (17) exists.*

Proof. First, suppose that i is fixed. One can compute $I^*(i,j)$ in $\mathcal{O}(m)$ time. Then, one can compute the $c(i,j,q)$ for all q in $\mathcal{O}(\sigma(d))$ time. After

that, one compute the right-hand side of (17) for all p in $\mathcal{O}(\sigma(d))$ time. Note that $m = \mathcal{O}(\sigma(d))$. So, for fixed i, one can check in $\mathcal{O}(\sigma(d))$ time whether a facility p exists satisfying (17). Let us call this entire $\mathcal{O}(\sigma(d))$-time procedure an 'i-check'.

Now, using any of several standard sorting algorithms, one can sort the facilities in non-decreasing order of distance from j in $\mathcal{O}(m \log m)$ time. Using binary search, one can then find the desired location i by performing $\mathcal{O}(m)$ i-checks. The time taken to perform these i-checks is $\mathcal{O}(\sigma(d) \log m)$. This dominates the time taken to sort the facilities. □

The second reduction procedure, then, consists of doing the above for each client $j \in J$. The total time taken by the procedure is $\mathcal{O}(\sigma(d)n \log m)$. Although this running time is rather high, we have found that it is well worth it, since it typically leads to well over 95% of the variables being eliminated (see Section 6). The space used is $\mathcal{O}(\sigma(d))$.

4 Bound Computations

After applying the bound-free reduction procedures, the next step in our aggressive reduction scheme is to compute lower and upper bounds. This is explained in the following two subsections.

4.1 Lower bound

As mention in Section 2, the lower bound obtained by solving the LP relaxation of the 0-1 LP is typically strong, but solving the LP exactly is time-consuming, due to its large size and its degeneracy. For this reason, we expected that it would be necessary to use dual ascent/adjustment or Lagrangian relaxation to compute lower bounds. It turns out, however, that our bound-free reduction procedures are so effective at eliminating x variables, that the LP relaxation of the reduced problem can usually be solved in reasonable times with the simplex method.

Recall that $I(j)$ denotes the set of locations such that the variable x_{ij} has not yet been eliminated. The LP that we solve is the following:

$$\begin{aligned}
\min \quad & \textstyle\sum_{i \in I} f_i y_i + \sum_{j \in J} \sum_{i \in I(j)} c_{ij} x_{ij} && && (18)\\
\text{s.t.} \quad & \textstyle\sum_{i \in I(j)} x_{ij} \geq 1 && (\forall j \in J) && (19)\\
& y_i - x_{ij} \geq 0 && (\forall j \in J,\ i \in I(j)) && (20)\\
& x_{ij} \geq 0 && (\forall j \in J,\ i \in I(j)) && (21)\\
& y_i \geq 0 && (\forall i \in I). && (22)
\end{aligned}$$

Note that this LP has $\mathcal{O}(\sigma(d))$ variables and constraints. Note also that the assignment constraints have been changed from equations to inequalities, and we do not include explicit upper bounds of 1 on the variables. This

has no effect on the lower bound, but it reduces the size of the LP, and also ensures that the dual of the LP is in the same form as (6)–(10).

We found that, in practice, the dual simplex method solves the LP relaxation more quickly than the primal simplex method. This is probably because setting all variables to zero immediately yields a dual feasible solution, whereas one would have to use artificial variables to find a primal feasible solution.

We denote by (x^*, y^*) the optimal primal solution, and (v^*, w^*) the optimal dual solution. For reasons which will become clear in the Section 5, we will need access to (v^*, w^*) in our final two reduction procedures. Note however that, given v^*, one can re-construct w^* using the identity (11) for all $j \in J$ and $i \in I(j)$. Therefore it suffices to store only v^* in memory. This is useful because storing v^* takes up only $\mathcal{O}(n)$ space, whereas storing w^* would take $\mathcal{O}(\sigma(d))$ space.

In practice, the lower bound obtained from the LP is usually extremely strong. In fact, typically it is within 0.02% of optimal; see Section 6.

4.2 Upper bounds

Since the lower bound from the LP is typically very strong, it seems plausible that the optimal solution to the LP relaxation could have some useful information that could be exploited in a heuristic. This was the motivation for our two procedures for producing upper bounds.

The first heuristic is a modified version of the classical 'drop' heuristic of Feldman *et al.* [12]. In our version, all facilities with positive y^* values are opened initially, but then facilities that have small y^* values are the first to be considered for closure. In each iteration, we compute for suitable clients j the quantity δ_j, that represents the change in the cost that would occur if client j were re-assigned to the second nearest open facility. The heuristic is as follows:

Let $I^+ = \{i \in I : y_i^* > 0\}$.
Sort the facilities in I^+ in non-decreasing order of y^* value.
Store the sorted facility indices in a list L.
Temporarily open all facilities in I^+.
For each client $j \in J$:
    Temporarily assign j to the nearest facility in $I^+ \cap I(j)$.
For each facility i in L do:
    Let S be the set of clients currently assigned to i.
    For each $j \in S$ do:
        Let $T = (I^+ \cap I(j)) \setminus \{i\}$.
        If $T = \emptyset$, then set δ_j to infinity.
        Otherwise, set δ_j to $\min_{k \in T} c_{kj} - c_{ij}$.
    If $f_i \geq \sum_{j \in S} \delta_j$, then close facility i.

Output the cost of the resulting solution.

If implemented in a naive way, this first heuristic takes $\mathcal{O}(\sigma(d)m)$ time and $\mathcal{O}(\sigma(d))$ space. Using an appropriate data structure, however, it can be implemented to run in only $\mathcal{O}(\sigma(d))$ time. The key is to keep, for each client, a pointer to the second-nearest open facility, and update it only when necessary. A similar data structure was used in our earlier paper [21]. We omit details, for brevity.

Our second heuristic, which is more time-consuming, uses LP. The idea is to iteratively round fractional y variables to 1, until a 0-1 solution is obtained:

Let (x^*, y^*) be the optimal solution to the LP relaxation.
Repeat:
  Let $F := \{i \in I : 0 < y_i^* < 1\}$.
  Let $k := \arg\max\{y_i^* : i \in F\}$.
  Add the equation $y_k = 1$ to the LP.
  Re-optimise the LP via dual simplex.
  Update (x^*, y^*).
Until $y_i \in \{0, 1\}$ for all $i \in I$.
Output the cost of the resulting solution.

Although this second heuristic involves the solution of $\mathcal{O}(m)$ LPs, in practice, the number of LPs solved is very small. Moreover, the number of dual simplex pivots needed to re-optimise the LP in each iteration is also typically very small.

We take the best of the upper bounds from the two heuristics as our definitive upper bound. In practice, this upper bound is usually extremely strong. As with the lower bound, it is typically within 0.02% of optimal; see Section 6.

5 Bound-Based Reductions

In this section, we present our other two reduction procedures, which use the lower and upper bounds obtained with the procedures described in the previous section, together with certain information from the LP relaxation. Specifically, the first uses reduced costs, and the second uses shadow prices.

5.1 Reduction based on reduced costs

A standard procedure for eliminating variables in integer programming is *reduced-cost fixing*, due to Balas & Martin [2]. The basic idea is that, if an integer variable takes the value 0 in the LP solution, and its reduced cost exceeds the difference between the lower bound from the LP and the an upper bound, then that variable can be fixed at 0 permanently.

Observe that, in the case of the SPLP, forcing a variable x_{ij} to take the value 1 also forces the variable y_i to take the value 1. Using this fact, one can make reduced-cost fixing more powerful. This was shown already by Körkel [16] in the context of dual ascent (Subsection 2.4), but we will prove it in the context of LP.

First, we show how to compute reduced costs efficiently:

Lemma 4 *Let $v^* \in \mathbb{Q}_+^n$ be the vector of optimal dual prices for the constraints (19), that was obtained in Subsection 4.1. For any $j \in J$ and $i \in I(j)$, the reduced cost of x_{ij} is:*

$$\pi_{ij}^* = \max\left\{0, c_{ij} - v_j^*\right\},$$

and, for any $i \in I$, the reduced cost of y_i is:

$$\rho_i^* = f_i - \sum_{j \in J:\, i \in I(j)} \max\left\{0, v_j^* - c_{ij}\right\}.$$

Proof. From LP duality, the reduced cost of x_{ij} will equal the slack of the corresponding constraint (8). That is, it will equal $c_{ij} - v_j^* + w_{ij}^*$. Similarly, the reduced cost of y_i will equal the slack of the corresponding constraint (7), but with the dual variables omitted for all VUBs whose x variables have already been eliminated. That is, it will equal

$$f_i - \sum_{j \in J:\, i \in I(j)} w_{ij}^*.$$

Now, from (11), one can obtain an optimal dual vector $w^* \in \mathbb{Q}_+^{\sigma(d)}$ by setting w_{ij}^* to $\max\left\{0, v_j^* - c_{ij}\right\}$ for all $j \in J$ and all $i \in I(j)$. The results then follow immediately. □

Now we present our strengthened version of reduced-cost fixing:

Proposition 1 *Let LB be the lower bound obtained by solving the LP (18)–(22), let UB be an upper bound, and let π^* and ρ^* be the vectors of reduced costs defined in Lemma 4. If, for some $j \in J$ and $i \in I(j)$, we have*

$$\pi_{ij}^* + \rho_i^* > UB - LB,$$

then we eliminate x_{ij} without losing any optimal solutions.

Proof. We consider two cases. The first case is $\rho_i^* = 0$. From the definition of reduced costs, if we forced x_{ij} to take the value 1, the lower bound would increase by at least π_{ij}^*, and the result is immediate. The second case is $\rho_i^* > 0$. In this case, we can obtain an alternative optimal dual solution by increasing the value of w_{ij} by ρ_i^*. From Lemma 4, the effect of this change will be to cause the reduced cost of y_i to drop from ρ_i^* to zero and the reduced cost of x_{ij} to increase by ρ_i^*. We can then proceed as in the first case. □

Note that this third reduction procedure takes only $\mathcal{O}(\sigma(d))$ time. Moreover, the space complexity is only $\mathcal{O}(n)$, since the only thing that needs to be stored in the computer's memory throughout is the vector v^*. Nevertheless, the procedure is very powerful, and it seems to complement the bound-free reduction procedures nicely (see Section 6).

For the sake of completeness, we mention that an analog of Proposition 1 can be derived for use in the context of Lagrangian relaxation. Specifically, for a given vector $\lambda \in \mathbb{R}^n$ of Lagrangian multipliers, a given vector $r \in \mathbb{R}^m$ of Lagrangian reduced costs as defined in (14), a given lower bound LB as defined in (15), and an arbitrary upper bound UB, one can eliminate x_{ij} if

$$\max\{0, c_{ij} - \lambda_j\} + \max\{0, r_i\} > \text{UB} - \text{LB}.$$

We omit the (simple but tedious) proof for the sake of brevity.

5.2 Reduction based on dual prices

Our final reduction procedure uses the dual prices for the VUBs (20) to replace x variables by y variables:

Proposition 2 *Let LB be the lower bound obtained by solving the LP (18)–(22), let UB be an upper bound, and let $v^* \in \mathbb{Q}_+^n$ be the vector of optimal dual prices for the constraints (19), that was obtained in Subsection 4.1. If, for any $j \in J$ and $i \in I(j)$, we have*

$$w_{ij}^* = \max\left\{0, v_j^* - c_{ij}\right\} > UB - LB,$$

then we can replace the variable x_{ij} with the variable y_i in the 0-1 LP, without losing any optimal solutions. (The variable x_{ij}, the VUB $y_i - x_{ij} \geq 0$, and the binary condition $x_{ij} \in \{0, 1\}$ can then all be discarded.)

Proof. By definition, w_{ij}^* is the dual price for the VUB $y_i - x_{ij} \geq 0$. That means that, if the slack of the VUB in a feasible solution to the LP relaxation is equal to some $\epsilon > 0$, the cost of that LP solution must be at least LB $+ \epsilon w_{ij}^*$. Moreover, the slack of the VUB in a 0-1 solution can only be either 0 or 1. So, if the stated condition holds, any solution for which the slack is 1 has a cost of at least LB $+ w_{ij}^*$, and therefore cannot be optimal. It follows that the slack must be 0 in an optimal solution. This means in turn that x_{ij} must equal y_i in an optimal solution. □

This reduction procedure, like the one given in the previous subsection, takes only $\mathcal{O}(\sigma(d))$ time and $\mathcal{O}(n)$ space. It is worth noting that it works best when v_j^* is much larger than c_{ij}, whereas the one in the previous subsection works best when the reverse holds. In that sense, the two bound-based reduction procedures complement each other nicely.

Note that the 0-1 LP that results after the application of the last procedure is no longer of the form (1)-(5), because y variables now appear in the assignment constraints. For this reason, we do not store the reduced 0-1 LP using the data structure described at the start of Section 3. Instead, we simply store the final 0-1 LP in its entirety. This 0-1 LP can then be passed to a standard LP-based branch-and-bound solver, and solved to proven optimality. Once the optimal y vector is obtained, one can easily 'reconstruct' the optimal x vector, in $\mathcal{O}(\sigma(d))$ time, by simply assigning each client to its nearest open facility.

If the solver permits it, one can also pass the upper bound UB to it, and ask it to prune any branch-and-bound nodes whose lower bound exceeds UB. This typically leads to a reduction in the number of branch-and-bound nodes.

6 Computational Experiments

In this section we report the results of some computational experiments. Our four reduction procedures and three bounding procedures were coded in C, using Microsoft Visual Studio.Net 2008. We used routines from the Callable Library of IBM CPLEX version 12.1 to solve the LP relaxations and to run branch-and-bound. Specifically, the initial LP relaxation described in Subsection 4.1 was solved by dual simplex, the LPs were re-optimised using dual simplex in the iterated rounding heuristic described in Subsection 4.2, and the final reduced 0-1 LPs were solved with the mixed-integer optimiser. We used default parameter settings in all cases, with one exception: in the branch-and-bound phase, the precision was increased so that we were guaranteed to find the exact optimal value. The code was run on a 2.33 GHz PC with 3.25 Gb of RAM, operating under Windows XP.

6.1 Test instances

When designing our aggressive reduction scheme, we had in mind mainly instances in which facilities and customers are located in the Euclidean plane. For this reason, the majority of our test instances were created according to the scheme used in [1, 15]. This involves setting m equal to n, setting each facility and customer location to a random point in the unit square, setting each assignment cost to the Euclidean distance between the corresponding points, and taking facility costs from a uniform distribution.

Since the current leading algorithm is the one in [15], we consider the same four options as they do for the facility costs:

- Small and constant: all facility costs set to $\sqrt{n}/1000$.
- Medium and constant: all facility costs set to $\sqrt{n}/100$.

	Reduction 1		Reduction 2		Bounding		
$m = n$	%elim	time	%elim	time	%LB	%UB	time
1000	99.53	0.02	99.76	0.05	0.000	0.000	0.03
2000	99.43	0.07	99.79	0.39	0.002	0.002	0.13
3000	98.77	0.17	99.72	1.61	0.007	0.004	0.56
4000	98.50	0.30	99.73	3.59	0.006	0.007	1.13
5000	98.21	0.52	99.74	6.82	0.009	0.008	2.12
6000	97.94	0.83	99.74	11.50	0.007	0.009	3.61
7000	97.66	1.10	99.75	17.71	0.010	0.012	6.09
8000	97.38	1.40	99.75	26.28	0.009	0.006	8.03
9000	97.11	1.91	99.75	37.41	0.008	0.008	11.72
10000	96.83	3.84	99.76	51.70	0.009	0.011	17.83

Table 1: Results from first three phases — small facility costs

- Large and constant: all facility costs set to $\sqrt{n}/10$.
- Varied: facility costs uniformly distributed between $\sqrt{n}/1000$ and $\sqrt{n}/10$.

To avoid problems with rounding errors, all costs were then multiplied by 5000 and rounded down to the nearest integer.

Results for the above four families of instances are reported in the following four subsections. In Subsection 6.6, we report results for some other instances, taken from the literature.

6.2 Instances with small and constant facility costs

We begin with the random instances with small and constant facility costs. In this case, we were able to solve instances with $m = n$ up to 10000. Table 1 shows, for each problem size, the results obtained from the first three phases of our scheme. Specifically, for the first and second reduction procedures (Subsections 3.1 and 3.1), the cumulative percentage of x variables eliminated and the running time in seconds is shown. For the bounding phase (Section 4), we report the gap between the lower and upper bounds and the optimum, expressed as a percentage of the optimum, along with the running time in seconds. The figures given in each row are the average over five random instances.

We see that the bound-free reduction procedures are extremely effective for these instances, typically eliminating over 99.7% of the variables. Moreover, the lower and upper bounds are of extremely good quality, being consistently within a fraction of a percent of optimal.

Table 2 shows, for each problem size, the results obtained from the last three phases of our scheme. For the third and fourth reduction procedures

	Reduction 3		Reduction 4		Branch-and-Bound	
$m = n$	%elim	time	%elim	time	Nodes	time
1000	99.82	0.00	99.94	0.00	1.0	0.03
2000	99.89	0.00	99.96	0.00	1.0	0.19
3000	99.87	0.00	99.91	0.00	1.2	1.02
4000	99.86	0.00	99.88	0.00	4.8	3.15
5000	99.83	0.00	99.85	0.00	13.0	9.80
6000	99.83	0.00	99.84	0.00	19.4	19.67
7000	99.80	0.00	99.80	0.00	226.6	52.56
8000	99.84	0.00	99.85	0.00	96.2	54.87
9000	99.82	0.00	99.82	0.00	572.0	105.14
10000	99.81	0.00	99.81	0.01	398.2	140.24

Table 2: Results from last three phases — small facility costs

(Subsections 5.1 and 5.2), we again show the cumulative percentage of x variables eliminated and the running time. For the branch-and-bound phase, we report the number of branch-and-bound nodes and the running time.

We see that, due to the good bounds, the last two reduction procedures eliminate a significant proportion of the variables that remained after the first two reduction procedures. This renders the reduced instance quite easy to solve by branch-and-bound. The running times of the last two reduction procedures are negligible, and the times taken by branch-and-bound are reasonable, especially if one considers the huge size of the original instances. (Note that an instance with $m = n = 10000$ has over one hundred million variables.)

We remark that the previous best algorithm for these instances, due to Hansen *et al.* [15], runs into difficulties when n reaches 7000 or so. Given the exponential growth in running times exhibited by exact algorithms for the SPLP, we believe that our approach is genuinely superior to theirs, rather than being merely due to using a slightly faster machine. Moreover, the only thing preventing us from solving larger instances in this class was the memory available on the machine.

6.3 Instances with medium and constant facility costs

Next, we consider the instances with medium and constant facility costs. Since these instances were harder, we were able to solve only instances with $m = n$ up to 4000. The results are shown in Tables 3 and 4. Again, each figure is the average over five instances.

In this case, the first reduction procedure is significantly less effective than before, particularly for the larger instances. (The explanation is that, as the facility costs increase, the quantities Δ_j increase as well, which makes

	Reduction 1		Reduction 2		Bounding		
$m = n$	%elim	time	%elim	time	%LB	%UB	time
500	84.537	0.01	97.498	0.06	0.004	0.001	0.08
1000	74.559	0.02	97.714	0.44	0.009	0.010	0.45
1500	65.643	0.06	97.846	1.42	0.012	0.011	1.36
2000	57.389	0.11	97.985	3.18	0.004	0.013	3.30
2500	50.203	0.18	98.058	6.06	0.009	0.003	15.11
3000	43.597	0.28	98.127	10.21	0.010	0.003	47.75
3500	38.076	0.38	98.193	15.81	0.010	0.006	60.94
4000	28.134	0.49	98.249	23.37	0.016	0.005	173.968

Table 3: Results from first three phases — medium facility costs

	Reduction 3		Reduction 4		Branch-and-Bound	
$m = n$	%elim	time	%elim	time	Nodes	time
500	99.400	0	99.870	0	1.0	0.08
1000	99.373	0	99.647	0	4.6	0.74
1500	99.494	0	99.488	0	25.0	2.66
2000	99.395	0	99.555	0	4.0	3.11
2500	99.355	0	99.554	0	24.4	14.72
3000	99.447	0	99.421	0	74.0	98.29
3500	99.291	0	99.437	0	66.8	156.97
4000	99.057	0	99.152	0	202.2	573.83

Table 4: Results from last three phases — medium facility costs

	Reduction 1		Reduction 2		Bounding		
$m = n$	%elim	time	%elim	time	%LB	%UB	time
250	0.00	0.00	84.06	0.06	0.000	0.000	0.15
500	0.00	0.00	86.00	0.40	0.001	0.000	1.13
750	0.00	0.02	87.02	1.25	0.000	0.000	7.55
1000	0.00	0.03	87.88	2.90	0.009	0.001	27.11
1250	0.00	0.05	88.44	5.40	0.041	0.011	191.21
1500	0.00	0.06	89.13	8.84	0.011	0.017	173.65
1750	0.00	0.08	89.56	14.07	0.019	0.023	490.50

Table 5: Results from first three phases — large facility costs

	Reduction 3		Reduction 4		Branch-and-Bound	
$m = n$	%elim	time	%elim	time	nodes	time
250	98.059	0.00	99.837	0.00	1.0	0.17
500	97.807	0.00	99.866	0.00	1.0	0.01
750	97.299	0.00	99.887	0.00	1.0	0.02
1000	96.489	0.00	98.713	0.00	6.8	16.29
1250	93.068	0.02	94.572	0.02	28.2	587.50
1500	94.300	0.01	95.611	0.01	9.2	141.17
1750	94.436	0.00	96.258	0.00	20.4	804.45

Table 6: Results from last three phases — large facility costs

it less likely that any given assignment cost c_{ij} will be larger than Δ_j.) The other reduction procedures, and the bounding procedures, are slightly less effective as well.

Nevertheless, we believe that, on the whole, these results are very encouraging. Indeed, for instances in this class, Hansen *et al.* [15] were able to solve only instances with $m = n \leq 3000$. For us, the bottleneck was the memory limit in the LP solver in our version of CPLEX, which could cope only with two million variables.

6.4 Random instances with large and constant facility costs

Next, we consider the instances with large and constant facility costs. These instances were harder still, and we were able to solve only instances with $m = n$ up to 1750. The results are shown in Tables 5 and 6.

We see that, for these instances, the first reduction procedure failed to eliminate any variables at all. The other reduction procedures, and the bounding procedures, show some deterioration as well. Nevertheless, these results are again very good. For comparison, for instances in this class,

	Reduction 1		Reduction 2		Bounding		
$m = n$	%elim	time	%elim	time	%LB	%UB	time
2000	91.668	0.08	98.368	2.93	0.001	0.000	1.35
4000	91.776	0.33	98.874	16.57	0.001	0.002	5.52
6000	91.111	1.01	99.050	50.43	0.002	0.001	13.83
8000	91.121	1.53	99.182	111.08	0.001	0.000	25.49
10000	90.347	4.38	99.270	205.31	0.003	0.002	41.38
12000	89.681	3.78	99.322	351.48	0.003	0.003	64.14
14000	89.024	6.38	99.360	567.63	0.003	0.002	91.69
16000	88.582	9.18	99.400	802.52	0.001	0.001	118.16
18000	88.113	19.84	99.430	1093.50	0.003	0.003	152.43

Table 7: Results from first three phases — varied facility costs

	Reduction 3		Reduction 4		Branch-and-Bound	
$m = n$	%elim	time	%elim	time	nodes	time
2000	99.876	0.01	99.961	0.01	1.0	0.02
4000	99.927	0.00	99.971	0.00	1.0	0.27
6000	99.942	0.01	99.967	0.01	1.6	1.00
8000	99.961	0.02	99.985	0.02	1.0	0.12
10000	99.959	0.02	99.974	0.02	3.2	5.31
12000	99.961	0.02	99.971	0.02	4.6	8.58
14000	99.951	0.03	99.956	0.03	4.0	26.66
16000	99.973	0.04	99.987	0.04	2.0	6.66
18000	99.976	0.04	99.979	0.04	4.0	16.33

Table 8: Results from last three phases — varied facility costs

Hansen *et al.* [15] were able to solve only instances with $m = n \leq 1400$. Again, for us, the bottleneck was the LP solver.

6.5 Random instances with varied facility costs

Next, we consider the instances with varied facility costs. These instances turned out to be very easy, and we were able to solve instances with $m = n$ up to 18000. The results are in Tables 7 and 8.

We see that all reduction procedures work extremely well for these instances. An intuitive explanation for this is the following. When m is very large, there is a high probability that every customer is close to a cheap facility. Then, the variable c_{ij} has a high chance of being eliminated, in the first three reduction procedures, if facility i is expensive.

We remark that, for instances in this class, Hansen *et al.* [15] solved only instances with $n \leq 15000$. Again, for us, the bottleneck was the LP solver.

	Reduction 1		Reduction 2		Bounding		
$m = n$	%elim	time	%elim	time	%LB	%UB	time
100	0.00	0.00	5.174	0.03	2.855	0.261	0.71
200	0.00	0.00	2.300	0.25	4.203	0.152	9.91
300	0.00	0.00	1.508	0.92	4.040	0.113	69.96
500	0.00	0.00	1.181	4.26	(7.083)		678.97
1000	0.03	0.00	0.601	38.87	—	—	—
2000	0.09	0.00	0.294	365.63	—	—	—

Table 9: Results from first three phases — `M*` instances

6.6 Other instances

Some other instances of the SPLP are available in the uncapacitated facility location library, `UflLib`. It can be found on the web at:

`http://www.mpi-inf.mpg.de/departments/d1/projects/benchmarks/UflLib/`

Some of these instances, such as the `Euclidean`, `K-median` and `ORLIB` instances, are constructed using a similar procedure to the one described above. For those instances, the results obtained with our reduction scheme were similar to those given in the previous subsections. The other instances, however, were constructed in different ways.

For the sake of brevity, we do not report results for all of the instances in `UflLib`. Instead, we report results for two sets of instances that proved to be especially difficult for our scheme. These were the so-called `M*` instances of Kratica *et al.* [18], and the `KG` instances, which were created by Ghosh [14] using a similar scheme to that of Koerkel [16].

In the `M*` instances, there is a negative correlation between the facility costs and the assignment costs. (More precisely, f_i is small when $\sum_{j \in J} c_{ij}$ is large, and vice-versa.) There are 22 `M*` instances, all of which have $m = n$. For values of m of 100, 200, 300 and 500, there are 5 instances each. There is also an instance with $m = 1000$ and another with $m = 2000$. The results for these instances are displayed in Tables 9 and 10. For the instances with $m \leq 500$, each figure is the average over the five instances. The missing values occur either because the optimal solution values are unknown, or because we were unable to solve the LP relaxation, due to time and memory problems. The value in parentheses in Table 9, for $m = 500$, is the average percentage gap between the upper and lower bounds found by our bounding procedures.

We see that the first, second and fourth reduction procedures have little or no effect, and the third reduction procedure works well only when $m = 100$. As a result, the LP and branch-and-bound solvers take an excessive amount of time and memory. Also, the lower bounds are poor. These factors

	Reduction 3		Reduction 4		Branch-and-Bound	
$m = n$	%elim	time	%elim	time	nodes	time
100	42.938	0.00	42.938	0.00	31.6	3.17
200	3.185	0.00	3.185	0.00	122.4	189.96
300	1.508	0.00	1.508	0.00	148.0	1449.91
500	1.181	0.00	1.181	0.00	—	—
1000	—	—	—	—	—	—
2000	—	—	—	—	—	—

Table 10: Results from last three phases — `M*` instances

combine to make the instances with $m \geq 500$ unsolvable with our approach. This is actually not surprising, since the `M*` instances were explicitly designed to have a large number of near-optimal solutions [18], which makes it hard to identify variables that could not be in an optimal solution.

In the `KG` instances, the facility and assignment costs are taken from uniform distributions. These instances come in three sizes, with $m = n \in \{250, 500, 750\}$. They also come in two kinds, 'symmetric' and 'asymmetric'. Also, instances named 'a', 'b' and 'c' have small, medium and large facility costs, respectively. For each of the resulting 18 combinations, there are five instances in the test set. This makes 90 instances in total.

The results for these instances are displayed in Tables 11 and 12. The figures in each row are the average over the five instances.

We see that the first reduction procedure is effective for the instances with small facility costs, but of no use at all for the other instances. The second reduction procedure is of some value, but the benefit decreases as the facility costs get larger. The gaps between the lower and upper bounds are comparatively large, and they increase as the facility costs increase. Due to these large gaps, the last two reduction procedures are of no benefit at all. The net effect is that we are able to solve only the smallest instances with small facility costs (of which there are ten). Although this may seem unimpressive, it is actually quite an achievement. By comparison, the algorithm in [6] managed to solve only two of them.

7 Conclusion

In this paper, we have demonstrated that 'aggressive reduction', originally proposed in the context of the quadratic knapsack problem, works very well when applied to planar Euclidean instances of the Simple Plant Location Problem. Using our five-phase scheme, in conjunction with the CPLEX MIP solver, we are able to solve to proven optimality larger instances than any previously solved in the literature. On the other hand, the `M*` and `KG`

Set	Reduction 1		Reduction 2		Bounding		
	%elim	time	%elim	time	%LB	%UB	time
sym-250-a	87.635	0.00	88.455	0.03	0.127	0.013	1.32
sym-500-a	88.316	0.00	88.706	0.26	(0.173)		26.83
sym-750-a	88.570	0.00	88.862	0.90	(0.157)		233.89
sym-250-b	0.000	0.00	15.286	0.29	(1.049)		48.93
sym-500-b	0.000	0.01	12.237	2.39	(0.990)		1356.76
sym-750-b	0.000	0.01	10.822	8.10	—	—	—
sym-250-c	0.000	0.00	1.533	0.51	(3.475)		44.27
sym-500-c	0.000	0.01	1.257	3.97	(3.177)		1183.62
sym-750-c	0.000	0.01	1.084	13.30	—	—	—
asym-250-a	87.575	0.00	88.392	0.03	0.133	0.034	1.35
asym-500-a	88.229	0.00	88.645	0.25	(0.156)		25.87
asym-750-a	88.615	0.02	88.902	0.91	(0.171)		252.78
asym-250-b	0.000	0.00	15.189	0.28	(1.065)		38.52
asym-500-b	0.000	0.01	12.162	2.41	(1.000)		1561.52
asym-750-b	0.000	0.02	10.913	8.11	—	—	—
asym-250-c	0.000	0.00	1.531	0.51	(3.340)		39.05
asym-500-c	0.000	0.00	1.241	3.88	(3.299)		1223.95
asym-750-c	0.000	0.00	1.089	13.30	—	—	—

Table 11: Results from first three phases — `KG` instances

	Reduction 3		Reduction 4		Branch-and-Bound	
Set	%elim	time	%elim	time	nodes	time
sym-250-a	88.455	0.00	88.455	0.00	23143.2	796.28
sym-500-a	88.706	0.00	88.706	0.00	—	—
sym-750-a	88.862	0.00	88.862	0.00	—	—
sym-250-b	15.286	0.00	15.286	0.00	—	—
sym-500-b	12.237	0.00	12.237	0.00	—	—
sym-750-b	—	—	—	—	—	—
sym-250-c	1.533	0.00	1.533	0.00	—	—
sym-500-c	1.257	0.01	1.257	0.01	—	—
sym-750-c	—	—	—	—	—	—
asym-250-a	88.392	0.00	88.392	0.00	58314.2	2069.62
asym-500-a	88.645	0.00	88.645	0.00	—	—
asym-750-a	88.902	0.00	88.902	0.00	—	—
asym-250-b	15.189	0.00	15.189	0.00	—	—
asym-500-b	12.162	0.00	12.162	0.00	—	—
asym-750-b	—	—	—	—	—	—
asym-250-c	1.531	0.00	1.531	0.00	—	—
asym-500-c	1.241	0.01	1.241	0.01	—	—
asym-750-c	—	—	—	—	—	—

Table 12: Results from last three phases — `KG` instances

instances present a major challenge to our approach.

As mentioned in the previous section, the only thing that prevented us from solving still larger planar Euclidean instances was limited computer memory. To address this, one could perhaps investigate the possibility of adding another reduction step, based on dual ascent/adjustment or Lagrangian relaxation, either immediately before or immediately after the second bound-free reduction procedure. Another possible topic for future research would be to add a local search step, to improve the upper bounds found by our heuristic. Finally, one could investigate the generation of cutting planes to strengthen the LP relaxation, either in the bounding phase or in the final branch-and-bound phase.

Acknowledgement: The first author was supported in part by the Engineering and Physical Sciences Research Council under grant EP/D072662/1.

References

[1] S. Ahn, C. Cooper, G. Cornuéjols & A.M. Frieze (1988) Probabilistic analysis of a relaxation for the p-median problem. *Math. Oper. Res.*, 13, 1–31.

[2] E. Balas & C.H. Martin (1980) Pivot and complement — a heuristic for 0-1 programming. *Mgmt. Sci.*, 26, 86–96.

[3] M. Balinski (1965) Integer programming: methods, uses, computation. *Mgmt. Sci.*, 12, 254–313.

[4] J.E. Beasley (1990) OR-Library: distributing test problems by electronic mail. *J. Opl Res. Soc.*, 41, 1069–1072.

[5] J.E. Beasley (1993) Lagrangian heuristics for location problems. *Eur. J. Opl Res.*, 65, 383–399.

[6] C. Beltran-Royo, J. Vial, A. Alonso-Ayuso (2012) Semi-lagrangian relaxation applied to the uncapacitated facility location problem. *Comput. Opt. & Appl.*, 51, 387–409.

[7] O. Bilde & J. Krarup (1977) Sharp lower bounds and efficient algorithms for the simple plant location problem. *Ann. Discr. Math.*, 1, 79–88.

[8] G. Cornuéjols, G.L. Nemhauser & L.A. Wolsey (1980) A canonical representation of simple plant location problems and its applications. *SIAM. J. Alg. Discr. Meth.*, 1, 261-272.

[9] G. Cornuéjols, G.L. Nemhauser & L.A. Wolsey (1990) The uncapacitated facility location problem. In: P.B. Mirchandani & R.L. Francis (eds.), *Discrete Location Theory*, pp. 1-54. New York: Wiley.

[10] M.A. Efroymson & T.L. Ray (1966) A branch-and-bound algorithm for plant location. *Oper. Res.*, 14, 361–368.

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

[12] E. Feldman, F.A. Lehrer & T.L. Ray (1966) Warehouse location under continuous economies of scale. *Mgmt. Sci.*, 12, 670–684.

[13] R. Galvão & L. Raggi (1989) A method for solving to optimality uncapacitated location problems. *Ann. Oper. Res.*, 18, 225–244.

[14] D. Ghosh (2003) Neighborhood search heuristics for the uncapacitated facility location problem. *Eur. J. Oper. Res.*, 150, 150–162.

[15] P. Hansen, J. Brimberg, D. Urosevic & N. Mladenovic (2007) Primal-dual variable neighborhood search for the simple plant-location problem. *INFORMS J. on Comput.*, 19, 552–564.

[16] M. Körkel (1989) On the exact solution of large-scale simple plant location problems. *Eur. J. Opl Res.*, 39, 157–173.

[17] J. Krarup & P.M. Pruzan (1983) The simple plant location problem: survey and synthesis. *Eur. J. Opl Res.*, 12, 36–81.

[18] J. Kratica, D. Tosic, V. Filipovic & I. Ljubic (2001) Solving the simple plant location problem by genetic algorithm. *RAIRO Oper. Res.*, 35, 127–142.

[19] M. Labbé & F. Louveaux (1997) Location problems. In M. Dell'Amico, F. Maffioli & S. Martello (eds.) *Annotated Bibliographies in Combinatorial Optimization*, pp. 261–281. Chichester: Wiley.

[20] M. Labbé, D. Peeters & J.-F. Thisse (1995) Location on networks. In M.0. Ball, T.L. Magnanti, C.L. Monma & G.L. Nemhauser (eds.) *Network Models*, pp 551–624. Handbooks in Operations Research and Management Science, vol. 8. Amsterdam: North-Holland.

[21] A.N. Letchford & S.J. Miller (2012) Fast bounding procedures for large instances of the simple plant location problem. *Comput. & Oper. Res.*, 39, 985–990.

[22] J.G. Morris (1978) On the extent to which certain fixed-charge depot location problems can be solved by LP. *J. Oper. Res. Soc*, 29, 71–76.

[23] W.D. Pisinger, A.B. Rasmussen & R. Sandvik (2007) Solution of large quadratic knapsack problems through aggressive reduction. *Informs J. Comput.*, 19, 280–290.

[24] C. ReVelle (1993) Facility siting and integer friendly programming. *Eur. J. Oper. Res.*, 65, 147–158.

[25] L. Schrage (1975) Implicit representation of variable upper bounds in linear programming. *Math. Program. Study*, 4, 118–132.

[26] M.J. Todd (1982) An implementation of the simplex method for linear programming problems with variable upper bounds. *Math. Program.*, 23, 34–49.

[27] T.J. Van Roy & D. Erlenkotter (1982) A dual based procedure for dynamic facility location. *Mgmt. Sci.*, 28, 1091–1105.

[28] V. Verter (2011) Uncapacitated and capacitated facility location problems. In H.A. Eiselt & V. Marianov (eds) *Principles of Location Science*, pp. 25–37. Heidelberg: Springer.