

An Aggressive Reduction Scheme for the Simple Plant Location Problem

Adam N. Letchford* Sebastian J. Miller

November 2011

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, using information obtained from a sequence of fast lower- and upper-bounding procedures. The hope is 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, also known as the uncapacitated facility location problem. The scheme involves a pre-processing phase, a dual ascent phase, a Lagrangian phase, and a Linear Programming phase. Using this 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.

1 Introduction

In the *Simple Plant Location Problem* (SPLP) we have a set I of locations and a set J of clients. For any location $i \in I$, the fixed cost of opening a facility at i is f_i . For any location $i \in I$ and any client $j \in J$, the cost of serving client j from an open facility at location i is c_{ij} . The task is to decide where to open facilities, and to assign each client to exactly one open facility, such that the total cost is minimised.

The SPLP is a well-known NP -hard combinatorial optimisation problem that has received a great deal of attention. A survey of early work on the SPLP (still relevant today) is given by Krarup & Pruzan [13]. More recent surveys include Cornuéjols *et al.* [6] and Labbé & Louveaux [16]. We remark that, in some papers, the SPLP is called the *Uncapacitated Facility Location Problem* or UFLP.

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

Pisinger, Rasmussen and Sandvik [20] 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 information obtained from a sequence of fast lower- and upper-bounding procedures. The hope is that the reduced instance will then be small enough to be solved by an exact algorithm.

Pisinger *et al.* presented an aggressive reduction scheme for the quadratic knapsack problem. Here, we present an aggressive reduction scheme for the SPLP, which uses a combination of dual ascent, Lagrangian relaxation, linear programming and various primal heuristics. Using this scheme, in conjunction with the CPLEX MIP solver, we are able to quickly solve instances with a few thousand locations and clients. For larger instances, it can compute tight lower and upper bounds, and significantly reduce the size.

The structure of the paper is as follows. In the next section, we briefly review the relevant literature. In Section 3, we describe a simple pre-processing procedure, which proves to be useful when facility costs are small. In Section 4, we describe the first phase of our scheme, which is based on dual ascent. In Section 5, we describe the second phase, which is based on Lagrangian relaxation and subgradient optimisation. In Section 6, we describe the third phase, which is based on linear programming and iterated rounding. The results of some computational experiments are given in Section 7, and concluding remarks are given in Section 8.

We assume throughout the paper that the f_i and c_{ij} are positive integers.

2 Literature Review

In this section, we review the relevant literature. Since the literature on the SPLP is vast, we have been highly selective in the works that we cite.

2.1 Zero-one linear programming formulation

The accepted integer programming formulation of the SPLP is the following one due to Balinski [2]:

$$\min \quad \sum_{i \in I} f_i y_i + \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \quad (1)$$

$$\text{s.t.} \quad \sum_{i \in I} x_{ij} = 1 \quad (\forall j \in J) \quad (2)$$

$$y_i - x_{ij} \geq 0 \quad (\forall i \in I, j \in J) \quad (3)$$

$$x_{ij} \in \{0, 1\} \quad (\forall i \in I, j \in J) \quad (4)$$

$$y_i \in \{0, 1\} \quad (\forall i \in I). \quad (5)$$

This is a Zero-One Linear Program (0-1 LP). The interpretation of the variables is that x_{ij} takes the value 1 if and only if client j is assigned to a facility at location i , and y_i takes the value 1 if and only if a facility is

opened at location j . The constraints (2) and (3) will be called *assignment constraints* and *variable upper bounds*, respectively.

The LP relaxation of this 0-1 LP is obtained by simply replacing the binary conditions (4) and (5) with non-negativity. (There is no need to impose upper bounds of 1 on the variables, since these will be satisfied automatically.) A key feature of the SPLP, is that the LP relaxation typically gives a very good lower bound, and is often even integral [1, 7, 19, 21]. On the other hand, the presence of the variable upper bounds makes the LP highly degenerate, in both the primal and the dual. Specialised methods have been devised to cope with variable upper bounds [22, 24], but the most successful approaches to the SPLP have been based on fast heuristics for the dual, as explained in the following subsection.

2.2 Dual ascent and dual adjustment

In their seminal paper, Bilde & Krarup [5] devised an effective lower-bounding procedure, called *dual ascent*, for the SPLP. The starting point is that the dual of the LP relaxation takes the following form:

$$\max \quad \sum_{j \in J} v_j \quad (6)$$

$$\text{s.t.} \quad \sum_{j \in J} w_{ij} \leq f_i \quad (\forall i \in I) \quad (7)$$

$$v_j - w_{ij} \leq c_{ij} \quad (\forall i \in I, j \in J) \quad (8)$$

$$v_j \geq 0 \quad (\forall j \in J) \quad (9)$$

$$w_{ij} \geq 0 \quad (\forall i \in I, j \in J). \quad (10)$$

Here, the v_j and w_{ij} are the dual variables for the assignment constraints and variable upper bounds, respectively. Then, Bilde and Krarup observed that there always exists an optimal solution to the dual in which $w_{ij} = \max\{0, v_j - c_{ij}\}$ for all i and j . This leads to the following so-called *condensed* dual:

$$\max \quad \sum_{j \in J} v_j \quad (11)$$

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

Dual ascent, then, is a fast and simple heuristic for finding good feasible solutions to the condensed dual. The 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. To keep the number of iterations polynomially bounded, each dual variable v_j is restricted to take a value that is equal to c_{ij} for some $i \in I$, except in the final iteration.

In our own paper [17], we showed that dual ascent runs in $\mathcal{O}(m^2n)$ time, where m is the number of locations and n the number of clients. We described an improved version, which is faster in practice but has the same

worst-case running time. We also devised a modified dual ascent heuristic which runs in only $\mathcal{O}(mn \log m)$ time, yet still produces reasonably good lower bounds.

Erlenkotter [8] proposed an iterative method, called ‘dual adjustment’, for improving the dual solution generated by dual ascent. Several enhancements were also proposed by Körkel [12]. More recently, Hansen *et al.* [11] presented an effective variable neighborhood search (VNS) heuristic. For the sake of brevity, we do not go into details.

2.3 Lagrangian relaxation

Beasley [4] proposed to compute lower bounds using Lagrangian relaxation. 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’:

$$r_i = f_i + \sum_{j \in J} \min\{0, c_{ij} - \lambda_j\}, \quad (13)$$

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

$$\sum_{i \in I} \min\{0, r_i\} + \sum_{j \in J} \lambda_j. \quad (14)$$

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

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

Beasley uses a standard subgradient method to solve this problem approximately.

2.4 Primal heuristics

Now we turn our attention to primal heuristics (i.e., heuristics for producing feasible integral solutions, and therefore upper bounds).

The early primal heuristics were all simple greedy heuristics, based on iteratively either adding or dropping facilities (e.g., [9, 15, 18]). A few meta-heuristics have also been proposed (e.g., [14, 23]). We, however, are more interested in heuristics which attempt to exploit the dual solutions found by either dual ascent or dual adjustment.

Given a feasible solution $\bar{v} \in \mathbb{Z}_+^n$ to the condensed dual, observe that the quantity

$$s_i = f_i - \sum_{j \in J} \max\{0, \bar{v}_j - c_{ij}\} \quad (15)$$

can be viewed as an estimate of the reduced cost of the variable y_i in the primal. Therefore, in [5, 8] it was suggested to open temporarily all facilities for which $s_i = 0$, assign each client to the closest open facility, and then close any open facility that does not have any client assigned to it.

In [12, 25], it was proposed to run the heuristic of [5, 8] after each iteration of dual adjustment. This significantly improves the upper bounds, at the cost of a slight increase in running time. More recently, Hansen *et al.* [11] proposed an effective primal VNS heuristic, designed to work in conjunction with their dual one (mentioned at the end of Subsection 2.2).

Beasley [4] proposes a natural Lagrangian primal heuristic: each time the multipliers are updated, all facilities for which r_i is negative are opened temporarily and one then precedes as in [5, 8]. The resulting bounds are of similar quality to those of [12, 25].

In our paper [17], we proposed a ‘drop’ heuristic, in which all facilities are initially opened, and then facilities are examined in non-increasing order of s_i , being closed whenever this causes the overall cost to decrease. This heuristic, which can be implemented to run in $\mathcal{O}(m(n + \log m))$ time, also gives rather good results.

2.5 Problem reduction

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

Körkel [12] showed how to apply problem reduction to the SPLP, within a dual ascent or dual adjustment context. As in the previous subsections, let $\bar{v} \in \mathbb{R}^n$ denote a feasible solution to the condensed dual, let $\text{LB} = \sum_{i \in I} \bar{v}_i$ denote the corresponding lower bound, let UB^* denote the best upper bound found so far, and let s_i for $i \in I$ be as in equation (15). Then for any $i \in I$ such that $s_i > \text{UB}^* - \text{LB}$, 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 $s_i + \max\{0, c_{ij} - v_j\} > \text{UB}^* - \text{LB}$, the variable x_{ij} can be permanently fixed to 0.

Beasley [4] gave a slightly different problem reduction procedure, for use in a Lagrangian context. It uses the Lagrangian reduced costs r_i given in equation (13). Namely, if r_i is positive and $\text{LB} + r_i > \text{UB}^*$ for any i , then y_i can be permanently fixed to 0 and, if r_i is negative and $\text{LB} - r_i > \text{UB}^*$ for any i , then y_i can be permanently fixed to 1.

In our experiments, we have found that the most useful kind of problem reduction in the case of the SPLP is to eliminate x variables, by permanently

fixing them to 0. We will show in Section 5 how to eliminate x variables within the Lagrangian context.

3 A Simple Pre-Processing Procedure

In this section, we describe a pre-processing procedure which, despite being very simple, turns out to be very useful under certain conditions. The procedure is based on the following lemma:

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

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

We 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 we 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). \square

One can perform the elimination described in this lemma easily in $\mathcal{O}(mn)$ time, as follows:

```

For  $j = 1, \dots, n$  do:
  Set  $\Delta_j := +\infty$ 
  For  $i = 1, \dots, m$  do:
    If  $(c_{ij} + f_i < \Delta_j)$ 
      Set  $\Delta_j := c_{ij} + f_i$ .
  For  $i = 1, \dots, m$  do:
    If  $(c_{ij} \geq \Delta_j)$ 
      Eliminate  $x_{ij}$ .

```

We will show in Section 7 that this pre-processing procedure performs remarkably well when the facility costs are small relative to the costs of assigning clients to facilities. A partial explanation is as follows. Suppose all locations are randomly distributed points in the unit square, and that the average cost of a facility is \bar{f} . If m and n are very large, then Δ_j will be approximately equal to \bar{f} for all $j \in J$. Then, for any given j , one can expect to eliminate the variable x_{ij} for all locations i lying outside a circle of radius \bar{f} centered on client j . So, if \bar{f} is significantly less than $\sqrt{2}$, we can expect to eliminate a large proportion of the x variables.

Remark: The above pre-processing procedure, along with all of our other

routines, outputs the reduced instance in a ‘compact’ form. Specifically, instead of storing the c_{ij} values in an $m \times n$ matrix, we store the following for each client: (i) the ‘degree’ of the client, which is the number of variables x_{ij} which have not yet been eliminated, (ii) a list of the indices in I associated with those variables, and (iii) a list of the associated c_{ij} values.

4 The Dual Ascent Phase

After the pre-processing step, we move into the first main phase of our reduction scheme, which is based on dual ascent. We use the so-called ‘enhanced’ dual ascent procedure described in [17]. This procedure can be summarised as follows:

Read in m and n , the facility costs, the client degrees, and the two lists for each client, and store them in memory.

Declare an array v of dimension n , to store the values of the dual values v_j .

Declare an array s of dimension m , to store the slacks of the constraints (12).

For each client j , do:

Sort the c_{ij} values in non-decreasing order.

Let c_j^k denote the k th value in the sorted list.

Let $d(j)$ be the degree of client j .

Set $c_j^{d(j)+1}$ to infinity (or a suitable large number).

Declare an array k of dimension n , with the following interpretation: if $k(j) = t$, where $t \in \{1, \dots, d(j)\}$, it means that $c_j^t \leq v_j \leq c_j^{t+1}$.

Using binary search, compute the largest integer t such that v_j can be set to c_j^t for all j while still satisfying the constraints (12). Let t^* denote this value.

For each client j , do:

Set $v_j := c_j^{t^*}$ and $k(j) := t^*$.

For each location i , do:

Set $s_i := f_i - \sum_{j \in J} \max\{0, v_j - c_{ij}\}$.

Label all clients ‘unblocked’.

Repeat:

For each unblocked client j , do:

Let Δ_j be the minimum of s_i over all i for which $v_j \geq c_{ij}$.

If $\Delta_j = 0$, client j is blocked.

If $v_j + \Delta_j$ is greater than $c_j^{k(j)+1}$

Set $\Delta_j := c_j^{k(j)+1} - v_j$.

Increment $k(j)$ by one.

If $\Delta_j > 0$

Decrease s_i by Δ_j for all locations i such that $v_j \geq c_{ij}$.

Increase v_j by Δ_j .

Until all clients are blocked.

It is shown in [17] that this ascent procedure runs in $\mathcal{O}(m^2n)$ time. Although

this is rather high, it performs very quickly in practice. In particular, it typically takes about half of the time of the original procedure of Bilde & Krarup [5].

When the ascent procedure ends, we store the final dual solution v , along with the corresponding lower bound given by (11). Then, to compute upper bounds, we use the so-called ‘multi-drop’ scheme that we presented in [17]. This involves calling the modified drop heuristic, described at the end of Subsection 2.4, before each iteration of the ‘repeat’ loop in the enhanced ascent procedure. The best upper bound found is stored.

Now that we have a dual solution, a lower bound and an upper bound, we can eliminate variables, following the procedure of Körkel [12] described in Subsection 2.5. We decided, however, to eliminate only x variables, and even then, to fix them only to zero rather than one. Restricting the reduction in this way means that the reduced instance is in the same format as the instance received from the pre-processing phase, which makes things easier to implement. In any case, this kind of reduction turned out to be the most important by far.

5 The Lagrangian Relaxation Phase

The second main phase of our reduction scheme is based on Lagrangian relaxation. We use a modified version of the procedure of Beasley [4]. Before presenting it, we need to define the following notation:

- LB and UB are the current lower and upper bounds,
- LB* and UB* are the best lower and upper bounds found so far,
- N is the number of subgradient iterations since LB* last increased,
- ℓ is the current step length,
- for each $j \in J$, G_j is the amount by which the j th assignment constraint (2) is violated by the solution to the current relaxed problem.

Also, recall that λ_j is the value of the j th Lagrangian multiplier, and r_i is the ‘Lagrangian reduced cost’ of facility i . The value of r_i is defined by equation (13), but it should be borne in mind that many of the x variables will have been eliminated in previous phases, and therefore should be ignored.

Our procedure can then be described as follows:

Read in the reduced instance obtained at the end of the ascent phase.

Set UB* to the best upper bound found in the ascent phase, N to 0, and ℓ to 1.

Set LB* to the lower bound obtained at the end of the ascent phase.

For each client j :

Set λ_j to the value that the dual variable v_j had at the end of the ascent phase.

Repeat:

For all facilities i :

 Compute r_i according to (13).

Compute LB according to (14).

If $LB > LB^* + 10$, set N to 0 and LB^* to LB. Else set N to N+1.

Compute UB by running a modified drop heuristic (see below).

If $UB < UB^*$, set UB^* to UB.

If $N = 30$, set ℓ to half its value and reset N to 0.

If $UB^* = LB^*$, stop.

For each $j \in J$:

 Let $G_j = 1 - |\{i \in I : r_i < 0, c_{ij} < \lambda_j\}|$.

If all of the G_j are zero, or $\ell \leq 10^{-5}$, stop.

For all $j \in J$:

 Increase λ_j by $\ell(1.05UB^* - LB)G_j / (\sum_{j \in J} G_j^2)$.

The major differences between this procedure and that of Beasley are as follows:

- Beasley initialises the Lagrangian multipliers in a different way, setting λ_j to the minimum of c_{ij} over all i . We found that our initialisation causes the bounds LB and UB to converge to near-optimal values more quickly.
- Beasley sets the initial and final values of ℓ to 2 and 0.005, respectively. Since our initial Lagrangian multipliers are better, and our instances are larger, it turned out to be more effective to set them to lower values.
- Whereas Beasley uses the heuristic described in Subsection 2.4 to compute the upper bound UB, we use a modified drop heuristic. This is similar to the drop heuristic described in the last paragraph of Subsection 2.4, except that facilities are examined in non-increasing order of r_i , rather than non-increasing order of s_i . In our experience, this tends to give better solutions.
- Beasley resets N to zero if $LB > LB^*$. For instances of the size considered in this paper, it is preferable to reset N only if $LB > LB^* + 10$. This reduces the running time substantially, while having a negligible effect on bound quality.

At the end of the Lagrangian procedure, we have new (typically substantially improved) lower and upper bounds LB^* and UB^* . It then proves very useful to perform a new round of eliminating x variables. To do this, we use the following proposition:

Proposition 1 *Let $\lambda \in \mathbb{R}^n$ be a vector of Lagrangian multipliers, let $r \in \mathbb{R}^m$ be the corresponding vector of Lagrangian reduced costs, let LB be the corresponding lower bound, and let UB^* be the best upper bound found so far. Let x_{ij} be a variable that has not yet been eliminated, and suppose that:*

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

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

Proof. We consider three cases. First, suppose that $c_{ij} - \lambda_j > 0$ and $r_i > 0$. Then, both x_{ij} and y_i take the value 0 in all optimal solutions to the relaxed problem. If we force x_{ij} to take the value 1 instead, this will also force y_i to take the value 1. Thus, the lower bound will increase by $c_{ij} - \lambda_j + r_i$.

Second, suppose that $c_{ij} - \lambda_j < 0$ and $r_i > 0$. Then, y_i takes the value 0 in all optimal solutions to the relaxed problem, which forces x_{ij} to take the value 0 as well. As before, if we force x_{ij} to take the value 1, this will force y_i to take the value 1 as well. The effect, however, will be to cause the lower bound to increase by r_i .

Third, suppose that $c_{ij} - \lambda_j > 0$ and $r_i < 0$. Then, x_{ij} takes the value 0 and y_i takes the value 1 in all optimal solutions to the relaxed problem. If we force x_{ij} to take the value 1 instead, the lower bound will increase by $c_{ij} - \lambda_j$.

In all three cases, if the change in the solution causes the lower bound to exceed UB^* , then we know all feasible solutions to the SPLP satisfying $x_{ij} = 1$ cost more than UB^* . \square

We found that this reduction rule is very powerful on many instances.

6 The Linear Programming Phase

The third and final main phase of our reduction scheme is based on Linear Programming.

The LP relaxation of the reduced SPLP instance is obtained by taking the formulation (1)–(5) in Subsection 2.1, replacing the binary conditions (4) and (5) with the trivial bounds $x \in [0, 1]^{m \times n}$ and $y \in [0, 1]^m$, and omitting the x variables which have been eliminated in previous phases. From LP duality, the lower bound obtained by solving this LP cannot be worse than that obtained by dual ascent. Moreover, it cannot be worse than that obtained with Lagrangian relaxation, since the Lagrangian subproblem clearly possesses the integrality property (see Geoffrion [10]).

We have found that, in practice, the dual simplex method solves the LP relaxation more quickly than the primal simplex method.

As usual, we let LB^* and UB^* denote the best bounds found so far. We begin by running the following ‘iterated rounding’ heuristic, which attempts

to exploit the tightness of the LP to improve both bounds:

Read in the reduced instance obtained at the end of the previous phase.

Read in the current best upper bound UB^* .

Solve the LP relaxation via the dual simplex method.

Let LB^* be the lower bound obtained.

Repeat:

Let (x^*, y^*) be the current solution to the LP.

Let $F := \{i \in I : 0 < y_i^* < 1\}$ and $k := \arg \max\{y_i^* : i \in F\}$.

Replace the bounds $0 \leq y_k \leq 1$ in the LP with the equation $y_k = 1$.

Re-optimize the LP via dual simplex.

Until $y_i \in \{0, 1\}$ for all $i \in I$.

Let UB be the cost of the final LP solution.

If $UB < UB^*$, set UB^* to UB .

Output LB^* and UB^* .

We have found that this procedure usually leads to significant further improvements to both bounds. It is therefore worthwhile performing another round of eliminating x variables. To do this, we use the following proposition:

Proposition 2 *Let LB^* be the lower bound obtained by solving the LP relaxation, and let π and ρ be the corresponding reduced-cost vectors for the x and y variables, respectively. Let x_{ij} be a variable that has not yet been eliminated, and suppose that:*

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

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

Proof. In order to provide reduced-cost vectors, the LP solver must have found an optimal solution to the dual LP (6)–(10). From LP duality, the reduced cost π_{ij} will be equal to the slack of the corresponding constraint (9), and the reduced cost ρ_i will be equal to the slack of the corresponding constraint (8).

We now 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 . The effect of this change will be to decrease the slack of the corresponding constraint (9) to zero, and increase the slack of the corresponding constraint (8) by ρ_i . Accordingly, the reduced cost ρ_i will drop to zero and the reduced cost π_{ij} will increase by ρ_i . We can then proceed as in the first case. \square

The computational results given in the next section demonstrate that this reduction procedure is also very useful for many instances.

Once all of our reduction procedures have been applied, we have a lower bound, an upper bound, and a reduced instance. If the reduced instance is small enough, one can simply pass the reduced 0-1 LP formulation to a commercial mathematical programming package, and solve it by LP-based branch-and-bound. If the package permits it, one can also pass it the best upper bound found so far, which may lead to a reduction in the number of branch-and-bound nodes.

7 Computational Experiments

In this section we report the results of some computational experiments. Our reduction 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. The code was run on a 2.33 GHz PC with 3.25 Gb of RAM, operating under Windows XP.

In our aggressive reduction scheme, the standard order of running our routines is as follows: pre-processing, enhanced ascent with modified drop, Lagrangian relaxation with modified drop, LP relaxation and iterated round-ing, branch-and-bound. It turns out, however, that it is not always best to run all of the routines. In particular, the best choice of routines seems to depend on the size of the facility costs. We explain this in the following subsections.

7.1 Test instances

Some small SPLP instances can be found in the OR Library [3]. Almost all of them are trivial by modern standards, but there are three instances $m = 100$ and $n = 1000$ that yield meaningful results. In order to create larger instances, we followed the scheme used in [1, 11]. This involves setting m equal to n , setting each facility and customer location to a random point in the unit square, setting assignment costs equal 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 [11], 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 $\sqrt{n}/100$.
- Large and constant: all facility costs set to $\sqrt{n}/10$.

$m = n$	lower bounds			upper bounds		
	Enh	LR	LP	Enh	LR	LP
a	0.37	0.004	0.000	0.00	0.000	0.000
b	1.13	0.006	0.000	0.47	0.000	0.000
c	1.11	0.059	0.048	0.03	0.033	0.033

Table 1: Percentage gaps of lower and upper bounds for OR-Lib instances

$m = n$	Variable Elimination %			Time (s)			
	Enh	LR	LP	Enh	LR	LP	B & B
a	93.08	98.60	99.00	0.078	0.765	0.187	—
b	42.13	98.37	99.00	0.047	1.249	0.203	—
c	52.96	91.09	92.37	0.032	1.546	0.468	2.390

Table 2: Cumulative elimination and running times for OR-Lib instances

- Varied: facility costs uniformly distributed between $\sqrt{n}/1000$ and $\sqrt{n}/10$.

Since the ascent procedures require costs to be integers, all costs were then multiplied by 5000 and rounded down to the nearest integer.

7.2 The OR-Lib instances

We began by testing the algorithm on the three largest instances from the OR Library [3], which all have $m = 100$ and $n = 1000$. Table 1 displays, for each instance, the percentage gap between the optimum and both bounds (lower and upper) at the end of each phase of our scheme. The column headings ‘Enh’, ‘LR’, ‘LP’ refer to enhanced ascent, Lagrangian relaxation, and Linear Programming (with iterated rounding), respectively. (No pre-processing was possible for these instances.)

It can be seen that the LR and LP phases produce remarkably tight lower and upper bounds. The enhanced ascent procedure also yields tight upper bounds, but weaker lower bounds. Note also that the first two instances are already solved by the end of phase 3.

Table 2 shows, for each instance, the cumulative percentage of the variables that have been eliminated and the computing time for each phase. The column heading ‘B & B’ refers to branch-and-bound. It can be seen that the reduction procedures eliminate almost all of the variables, in just seconds or fractions of a second. Moreover, after they have been applied, the third instance is solved by branch-and-bound in under 3 seconds.

$m = n$	Eliminated %		Gap %		Time (s)		
	PrePro	LP	LB	UB	PrePro	LP	B & B
1000	99.53	99.86	0.000	0.000	0.025	0.091	0.094
2000	99.43	99.91	0.004	0.002	0.094	0.463	0.177
3000	98.77	99.88	0.009	0.001	0.209	3.244	0.650
4000	98.50	99.84	0.010	0.003	0.375	8.731	1.737
5000	98.21	99.74	0.012	0.005	0.609	19.405	6.147
6000	97.94	99.74	0.009	0.006	0.922	39.713	10.984
7000	97.66	99.52	0.012	0.010	1.283	77.280	44.191
8000	97.38	99.73	0.009	0.004	1.891	110.480	37.076

Table 3: Results for instances with small and constant facility costs

7.3 Instances with small and constant facility costs

Now we turn our attention to the random instances whose facility costs were set to $\sqrt{n}/1000$. For these instances, it proved most effective to use a 3-step approach, including only the pre-processing, LP and branch-and-bound phases.

Table 3 shows, for each problem size, the % of variables eliminated after the pre-processing and LP phases, the percentage gap between the optimum and both bounds (lower and upper) after the LP phase, and the times for each of the 3 procedures.

For these instances, pre-processing is extremely effective, eliminating the majority of the variables in a matter of seconds. The LP phase takes a bit longer, but produces such tight bounds that it is possible to eliminate the majority of the remaining variables. This then renders the instances quite easy to solve via branch-and-bound.

We remark that the previous best algorithm, due to Hansen *et al.* [11], runs into difficulties when n reaches 7000 or so. We also remark that the only thing preventing us from solving larger instances in this class was the memory limit of the LP solver.

7.4 Instances with medium and constant facility costs

Next, we consider the instances whose facility costs were set to $\sqrt{n}/100$. These instances are harder to solve, and it proved effective to use a 4-step approach, using pre-processing, LR, LP and branch-and-bound.

Table 4 shows, for each problem size, the percentage gap between the optimum and both bounds (lower and upper) after the LR and LP procedures. We see that both procedures yield remarkably tight bounds.

Table 5 shows, for each problem set, the % of variables eliminated after the Pre-Pro, LR and LP phases, and the times for each of the 4 procedures.

$m = n$	lower bounds		upper bounds	
	LR	LP	LR	LP
500	0.004	0.004	0.001	0.000
1000	0.013	0.012	0.007	0.005
1500	0.013	0.012	0.006	0.004
2000	0.008	0.007	0.006	0.004
2500	0.015	0.011	0.005	0.001
3000	0.019	0.011	0.008	0.001
3500	0.024	0.017	0.037	0.012

Table 4: Percentage gaps of lower and upper bounds, to two significant figures, for instances with medium and constant facility costs

$m = n$	Variable Elimination %			Time (s)			
	Pre	LR	LP	Pre	LR	LP	B & B
500	84.54	99.66	99.70	0.009	3.116	0.133	0.117
1000	74.56	99.55	99.64	0.025	18.566	0.219	0.895
1500	65.64	99.54	99.64	0.053	52.785	0.469	2.223
2000	57.39	99.61	99.70	0.097	111.285	1.059	3.465
2500	50.20	99.36	99.59	0.163	223.108	6.094	26.225
3000	43.60	98.96	99.45	0.219	349.736	32.697	108.462
3500	38.13	95.21	98.48	0.284	481.828	1057.262	874.224

Table 5: Further results for instances with medium and constant facility costs

It can be seen that pre-processing does not work so well here, as might be expected, but the LR and LP phases still eliminate a substantial proportion of the variables. Running times are however much longer in this case than in the previous cases.

We remark that, for instances in this class, Hansen *et al.* [11] solved only instances with $n \leq 3000$. Again, it was the memory limit of the LP solver that prevented us from solving still larger instances.

7.5 Random instances with large and constant facility costs

Next, we consider the instances whose facility costs were set to $\sqrt{n}/10$. These instances are still harder, since no variables can be eliminated via pre-processing. We therefore decided to call the enhanced ascent procedure before invoking LR, LP and branch-and-bound.

Table 6 shows, for each problem set, the percentage gap between the optimum and both bounds (lower and upper) after the enhanced ascent, LR

$m = n$	lower bounds			upper bounds		
	Enh	LR	LP	Enh	LR	LP
500	1.01	0.002	0.001	0.88	0.000	0.000
1000	0.95	0.019	0.009	1.03	0.004	0.000
1500	0.99	0.026	0.015	1.16	0.026	0.000

Table 6: Percentage gaps of lower and upper bounds for instances with large and constant facility costs

$m = n$	Variable Elimination %			Time (s)			
	Enh	LR	LP	Enh	LR	LP	B & B
500	5.04	99.72	99.74	0.250	5.502	0.032	0.047
1000	0.26	99.05	99.36	1.388	37.052	2.682	37.446
1500	0.00	96.75	98.35	4.225	108.251	32.546	72.978

Table 7: Results for instances with large and constant facility costs

and LP phases. Once again, the LR and LP phases yield excellent bounds, whereas enhanced ascent yields slightly poorer ones.

Table 7 shows, for each problem set, the % of variables eliminated after each of the three reduction phases, and the times for each of the four procedures. The enhanced ascent phase does not eliminate a significant number of variables, but we kept it in since it provides good starting multipliers for the LR phase. The LR phase itself is time-consuming, but this is worth it, since the reduction is substantial.

We remark that, for instances in this class, Hansen *et al.* [11] solved only instances with $n \leq 1400$. Again, it was the memory limit of the LP solver that prevented us from solving still larger instances.

7.6 Random instances with varied facility costs

In these instances all facility costs were uniformly distributed between $\sqrt{n}/1000$ and $\sqrt{n}/10$. These instances are relatively easy to solve. One can solve instances with $m = n \leq 7000$ using only pre-processing, enhanced ascent, LP, and branch-and-bound. For larger values of m and n , it is helpful to use LR, to ease the memory burden on the LP solver.

Table 8 shows, for each problem set, the percentage gap between the optimum and both bounds after the enhanced ascent, LR and LP phases. As before, the LR and LP phases yield excellent bounds, whereas enhanced ascent yields slightly poorer ones.

Table 9 shows, for each problem set, the % of variables eliminated after

$m = n$	lower bounds			upper bounds		
	Enh	LR	LP & IR	Enh	LR	LP & IR
1000	0.59	-	0.011	0.07	-	0.000
2000	0.71	-	0.001	0.12	-	0.000
3000	0.76	-	0.000	0.10	-	0.001
4000	0.81	-	0.001	0.11	-	0.002
5000	0.81	-	0.001	0.14	-	0.000
6000	0.86	-	0.002	0.11	-	0.000
7000	0.90	-	0.001	0.12	-	0.004
8000	0.83	0.009	0.001	0.10	0.000	0.000
9000	0.86	0.011	0.002	0.11	0.000	0.000
10000	0.88	0.013	0.003	0.14	0.000	0.000
11000	0.91	0.014	0.003	0.14	0.000	0.000
12000	0.98	0.014	0.004	0.10	0.001	0.000
13000	0.97	0.009	0.000	0.14	0.001	0.000
14000	1.03	0.013	0.003	0.17	0.000	0.000
15000	0.95	0.013	0.004	0.19	0.002	0.001
16000	0.89	0.011	0.001	0.15	0.000	0.000

Table 8: Percentage gaps of lower and upper bounds, to two significant figures, for instances with varied facility costs

the Pre-Processing, Enhanced Ascent, Pre-Pro, LR and LP phases, and the times for each of the 5 procedures.

For these instances, all four reduction phases are extremely effective. For the smaller instances, the bulk of the time is spent in the ascent phase. For the larger instances, it is spent in the LR phase.

We remark that, for instances in this class, Hansen *et al.* [11] solved only instances with $n \leq 15000$. In this case, it was the memory limit of Microsoft Visual Studio that prevented us from solving still larger instances.

8 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 the simple plant location problem. Using our four-phase approach, in conjunction with the CPLEX MIP solver, we are able to solve to proven optimality larger instances than any previously solved in the literature. In particular, when facility costs are small relative to assignment costs, one can easily solve instances with many thousands of locations and clients.

As mentioned in the previous section, the only thing that prevented us

$m = n$	Variable Elimination %				Time (s)				
	Pre	Enh	LR	LP	Pre	Enh	LR	LP	B & B
1000	89.06	98.01	–	99.85	0.019	0.784	–	0.347	0.078
2000	93.74	98.50	–	99.93	0.094	4.875	–	1.191	0.094
3000	87.62	96.76	–	99.95	0.234	12.281	–	7.875	0.266
4000	91.66	97.63	–	99.96	0.393	22.624	–	12.212	0.266
5000	91.18	97.30	–	99.97	0.628	42.760	–	24.568	0.664
6000	91.20	97.25	–	99.97	0.956	64.432	–	38.933	0.526
7000	95.40	98.39	–	99.96	1.319	97.202	–	42.020	2.672
8000	90.53	96.90	99.94	99.98	1.784	134.330	589.487	1.219	0.328
9000	91.59	97.22	99.92	99.98	2.481	185.071	770.401	1.738	0.802
10000	91.19	96.83	99.92	99.98	3.088	222.231	1092.675	2.500	1.907
11000	90.40	96.20	99.88	99.98	3.867	280.712	1166.571	5.136	1.469
12000	90.51	96.30	99.86	99.98	4.900	376.255	1489.783	8.459	2.297
13000	87.97	95.08	99.89	99.99	5.582	435.144	1729.484	6.741	0.461
14000	90.04	95.49	99.92	99.98	7.512	515.540	2076.677	6.106	1.281
15000	88.76	95.05	99.81	99.98	9.267	589.493	2426.238	20.453	4.594
16000	85.77	94.17	99.91	99.99	12.556	786.059	2381.691	9.631	1.104

Table 9: Results for instances with varied facility costs

from solving still larger instances was limited computer memory. To address this, one could perhaps investigate the possibility of some kind of column and/or row generation scheme, in which one starts with a small subset of the variables and/or constraints, and then generates others only as and when needed. Note however that column and row generation would probably have to be implemented in different ways at different phases of the scheme, since dual ascent, Lagrangian relaxation and Linear Programming are quite distinct kinds of procedure.

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] M. Balinski (1965) Integer programming: methods, uses, computation. *Mgmt. Sci.*, 12, 254–313.
- [3] J.E. Beasley (1990) OR-Library: distributing test problems by electronic mail. *J. Opl Res. Soc.*, 41, 1069–1072.

- [4] J.E. Beasley (1993) Lagrangian heuristics for location problems. *Eur. J. Opl Res.*, 65, 383–399.
- [5] O. Bilde & J. Krarup (1977) Sharp lower bounds and efficient algorithms for the simple plant location problem. *Ann. Discr. Math.*, 1, 79–88.
- [6] 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.
- [7] M.A. Efroymsen & T.L. Ray (1966) A branch-and-bound algorithm for plant location. *Oper. Res.*, 14, 361–368.
- [8] D. Erlenkotter (1978) A dual-based procedure for uncapacitated facility location. *Oper. Res.*, 26, 992–1009.
- [9] E. Feldman, F.A. Lehrer & T.L. Ray (1966) Warehouse location under continuous economies of scale. *Mgmt. Sci.*, 12, 670–684.
- [10] A.M. Geoffrion (1974) Lagrangean relaxation for integer programming. *Math. Program. Study*, 2, 82–114.
- [11] 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.
- [12] M. Körkel (1989) On the exact solution of large-scale simple plant location problems. *Eur. J. Opl Res.*, 39, 157–173.
- [13] J. Krarup & P.M. Pruzan (1983) The simple plant location problem: survey and synthesis. *Eur. J. Opl Res.*, 12, 36–81.
- [14] J. Kratica, D. Tomic, V. Filipovic & I. Ljubic (2001) Solving the simple plant location problem by genetic algorithm. *RAIRO Oper. Res.*, 35, 127–142.
- [15] A.A. Kuehn & M.J. Hamburger (1963) A heuristic program for locating warehouses. *Mgmt. Sci.*, 9, 643–666.
- [16] 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.
- [17] 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.

- [18] A.S. Manne (1964) Plant location under economies-of-scale — decentralization and computation. *Mgmt. Sci.*, 11, 213–235.
- [19] 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.
- [20] W.D. Pisinger, A.B. Rasmussen & R. Sandvik (2007) Solution of large quadratic knapsack problems through aggressive reduction. *Informs J. Comput.*, 19, 280–290.
- [21] C. ReVelle (1993) Facility siting and integer friendly programming. *Eur. J. Oper. Res.*, 65, 147–158.
- [22] L. Schrage (1975) Implicit representation of variable upper bounds in linear programming. *Math. Program. Study*, 4, 118–132.
- [23] M. Sun (2006) Solving the uncapacitated facility location problem using tabu search. *Comput. Oper. Res.*, 33, 2563–2589.
- [24] M.J. Todd (1982) An implementation of the simplex method for linear programming problems with variable upper bounds. *Math. Program.*, 23, 34–49.
- [25] T.J. Van Roy & D. Erlenkotter (1982) A dual based procedure for dynamic facility location. *Mgmt. Sci.*, 28, 1091–1105.