Computers & Operations Research 39 (2012) 985–990

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Fast bounding procedures for large instances of the Simple Plant Location Problem

Adam N. Letchford *, Sebastian J. Miller

Department of Management Science, Lancaster University, Lancaster LA1 4YX, United Kingdom

ARTICLE INFO

Available online 29 June 2011

Keywords:
Combinatorial optimisation
Facility location
Dual ascent

ABSTRACT

Some new, simple and extremely fast bounding procedures are presented for large-scale instances of the Simple Plant Location Problem. The lower-bounding procedures are based on dual ascent. The fastest of them runs in $\mathcal{O}(mn \log m)$ time, where m and n are the number of locations and clients, respectively. The upper-bounding procedures are based on iteratively dropping facilities, and the fastest of them runs in $\mathcal{O}(m(n+\log m))$ time. Extensive computational results show that, in practice, the procedures give very good bounds extremely quickly.

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 and Pruzan [20]. More recent surveys include Cornuéjols et al. [10] and Labbé and Louveaux [23]. We remark that, in some papers, the SPLP is called the *Uncapacitated Facility Location Problem* or UFLP.

The SPLP is normally formulated as the following Zero–One Linear Program (0–1 LP):

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

Here, x_{ij} is a binary variable, taking the value 1 if and only if client j is assigned to a facility at location i, and y_i is a binary variable, taking the value 1 if and only if a facility is opened at location j.

A key feature of this 0–1 LP is that its LP relaxation is typically quite tight. Moreover, the dual of the LP relaxation can be solved to near-optimality very quickly using dual ascent [5], dual adjustment [11] or Lagrangian relaxation [4]. Even today, these dual-based procedures remain the methods of choice.

Now, let m denote the number of locations and n the number of clients. It is not hard to show (see Section 3.1) that the dual ascent method runs in $\mathcal{O}(m^2 n)$ time. This is of course polynomially bounded, but it can be excessively high when m or n is large. In this paper, we show how a simple modification to the algorithm can significantly improve its speed in practice. We also present an alternative dual ascent algorithm, which runs in $\mathcal{O}(mn \log m)$ time, yet produces bounds of a similar quality.

In addition, we present a new upper-bounding procedure – based on iteratively dropping facilities in non-increasing order of reduced costs – that runs in $\mathcal{O}(m(n+\log m))$ time. Because it is so fast, we can safely call it several times, and keep the best upper bound generated. We ensure, however, that it is called no more than $\mathcal{O}(\log m)$ times, to keep the running time small.

We remark that there exist some other effective lower- and upper-bounding procedures for the SPLP, which we review in Section 2. The emphasis in this paper is on procedures that are extremely fast (in both theory and practice), conceptually simple, and easy to implement.

The structure of the paper is as follows. In Section 2, the relevant literature is reviewed. In Section 3, the running time of the classical dual ascent procedure is analysed, and two modified versions are presented. In Section 4, the new upper-bounding procedure is presented, along with an analysis of its running time. In Section 5, some encouraging computational results are presented. Finally, concluding remarks are given in Section 6.

2. Literature review

We now review the literature. We cover lower bounds in Section 2.1, heuristics in Section 2.2, and exact methods in Section 2.3.

* Corresponding author.
E-mail addresses: a.n.letchford@lancaster.ac.uk (A.N. Letchford), s.miller2@lancaster.ac.uk (S.J. Miller).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.06.014

2.1. Lower bounds

After some simplification, the LP relaxation of the above 0–1 LP can be written in the following form:

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

The lower bound from this relaxation is typically very tight (e.g., Ahn et al. [1], Morris [27], Mladenovic et al. [26]), but solving the LP exactly can be time-consuming. Specialised solution methods have been developed (e.g., Todd [31], Conn and Cornuéjols [9]), but the most successful lower-bounding procedures for the SPLP are based on solving the dual heuristically.

The dual takes the following form:

$$\begin{aligned}\max\quad & \sum_{j\in J} v_j\\ \text{s.t.}\quad & \sum_{j\in J} w_{ij} \le f_i \quad (\forall i\in I)\\ & v_j - w_{ij} \le c_{ij} \quad (\forall i\in I, j\in J)\\ & v_j \ge 0 \quad (\forall j\in J)\\ & w_{ij}\ge 0 \quad (\forall i\in I, j\in J).\end{aligned}$$

Bilde and Krarup [5] and Erlenkotter [11] independently 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:

$$\begin{aligned}\max\quad & \sum_{j\in J} v_j\\ \text{s.t.}\quad & \sum_{j\in J} \max\{0, v_j - c_{ij}\} \le f_i \quad (\forall i\in I).\end{aligned} \tag{1}$$

Bilde and Krarup [5] and Erlenkotter [11] independently proposed a 'dual-ascent' heuristic for finding a feasible solution to the condensed dual. It begins by setting v_j to the cost of assigning client j to the nearest facility. (This corresponds to setting all of the w variables to zero.) It then iteratively tries to increase each v_j value to the cost of the next least expensive facility, or, if this renders the solution infeasible, by as much as possible while maintaining feasibility.

The starting point of the algorithm is to sort the c_{ij} values, for each client j, in non-decreasing order. For $j = 1, \ldots, n$ and $k = 1, \ldots, m$, we let c_j^k denote the kth cost in the jth sorted list. We also use the convention $c_j^{m+1} = \infty$ for $j = 1, \ldots, n$. For a given dual solution $(v_1, \ldots, v_n)$, we let s_i denote the slack of the ith constraint of the form (1). We also let $k(j)$ denote, for each client j, the current minimum value of k such that $v_j \le c_j^k$. A location i is called *blocked* if $s_i = 0$. A client j is called *blocked* if there exists a blocked location i such that $v_j \ge c_{ij}$. If client j is blocked then the dual value v_j cannot be increased.

A high-level description of the algorithm is as follows:

For each client j, do:
  Sort the c_{ij} values in non-decreasing order.
  Set $v_j := c_j^1$ and $k(j) := 1$.
For each location i:
  Set $s_i := f_i$.
Repeat the following until all clients are blocked:
  For each unblocked client j, do:
    Let Δ_j be the minimum of s_i over all i for which $v_j \ge c_{ij}$.
    If $\Delta_j = 0$, client j is blocked.
    If $v_j + \Delta_j$ is greater than $c_j^{k(j)+1}$
      Set Δ_j to the difference between $c_j^{k(j)+1}$ and v_j.
      Increment $k(j)$ by one.
    If $\Delta_j > 0$
      Decrease s_i by Δ_j for all locations i such that $v_j \ge c_{ij}$.
      Increase v_j by Δ_j.

At the end of this procedure, the sum of the v_j variables is a valid lower bound for the SPLP.

Erlenkotter [11] also proposed a 'dual adjustment' procedure that can, and often does, improve the dual solution. Other heuristics for solving the dual were proposed, for example, by Körkel [19], Beasley [4], Jain and Vazirani [17] and Thorup [30]. For the sake of brevity, we do not review these developments in detail.

2.2. Heuristics

A wide variety of primal heuristics (i.e., heuristics for producing feasible integral solutions to the 0–1 LP) have been proposed in the literature. In the early literature, the heuristics were all of a simple greedy nature, in which facilities were either iteratively added or dropped. Examples include the 'Add' and 'Bump-and-Shift' heuristic of Kuehn and Hamburger [22], the local search heuristic of Manne [25], and the 'Drop' heuristic of Feldman et al. [12]. Later on, some meta-heuristic approaches were explored, such as genetic algorithms [21] and tabu search [29]. Also, there is an extensive literature on *approximation algorithms*, i.e., heuristics that have a proven *a priori* performance guarantee (e.g., Jain and Vazirani [17], Mahdian et al. [24], Byrka and Aardal [6]).

A completely different class of primal heuristics consists of those that attempt to exploit good dual solutions. Suppose that a feasible dual vector $v^* \in \mathbb{Z}_+^n$ has been obtained by one of the dual heuristics mentioned in the previous section. Observe that the quantity $s_i = f_i - \sum_{j\in J}\max\{0, v_j^* - c_{ij}\}$, called the 'slack' in the dual context, can be viewed as an estimate of the reduced cost of the variable y_i in the primal. This led Bilde and Krarup [5] and Erlenkotter [11] to propose the following simple primal heuristic: temporarily open a facility at every blocked location, assign each client to the closest open facility, and then close any open facility that does not have any client assigned to it.

When this primal heuristic is applied to small instances, the resulting upper bound is typically quite good. Moreover, if multiple dual solutions are available (for example, if a sequence of dual adjustments is made), then multiple primal solutions can be obtained, thus potentially leading to even better upper bounds [11,19]. A more sophisticated variant of this primal–dual scheme can be found in Hansen et al. [16]. See also Beasley [4] for a similar heuristic scheme, adapted to the Lagrangian setting.

2.3. Exact methods

A standard way of solving combinatorial optimisation problems to proven optimality is the branch-and-cut technique of Padberg and Rinaldi [28]. A branch-and-cut algorithm for the SPLP (in fact for a generalisation of it) was presented in Caprara and Salazar-González [7]. This algorithm works well for very hard instances, but for most instances it is better to use a branch-and-bound framework in which fast combinatorial procedures are used to produce the lower and upper bounds at each node of the enumeration tree.

Erlenkotter [11] embedded his dual ascent/dual adjustment procedure within a branch-and-bound scheme, using depth-first search and branching on y variables. Some effective improvements to his approach were suggested by Van Roy and Erlenkotter [32] and Körkel [19]. The most effective of these were an improved branching rule, the generation of more primal solutions, and some rules for eliminating variables on the basis of estimated reduced costs.

A similar scheme was proposed by Beasley [4], but using Lagrangian relaxation to compute lower and upper bounds instead of dual ascent and dual adjustment.

Finally, we mention that some authors have explored exact solution methods based on the idea of converting the SPLP into a pseudo-Boolean optimisation problem [13–15]. This approach seems to work best when applied to very sparse instances. (An SPLP instance is said to be *sparse* if $c_{ij}=\infty$ for the majority of pairs i,j.)

3. New dual-based procedures

3.1. Motivation

To our knowledge, an explicit analysis of the running time of the classical dual ascent procedure has not appeared in the literature. Fortunately, the analysis is straightforward, as shown in the following lemma and proposition:

Lemma 1. *The initial sorting of the assignment costs can be performed in* $\mathcal{O}(mn\log m)$ *time.*

Proof. The sorting of the assignment costs for an individual client can be performed in $\mathcal{O}(m\log m)$ time using heapsort [33]. This can be performed for each of the n clients. □

Proposition 1. *The classical dual ascent procedure runs in* $\mathcal{O}(m^2n)$ *time.*

Proof. The number of times that we encounter a positive Δ_j value is $\mathcal{O}(mn)$, and, each time this happens, we have to update $\mathcal{O}(m)$ slacks. The updating of the slacks is the bottleneck of the procedure. □

For very large SPLP instances, this can be rather time-consuming, especially if one wishes to embed the procedure within a branch-and-bound framework. We remark that very large values of m and n can arise in real-life applications. Indeed, large values of m can arise when a continuous location problem is discretised (i.e., when a location problem with an infinite number of locations is approximated by an SPLP instance with a large, but finite, set of locations). Large values of n can arise simply because some companies have thousands of clients.

As for the dual adjustment procedure, we have not seen a formal analysis of its running time. We suspect that it can be implemented to run in $\mathcal{O}(m^2n^2)$ time, but we have not managed to prove it. In any case, its running time is unattractive for very large instances.

These considerations led to our search for faster ascent procedures.

3.2. Enhanced ascent procedure

We now present an enhanced version of the classical dual ascent procedure, which works significantly faster in practice.

A key concept in the enhanced procedure is that of the *base level*. The base level is the largest value of k, with $1\le k\le m$, such that it is feasible to set v_j to c_j^k for all j. We have observed that, on instances with large m, the dual ascent procedure spends well over half the time incrementing the $k(j)$ until they all reach the base level. (A similar observation was made by Körkel [19].) Fortunately, we have the following result:

Lemma 2. *Let* k^* *denote the base level. One can compute* k^* *in* $\mathcal{O}(nk^*\log k^*)$ *time.*

Proof. First, let k be a fixed integer, with $1\le k\le m$. Simply by checking the constraints (1), one can check in $\mathcal{O}(nk)$ time whether it is feasible to set the dual variable v_j to c_j^k for each client $j\in J$.

Now, starting with $k=1$ and iteratively doubling k, one can find in $\mathcal{O}(nk^*\log k^*)$ time a value r such that $2^r\le k^*\le\min\{m,2^{r+1}\}$.

Next, by performing binary search over the interval $[2^r,\min\{m,2^{r+1}\}]$, one can determine the exact value of k^* in $\mathcal{O}(nk^*\log k^*)$ time. □

Since $k^*\le m$, the time taken to compute the base level is dominated by the initial sorting of the assignment costs (Lemma 1).

Once the base level has been computed by binary search, one can immediately set all dual values to the base level, and then use standard dual ascent to complete the ascent process. This is what we call the *enhanced ascent procedure*. Note that the time taken after the base level has been found can still be significant, so that the running time of the enhanced procedure remains $\mathcal{O}(m^2n)$. Nevertheless, as we will see, the running time is frequently substantially reduced in practice.

3.3. Fast ascent procedure

When m and n are extremely large, even the enhanced ascent procedure can be impractical. In this section, we present a new ascent procedure, called *fast ascent*, that runs in $\mathcal{O}(mn\log m)$ time. This is the same time as that taken for the initial sorting.

We will need the following lemma:

Lemma 3. *Suppose that we are given a feasible solution* v^* *to the condensed dual, and we have already computed the corresponding slacks* s_i *for all* $i\in I$*. For any given client* $j\in J$*, we can compute in* $\mathcal{O}(m)$ *time the maximum amount by which* v_j *can be increased, while maintaining feasibility.*

Proof. It suffices to compute, for all $i\in I$, the maximum amount by which v_j can be increased, while maintaining non-negativity of the slack s_i. □

Thus, to obtain the desired running time bound, it suffices to ensure that each client's dual value is increased no more than $\mathcal{O}(\log m)$ times. To achieve this, we perform a kind of binary search.

We are now ready to present the fast ascent procedure:

For each client j, do:
  Sort the c_{ij} values in non-decreasing order using heapsort.
Compute the base level k^* by binary search, as described above.
Set $v_j := c_j^{k^*}$ and $k(j) := k^*$ for all j, and update the slacks s_i accordingly.
Repeat the following until all clients are blocked:
  For each unblocked client j, do:
    Let Δ_j be the largest amount by which v_j can be increased.
    If $\Delta_j=0$, client j is blocked.
    If $v_j+\Delta_j$ is greater than $c_j^{k(j)+1}$
      Find the largest value $k'(j)$ such that $c_j^{k'(j)}$ is no larger than $v_j+\Delta_j$.
      Set $k(j) := \left\lceil \frac{k(j)+k'(j)}{2} \right\rceil$.
      Set Δ_j to the difference between $c_j^{k(j)}$ and v_j.
    If $\Delta_j>0$
      Increase v_j by Δ_j.
      Update the slacks s_i.

As mentioned above, the time taken for the sorting and for computing the base level is $\mathcal{O}(mn\log m)$. Now consider the remainder of the algorithm. For each client j, the quantity $v_j+\Delta_j$ is non-increasing, which implies that $k'(j)$ is non-increasing as well. Since we repeatedly set $k(j)$ to the mean of $k(j)$ and $k'(j)$, the total number of times v_j is updated is $\mathcal{O}(\log m)$ as desired.

Remark 1. Instead of replacing $k(j)$ with $\lceil(k(j)+k'(j))/2\rceil$, as described above, we can replace it with

$$\left\lceil\frac{(t-1)k(j)+k'(j)}{t}\right\rceil,$$

where $t>1$ is an arbitrary constant. The running time remains $\mathcal{O}(mn\log m)$, but the logarithm is to the base $t/(t-1)$. As t increases, the fast ascent routine becomes more and more similar to the enhanced routine described in the previous subsection, but the running time increases.

4. A new scheme for producing primal solutions

Now we turn our attention to primal heuristics, i.e., heuristic procedures for producing good integer feasible solutions to the SPLP. Since we are dealing with very large instances of the SPLP, we are interested in heuristics that run extremely quickly.

Recall from Section 2.2 that a primal heuristic based on opening 'blocking' facilities was described in [5,11]. It is not hard to show that this heuristic, when applied to a single given dual solution, can be implemented to run in $\mathcal{O}(mn)$ time. So, we considered simply applying this heuristic to the dual solution obtained from one of our dual ascent procedures. Our preliminary computational experiments revealed, however, that this approach usually performs very poorly.

We propose to use instead the following simple heuristic scheme, which is based on iteratively 'dropping' facilities:

Sort the locations according to some pre-specified criterion.
Temporarily open a facility at every location (i.e., set $y^*=1$ for all $i\in I$).
Assign each client to its nearest open facility.
For each location in the sorted list do:
  Evaluate the effect on the cost of closing the facility at that location.
  If closing the facility would lead to a cost saving, close it.

The following proposition shows that this 'drop heuristic' can be implemented so that it runs very quickly:

Proposition 2. *Suppose that the initial sorting of assignment costs has already been performed (i.e, for each client j, the locations have already been sorted in non-decreasing order of c_{ij}). Suppose also that the criterion for sorting the locations has been specified in advance. Then the drop heuristic can be implemented to run in $\mathcal{O}(mn+m\log m)$ time.*

Proof. The first step is to sort the locations according to the specified criterion, which can be performed in $\mathcal{O}(m\log m)$ time.

Now, we construct an array of length n, called `nearest`, with the following interpretation. At any stage of the algorithm, if `nearest` $[j]=k$, it means that the closest open facility to client j is the kth nearest facility to client j. We also construct another array of length n, called `second_nearest`, which is similar to `nearest`, but stores the level of the *second* nearest open facility to each client.

At the start of the heuristic, all facilities are open, and each client is assigned to the nearest facility. Therefore we initialise `nearest` $[j]:=1$ and `second_nearest` $[j]:=2$ for all $j\in J$.

We now scan through the list of facilities. For each facility i, we scan through the list of clients. For each client such that `nearest` $[j]=i$, if any, we evaluate the effect on the cost of re-assigning client j to the second closest open facility. If we then decide to close facility i, we scan through the clients again, and update the `nearest` and `second_nearest` arrays.

Now, the largest possible value of any entry in the two arrays is m, and the value of each entry can only increase, not decrease. Therefore the total amount of work spent in scanning and updating arrays is $\mathcal{O}(mn)$. □

The crucial choice to be made, of course, is the criterion by which the locations are to be sorted. We have experimented with three different criteria:

1. Sort in non-increasing order of fixed cost f_i.
2. Sort in non-increasing order of s_i, where s is the vector of slacks obtained at the end of the fast ascent procedure described in the previous subsection.
3. Sort in non-increasing order of s'_i, where s' is the vector of slacks obtained at the base level k^*, i.e., when $v_j=c_j^{k^*}$ for all j.

We call these approaches the *standard*, *fast* and *base* drop heuristics, respectively. The results given in the next section indicate that fast drop usually produce better upper bounds than standard and base drop, but that there is no clear winner among standard and base drop.

We have also experimented with the following more sophisticated approach, which we call the *multi-drop* heuristic: after each major iteration of the fast ascent procedure, the facilities are sorted in non-increasing order of their current slack values, and the drop heuristic is invoked. The best upper bound is then taken over all major iterations. By 'major iteration', we mean a single loop through all the clients. The number of major iterations is $\mathcal{O}(\log m)$, which ensures that the total time taken by multi-drop is $\mathcal{O}(mn\log m+m\log^2 m)$.

Note that the upper bound given by multi-drop is guaranteed to be at least as good as the best of the upper bounds given by fast and base drop. We will see in the next section that, in fact, all drop heuristics give better upper bounds than the heuristic of [5,11], and multi-drop in particular gives excellent bounds.

5. Computational experiments

In this section, we report the results of some computational experiments.

We began by testing the procedures on the three largest instances in the OR library [3], which are taken from Beasley [2]. For these instances, the assignment costs are distances between random points in the plane, but with small random perturbations. All three instances have $m=100$ and $n=1000$, and the optimal solutions are known for all of them [3].

Table 1 reports, for each instance, the percentage gap between various lower bounds and optimum. The lower bounds were obtained using four different dual ascent procedures: classical, enhanced, fast with $t=2$ and fast with $t=10$. (Recall that the classical and enhanced ascent procedures yield the same dual solution, and therefore the same lower bound.)

We see that the lower bounds from classical/enhanced ascent are very good, whereas the lower bounds from fast ascent are competitive only when the larger value for the parameter t is selected.

The running times for the ascent procedures were negligible (less than 15 ms) in every case.

Table 2 reports the results obtained when applying the various primal heuristics to the OR-Lib instances. For each instance, we display the instance name and the average percentage gaps

Table 1
Percentage gaps obtained when applying dual ascent routines to OR-Lib instances with $m=100$ and $n=1000$.

Name	% gap of lower bound		
	Class./enh.	fast2	fast10
a	0.37	3.08	0.47
b	1.13	13.14	2.29
c	1.11	8.52	1.80
Mean	0.87	8.25	1.52

Table 2
Percentage gaps obtained when applying primal heuristics to OR-Lib instances with $m=100$ and $n=1000$.

Name	Block	S-drop	B-drop	F-drop2	F-drop10	M-drop2	M-drop10
a	9.10	12.57	1.11	1.11	0.00	0.00	0.00
b	14.01	5.55	7.91	6.13	2.82	2.82	1.08
c	3.78	4.52	3.64	3.72	0.20	0.03	0.03
Mean	8.96	7.55	4.22	3.65	1.01	0.95	0.37

Table 3
Running times and percentage gaps obtained when applying ascent routines to new large instances.

$m=n$	Running time (s)				% gap of lower bound		
	Classical	Enhanced	fast2	fast10	Classical	fast2	fast10
500	0.076	0.061	0.016	0.022	1.15	4.07	1.54
1000	0.411	0.323	0.056	0.111	0.95	3.84	1.48
1500	1.150	0.916	0.131	0.276	0.70	4.42	1.26
2000	2.530	1.998	0.234	0.469	0.63	4.78	1.43
2500	5.724	4.756	0.388	0.855	0.60	4.76	1.32
3000	9.978	8.227	0.575	1.358	0.84	5.38	1.76
Mean	3.312	2.714	0.233	0.515	0.81	4.54	1.47

Table 4
Percentage gaps obtained when applying primal heuristics to new large instances.

$m=n$	Block	S-drop	B-drop	F-drop2	F-drop10	M-drop2	M-drop10
500	20.06	7.96	7.68	2.21	1.77	1.29	0.52
1000	21.40	8.25	11.08	1.78	2.05	0.81	0.79
1500	25.33	9.11	13.40	3.09	2.59	1.21	0.62
2000	27.62	8.33	11.90	2.71	1.94	1.37	0.68
2500	27.73	8.74	17.79	2.58	1.40	1.28	0.62
3000	26.84	9.36	14.19	2.59	1.80	1.42	0.72
Mean	24.83	8.63	12.67	2.49	1.93	1.23	0.66

between the upper bounds and the optimum for seven different primal heuristics: the one based on 'blocking' facilities, given in [5,11], standard drop, base drop, fast drop with $t=2$ and $t=10$, and multi-drop with $t=2$ and 10.

We see that the heuristic based on blocking facilities performs surprisingly poorly. The drop heuristics do bit better, and the multi-drop heuristics are the clear winner. Again, the running times were negligible.

Next, we decided to test the procedures on some larger random instances. As usual in the literature (e.g., Ahn et al. [1], Hansen et al. [16]), these instances were created by 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. In our case, the facility costs are random numbers between $\sqrt{n}/3$ and $\sqrt{n}/2$. (Since the ascent procedures require costs to be integers, all costs were then multiplied by 5000 and rounded down to the nearest integer.)

To begin with, we created some instances with m and n ranging from 500 to 3000. We were able to solve each of these instances to proven optimality by running dual ascent, eliminating variables as in [4,19], and then running ILOG CPLEX 12.0 for a long period of time. Although time-consuming, this enables us to assess precisely the quality of the various lower and upper bounds.

Tables 3 and 4 present the results for these instances. Each row represents the average over 10 random instances. We remark that, for the sake of brevity, we do not report running times for the primal heuristics. As it turned out, these times were almost always negligible in comparison with the time taken by the dual ascent routines. The only exception was in the case of the multi-drop heuristic, for which the running times were similar to the time taken by the fast ascent algorithm (with the same value of t).

We see that the running times are quite small for these instances, and usually below 1 s in the case of our fast procedures. As before, the fast drop and multi-drop heuristics perform very well, but one needs to use a reasonably large value of t to obtain good lower bounds with the fast ascent procedure.

Finally, we ran the procedures on some large-scale instances, with m and n ranging from 5000 to 15,000. The results are shown in Table 5. Each row of the table represents the average over five instances. As before, the times do not include the times for the primal heuristics, which were in all cases much smaller than the times taken by the ascent algorithms. We remark that, since these instances were too large to solve to proven optimality, we report the percentage gap between the lower and upper bounds. Moreover, for the sake of space, we do not report the gaps for the standard and base drop heuristics (which perform poorly in any case).

For these instances, the running times and percentage gaps are noticeably larger. Moreover, we were surprised to see just how poorly the classical 'blocking' heuristic performs. On the other

Table 5
Running times and percentage gaps obtained when testing larger instances.

Size	Running time (s)				Total % gap				
	Classical	Enhanced	fast2	fast10	Block	F2	F10	M2	M10
5000	31.853	23.269	1.753	4.753	32.31	6.98	4.41	6.09	3.16
7500	98.304	72.954	4.410	11.297	40.41	7.95	5.43	7.09	3.76
10,000	192.779	140.466	9.569	23.072	45.63	8.23	5.01	7.43	3.15
12,500	318.058	234.707	23.928	54.450	47.24	9.44	4.78	7.65	3.49
15,000	494.387	353.172	54.664	122.70	47.15	9.32	5.81	7.87	3.62
Mean	227.076	164.914	18.864	43.154	42.55	8.39	5.09	7.23	3.44

hand, we think it is promising that, using our new methods, one can obtain lower and upper bounds differing by under 4% in a couple of minutes even when m and n are as large as 15,000.

Finally, we remark that we would have liked to run our algorithms on even larger instances, but ran into memory limitations.

6. Conclusion

Large-scale SPLP instances can arise when there are thousands of clients, or when a continuous location problem is discretised. We have shown that it is possible to compute quickly reasonably good lower bounds, and very good upper bounds, for such instances. In a subsequent paper, we will show how to embed our fast lower- and upper-bounding procedures in a sophisticated scheme for solving large-scale instances to proven (near-)optimality.

We end the paper by making some remarks about sparsity (see Section 2.3 for a definition). For a given client j, let d_j denote the number of locations for which c_{ij} is finite. Also define:

$$\sigma = \sum_{j \in J} d_j \quad \text{and} \quad \overline{d} = \max_{j \in J}\{d_j\}.$$

It is not hard to implement the classical and fast ascent algorithms so that they run in $\mathcal{O}(\sigma\overline{d})$ and $\mathcal{O}(\sigma \log \overline{d})$ time, respectively. Moreover, sparse instances consume significantly less memory. For this reason, we believe that our new algorithms could be used to tackle sparse instances of extremely large size.

Acknowledgements

The first author was supported in part by the Engineering and Physical Sciences Research Council under Grant EP/D072662/1. Thanks are due to an anonymous referee for some helpful suggestions.

References

[1] Ahn S, Cooper C, Cornuéjols G, Frieze AM. Probabilistic analysis of a relaxation for the p-median problem. Mathematics of Operations Research 1988;13:1–31.
[2] Beasley JE. An Algorithm for solving large capacitated warehouse location problems. European Journal of Operational Research 1988;33:314–25.
[3] Beasley JE. OR-Library: distributing test problems by electronic mail. Journal of the Operational Research Society 1990;41:1069–72.
[4] Beasley JE. Lagrangian heuristics for location problems. European Journal of Operational Research 1993;65:383–99.
[5] Bilde O, Krarup J. Sharp lower bounds and efficient algorithms for the Simple Plant Location Problem. Annals of Discrete Mathematics 1977;1:79–88.
[6] Byrka J, Aardal K. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. SIAM Journal on Computing 2010;39:2212–31.
[7] Caprara A, Salazar-González JJ. A branch-and-cut algorithm for a generalization of the uncapacitated facility location problem. TOP (Journal of the Spanish Society of Statistics and Operations Research) 1996;4:135–63.
[9] Conn AR, Cornuéjols G. A projection method for the uncapacitated facility location problem. Mathematical Programming 1990;46:273–98.
[10] Cornuéjols G, Nemhauser GL, Wolsey LA. The uncapacitated facility location problem. In: Mirchandani PB, Francis RL, editors. Discrete location theory. New York: Wiley; 1990. p. 1–54.
[11] Erlenkotter D. A dual-based procedure for uncapacitated facility location. Operational Research 1978;26:992–1009.
[12] Feldman E, Lehrer FA, Ray TL. Warehouse location under continuous economies of scale. Management Science 1966;12:670–84.
[13] Goldengorin B, Ghosh D, Sierksma G. Branch and peg algorithms for the Simple Plant-Location Problem. Computers & Operations Research 2003;30:967–81.
[14] Goldengorin B, Tijssen GA, Ghosh D, Sierksma G. Solving the Simple Plant Location Problem using data correcting approach. Journal of Global Optimization 2003;25:377–406.
[15] Hammer PL. Plant location: a pseudo-Boolean approach. Israel Journal of Technology 1968;6:330–2.
[16] Hansen P, Brimberg J, Urosevic D, Mladenovic N. Primal–dual variable neighborhood search for the Simple Plant-Location Problem. INFORMS Journal on Computing 2007;19:552–64.
[17] Jain K, Vazirani VV. Approximation algorithms for metric facility location and k-median problems using the primal–dual schema and Lagrangian relaxation. Journal of the ACM 2001;48:274–96.
[19] Körkel M. On the exact solution of large-scale Simple Plant Location Problems. European Journal of Operational Research 1989;39:157–73.
[20] Krarup J, Pruzan PM. The Simple Plant Location Problem: survey and synthesis. European Journal of Operational Research 1983;12:36–81.
[21] Kratica J, Tosic D, Filipovic V, Ljubic I. Solving the Simple Plant Location Problem by Genetic Algorithm. RAIRO Operations Research 2001;35:127–42.
[22] Kuehn AA, Hamburger MJ. A heuristic program for locating warehouses. Management Science 1963;9:643–66.
[23] Labbé M, Louveaux F. Location problems. In: Dell'Amico M, Maffioli F, Martello S, editors. Annotated bibliographies in combinatorial optimization. Chichester: Wiley; 1997. p. 261–81.
[24] Mahdian M, Ye Y, Zhang J. Approximation algorithms for metric facility location problems. SIAM Journal on Computing 2006;36:411–32.
[25] Manne AS. Plant location under economies-of-scale—decentralization and computation. Management Science 1964;11:213–35.
[26] Mladenovic N, Brimberg J, Hansen P. A note on duality gap in the Simple Plant-Location Problem. European Journal of Operational Research 2006;174:11–22.
[27] Morris J. On the extent to which certain fixed-charge depot location problems can be solved by LP. Journal of the Operational Research Society. 1978;29:71–6.
[28] Padberg MW, Rinaldi G. Optimization of a 532 city symmetric traveling salesman problem by branch-and-cut. Operations Research Letters 1987;6:1–7.
[29] Sun M. Solving the uncapacitated facility location problem using tabu search. Computers & Operations Research 2006;33:2563–89.
[30] Thorup M. Quick and good facility location. In: Proceedings of the 14th annual ACM-SIAM symposium on discrete algorithms. Philadelphia: SIAM; 2003. p. 178–85.
[31] Todd MJ. An implementation of the simplex method for linear programming problems with variable upper bounds. Mathematical Programming 1982;23:34–49.
[32] Van Roy TJ, Erlenkotter D. A dual based procedure for dynamic facility location. Management Science 1982;28:1091–105.
[33] Williams JWJ. Algorithm 232—heapsort. Communications of the ACM 1964;7:347–8.