Solving Single-Source Capacitated Facility Location Problems using Repeated Matching

Mikael Rönnqvist
Department of Engineering Science, The University of Auckland
PB 92019 Auckland, New Zealand
email m.ronnqvist@auckland.ac.nz

Abstract

Facility location models form an important class of integer programming problems, with application in many areas such as the distribution and transportation industries. An important class of solution methods for these problems are so-called Lagrangian heuristics which have shown to produce high quality solutions and at the same time are robust. The general facility location problem can be divided into a number of special problems depending on the properties assumed. In the capacitated location problem each facility has a specific capacity. We describe a new solution approach for the capacitated facility location problem when each customer is served by a single facility. The approach is based on a repeated matching algorithm which essentially solves a series of matching problems until certain convergence criteria are satisfied. The method generates feasible solutions in each iteration in contrast to Lagrangian heuristics where problem dependent heuristics must be adopted to construct a feasible solution. Preliminary numerical results show that the approach produces solutions equal in quality to the best Lagrangian heuristics.

1 Introduction

Facility location models are used in many areas such as the distribution and transportation industries. These models have received a great deal of attention in the research literature. Given a set of potential locations for facilities and a set of customers, the facility location problem is to assign facilities in such a way that the total cost for assigning facilities and satisfying the customer demands is minimized. The facility location problem can be classified into different categories depending on the restrictions assumed. In the uncapacitated facility location problem each facility is assumed to have no limits on its capacity. Here, each customer receives all required demand from one facility. When each facility has a limited capacity the problem is called the capacitated facility location problem. The Single-Source Capacitated Facility Location (SSCFL) problem is a special case of the capacitated facility location problem in which each customer can only be supplied from one facility.
One of the most common and successful approaches to solve the SSCFL problem is to use so-called Lagrangian heuristics. These heuristics are based on a Lagrangian relaxation and solving the related Lagrangian dual problem. Here a sequence of smaller and simpler subproblems is solved for which efficient solution procedures have been developed. To ensure the construction of feasible solutions some heuristic procedure must be adopted which, given a subproblem solution and/or the dual solution, attempts to generate a feasible solution. Klincewicz and Luss [7] present an algorithm that is based upon relaxing the constraints on the facility capacities. The corresponding Lagrangian subproblems then become uncapacitated facility location problems. Pirkul [8], Barcelo and Casanovas [2] and Sridharan [9] develop algorithms based upon relaxing the customer assignment constraints. By this relaxation a number of knapsack problems is constructed in each iteration. Beasley [3] presents a relaxation of both the capacity constraints and the customer assignment constraints. Beasley also give a comprehensive comparison between various Lagrangian heuristics to facility location problems. For the problem considered in this paper Beasley found that the approach developed by Pirkul [8] provides the best feasible solutions, followed by Beasley [3], and then Klincewicz and Luss [7].

Repeated matching has previously been used to obtain heuristic solutions to several routing and scheduling problems. The term repeated matching is used to describe the solving of a succession of related matching problems. The solution to one problem generates a new matching problem, and the sequence of problems typically terminates when the optimal matching leaves all elements unmatched. It has been used by Forbes [6] in bus crew scheduling, and by Desrochers and Verhoog [4] in work on the fleet size and mix vehicle routing problem. Altinkemer and Gavish [1] and Wark and Holt [10] use it as the basis for solving vehicle routing problems. It has also been used in aircrew scheduling by Wark et al. [11].

In this paper, we consider a solution procedure based on repeated matching to solve the SSCFL problem. In section 2 we formulate the mathematical model of the SSCFL problem. The concept of repeated matching and how it can be adopted for the problem is then described in section 3. An important part of the approach is the computation of matching costs, this is given in section 4. The overall solution procedure is then summarized in section 5. We then give some preliminary numerical results in section 6, and finally make some conclusions.

2 Problem formulation

To formulate the mathematical model for the SSCFL problem we introduce the following notation. Let \( m \) be the number of potential facilities, \( n \) the number of customers, \( a_j \) the demand of customer \( j \), \( b_i \) the capacity of facility \( i \), and \( c_{ij} \) the cost for assigning customer \( j \) to facility \( i \). Also introduce the following decision variables.

\[
y_i = \begin{cases} 
1 & \text{if facility } i \text{ is open} \\
0 & \text{otherwise}
\end{cases}
\]

\[
x_{ij} = \begin{cases} 
1 & \text{if facility } i \text{ services customer } j \\
0 & \text{otherwise}
\end{cases}
\]

The Integer Programming (IP) problem can then be stated as
The objective function is to minimize the cost consisting of the cost of establishing facilities and the cost of assigning customers to open facilities. The first constraint set ensures that the customer demand serviced by a certain facility cannot exceed its capacity. The second constraint set ensures that each customer is assigned to exactly one single facility and the third set ensures that the assignments are made only to open facilities. The last set is the integrality requirement.

3 The Repeated Matching Algorithm

3.1 Matching problems

The basic matching problem may be described as follows. Given a set $A$ of $k$ elements \{\(a_1, a_2, \ldots, a_k\)$ (where $k$ is even), a perfect matching on $A$ is a matching of elements in $A$ such that each $a_i \in A$ is matched with one and only one $a_j \in A$ ($j \neq i$). Thus a perfect matching on $A$ is a partition of $A$ into pairs of matched objects. Given costs $d_{ij}$ of matching $a_i$ with $a_j$ ($i, j = 1, \ldots, k; i \neq j$), where $d_{ij} = d_{ji}$, the minimum cost perfect matching problem is to find a perfect matching on $A$ for which the sum of the costs of matching the pairs of matched elements in $A$ is minimized. It is, however, useful to extend the concept of the perfect matching problem to admit the possibility of an element being matched with itself, effectively remaining unmatched, which we describe as self-matching. This more general minimum matching problem can be formulated as an IP problem by introducing the following decision variables.

\[
z_{ij} = \begin{cases} 1 & \text{if } a_i \text{ is matched with } a_j \\ 0 & \text{if } a_i \text{ is not matched with } a_j. \end{cases}
\]

In this formulation $d_{ii}$ must be twice the actual cost of self-matching, as the cost of true matchings is counted twice in the objective function. We note that $k$ need not be even in this formulation.
3.2 Notation and terminology

In order to use the repeated matching algorithm for solving problem [P] we need to make a reformulation of the problem. Let \( I = \{1, 2, \ldots, m\} \) be the set of all facilities, \( J = \{1, 2, \ldots, n\} \) the set of all customers, and \( K = I \times G \), where \( G \) is the set of all nonempty subsets of \( J \).

We say \((i, C) \in K\) is feasible if \( \sum_{j \in C} a_{ij} \leq b_i \), and let \( F \) be the set of all feasible members of \( K \). A packing is a subset \( P \) of \( F \) with the property that

\[
(i_1, C_1), (i_2, C_2) \in P \Rightarrow C_1 \cap C_2 = \emptyset \text{ and } i_1 \neq i_2. \tag{1}
\]

Given a packing \( P \), let \( L_1 = \{i \mid (i, C) \notin P\} \), \( L_2 = \bigcup_{(i, C) \notin P} C \), and \( L_3 = P \). Each of these sets has cardinality \( n_A \), \( n_B \), and \( n_C \) respectively. The meaning of these sets are that the set \( L_1 \) consists of all facilities that are unused and \( L_2 \) consists of all customers which are not assigned to a facility. The set \( L_3 \) consists of all used facilities with their assigned customers. The cost of the packing is

\[
Mn_B + \sum_{(i, C) \in P} f_i + \sum_{(i, j) \mid (i, C) \in P, j \in C} c_{ij}, \tag{2}
\]

where \( M \) is some large number.

By using the concept of unassigned customers we may have a packing which does not necessarily correspond to a feasible solution of problem [P]. If the set \( L_2 \) is non-empty then the packing violates the assignment constraints for customers in problem [P]. In the repeated matching approach we want to match elements of \( L_1, L_2 \) and \( L_3 \) with each other so as to generate new sets \( L_1', L_2' \) and \( L_3' \) so that the cost of the new packing is improved. By choosing a large value of \( M \) the matching will tend to decrease the number of elements in \( L_2 \) (as in a penalty function approach). From the numerical experiments we find that unassigned customers is not a problem because they are only a part of a packing in the early iterations of the algorithm.

4 Matching costs

Given a packing we need to compute cost coefficients, \( d_{ij} \), for formulation [M]. The dimension of the cost matrix will change dynamically as the packing changes. At any instance the dimension of the cost matrix is determined by \( n_A + n_B + n_C \). Different properties can be used for finding the matching costs when the different elements in \( L_1, L_2 \) and \( L_3 \) are matched. The cost matrix will consist of nine submatrices. We refer these to as block 1, \ldots, block 9. However, since the cost matrix is symmetric we only need to investigate six of the blocks given in formula (3).

\[
d = \begin{bmatrix}
[L_1 - L_1] & [-] & [-] \\
[L_2 - L_1] & [L_2 - L_2] & [-] \\
\end{bmatrix} = \begin{bmatrix}
[1] & [-] & [-] \\
\end{bmatrix} \tag{3}
\]

To compute the matching costs for blocks 1, 2, and 3 is very straightforward. The main reason is that the feasibility checking is trivial. For the remaining blocks we have a more complicated situation. Here we may swap customers between open facilities. At the same time we need to ensure feasibility with respect to the capacity constraint of each facility. It turns out that block 6 is the most general block and that blocks 4 and 5 are modified cases. We will now discuss how the cost coefficients can be found for each of the blocks.
4.1 Block 1-3

A matching of an unused facility with another unused facility (block 1) is not feasible and the cost can therefore be set to $\infty$. A self-matching, i.e., where a facility is matched with itself give a cost of 0 (since it remain unmatched). A matching of an unassigned customer with an unused facility (block 2) is feasible if the capacity of the facility is not exceeded by the demand of the customer. The cost is then just the sum of opening the facility and the cost of assigning the customer. A matching of an unassigned customer with another unassigned customer (block 3) is not possible and hence the cost is $\infty$. A self-matching has cost $2M$.

4.2 Block 6

To find the cost of a matching of two elements $(i_1, C_1), (i_2, C_2) \in P$ we split the matching into three different cases. The first two correspond to the case when all customers are assigned to one of the two facilities. The third case is when some swapping of customers between the two facilities occurs. Cases 1 and 2 are easy to check by comparing the total demand from the customers with the capacity of each facility. In case 3 we need to find the best swapping of customers. This is, however, an IP problem and to formulate it we introduce the following variables.

$$\begin{cases} 
1 & \text{if customer } j \in C_1 \text{ swaps to facility } i_2 \\
0 & \text{otherwise}
\end{cases}$$

$$\begin{cases} 
1 & \text{if customer } j \in C_2 \text{ swaps to facility } i_1 \\
0 & \text{otherwise}
\end{cases}$$

The problem has two constraints which state that the surplus capacities at the two facilities must not be exceeded by the swapping of customers. This problem has a special structure and we can actually reformulate the problem into an interval knapsack problem (which has one constraint).

$$\text{[K]} \quad \min \left\{ \sum_{j \in C_1} g_j w_j + \sum_{j \in C_2} h_j z_j \right\}$$

s.t. \(-\delta_w \leq \sum_{j \in C_1} -a_j w_j + \sum_{j \in C_2} a_j z_j \leq \delta_w \)

$$w_j, z_j \in \{0, 1\} \quad \forall j$$

Here $g_j$ is the cost if customer $j \in C_1$ swaps facility to facility $i_2$, $h_j$ is the cost if customer $j \in C_2$ swaps facility to facility $i_1$, and $\delta_w$ and $\delta_z$ are the surplus capacities at facility $i_1$ and $i_2$ respectively. Problem [K] can be solved by a number of different methods. We have used a dynamic programming procedure for its solution. To find the correct matching cost we then compare the costs corresponding to the three different cases and select the smallest.

4.3 Block 4-5

To find the costs for block 4 and 5 we modify the computations for block 6 (in terms of the elements $(i_1, C_1)$ and $(i_2, C_2)$). To match an element $(i_1, C_1) \in L_3$ with an empty facility (block 4) we simply have no customer in the set $C_2$. The surplus capacity of facility $i_2$ is given by $b_{i_2}$ (since it is unused). When we match a composite with an
unassigned customer (block 5) we have two possibilities. The first is that by assigning
the unassigned customer to the facility the capacity is not exceeded. If the capacity is
exceeded, then some customers must be unassigned. This can be done by forcing the
unassigned customer to the set $C_1$ which now is enlarged with one element. To ensure
a feasible solution we include a “pseudo” facility $i_2$ with a capacity equal the demand
of the unassigned customer.

5 Algorithm

We are interested of finding solutions to [P] by exploring different packings. The dif­
ferent packings are obtained using a heuristic method based on the ideas used in Wark
and Holt [10]. The heuristic uses the repeated matching approach to form new feasible
packings from previously determined feasible packings. The process is started by taking
any feasible packing and a sequence of feasible packings of decreasing cost is formed.
When repeated matching fails to yield any further reduction in cost, the packing is split
so that customers previously assigned to certain facilities become unassigned. After
the splitting, a new sequence of packings are generated to facilitate further progress.
This process is then repeated until no further progress can be made in a fixed number
of splits. A scheme of the algorithm is as follows.

Initialize Set $Best\_main=\infty$, $Best\_Local=\infty$, $k=1$, $l=1$, $Max\_split=m$.

Find an initial solution (a feasible packing).

Main iteration $k$:

1. Local iteration $l$:
   a. Compute matching costs, $d$.
   b. Solve matching problem [M].
   c. Update $Best\_Local$:
      (i) If no improvement, apply splitting and go to step 2.
      (ii) If improvement, update $Best\_Local$ and set $l = l + 1$.
      Then return to step a.

2. Update $Best\_main$
   a. If no improvement in $m$ main iterations, stop.
   b. If improvement update $Best\_main$, set $k = k + 1$,
      $Best\_Local=\infty$ and return to step 1.

End of algorithm

The splitting is needed when no improved solution can be found. Here, we take
the current solution and remove some customers from one or more facilities. One
alternative for splitting is the following. For each element $(i, C) \in L_3$, we generate a
random number $\alpha \in [0, 1]$. Then for each customer $j \in C$ we generate another random
number $\beta \in [0, 1]$. If $\beta \leq \alpha$ then we remove the customer from the set $C$ and insert it
in the set $L_2$. 
6  Numerical results

The proposed algorithm has been implemented and tested on a number of test problems. The size of the test problems is given in Table 1.

<table>
<thead>
<tr>
<th>Test problem</th>
<th>m</th>
<th>n</th>
<th># of 0/1 variables</th>
<th># of constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>90</td>
<td>900</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>80</td>
<td>1600</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>70</td>
<td>2100</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>90</td>
<td>900</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>100</td>
<td>3000</td>
<td>130</td>
</tr>
</tbody>
</table>

Table 1: Data for the five test examples.

To illustrate the overall convergence we show results from test problem 2. Table 2 gives the last Best_local value and the Best_main value at each main iteration. The final value is found in main iteration four and then the algorithm needs nine additional iterations before it terminates (Max_split was set to ten in all tests.).

<table>
<thead>
<tr>
<th>Main iter.</th>
<th>Best_local</th>
<th>Best_main</th>
<th>Main iter.</th>
<th>Best_local</th>
<th>Best_main</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5753</td>
<td>5753</td>
<td>8</td>
<td>5722</td>
<td>5666</td>
</tr>
<tr>
<td>2</td>
<td>5741</td>
<td>5741</td>
<td>9</td>
<td>5722</td>
<td>5666</td>
</tr>
<tr>
<td>3</td>
<td>5753</td>
<td>5741</td>
<td>10</td>
<td>5722</td>
<td>5666</td>
</tr>
<tr>
<td>4</td>
<td>5666</td>
<td>5666</td>
<td>11</td>
<td>5735</td>
<td>5666</td>
</tr>
<tr>
<td>5</td>
<td>5666</td>
<td>5666</td>
<td>12</td>
<td>5785</td>
<td>5666</td>
</tr>
<tr>
<td>6</td>
<td>5839</td>
<td>5666</td>
<td>13</td>
<td>5795</td>
<td>5666</td>
</tr>
<tr>
<td>7</td>
<td>5741</td>
<td>5666</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: The values, Best_main and Best_local, obtained in each main iteration in solving test problem 2.

Some computational aspects from solving the five test examples can be found in table 3. To perform one main iterations we need, in average, 10-11 local iterations.

<table>
<thead>
<tr>
<th>Test problem</th>
<th>CPU time* (sec)</th>
<th># Main iter.</th>
<th># Local iter.</th>
<th>% CPU time costing</th>
<th>% CPU time matching</th>
<th>Repeated matching</th>
<th>Pirkul</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.6</td>
<td>11</td>
<td>127</td>
<td>46</td>
<td>53</td>
<td>6228</td>
<td>6228</td>
</tr>
<tr>
<td>2</td>
<td>27.0</td>
<td>14</td>
<td>153</td>
<td>82</td>
<td>17</td>
<td>5666</td>
<td>5672</td>
</tr>
<tr>
<td>3</td>
<td>5.9</td>
<td>11</td>
<td>100</td>
<td>29</td>
<td>67</td>
<td>5317</td>
<td>5364</td>
</tr>
<tr>
<td>4</td>
<td>12.3</td>
<td>16</td>
<td>186</td>
<td>75</td>
<td>24</td>
<td>7045</td>
<td>7029</td>
</tr>
<tr>
<td>5</td>
<td>30.7</td>
<td>11</td>
<td>113</td>
<td>74</td>
<td>25</td>
<td>7142</td>
<td>7163</td>
</tr>
</tbody>
</table>

* The tests was done on a Silicon Graphic workstation.

Table 3: Computational results from solving the five test examples.

This seems to be independent of problem size. The proportion of the time spent
on computing the matching costs, which involves the solution of knapsack problems, ranges from 29% up to 82%. Corresponding values for the solution of the matching problems is 17% up to 67%. We also include a comparison with Pirkul’s method since this has shown to be the best Lagrangian heuristic. We find that our approach gives better values in three test problems, worse in one, and equal in one. Preliminary results on a larger set of test problems show that the two methods are very close in terms of quality of the feasible solutions.

7 Conclusions

We have proposed a new approach to solve the SSCFL problem. It is a heuristic method based on solving a sequence of related matching problems. In each of the iterations we solve a matching problem to optimality and to find the related matching costs we solve a number of knapsack problems. Preliminary numerical results indicates that the approach is at least as efficient as the best Lagrangian heuristic.

Future work includes making a more comprehensive comparison with other methods for solving the problem. To combine a dual approach with the proposed method is very interesting. In a combined scheme the dual scheme would generate lower bounds and the proposed scheme would generate feasible solutions. The solution from each Lagrangian subproblem would here act as an initial solution. The fact that the subproblem solution is, in general, infeasible can easily be handled by the approach.

References