The heuristic concentration-integer and its application to a class of location problems

Vladimir Marianov\textsuperscript{a,*,1}, Michelle Mizumori\textsuperscript{b,1}, Charles ReVelle\textsuperscript{*}

\textsuperscript{a}College of Engineering, Pontificia Universidad Católica de Chile, Víctor Mackenna 4860, Santiago, Chile
\textsuperscript{b}The Johns Hopkins University, Baltimore, MD, USA

Available online 10 March 2008

Abstract

We propose a new metaheuristic called heuristic concentration-integer (HCI). This metaheuristic is a modified version of the heuristic concentration (HC), oriented to find good solutions for a class of integer programming problems, composed by problems in which \( p \) elements must be selected from a larger set, and each element can be selected more than once. These problems are common in location analysis. The heuristic is explained and general instructions for rewriting integer programming formulations are provided, that make the application of HCI to these problems easier. As an example, the heuristic is applied to the maximal availability location problem (MALP), and the solutions are compared to those obtained using linear programming with branch and bound (LP + B&B). For one-third of the instances of MALP, LP + B&B can be allowed to run until the computer is out of memory without termination, while HCI can find good solutions to the same instances in a reasonable time. In one such case, LP-IP was allowed to run for nearly 100 times longer than HCI and HCI still found a better solution. Furthermore, HCI found the optimal solution in 33.3\% of cases and had an objective value gap of less than 1\% in 76\% of cases. In 18\% of the cases, HCI found a solution that is better than LP+B&B. Therefore, in cases where LP + B&B is unreasonable due to time or memory constraints, HCI is a valuable tool.

© 2008 Elsevier Ltd. All rights reserved.

Keywords: Facility location; Heuristics; Integer programming

1. Introduction

In the \( p \)-median problem on a network (Hakimi [1,2], ReVelle and Swain [3]), locations must be selected for \( p \) facilities on a network, so that the average distance between each demand point and its closest facility is minimized. Heuristic concentration (HC) is a two stage metaheuristic originally designed to approach large instances of the \( p \)-median problem (Rosing and ReVelle [4]). This heuristic has been applied to this problem with excellent results, outperforming well-established heuristics such as Tabu Search (Rosing et al. [5]).

When applying the HC to the \( p \)-median, its first stage consists of \( q \) runs of a fairly simple base heuristic, such as the one-opt heuristic of Teitz and Bart [6], with some added random or varying component, usually a randomly generated initial feasible solution. This variation allows the multiple runs to find different good solutions, whereas if the algorithm...
contained no varying element, each run would arrive at the same solution. Each solution contains a set of good locations of the facilities. The multiple good solutions are used to create the Concentration Set, which lies at the heart of HC. This Concentration Set is then used to apply a more complex algorithm in stage 2.

The Concentration Set, CS, is the union of the best \( m \) solutions from all of the runs of the base heuristic of the first stage. In other words, it is every item (location) that was selected in any of the top \( m \) (\( m \leq q \)) unique solutions. This CS will be the reduced eligible or candidate set for the second stage, since there is reason to believe that the elements of CS must have some reasonably good quality to have been selected to be in the top solutions, and therefore the optimal solution is likely to be a subset of CS.

The second stage of the original version of HC consists of a single run of an exact method, as linear programming plus branch and bound (LP + B&B), applied over the Concentration Set. At the end of this second stage, an optimal solution is found, constrained to choosing elements belonging to the CS. It is important to note, however, that although an exact solution method was used, the optimal solution over the reduced eligible set may not be the same as the optimal solution to the problem solved over the entire eligible set.

A later version of HC, also intended for solving the \( p \)-median, was developed by Rosing et al. [7]. In this modified version, called “Gamma Heuristic”, the first stage is the same, but an additional Concentration Set is defined after the first stage: the concentration set \( CS_0 \), which is the intersection of the best \( m \) unique solutions from all of the runs of the first stage of the heuristic. This differs from the Concentration Set CS, because it only includes an item if that item occurred in every single one of the best solutions. Again, it is reasonable to believe that, if an item was selected in all of these good solutions, it will also be selected in the optimal solution. Therefore, the members of \( CS_0 \) are required to be selected in every solution considered in the second stage. Naturally, \( CS_0 \) contains a feasible number of items, since all of them are present in a number of feasible solutions.

As opposed to HC, where the second stage is an exact method, the second stage of the Gamma Heuristic consists of \( s \) runs of a more complex, non-exact algorithm, using the concentration sets described above. The CS is used as the set of all possible items to choose, while the items contained in the \( CS_0 \) are forced to belong to all the solutions. Notice that if this algorithm has no variation, for example, when using an exact method as in the original HC, there is no point in repeating \( s \) times the algorithm. However, if there is a varying element to the algorithm, repetition can lead to better solutions.

An optional third stage of the Gamma Heuristic takes the best solution from the second stage and tries to improve upon it by performing another heuristic on the eligible set, possibly again the one-opt method. Thus, the third stage provides the opportunity for further improvement.

For examples and analyses of different versions of HC, see Rosing and ReVelle [4], Rosing et al. [5,7], Rosing [8], Rosing and Hodgson [9], and Mizumori et al. [10].

The \( p \)-median belongs to a much larger family of combinatorial optimization problems, that can be described as picking at most \( p \) elements from a large set of “candidate” elements and putting them into a smaller set \( P \), in such a way as to optimize an objective that depends on the quality of the \( p \) elements in the set \( P \). Besides the \( p \)-median, this family contains other location problems, as the discrete \( p \)-center problem; the maximal covering location problem (MCLP) of Church and ReVelle [11]; the maximum availability location problem (MALP) of ReVelle and Hogan [12] and its derivates, as the queuing maximum availability location problem (QMALP) of Marianov and ReVelle [13]; the maximum expected covering location problem (MEXCLP) of Daskin [14] and many other similar problems.

In the \( p \)-median, each demand point is assigned to a single facility, specifically the closest one. In the MCLP, coverage of a demand is achieved if it has at least one facility within a preset neighborhood. In both cases, there is no point in locating more than a facility at the same place. On the other hand, in MALP, coverage frequently requires more than one facility in the neighborhood, and the optimal solution will most likely include sites with more than one facility. In other words, in MALP the same candidate location will be selected more than once and co-location of servers or facilities at the same site will result. The same happens in MEXCLP.

The method we propose here, called heuristic concentration integer (HCI), is a two-stage general procedure that can, in a number of instances, solve integer programming problems as the ones just described, often with optimal solutions, HCI is an outgrowth of the HC of Rosing and ReVelle [4] and the Gamma Heuristic of Rosing et al. [7]. In its second phase, HCI uses a heuristic, as does the Gamma Heuristic. Although it also could have an optional third stage, we use only two stages. In its first stage, HCI repeatedly uses a one-opt procedure, each time using a different initial feasible solution of the problem, and saving the best solution found. In its second stage, HCI uses a two-opt heuristic, using as candidate points all the locations found in any solution in the first stage. The main difference between HCI and
Gamma Heuristic is its capability of repeatedly picking the same element, while in Gamma Heuristic each element can be selected only once. Another difference is the way that the concentration sets are defined. Now, CS must be defined as the set of any location that was selected at least once in any solution of the first stage. Similarly, the CS₀ is defined as any site that was selected in all of the solutions.

Using this procedure, the problems need not be linear integer problems or even linearizable. Furthermore, since similar heuristics have been so far applied only to the \( p \)-median problem, we also propose some general rules for rewriting different integer programming formulations, in such a way that it is easier to apply HCI for finding the solution of the corresponding problems. Note that the formulations so obtained, are in general nonlinear. These nonlinear formulations allow a faster and easier evaluation of the quality of the solutions, not only when applying HCI, but also other heuristics, such as one-opt, two-opt, genetic algorithms, and, in general, any heuristics that requires repeated evaluation of the solutions.

Finally, we apply this algorithm to the MALP or the QMALP models, both of which seek to find the best location of \( p \) facilities, in the presence of congestion, so as to provide timely service. These models are of use, for example, in the strategic design of emergency medical services. The goal of the classic MCLP was to locate a fixed number of facilities in such a way that as much demand as possible had at least one server located within a preset distance or travel time. Among other natural applications of this model, were finding the optimal locations of emergency medical vehicles, pizza delivery parlors and service centers needing a response within a preset time limit. Coverage of a demand was fulfilled when the demand had at least one server closer than the maximum allowed distance. However, the location of a facility or a server within a preset distance does not guarantee service within time limits, because the server or facility might be busy at the time a call comes in. The MALP generalized this concept of coverage through the new concept of server or service availability. In the MALP, instead of the demand being considered as served (covered) by having at least one facility located within a preset distance, it is considered as served when there is a percentage of time, larger than a certain minimum value, during which at least one server is idle, within the preset distance. In other words, a demand node is counted as served only if a facility or server is available, within time or distance \( S \), with probability \( z \) or more. This model seeks to maximize the population having service available within a desired travel time with a stated reliability, given that only \( p \) servers are to be located.

In ReVelle and Hogan’s [12] MALP, increased service availability for a demand is achieved by locating a higher number of servers in a neighborhood of that demand, defined by the distance standard \( S \). In order to compute the required number of servers in the neighborhood, an explicit probabilistic modeling of the system needs to be done. Once the number of servers is computed, a constraint on the availability of servers for each demand node is formulated, which utilizes binomial distribution. Since binomial distribution has an implicit assumption of independence between the occupancy states of servers, even if they serve the same region, Marianov and ReVelle[13] relaxed this assumption in MALP, considering each neighborhood as an M/M/s-loss queuing system, computing the required number of servers in the neighborhood accordingly. Their model was the QMALP.

ReVelle and Hogan [12] as well as Marianov and ReVelle [13] solved their models using LP + B&B. Although this is a good choice for small size instances, the solution time increases significantly with the size of the problem, and efficient heuristic methods are required for solving large instances of the problem. However, the only heuristic method proposed so far for solving MALP-like models has been simulated annealing; Galvão et al. [15] used this heuristic on their Extended MALP or EMALP.

2. Heuristic concentration-integer (HCI)

When the problem consists of picking elements from a set of candidate elements as described before, HC is a heuristic that allows the selection of each element just once. For example, if the problem consists of finding optimal locations for a set of facilities, every candidate location can be occupied by at most one facility. This is represented by a binary variable taking a value of 1 if there is a facility or 0 otherwise. However, in some problems, each element can be selected more than once. Using the same example as before, a candidate location could accommodate several facilities, or servers, and in the integer programming formulation of the problem this could be represented by an integer variable taking the value of the number of facilities co-located at the same site, as opposed to a binary variable representing one or no facilities in the same location. For these cases, we use our integer version of the HC—the HCI.

HCI uses the same basic structure as the Gamma Heuristic version of the binary HC. In the first stage, a simple heuristic, for example, a one-opt algorithm, is used. This begins with a random selection of \( p \) elements from the eligible
set of \( n \) elements. Notice that in our integer version, an element may be chosen more than once and thus \( p \) may be greater than \( n \). The objective value of this random start is then calculated.

In an exchange, we first choose one of the elements that have been selected at least once in the current solution. We then reduce the number of times that it is selected by one and iteratively increase the selection of each of the other elements by one, and calculate the objective value. We then take the exchange that had the best objective value if it was an improvement over the objective value of the current solution. When there is no one-for-one exchange that can improve the objective value, we have completed a run of one-opt. We do \( q \) runs of the one-opt algorithm, each with a new random start, and save the best solutions of each run.

In the HCI we do not use a CS\( _0 \), although in some cases, the use of a set of forced locations (or elements in the solution set) could speed the procedure up. The main Concentration Set is again the set of all elements that were chosen at least once in any of the top \( m \) \((m \leq q)\) unique solutions from the first stage.

In the second stage, we use a more complex algorithm; in our case, the two-opt algorithm, with \( s \) random initial starts. In the two-opt algorithm, we do two-for-two exchanges. In the binary two-opt, at each exchange, two elements that were selected in the current solution are de-selected, and two elements that did not belong to the solution are added to it. In HCI, each element can be selected more than once. In this case, at each exchange, the presence of a single element in the solution could be increased twice, and also decreased twice if it was selected twice to belong to the solution. In other words, the increases can be either increasing a single element in the Concentration Set by two or increasing two different elements in the Concentration Set by one. Either way, we take the best exchange if it was an improvement over the current objective value. We do not use a third stage.

A pseudo-code representation of the procedure is as follows:

**Procedure** HCI \((n; p; \text{coordinates of demand points}; \text{parameters})\)

Begin first stage

\[
\text{for } i = 1 \text{ to } q \\
\quad \text{// Find } i\text{th random initial solution} \\
\quad \text{for } k = 1 \text{ to } p \\
\quad \\n\quad \text{Locate } k\text{th facility at random, at any of the } n \text{ candidate locations;} \\
\quad \text{endfor} \\
\quad \text{Compute Objective} \\
\quad \text{Save Objective, Locations} \\
\quad \\n\quad \text{// Improve } i\text{th initial solution using One-Opt} \\
\quad \text{Call One-Opt} \\
\quad \\n\quad \text{// Save the } i\text{th improved solution} \\
\quad \text{Objective } (i) = \text{Objective} \\
\quad \text{Locations } (i) = \text{Locations} \\
\quad \text{endfor} \\
\text{// Save the } m \text{ best solutions into the Concentration Set} \\
\text{for } j = 1 \text{ to } m \\
\quad \text{Objective } (j) = j\text{th best objective } (i) \\
\quad \text{Locations } (j) = \text{Locations corresponding to } j\text{th best objective} \\
\text{Endfor}
\]

End first stage

Begin second stage

\[
\text{for } i = 1 \text{ to } s \\
\quad \text{// Find } i\text{th random initial solution} \\
\quad \text{for } k = 1 \text{ to } p \\
\quad \text{Locate } k\text{th facility randomly at any of the locations} \\
\quad \text{belonging to Concentration Set;} \\
\quad \text{endfor}
\]

endfor
Compute Objective
Save Objective, Locations
\\ Improve $i$th initial solution using Two-Opt
Call Two-Opt
\\ Save the $i$th improved solution
Objective ($i$) = Objective
Locations ($i$) = Locations
endfor
\\ Pick the best solution
Objective = best objective
Locations = Locations corresponding to best objective
End second stage
End the procedure

Function One-Opt
for $k = 1$ to $p$
  for $m = 1$ to $n$
    Move $k$th facility to position $m$
    Compute Objective
    If Objective is the best so far
      Save Objective, Locations
    endif
  endfor
endfor

Function Two-Opt
for $k = 1$ to $p$
  for $l = 1$ to $p$
    for $m = 1$ to $|Concentration Set|$
      for $r = 1$ to $|Concentration Set|$
        Move $k$th facility to position $m$
        Move $l$th facility to position $r$
        Compute Objective
        If Objective is the best so far
          Save Objective, Locations
        endif
      endfor
    endfor
  endfor
endfor

3. Rewriting integer formulations

When applying the HCI, or any other iterative procedure, repeated evaluations of the objective are required. Usual integer programming formulations are not intended for fast objective evaluations: they just represent logical or algebraic relations between the variables in a linear fashion. Furthermore, if these formulations are used as a starting formulation for finding solutions through heuristic procedures, at each step, these procedures must check if the constraints hold, which slows down the process. We propose some general rewriting or reformulation rules for the class of problems we are dealing with, which have a twofold impact: firstly, these rules are a systematic way of transforming integer programming formulations into models that are very well suited for heuristics. Secondly, the resulting models eliminate the need of checking if some of the constraints hold, and integrate these constraints into the objective, making it nonlinear
but easier to evaluate. In many cases, the problems in this class can be transformed into nonlinear objective knapsack
tasks, in which objects in the knapsack have the same weight (or volume) but different economic value.

The goal of the rewriting procedure is to replace as many variables as possible in the objective, leaving the objective
written only in terms of few variables. Many combinatorial optimization models, including some of those common in
location analysis, fall into the family of problems that can be rewritten as we propose.

These rewriting rules apply to optimization problems in which:

- exactly (or at most) \( p \) elements must be selected from a set,
- any selection of the \( p \) elements is feasible, and
- variables are binary or integer. In some cases, other types of variables can also be considered, as in the \( p \)-center.

In an integer programming formulation, the first condition is usually enforced by a constraint that specifies the number
of elements to be selected, i.e. a constraint of the type

\[ \sum_j x_j = p \]

Note that in this constraint, the elements to be selected are represented by the variables \( x_j \), one for each element. If
each element can be selected only once, the variables will be binary, while if elements can be selected more than once,
the variables will be integer.

The second condition means that there is no need to check feasibility at each step of the procedure, as it would
happen for example if there are capacity constraints, or constraints that enforce special conditions, including equilibrium
conditions between demand and supply, or equilibrium between congestion-sensitive demand arrival rate and queuing
at the facilities, and the like. In these cases, if a heuristic as HCI is to be used, sub-problems must be solved at each
step of it, to assure that capacity is not overflowing or that equilibrium is reached.

Unfortunately, nothing too general can be said so far about constraints that enforce special conditions or put a limit on
capacity, because these can take many different forms. In some particular cases, it is possible to do some simplification
and rewriting; for example, linear equivalent constraints can be found for nonlinear capacity-like constraints, as did
Marianov and ReVelle [16] in their Queuing PLSCP.

Once we rule out special constraints, we leave only constraints that define a variable as a logical function of other
variables, or represent logical relations between the variables (“logical constraints”). These logical constraints can
always be rewritten and some of the variables can be replaced in the objective by functions of the selection or location
variables \( x_j \).

### 3.1. Equality constraints

In general, logical equality constraints have the linear or nonlinear form:

\[ y_i = f(x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_n) \quad \text{for all } i. \]

In this case, variable \( y_i \) can be directly replaced into the objective or in other constraints by function \( f(\bullet) \). If this
function is difficult to evaluate, unless the constraint is unfeasible or ill posed, it is still possible to represent logical
constraints in simple forms, because variable \( y_i \) will take only values 0 or 1. In fact, if the logical constraint is an
equation and all variables are binary, it can be represented as a truth table, i.e. a table that shows the value of one
variable as a function of the remaining variables in the equation, as the example shown in Fig. 1, in which variable \( m \)
is a function of variables \( a, b, \) and \( c \).

The truth table can be computed once at the beginning of the solution procedure, and used to evaluate the variable
each time the objective has to be evaluated. Note that the truth table can be as large as needed. If it is too large, once it
is found, the relations between variables can be simplified using Karnaugh maps (used in Boolean algebra, Karnaugh
[17]), and can be always expressed as sums of products or products of sums of the variables in the function \( f(\bullet) \), \( x_j \)
and \( z_j \) or their complements \( (1 - x_j) \) and \( (1 - z_j) \). This procedure has been used for a long time for the design of
digital electronic logic gates.

Once the function \( f(\bullet) \) is simplified and written as a sum of products, it can be replaced everywhere by this sum of
products. If it is part of the objective, its evaluation is extremely simple and straightforward: each product is replaced
by 0 every time at least one of its factors (variables that multiply each other) in it is 0, and replaced by 1 otherwise, while a sum is replaced by 1 if at least one of the addends are 1, and by 0 otherwise. If variable $y_i$ is present in any other constraint, it can also be replaced by this sum of products, and then, proceed with any further rewriting of this constraint.

3.2. Inequality constraints

If a model has inequality logical constraints as

$$y_i \leq f_i(x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_n) \quad \text{for all } i$$

and

$$z_j \geq g_j(x_1, x_2, \ldots, x_n) \quad \text{for all } j,$$

in order to replace variables $y_i$ and $z_j$, these constraints can be rewritten as

$$y_i = \min\{1, f_i(x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_n)\} \quad \text{for all } i,$$

$$z_j = \max\{0, g_j(x_1, x_2, \ldots, x_n)\} \quad \text{for all } j$$

and

$$y_i = \min\{1, f(x_1, x_2, \ldots, x_n, \max\{0, g_1(x_1, x_2, \ldots, x_n)\}, \ldots, \max\{0, g_n(x_1, x_2, \ldots, x_n)\})\}.$$ 

As an example, Table 1 shows some typical constraints in combinatorial problems, and how these constraints are rewritten so variables can be replaced in the objective.

Using reformulations, we could rewrite, for example, the following model, to make it suitable for a fast objective evaluation:

$$\text{Max} \quad Z = \sum_i W_i u_i$$

s.t. $u_i \leq \sum_j u_{ij} \quad \forall i$, (1)

$$b_i u_{ij} \leq \sum_k x_{jk} \quad \forall i, j,$$ (2)

$$x_{jk} \leq x_k \quad \forall j, k,$$ (3)

$$\sum_j x_j = p,$$ (4)

$$u_i, y_{ij}, x_{jk}, x_j = 0, 1 \quad \forall i, j, k.$$ (4)
Table 1
Example constraints.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Logical meaning</th>
<th>Replace in the objective by</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u_i \leq \sum_j u_{ij} )</td>
<td>Variable ( u_i ) is not allowed to take value 1 unless the RHS is greater than or equal to 1</td>
<td>( u_i = \min \left{ 1, \sum_j u_{ij} \right} )</td>
</tr>
<tr>
<td>( x_{jk} = x_k x_j )</td>
<td>Variable ( x_{jk} ) takes value 1 only if both variables ( x_k ) and ( x_j ) are 1</td>
<td>( x_{jk} = \min {x_k, x_j} )</td>
</tr>
<tr>
<td>( z_{jk} \leq x_k )</td>
<td>Variable ( z_{ij} ) is not allowed to take value 1 if either or both variables ( x_k ) and ( x_j ) are 0. This form is used when variable ( z_{ij} ) must take the value 1 when both ( x_k ) and ( x_j ) are 1</td>
<td>( z_{jk} = \min {x_k, x_j} ) or ( z_{jk} = \max {0, x_k + x_j - 1} )</td>
</tr>
<tr>
<td>( z_{jk} \geq x_j + x_k - 1 )</td>
<td>Variable ( z_{ij} ) is not allowed to take value 0 if both variables ( x_k ) and ( x_j ) are 1. This form is used when variable ( z_{ij} ) must take the value 0 if either or both variables ( x_k ) and ( x_j ) are 0</td>
<td>( z_{jk} = \min {x_k, x_j} )</td>
</tr>
<tr>
<td>( u_{ij} \leq c_{ij} x_j + \sum_k c_{ik} x_{jk} )</td>
<td>Variable ( u_{ij} ) is not allowed to take value 1 unless the RHS is greater than or equal to 1</td>
<td>( u_{ij} = \min \left{ 1, \frac{1}{b_i} \left[ x_j + \sum_k x_{jk} \right] \right} )</td>
</tr>
<tr>
<td>( b_i u_{ij} \leq \sum_k x_{jk} )</td>
<td>Variable ( u_{ij} ) is not allowed to take value 1 unless the RHS is greater than or equal to ( b_i )</td>
<td>( u_{ij} = \min \left{ 1, \frac{1}{b_i} \left[ \sum_k x_{jk} \right] \right} )</td>
</tr>
</tbody>
</table>

Note that, if all parameters are positive and if the right-hand side of constraint (1) is zero, the variable \( u_i \) is zero. If the right-hand side is one or more, this variable takes the value of 1, because it is being maximized in the objective (assuming that the weights \( W_i \) are non-negative). The same condition can be written as

\[
\begin{align*}
  u_i &= \min \left\{ 1, \sum_j u_{ij} \right\} \quad \forall i. \\
\end{align*}
\]

Considering that it is enough for one variable \( u_{ij} \) to take the value 1 for variable \( u_i \) to be also 1, we can further rewrite this constraint as

\[
\begin{align*}
  u_i &= \min \left\{ 1, \max_j \{u_{ij}\} \right\} \quad \forall i. \quad \text{(1bis)}
\end{align*}
\]

Using the same reasoning and assuming positive \( b_i \)'s, constraint (2) can be rewritten as

\[
\begin{align*}
  u_{ij} &\leq \left[ \frac{1}{b_i} \left[ x_j + \sum_k x_{jk} \right] \right] \quad \forall i, j, \\
  u_{ij} &= \min \left\{ 1, \frac{1}{b_i} \left[ \sum_k x_{jk} \right] \right\} \quad \forall i, j. \quad \text{(2bis)}
\end{align*}
\]

Both constraints (1bis) and (2bis) can be merged in the following equation:

\[
\begin{align*}
  u_i &= \min \left\{ 1, \max_j \{\min \left\{ 1, \frac{1}{b_i} \left[ x_j + \sum_k x_{jk} \right] \right\}\} \right\}.
\end{align*}
\]

Constraint (3) can be written as

\[
\begin{align*}
  x_{jk} &= \min \{x_k, x_j\}
\end{align*}
\]
and the full problem can be formulated as

\[
\text{Max } Z = \sum_i W_i \text{Min} \left\{ 1, \text{Max} \left\{ 1, \left[ c_{ij} x_j + \sum_k c_{ik} \text{Min} \{x_k, x_j\} \right] \right\} \right\} \\
\text{s.t. } \sum_j x_j = p, \\
\quad x_j = 0, 1 \quad \forall j.
\]

Note that the problem is now a nonlinear knapsack type of formulation. This problem can be solved using any heuristic that uses repeated selection of \( p \) elements, represented by variables \( x_j \), and corresponding objective evaluations.

### 3.3. Special cases

There are some special cases, in which, rather than starting with the integer programming formulation and rewriting each constraint, logical relations are directly embedded into the final formulation. Examples of these cases are the \( p \)-median and \( p \)-center problems. The most well-known \( p \)-median formulation is due to ReVelle and Swain [3]:

\[
\text{Min } Z = \sum_{i,j} a_i d_{ij} y_{ij} \\
\text{s.t. } y_{ij} \leq x_j \quad \forall i, j, \\
\sum_j y_{ij} \leq 1 \quad \forall i, \\
\sum_j x_j = p, \\
y_{ij}, x_j = 0, 1 \quad \forall i, j,
\]

where \( i \) is the index of demand points, \( j \) the index of available server locations, \( p \) the number of servers to be located, \( x_j = 1, 0 \), it is 1 if a server is located at candidate location \( j \), and 0 otherwise, \( y_{ij} = 1, 0 \), it is 1 if demand point \( i \) is assigned to a facility at \( j \), 0 otherwise, \( a_i \) the demand at demand point \( i \), and \( d_{ij} \) the distance between nodes \( i \) and \( j \).

The objective in this model minimizes the total distance between demand nodes and their closest facilities. Consequently, it has two functions. In the first place, together with the first two constraints it optimizes the assignment, that is, once \( p \) facilities have been located, the model assigns each demand to its closest facility. In the second place, it indirectly pushes the facilities to locate on the sites that minimize total weighted distance.

If facilities are already located, the assignment problem consists in connecting each demand \( i \) to the existing facility \( j \) that corresponds to the minimum demand-facility distance, that is, finding \( \min_j \{d_{ij} x_j\} \). The objective captures this minimum value for all demands, while the constraint describes the location problem:

\[
\text{Min } Z = \sum_i a_i \min_j \{d_{ij} x_j\} \\
\text{s.t. } \sum_j x_j = p, \\
\quad x_j = 0, 1 \quad \forall j.
\]
This formulation is suitable for solving through any heuristic that requires repeated evaluation of the objective. Rewriting the $p$-center problem for heuristic solution is analogous. The classical formulation is as follows:

Min $Z$

s.t. $\sum_j a_i d_{ij} y_{ij} \leq Z \quad \forall i,$

$\sum_j y_{ij} = 1 \quad \forall i,$

$y_{ij} \leq x_j \quad \forall i, j,$

$\sum_j x_j = p,$

$y_{ij}, x_j = 0, 1 \quad \forall i, j,$

where the variables and parameters are the same as in the $p$-median problem. In the $p$-center, there is again a location problem and an assignment problem. The difference with the $p$-median is that although the assignment of each demand is again to the closest facility, the objective minimizes the maximum distance between a demand and its closest facility. In other words, the new formulation is:

Min $Z = \max_i \left\{ \min_j \{d_{ij} x_j\} \right\},$

s.t. $\sum_j x_j = p,$

$x_j = 0, 1 \quad \forall j.$

4. Review of MALP and QMALP

MALP and QMALP have two different formulations. We will call the first formulation the ‘integer-coefficient’ model. This model uses the following notation:

$i$ the index of demand points

$j$ the index of available server locations

$N_i$ neighborhood of $i$, that is, the set of locations $j$ that are within standard distance of demand $i$

$p$ the number of servers to be located

$x_j$ 1, 0, it is 1 if a server is located at candidate location $j$, and 0 otherwise

$y_i$ 1, 0, it is 1 if demand point $i$ has $b_i$ servers in its neighborhood, 0 otherwise

$h_i$ demand at demand point $i$ and

$b_i$ number of facilities needed at demand point $i$ so that the required availability of service is fulfilled

The integer-coefficient model can thus be defined mathematically as:

Max $Z = \sum_{i \in I} h_i y_i$

s.t. $b_i y_i \leq \sum_{j \in N_i} x_j \quad \forall i,$

$\sum_{j \in J} x_j = p,$

$y_i, x_j = 0, 1 \quad \forall i, j.$

In order to estimate $b_i$, the required number of servers in $N_i$ for $\alpha$ percent availability, the MALP assumes that busy probabilities of servers in the neighborhood follow a binomial distribution, that is, there is independence of the
probabilities of different servers being busy. This number $b_i$ is computed numerically, by finding the smallest integer which satisfies

$$1 - \left( \frac{\rho_i}{b_i} \right)^{b_i} \geq \alpha,$$

where $\rho_i$ is the utilization ratio, i.e. call arrival rate divided by service rate, in the neighborhood $N_i$.

Marianov and ReVelle [13] relaxed the independence assumption in MALP, considering each neighborhood as an M/M/s-loss queuing system, and computing the parameter $b_i$ as the smallest integer satisfying

$$\frac{(1/b_i!)^b_i}{1 + \rho_i + (1/2!)^p_i^2 + \cdots + (1/b_i!)^p_i^{b_i}} \leq 1 - \alpha.$$

In order to apply HCI to this version of the problem, it has to be rewritten as follows:

Max $Z = \sum_{i \in I} h_i \min \left\{ 1, \left[ \frac{\sum_{j \in N_i} x_j}{b_i} \right] \right\}$

s.t. $\sum_{j \in J} x_j = p$,

$x_j = 0, 1 \ \forall j$.

Through numerical experience, Marianov and ReVelle [13] found that this integer-coefficient formulation tended to lead to many fractional solutions and thus extensive branching and bounding. They therefore formulated a ‘counting variables’ version, which has more variables and constraints, but was expected to be more integer-friendly. This version uses variables that are similar to those of the MEXCLP.

The counting variable formulation of the MALP would use a new variable $u_{ik} = 1, 0$, it is 1 if demand point $i$ has a $k$th server in its neighborhood, 0 otherwise.

The mathematical formulation would then be:

Max $Z = \sum_{i \in I} h_i u_{ib_i}$

s.t. $\sum_{k=1}^{b_i} u_{ik} \leq \sum_{j \in N_i} x_j \ \forall i$,

$u_{ik} \leq u_{i(k-1)} \ \forall i, \forall k = 2, 3, \ldots, b_i$,

$\sum_{j \in J} x_j = p$,

$u_{ib_i}, x_j = 0, 1 \ \forall i, j$.

Marianov and ReVelle [13] also noticed that only the $x_j$ and the $u_{ib_i}$ need to be defined as binary variables. When $u_{ib_i}$ is equal to 1, all of the other $u_{ik}$ for the same species will also be forced to 1 by the second set of constraints. When $u_{ib_i}$ is equal to 0, the values of the other $u_{ik}$ for the same species are unimportant because they have no impact on the objective value or the values of any other variables.

There is no point in rewriting this counting variable formulation, since it leads to the same new formulation as the integer-coefficient version. However, there are cases in other problems in which the counting variables are needed.
For example, in the MEXCLP, the objective includes all these counting variables. Its formulation is

\[
\begin{align*}
\text{Max} & \quad Z = \sum_{k} \sum_{i \in I} h_i q_k u_{ik} \\
\text{s.t.} & \quad \sum_{k=1}^{b_i} u_{ik} \leq \sum_{j \in N_i} x_j \quad \forall i, \\
& \quad \sum_{j \in J} x_j = P, \\
& \quad u_{ik}, x_j = 0, 1 \quad \forall i, j, k.
\end{align*}
\]

In order to obtain the formulation for heuristic purposes, the MEXCLP can be rewritten as:

\[
\begin{align*}
\text{Max} & \quad Z = \sum_{k} \sum_{i \in I} h_i q_k u_{ik} \\
\text{s.t.} & \quad k u_{ik} \leq \sum_{j \in N_i} x_j \quad \forall i, k = 1, 2, 3, \ldots, b_i, \\
& \quad \sum_{j \in J} x_j = P, \\
& \quad u_{ik}, x_j = 0, 1 \quad \forall i, j, k.
\end{align*}
\]

And the rewriting proceeds as in the integer coefficient MALP.

Small instances of MALP and QMALP can be easily solved using LP + B&B, but the solution time increases rapidly with the size of the problem. This is due to the availability constraint of MALP and QMALP:

\[
b_i y_i \leq \sum_{j \in N_i} x_j.
\]

If \( r < b_i \) servers are located in the neighborhood of demand \( i \), the right-hand side will take the value \( r \) in the solution. Since \( y_i \) is maximized in the objective, this variable will tend to take the maximum possible value, in this case, the non-integer value \( r/b_i \). The LP + B&B solver will necessarily branch on variable \( y_i \). Consequently, the model is not as integer friendly as MCLP, and heuristic methods are required for solving large instances of this problem.

A similar effect takes place when using the counting variable formulation.

5. Computational experiments with the MALP

We solved some instances of the integer-coefficient MALP using both HCI and LP + B&B. HCI was coded in Visual Basic on a 1.6 GHz, Pentium 4 PC with 384 MB of memory. The Linear Programming models were solved using CPLEX 7.5\(^2\) on the same computer.

Different authors have used different values for the parameters \( q, m \) and \( s \). For the parameter \( m \), Rosing and ReVelle [4] use a value of 5 while Rosing and Hodgson [9] experiment with values of \( m \) of 5, 10, and 15, and Rosing [8] chose \( m = 10 \). All of these parameter values have been used for the \( p \)-median problem. Although they constitute good initial choices, they could change for different problems. As \( m \) increases, the Concentration Set gets larger. The chances that the optimal solution (or at least a better solution) can be found in the second stage are better, because there are more options to choose from. However, a larger Concentration Set also increases the run time of the algorithm by increasing the number of eligible solutions.

Values of \( q \) ranging from 25 (Rosing and Hodgson [9]) to 200 (Rosing and ReVelle [4]) have been used. The value of \( q \) indirectly influences the size of the Concentration Set. The larger the value of \( q \), the greater the probability that the “best” solutions are good. However, it must be kept in mind that solutions that are close to optimality have a greater chance of being similar to each other. Consequently, for a given \( m \) (number of best solutions that are considered for the

Table 2
CPLEX results.

<table>
<thead>
<tr>
<th>Coverage</th>
<th>CPLEX time</th>
<th>Pop Cov</th>
<th>Pop Cov</th>
<th>Server locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>distance</td>
<td>(s)</td>
<td>(%)</td>
<td>(%)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>1750</td>
<td>22.87</td>
<td>7 44 61 64 82 82 103 103 103 103</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>1998</td>
<td>26.11</td>
<td>22 22 44 44 61 91 103 103 103 103</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>2169</td>
<td>28.35</td>
<td>7 7 7 7 13 13 18 18 91 120</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>3863</td>
<td>50.49</td>
<td>7 7 7 7 51 51 51 51 91</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>6130</td>
<td>80.12</td>
<td>7 8 13 50 68 76 79 79 88 109</td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>6772</td>
<td>88.51</td>
<td>Stopped after 3 h 15 min Within 5% of the best node</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>44</td>
<td>93.53</td>
<td>7 13 13 14 60 74 79 114 134 138</td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>14</td>
<td>98.84</td>
<td>12 13 14 15 37 50 51 62 79 109</td>
</tr>
<tr>
<td></td>
<td>140</td>
<td>0.04</td>
<td>100.00</td>
<td>1 1 7 7 14 34 34 77 138 138</td>
</tr>
<tr>
<td>15</td>
<td>50</td>
<td>2184</td>
<td>30.81</td>
<td>18 18 25 44 61 64 103 103 103 103 103 103 103 103 109 125 126 142 142 142</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>2307</td>
<td>37.01</td>
<td>22 22 25 25 25 30 44 44 61 103 103 103 103 109 109 109</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>688</td>
<td>39.62</td>
<td>7 7 7 7 18 18 44 46 66 70 94 111 121 142 142</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>4854</td>
<td>63.44</td>
<td>Stopped after 2 h 55 min Within 1.93% of the best node</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>7662</td>
<td>96.44</td>
<td>3 7 7 13 15 16 37 42 43 51 74 79 81 84 91 92 138</td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>7642</td>
<td>99.88</td>
<td>1 7 7 14 16 42 74 79 84 90 103 115 122 142 146</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>7651</td>
<td>100.00</td>
<td>3 7 14 29 42 43 68 74 84 102 120 122 127 141</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>0.1</td>
<td>100.00</td>
<td>3 7 14 29 42 43 68 74 84 102 120 122 127 141</td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>2184</td>
<td>30.81</td>
<td>18 18 25 44 61 64 103 103 103 103 103 103 103 103 109 125 126 142 142 142</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>2307</td>
<td>37.01</td>
<td>22 22 25 25 25 30 44 44 61 103 103 103 103 109 109 109 109</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>688</td>
<td>39.62</td>
<td>7 7 7 7 18 18 44 46 66 70 94 111 121 142 142</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>4854</td>
<td>63.44</td>
<td>Stopped after 2 h 55 min Within 1.93% of the best node</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>7662</td>
<td>96.44</td>
<td>3 7 7 13 15 16 37 42 43 51 74 79 81 84 91 92 138</td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>7642</td>
<td>99.88</td>
<td>1 7 7 14 16 42 74 79 84 90 103 115 122 142 146</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>7651</td>
<td>100.00</td>
<td>3 7 14 29 42 43 68 74 84 102 120 122 127 141</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>0.1</td>
<td>100.00</td>
<td>3 7 14 29 42 43 68 74 84 102 120 122 127 141</td>
</tr>
<tr>
<td>25</td>
<td>50</td>
<td>756</td>
<td>44.18</td>
<td>7 18 18 23 24 24 24 25 31 13 44 62 75 101 103 103 103 103 109 109 111 111 111 113</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>10,224</td>
<td>45.24</td>
<td>Stopped after 2 h 51 min Within 5.5% of the best node</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>15,648</td>
<td>48.03</td>
<td>Stopped after 4 h 20 min Within 3.18% of best node</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>18,000</td>
<td>74.33</td>
<td>Stopped after 5 h 00 min Within 2.98% of best node</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>123</td>
<td>100.00</td>
<td>1 3 7 7 13 15 16 37 42 43 51 74 79 81 84 91 92 138</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>7651</td>
<td>100.00</td>
<td>1 3 7 7 13 15 16 37 42 43 51 74 79 81 84 91 92 138</td>
</tr>
<tr>
<td>30</td>
<td>50</td>
<td>1379</td>
<td>49.25</td>
<td>7 7 15 18 18 24 24 24 31 41 43 43 37 46 75 75 83 103 103 103 103 105 105 105 109 109 109 111 111 111 139</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>11,389</td>
<td>59.00</td>
<td>Stopped after 3 h 09 min Within 12% of the best node</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>13,522</td>
<td>62.14</td>
<td>Stopped after 3 h 48 min Within 8.88% of the best node</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>220</td>
<td>92.94</td>
<td>1 3 7 7 14 16 18 25 32 35 41 43 44 47 47 50 51 51 60 66 67 67 68 68 74 74 79 80</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>72</td>
<td>100.00</td>
<td>2 2 12 13 14 18 19 26 26 41 60 60 66 101 102 102 113 116 120 120 128 128</td>
</tr>
</tbody>
</table>

Concentration Set), the Concentration Set grows at a decreasing rate with the number $q$ of times we run the one-opt algorithm.

As $p$ grows, the run time potentially increases. The effects of raising $p$ are due to the combinatorial nature of this parameter. The number of feasible solutions to the problem is $nC_p$, where $n$ is the number of nodes, and the notation $aC_b$ is the ‘choose’ function and equals $a!/b!(a - b)!$. Therefore, the number of combinations $nC_p$ peaks at $p = n/2$ and declines to either side of $n/2$. For $p < n/2$, the number of combinations increases as $p$ increases. In the HCI runs, however, only part of the solution space is analyzed. In fact, each of the stages of the algorithm uses polynomial time.

The initial random solution is found in a time that is $O\left( p \right)$ and the procedure is repeated $q$ times, which makes $O\left( pq \right)$. Each run of the one-opt procedure is $O\left( np \right)$.

Increasing $p$, $q$, and $s$ increases the run time.

By trying different values of the parameters $q$, $m$, and $s$, we found that the value of $q$ (the number of random starts of the one-opt heuristic) is not too relevant, as long as it is not less than 10. Above this value, facility locations tend
to repeat, mostly. We use $q = 30$, because the one-opt heuristic is fast and running it more times does not increase importantly the total solution time. Because of the same reason, the parameter $m$ (number of solutions from the first stage that are considered for construction of the Concentration Set) should have a value of 10 or above. We found that below 10, the solutions deteriorate. By using a value of 15 instead of 10, the frequency of iterations in the second stage that end in the best solution, increases. Finally, we used $s = 5$ random initial solutions for the two-opt algorithm in the second stage, which provides a good compromise between solution quality and run time.

We used the 150 nodes network of Galvao and ReVelle [18]. The number of servers needed for availability at each demand point, $b_i$, was randomly set between 1 and 6. The results of the runs are shown in Tables 2–4. In Tables 2 and 3, the first column is the number of facilities. The second column shows the coverage distance. The third column shows the run time for CPLEX and HCI, respectively. Columns 4 and 5 show the population coverage in absolute and percentage values. The last column shows the locations of the servers.
In Table 4, the first column is the number of facilities. The second column shows the coverage distance. The third and fourth columns show the run time for HCI and CPLEX, respectively. The numbers in italics indicate the CPLEX runs that stopped before the optimal solution was found. Columns 5 and 6 show the percentage of population coverage in the HCI and CPLEX solutions. Column 7 shows the gap between HCI and CPLEX solutions, defined as $(100 \times (\text{CPLEX Objective} - \text{HCI Objective})/\text{CPLEX Objective})$. In this column, the negative values (bold font) indicate that HCI found better objective values than CPLEX. In these cases, CPLEX stopped because the computer did run out of memory, without finding the optimal solution. The last column shows the gap between the best integer solution found by CPLEX and the upper bound, in those cases in which CPLEX stopped.

As Table 4 shows, in 11 out of 33 cases, HCI found the best known solution, either the optimal solution or an integer solution that is better than the best integer solution found by CPLEX. In 25 out of 33 cases, the gap between CPLEX and HCI solutions is less than 1% (or better). The biggest overall positive gap (CPLEX better than HCI) is 2.83%, while the biggest negative gap (HCI better than CPLEX) is 13.37%. The average gap is 0.05%.

It has been argued that LP + B&B (CPLEX, in our case) should always be used because even when it does not terminate, it still finds a good integer solution (Rodrigues and Gaston [19]). However, in our tests, in six cases where

<table>
<thead>
<tr>
<th>p</th>
<th>Coverage distance</th>
<th>HCI time (s)</th>
<th>CPLEX time (s)</th>
<th>Pop Cov HCI (%)</th>
<th>Pop Cov CPLEX (%)</th>
<th>GAP %</th>
<th>CPLEX gap %</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>50</td>
<td>12</td>
<td>1</td>
<td>22.87</td>
<td>22.87</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>30</td>
<td>40</td>
<td>26.11</td>
<td>26.11</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>17</td>
<td>64</td>
<td>28.13</td>
<td>28.35</td>
<td>0.78</td>
<td></td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>32</td>
<td>667</td>
<td>50.44</td>
<td>50.49</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>35</td>
<td>5891</td>
<td>79.69</td>
<td>80.12</td>
<td>0.54</td>
<td></td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>126</td>
<td>11,821</td>
<td>88.69</td>
<td>88.51</td>
<td>0.21</td>
<td>5.00</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>106</td>
<td>835</td>
<td>92.42</td>
<td>93.53</td>
<td>1.19</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>112</td>
<td>14</td>
<td>98.26</td>
<td>98.84</td>
<td>0.58</td>
<td></td>
</tr>
<tr>
<td></td>
<td>140</td>
<td>178</td>
<td>0.04</td>
<td>100.00</td>
<td>100.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>50</td>
<td>44</td>
<td>2184</td>
<td>30.71</td>
<td>30.81</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>133</td>
<td>2307</td>
<td>36.27</td>
<td>37.01</td>
<td>2.01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>118</td>
<td>688</td>
<td>38.83</td>
<td>39.62</td>
<td>1.98</td>
<td></td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>110</td>
<td>10,515</td>
<td>62.89</td>
<td>63.44</td>
<td>0.87</td>
<td>1.93</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>233</td>
<td>12</td>
<td>96.20</td>
<td>96.44</td>
<td>0.26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>345</td>
<td>8</td>
<td>99.18</td>
<td>99.88</td>
<td>0.71</td>
<td></td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>556</td>
<td>8</td>
<td>99.58</td>
<td>100.00</td>
<td>0.42</td>
<td></td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>663</td>
<td>0.1</td>
<td>100.00</td>
<td>100.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>143</td>
<td>143</td>
<td>37.13</td>
<td>37.59</td>
<td>1.22</td>
<td></td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>399</td>
<td>10,224</td>
<td>45.21</td>
<td>45.24</td>
<td>0.06</td>
<td>5.50</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>439</td>
<td>15,648</td>
<td>48.49</td>
<td>48.03</td>
<td>0.95</td>
<td>3.18</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>379</td>
<td>18,000</td>
<td>73.94</td>
<td>74.33</td>
<td>0.53</td>
<td>2.98</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>899</td>
<td>123</td>
<td>98.92</td>
<td>100.00</td>
<td>1.08</td>
<td></td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>1502</td>
<td>115</td>
<td>100.00</td>
<td>100.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>50</td>
<td>326</td>
<td>756</td>
<td>43.18</td>
<td>44.18</td>
<td>2.25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>809</td>
<td>19,965</td>
<td>52.97</td>
<td>46.73</td>
<td>13.37</td>
<td>23.37</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>925</td>
<td>10,068</td>
<td>54.32</td>
<td>53.52</td>
<td>1.49</td>
<td>10.00</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>730</td>
<td>13,718</td>
<td>84.63</td>
<td>84.14</td>
<td>0.22</td>
<td>1.35</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>1803</td>
<td>55</td>
<td>99.63</td>
<td>100.00</td>
<td>0.37</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>50</td>
<td>679</td>
<td>1379</td>
<td>48.43</td>
<td>49.25</td>
<td>1.67</td>
<td></td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>1398</td>
<td>11,389</td>
<td>60.27</td>
<td>59.00</td>
<td>2.15</td>
<td>12.00</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>2073</td>
<td>13,522</td>
<td>62.07</td>
<td>62.14</td>
<td>0.11</td>
<td>8.88</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>1327</td>
<td>220</td>
<td>90.31</td>
<td>92.94</td>
<td>2.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>3088</td>
<td>72</td>
<td>99.88</td>
<td>100.00</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td></td>
<td>3088</td>
<td>19,965</td>
<td>100.00</td>
<td>100.00</td>
<td>2.83</td>
<td>23.37</td>
</tr>
<tr>
<td>Min</td>
<td></td>
<td>12</td>
<td>0</td>
<td>22.87</td>
<td>22.87</td>
<td>−13.37</td>
<td>1.35</td>
</tr>
<tr>
<td>Av</td>
<td></td>
<td>599</td>
<td>4698</td>
<td>68.17</td>
<td>68.29</td>
<td>0.05</td>
<td>7.42</td>
</tr>
</tbody>
</table>
Cplex did not terminate because of computer memory problems, HCI found a better solution than CPLEX. Note that in these cases, CPLEX was left to run for long times, ranging from approximately 2 h and 50 min to 5 h and 30 min. From a different point of view, use of HCI solutions as bounds for CPLEX might help, but will not reduce the CPLEX run time importantly, because CPLEX takes usually a long time when branching over nodes that correspond to objective values that are close to the optimal solution.

There is also difference between the run times. Although there are some instances in which CPLEX terminated in a time that is shorter than the time it takes to HCI to solve the same instance, in average CPLEX requires 8 times more time than HCI and in the worst cases, it required a time that is between 95 and 167 times longer than the time required by HCI.

6. Conclusions

There is a large class of combinatorial problems in which \( p \) elements must be selected from a large set, in such a way that some criterion is optimized. The \( p \)-median, the \( p \)-center, the many different versions and variants of the MCLP are all examples of these problems in the facility location context.

We offer first some general rules for rewriting the combinatorial problems described above. These rules make it easier and faster to find solutions, when using any procedure that uses repeated objective evaluation. Examples of these procedures are the one-opt and two-opt exchange heuristics, genetic algorithms and so on. The formulations obtained by using these rules are in general nonlinear. These nonlinear formulations allow a faster and easier evaluation of the quality of the solutions. As a conclusion, we note that in some cases, a location model can be formulated in different forms, all of them carrying the same logical meaning. Nonlinear formulations are currently rejected, because they usually are too difficult to solve. However, we have shown that in some cases, nonlinear formulations are the best choice, particularly when we are dealing with heuristic procedures.

We next propose a two-stage procedure called HCI. This heuristic method can, in many instances, solve problems consisting of the selection of \( p \) elements, with good or even optimal solutions. In its first stage, HCI uses a very simple procedure repeated several times with random starts. In its second stage, a more elaborate method is used, starting from the \( s \) best solutions found in the first stage, and searching over a reduced solution set obtained in the first stage. As opposed to previously published heuristics in which each element can be selected only once, HCI is capable of picking repeatedly the same element from the initial set. Using HCI, the problems need not be linear integer problems or even linearizable. We conclude that HCI, together with an appropriate formulation, can be applied to a broader class of problems.

Finally, we apply this algorithm to the MALP, a model that is of use in the design of emergency medical services. This problem is solved using LP + B&B and also HCI. In many instances, the run time of the exact method is excessive, while HCI can find near-optimal solutions in much less time. Furthermore, in some cases, LP + B&B does not terminate on a PC, while HCI finds good solutions. In some of these cases, the solution found by HCI is the best. In conclusion, HCI, together with a nonlinear, condensed formulation, can be successfully applied to many location models.

Acknowledgments

This research was supported by a grant from the David and Lucille Packard Foundation, Interdisciplinary Science Program. We gratefully acknowledge their support. The first author also gratefully acknowledges partial funding by FONDECYT, project 1040606.

References


