Hindawi
Complexity
Volume 2017, Article ID 1702506, 15 pages
https://doi.org/10.1155/2017/1702506

Research Article

Parameter Tuning for Local-Search-Based Matheuristic Methods

Guillermo Cabrera-Guerrero,[1] Carolina Lagos,[1] Carolina Castañeda,[2] Franklin Johnson,[3] Fernando Paredes,[4] and Enrique Cabrera[5]

[1] *Pontificia Universidad Católica de Valparaíso, 2362807 Valparaíso, Chile*
[2] *Instituto Tecnológico Metropolitano, Calle 73 No. 76A-374, Vía al Volador, Medellín, Colombia*
[3] *Universidad de Playa Ancha, Casilla 34-V, Valparaíso, Chile*
[4] *Universidad Diego Portales, 8370109 Santiago, Chile*
[5] *CIMFAV-Facultad de Ingeniería, Universidad de Valparaíso, 2374631 Valparaíso, Chile*

Correspondence should be addressed to Guillermo Cabrera-Guerrero; guillermo.cabrera@pucv.cl

Received 20 June 2017; Revised 2 October 2017; Accepted 25 October 2017; Published 31 December 2017

Academic Editor: Kevin Wong

Algorithms that aim to solve optimisation problems by combining heuristics and mathematical programming have attracted researchers' attention. These methods, also known as *matheuristics*, have been shown to perform especially well for large, complex optimisation problems that include both integer and continuous decision variables. One common strategy used by matheuristic methods to solve such optimisation problems is to divide the main optimisation problem into several subproblems. While heuristics are used to seek for promising subproblems, exact methods are used to solve them to optimality. In general, we say that both mixed integer (non)linear programming problems and combinatorial optimisation problems can be addressed using this strategy. Beside the number of parameters researchers need to adjust when using heuristic methods, additional parameters arise when using matheuristic methods. In this paper we focus on one particular parameter, which determines the size of the subproblem. We show how matheuristic performance varies as this parameter is modified. We considered a well-known NP-hard combinatorial optimisation problem, namely, the capacitated facility location problem for our experiments. Based on the obtained results, we discuss the effects of adjusting the size of subproblems that are generated when using matheuristics methods such as the one considered in this paper.

1. Introduction

Solving mixed integer programming (MIP) problems as well as combinatorial optimisation problems is, in general, a very difficult task. Although efficient exact methods have been developed to solve these problems to optimality, as the problem size increases exact methods fail to solve it within an acceptable computational time. As a consequence, nonexact methods such as heuristic and metaheuristic algorithms have been developed to find good quality solutions. In addition, hybrid strategies combining different nonexact algorithms are also promising ways to tackle complex optimisation problems. Unfortunately, algorithms that do not consider exact methods cannot give us any guarantee of optimality and, thus, we do not know how good (or bad) solutions found by these methods are.

One hybrid strategy that combines nonexact methods are memetic algorithms, which are population-based metaheuristics that use an evolutionary framework integrated with local search algorithms [1]. Memetic algorithms provide lifetime learning process to refine individuals in order to improve the obtained solutions every iteration or generation; their applications have been grown significantly over the years in several NP-hard optimisation problems [2]. These algorithms are part of the paradigm of memetic computation, where the concept of meme is used to automate the knowledge transfer and reuse across problems [3]. A large number of memetic algorithms can be found in the literature. Depending on its implementation, memetic algorithms might (or might not) give guarantee of local optimality: roughly speaking, if heuristic local search algorithms are considered, then no optimality guarantee is given; if exact methods are considered as the local optimisers, then local optimality could be ensured.

To overcome the situation described above, hybrid methods that combine heuristics and exact methods to solve optimisation problems have been proposed. These methods, also known as matheuristics [4], have been shown to perform better than both heuristic and mathematical programming methods when they are applied separately. The idea of combining the power of mathematical programming with flexibility of heuristics has gained attention within researchers' community. We can found matheuristics attempting to solve problems arising in the field of logistics [5–9], health care systems [10–13], and pure mathematics [14, 15], among others. Matheuristics have been demonstrated to be very effective in solving complex optimisation problems. Some interesting surveys on matheuristics are [16, 17]. Although there is some overlapping between memetic algorithms and matheuristic ones, in this paper we have chosen to label the studied strategy as matheuristic, as we think matheuristics definition better fits the framework we are interested in.

Because of their complexity, MIP problems as well as combinatorial optimisation problems are often tackled using matheuristic methods. One common strategy to solve this class of optimisation problems is to divide the main optimisation problem into several subproblems. While heuristics are used to seek for promising subproblems, exact methods are used to solve them to optimality. One advantage of this approach is that it does not depend on the (non)linearity of the resulting subproblem. Instead, it has been pointed out that it is desirable that the resulting subproblem would be convex [11]. Having a convex subproblem would allow us to solve it to optimality, and thus comparing solutions obtained at each subproblem becomes more senseful. This strategy has been successfully applied to problems arising in fields as diverse as logistics and radiation therapy.

In this paper we aim to study the impact of parameter tuning on the performance of matheuristic methods as the one described above. To this end, the well-known capacitated facility location problem is used as an application of hard combinatorial optimisation problem. To the best of our knowledge, no paper has focused on parameter tuning for matheuristic methods.

This paper is organised as follows: Section 2 shows the general matheuristic framework we consider in this paper. Details on the algorithms that are used in this study are also shown in this section. In Section 3 the capacitated facility location problem is introduced and its mathematical model is described in Section 3.1. The experiments performed in this study are presented and the obtained results are discussed in Section 3.2. Finally, in Section 4 some conclusions are presented and the future work is outlined.

2. Matheuristic Methods

This section is twofold. We start by describing a general matheuristic framework that is used to solve both MIP problems and combinatorial optimisation problems and how it is different from other commonly used approaches such as memetic algorithms and other evolutionary approaches. After that, we present the local-search-based algorithms we consider in this work to perform our experiments. We finish this section by introducing the parameter we will be focused on in this study.

2.1. General Framework. Equations (1a) to (1f) show the general form of MIP problems. Hereafter we will refer to this problem as the MIP(x, y) problem or the *main problem*.

$$\text{MIP}(x,y): \min_{x,y} \quad f(x,y) \tag{1a}$$

$$\text{s.t.} \quad g_j(x,y) \le b_j \quad \text{for } j = 1,\dots,m, \tag{1b}$$

$$\overline{g_j}(x) \le \overline{b_j} \quad \text{for } j = 1,\dots,\overline{m}, \tag{1c}$$

$$\overline{\overline{g_j}}(y) \le \overline{\overline{b_j}} \quad \text{for } j = 1,\dots,\overline{\overline{m}}, \tag{1d}$$

$$y \in \{0,1\}^o, \tag{1e}$$

$$x \in \mathbb{R}^n, \tag{1f}$$

where $f(x, y)$ is an objective function, m is the number of inequality constraints on x and y, $\overline{m}$ is the number of inequality constraints on x, $\overline{\overline{m}}$ is the number of inequality constraints on y, o is the number of binary (≥ 0) decision variables y, and n is the number of continuous decision variables. Combinatorial optimisation problems, as the one we consider in this paper, can be easily obtained by either removing the continuous decision variable from the model or making it integer (i.e., $x \in \mathbb{Z}^o_{\ge}$).

Although there exist a number of exact algorithms that can find an optimal solution for the MIP(x, y), as the size of the problem increases, exact methods fail or take too long. Because of this, heuristic methods are used to obtain good quality solutions of the problem within an acceptable time. Heuristic methods cannot guarantee optimality though.

During the last two decades, the idea of combining heuristic methods and mathematical programming has received much more attention. Exploiting the advantages of each method appears to be a senseful strategy to overcome their inherent drawbacks. Several strategies have been proposed to combine heuristics and exact methods to solve optimisation problems such as the MIP(x, y) problem. For instance, Chen and Ting [18] and Lagos et al. [8] combine the well-known ant colony optimisation (ACO) algorithm and Lagrangian relaxation method to solve the single source capacitated facility location problem and a distribution network design problem, respectively. In these articles, Lagrangian multipliers are updated using ACO algorithm. Another strategy to combine heuristics and exact methods is to let heuristics seek for subproblems of MIP(x, y) which, in turn, are solved to optimality by some exact method. One alternative to obtain subproblems of MIP(x, y) is to add a set of additional constraints on a subset of binary decision variables. These constraints are of the form $y_i = 0$, with $i \in \mathcal{I}$ and $\mathcal{I}$ being the set of index that are restricted in subproblem MIP$_{\mathcal{I}}(x, y)$ (see (1a) to (1f)). The portion of binary decision variables y_i that are set to 0 is denoted by α (i.e., $\#\mathcal{I} = \alpha \times o$), with o being the number of binary variables y_i. Then, the obtained subproblem, which we call $\mathrm{MIP}_{\mathcal{I}}(x, y)$, is

$$\mathrm{MIP}_{\mathcal{I}}(x,y): \min_{x,y} \quad f(x,y) \tag{2a}$$

$$\text{s.t.} \quad g_j(x,y) \le b_j \quad \text{for } j = 1,\ldots,m, \tag{2b}$$

$$\overline{g_j}(x) \le \overline{b_j} \quad \text{for } j = 1,\ldots,\overline{m}, \tag{2c}$$

$$\overline{\overline{g_j}}(y) \le \overline{\overline{b_j}} \quad \text{for } j = 1,\ldots,\overline{\overline{m}}, \tag{2d}$$

$$y_i = 0 \quad i \in \mathcal{I}, \tag{2e}$$

$$y \in \{0,1\}^o, \tag{2f}$$

$$x \in \mathbb{R}^n. \tag{2g}$$

In this paper we assume that a constraint on the ith resource of vector y of the form $y_i = 0$, that is, $i \in \mathcal{I}$, means that such a resource is not available for subproblem $\mathrm{MIP}_{\mathcal{I}}(x, y)$. We can note that, as the number of constrained binary decision variables y_i associated with $\mathcal{I}$ increases, that is, as $(1-\alpha)$ increases, the subproblem $\mathrm{MIP}_{\mathcal{I}}(x, y)$ becomes smaller. Similarly, as the number of constrained binary decision variables y_i associated with $\mathcal{I}$ gets smaller, that is, $(1-\alpha)$ decreases, the subproblem $\mathrm{MIP}_{\mathcal{I}}(x, y)$ gets larger, as there are more available resources. It is also easy to note that, as $(1-\alpha)$ increases, we can obtain optimal solutions of the corresponding subproblem $\mathrm{MIP}_{\mathcal{I}}(x, y)$ relatively faster. However, quality of the obtained solutions is usually impaired as the solution space is restricted by the additional constraints associated with the set $\mathcal{I}$. Similarly, as $(1-\alpha)$ gets smaller, obtaining optimal solutions of the associated subproblems might take longer but the quality of the obtained solutions is, in general, greatly improved. Finally, when $\alpha = 0$ we have that subproblem $\mathrm{MIP}_{\mathcal{I}}(x, y)$ is identical to the main problem, $\mathrm{MIP}(x, y)$. We assume that optimal solution of subproblem $\mathrm{MIP}_{\mathcal{I}}(x, y)$, $(\hat{x}, \hat{y})$, is also feasible for the $\mathrm{MIP}(x, y)$ problem. Moreover, we can note that there must be a minimal set $\mathcal{I}$, for which an optimal solution $(\hat{x}, \hat{y})$ of the associated subproblem $\mathrm{MIP}_{\mathcal{I}}(x, y)$ is also an optimal solution (x^*, y^*) of the main problem $\mathrm{MIP}(x, y)$.

In some cases, the value of α might be predefined by the problem that is being solved. For instance, the problem of finding the best beam angle configuration for radiation delivery in cancer treatment (beam angle optimisation problem) usually sets the number of beams to be used in a beam angle configuration (see Cabrera-Guerrero et al. [11], Li et al. [19], and Li et al. [20]). This definition is made by the treatment planner and it does not take into account the algorithm performance but clinical aspects. Unlike this kind of problems, there are many other problems where the value of α is not predefined and, then, setting it to an efficient value is important for the algorithm performance. Figure 1 shows the interaction between the heuristic and the exact method.

One distinctive feature of matheuristics is that there exists an interaction between the heuristic method and the exact method. In the method depicted in Figure 1 we have that, on one hand, the heuristic method influences the solver by passing onto it a set of constraints, $\mathcal{I}$, that defines the subproblem to be solved by the exact method. On the other hand, we have that the solution obtained by the solver is returned to the heuristic method such that it can be used to influence the selection of the elements in the next set of constraints; that is, the solution of a subproblem might be used by the heuristic to obtain some useful information to generate next subproblems.

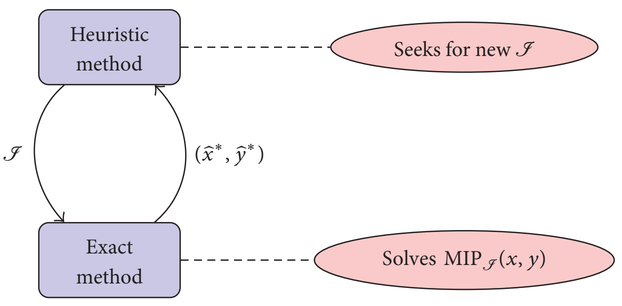

Heuristic method
Seeks for new $\mathcal{I}$
$\mathcal{I}$
$(\hat{x}^*, \hat{y}^*)$
Exact method
Solves $\mathrm{MIP}_{\mathcal{I}}(x, y)$

Figure 1: Interaction between a heuristic method and an exact method.

As mentioned above, there is one parameter that is not part of the set of parameters of the heuristic method nor part of the set of parameters of the exact method. This parameter, which we call α, only comes up after these two methods are posed together. Then, in this paper we are interested in how the choice of α can modify the performance of the proposed strategy in terms of the quality of the obtained solutions of the main problem. Since many different matheuristic frameworks might be proposed to solve the $\mathrm{MIP}(x, y)$ problem, we restrict this study to local-search-based matheuristics. Thus, in this study, three local-search-based matheuristic algorithms are implemented and their results compared. Additionally, we implement a very simple method which we call "blind algorithm" as a baseline for this study. Next sections explain these algorithms.

2.2. Local Search Algorithms. As mentioned in previous sections, the aim of this paper is not to provide a "state-of-the-art" algorithm to solve MIP problems but, instead, to study the effect that changing the size of subproblems has on the quality of the obtained solutions when using local-search-based matheuristic methods as the one described in Section 2.1. Thus, three local search algorithms are implemented, namely, steepest descent (SD), next descent (ND), and tabu search (TS). Local search algorithms need a neighbourhood $\mathcal{N}$ to be defined by means of a neighbourhood movement. We define the same neighbourhood movement for the all three methods.

Since the local search algorithms move on the subproblem space, that is, the local search algorithms look for promising subproblems $\mathrm{MIP}_{\mathcal{I}}(x, y)$, or, equivalently, look for promising sets $\mathcal{I}$, then we need to define a neighbourhood within the same search space. Thus, the neighbour set of $\mathcal{I}$, which we denote by $\mathcal{N}(\mathcal{I})$ or simply $\mathcal{I}'$, corresponds to all resources y_i such that they either are not considered in

```
     Input: $\alpha$ (portion of the resources included in $\mathcal{I}$)
     Output: $(\hat{x}, \hat{y})$ (locally optimal solution)
(1) begin
(2)     $k = 0$;
(3)     $\mathcal{I}_k$ = selectBinaryVariablesRandomly($\alpha$);
(4)     $(\hat{x}, \hat{y})$ = solve $\mathrm{MIP}_{\mathcal{I}_k}(x, y)$;
(5)     repeat
(6)         localOptimum = true;
(7)         $k = k + 1$;
(8)         foreach $\mathcal{I}'_{k-1} \in \mathcal{N}(I_{k-1})$ do
(9)             $(x', y')$ = solve $\mathrm{MIP}_{\mathcal{I}'_{k-1}}(x, y)$;
(10)            If $\mathrm{MIP}(x', y') < \mathrm{MIP}(\hat{x}, \hat{y})$ then
(11)                $(\hat{x}, \hat{y}) = (x', y')$;
(12)                $\mathcal{I}_k = \mathcal{I}'_{k-1}$;
(13)                localOptimum = false;
(14)    until localOptimum;
(15)    return $(\hat{x}, \hat{y})$;
```

Algorithm 1: Steepest descent algorithm.

the optimal solution of $\mathrm{MIP}_{\mathcal{I}}(x, y)$ or are actually part of $\mathcal{I}$. Since the number of resources y_i that meet these criteria might be above or below the number of elements set $\mathcal{I}'$ must contain according to the parameter α that is being considered, we randomly select among the variables that meet these criteria until the neighbour set $\mathcal{I}'$ is completed. This neighbourhood movement ensures that those resources y_i that are part of the optimal solution of subproblem $\mathrm{MIP}_{\mathcal{I}}(x, y)$, that is, $y_i \notin \mathcal{I}$ and $y_i = 1$, will continue to be available for sets $\mathcal{I}' \in \mathcal{N}(\mathcal{I})$. Thus, the set of resources y_i that can potentially be part of sets $\mathcal{I}' \in \mathcal{N}(\mathcal{I})$ is defined as follows:

$$\mathcal{I}' \in \mathcal{N}(\mathcal{I}) = \{y_i : y_i \in \mathcal{I} \text{ or } (y_i \notin \mathcal{I},\ \hat{y}_i = 0)\}, \quad (3)$$

where $(\hat{x}, \hat{y})$ is the optimal solution of $\mathrm{MIP}_{\mathcal{I}}(x, y)$. The initial set $\mathcal{I}$ local search methods start with is set randomly for the all three algorithms implemented in this paper.

2.2.1. Steepest Descent Algorithm. The steepest descent algorithm starts with an initial $\mathcal{I}_0$ for which its associated subproblem, $\mathrm{MIP}_{\mathcal{I}_0}(x, y)$, is labelled as the current subproblem. For this current subproblem, its entire neighbourhood $\mathcal{N}(\mathcal{I}_0)$ is generated, and the neighbour that leads to the best objective function value is selected. If the best neighbour subproblem is better than the current one, then the best neighbour is set as the new current solution. If the best neighbour is not better than the current subproblem, then the algorithm stops and the optimal solution of the current subproblem is returned as a locally optimal solution of the main $\mathrm{MIP}(x, y)$ problem. Algorithm 1 shows the pseudocode for the steepest descent algorithm implemented in this paper.

Although the steepest descent algorithm can be considered, in general, a deterministic algorithm, in the sense that, given an initial solution, it converges to the same local optima, in our implementation the algorithm does not visit all possible neighbours and, therefore, it becomes a stochastic local search. Since only one local optimum is generated at each run, we repeat the algorithm until the time limit is reached. Same is done for both ND and TS algorithms we introduce next.

2.2.2. Next Descent Algorithm. As mentioned in Section 2.2.1, the steepest descent might take too long to converge if the size of the neighbourhood is too large or solving an individual subproblem is too time-consuming. Thus, we include in this paper a local search algorithm called next descent that aims to converge faster than the steepest descent, without a major impact on the solution quality. Algorithm 2 shows the next descent method.

Just as in the steepest descent algorithm, the next descent algorithm starts with an initial solution, which is labelled as the current solution. Then, a random element from the neighbourhood of the current solution is selected and solved, and the objective function value of its optimal solution is compared to the objective function value of the current solution. Like in the SD algorithm, if the neighbour solution is not better than the current solution, another randomly selected set $\mathcal{I}'$ from the neighbourhood $\mathcal{N}(\mathcal{I}_{k-1})$ is generated and compared to the current solution. Unlike the SD algorithm, in the ND algorithm, if the neighbour is better than the current solution, then the neighbour is labelled as the new current solution and no other $\mathcal{I}' \in \mathcal{N}(\mathcal{I}_{k-1})$ is visited. The algorithm repeats these steps until the entire neighbourhood has been computed with no neighbour resulting in a better solution than the current one, in which case the algorithm stops.

2.2.3. Tabu Search Algorithm. Unlike the algorithms described in the previous sections, tabu search is a local search technique guided by the use of adaptive or flexible memory structures [5], and thus, it is inherently a stochastic local search algorithm. The variety of the tools and search principles introduced and described in [21] are such that the TS can be considered as the seed of a general framework for modern heuristic search [22]. We include TS as it has

```
     Input: $\alpha$ (portion of the resources included in $\mathcal{I}$)
     Output: $(\hat{x}, \hat{y})$ (locally optimal solution)
(1)  begin
(2)    $k = 0$;
(3)    $\mathcal{I}_k$ = selectBinaryVariablesRandomly($\alpha$);
(4)    $(\hat{x}, \hat{y})$ = solve $\mathrm{MIP}_{\mathcal{I}_k}(x, y)$;
(5)    repeat
(6)      localOptimum = true;
(7)      $k = k + 1$;
(8)      foreach $\mathcal{I}'_{k-1} \in \mathcal{N}(I_{k-1})$ do
(9)        $(x', y')$ = solve $\mathrm{MIP}_{\mathcal{I}'_{k-1}}(x, y)$;
(10)       if $\mathrm{MIP}(x', y') < \mathrm{MIP}(\hat{x}, \hat{y})$ then
(11)         $(\hat{x}, \hat{y}) = (x', y')$;
(12)         $\mathcal{I}_k = \mathcal{I}'_{k-1}$;
(13)         localOptimum = false;
(14)         break;
(15)   until localOptimum;
(16)   return $(\hat{x}, \hat{y})$;
```

Algorithm 2: Next descent algorithm.

been applied to several combinatorial optimisation problems (see, e.g., [5, 23–27]) including, of course, mixed integer programming problems as the one we consider in this study.

As in the algorithms introduced above, TS also starts with an initial set of constraints $\mathcal{I}_0$ for which its associated subproblem $\mathrm{MIP}_{\mathcal{I}_0}(x, y)$ is solved. Then, as in the SD algorithm, the "entire" neighbourhood of the current solution is computed. As explained before, since the neighbourhood $\mathcal{N}$ has a stochastic component, it is actually not possible to generate the entire neighbourhood of a solution; however, we use the term "entire" to stress the fact that all the *generated* neighbours of $\mathcal{N}$ are solved. This is different from the ND algorithm, where not all the generated neighbours are necessarily solved. The number of generated neighbours, ns, is, as in the algorithms above, set equal to 10. Moreover, TS implements a list called *tabu list* that aims to avoid cycles during the search. Each time a neighbourhood is generated, the warehouses removed from $I_{(k-1)}$ are marked as tabu. Once we have solved the subproblems $\mathrm{MIP}_{\mathcal{I}'_k}(x, y)$, with $\mathcal{I}'_k \in \mathcal{N}(I_k)$, its optimal solutions are ranked and the one with the best objective function value is chosen as candidate solution for the next iteration. If the candidate solution was generated using a movement within the tabu list, it should be discarded, and the next best neighbour of the list should be chosen as the new candidate solution. However, there is one exception to this rule: if the candidate solution is within the tabu list but its objective function value is better than the best objective function found so far by the algorithm, then the so called *aspiration criterion* is invoked and the candidate solution is set as the new current solution and passed on to the next iteration.

Unlike the algorithms introduced above, TS does not require next current solution to be better than the previous one. This means that it is able to avoid local optimal by choosing solutions that are more expensive as they allows the algorithm to visit other (hopefully promising) areas in the search space. However, in case the algorithm cannot make any improvement after a predefined number of iterations, a diversification mechanism is used to get out from low-quality neighbourhoods and "jump" to other neighbourhoods. The diversification mechanism implemented here is a restart method, which set the current solution to a randomly generated solution without losing the best solution found so far. Termination criterion implemented here is the time limit.

The tabu search implemented in this paper is as follows.

As Algorithm 3 shows, tabu search requires the following parameters:

(i) *Time limit*: total time the algorithm will perform.

(ii) *DiversBound*: total number of iteration without improvements on the best solution before diversification criterion (restart method) is applied.

(iii) *TabuListSize* (ts): size of *tabuList*. Number of iterations for which a specific movement remains banned.

2.2.4. Blind Algorithm. We finally implement a very simple heuristic method we call *blind* as it moves randomly at each iteration. Pseudocode of this method is presented in Algorithm 4.

As we can see, no additional intelligence is added to the blind algorithm. It is just a random search that, after a predefined number of iterations (or any other "stop criterion"), returns the best solution it found during its search. Thus, we can consider this algorithm as a baseline of this study.

We apply the all four algorithms described in this section to two prostate cases. Details on this case and the obtained results are presented in the next section.

2.3. Subproblem Sizing. All the algorithms above perform very different as the value of the input parameter α varies. On the one hand, setting α to a very small value provokes that either the solver fails to solve the subproblem because of lack of memory or it takes too long to find an optimal

```
Input: $\alpha$ (portion of the resources included in $\mathcal{I}^k$)
Output: $(\hat{x}, \hat{y})$ (approximately optimal solution)
(1) begin
(2)   $k = 0$;
(3)   $\mathcal{I}_k$ = selectBinaryVariablesRandomly($\alpha$);
(4)   $(x^c, y^c)$ = solve $\text{MIP}_k(x, y)$;  // set current sol
(5)   $(\hat{x}, \hat{y}) = (x^c, y^c)$; // set best sol
(6)   $\mathcal{Y} = \emptyset$; // tabu list initially empty
(7)   $noImprovementCounter = 0$;
(8)   repeat
(9)     $\mathcal{C} = \emptyset$;
(10)    $k = k + 1$;
(11)    foreach $\mathcal{I}'_{k-1} \in \mathcal{N}(I'_{k-1})$ do
(12)      $(x^n, y^n)$ = solve $\text{MIP}_{\mathcal{I}'_{k-1}}(x, y)$;
(13)      $\mathcal{C} = \mathcal{C} \cup \{(x^n, y^n)\}$;
(14)    Sort($\mathcal{C}$);
(15)    foreach $\{(x^n, y^n)\} \in \mathcal{C}$ do
(16)      if $isTabu((x^n, y^n))$ then
(17)        if $\text{MIP}(x^n, y^n) < \text{MIP}(\hat{x}, \hat{y})$ then
(18)          $\mathcal{Y} = updateTabuList(\mathcal{Y}, (x^c, y^c), (x^n, y^n), \text{ts})$;
(19)          $(x^c, y^c) = (x^n, y^n)$;
(20)          $(\hat{x}, \hat{y}) = (x^n, y^n)$;
(21)          $\mathcal{I}_k = \mathcal{I}^n_{k-1}$;
(22)          $noImprovementCounter = 0$;
(23)          break;
(24)      else
(25)        $\mathcal{Y} = updateTabuList(\mathcal{Y}, (x^c, y^c), (x^n, y^n))$;
(26)        $(x^c, y^c) = (x^n, y^n)$;
(27)        $\mathcal{I}_k = \mathcal{I}^n_{k-1}$ if $\text{MIP}(x^n, y^n) < \text{MIP}(\hat{x}, \hat{y})$ then
(28)          $(\hat{x}, \hat{y}) = (x^n, y^n)$;
(29)          $noImprovementCounter = 0$;
(30)        else
(31)          $noImprovementCounter++$;
(32)        break;
(33)    if $noImprovementCounter \geq diversBound$ then
(34)      $\mathcal{I}^k$ = selectBinaryVariablesRandomly($\alpha$);
(35)  until time limit is reached;
(36)  return $(\hat{x}, \hat{y})$;
```

Algorithm 3: Tabu search algorithm.

```
// To find an (approximately) optimal solution for MIP($x, y$)
Input: $\alpha$ (portion of the resources included in $\mathcal{I}^k$)
Output: $(\hat{x}, \hat{y})$ (approximately optimal solution)
(1) begin
(2)   $k = 0$;
(3)   $\mathcal{I}_k$ = selectBinaryVariablesRandomly($\alpha$);
(4)   $(\hat{x}, \hat{y})$ = solve ($\text{MIP}_{\mathcal{I}_k}(x, y)$);
(5)   while !stopCriterion do
(6)     $k = k + 1$;
(7)     $\mathcal{I}_k$ = selectBinaryVariablesRandomly($\alpha$);
(8)     $(x^c, y^c)$ = solve ($\text{MIP}_{\mathcal{I}_k}(x, y)$);
(9)     if $\text{MIP}(x^c, y^c) < \text{MIP}(\hat{x}, \hat{y})$ then
(10)      $(\hat{x}, \hat{y}) = (x^c, y^c)$;
(11)      $\mathcal{I}_k = \mathcal{I}_{k-1}$
(12)  return $(\hat{x}, \hat{y})$;
```

Algorithm 4: Matheuristic method using the blind algorithm.

solution of the subproblem. On the other hand, setting α close to the total number of binary decision variables provokes that the algorithm fails to find a solution for the generated subproblems as there is no feasible solution to it (i.e., the subproblems are too restrictive). Thus, we have to find a value of α such that exact methods can solve the obtained subproblems within an acceptable time. Further, α should allow local search methods to iterate as much as needed. Solving the obtained subproblems within few seconds is critical to matheuristic methods as they usually need several iterations before to converge to a good quality solution. This is especially true for matheuristics that consider local search or population-based heuristic methods. Then, there is a trade-off between the quality of the solution of subproblems and the time that is needed to generate such solutions. Therefore, finding a value of α that gives us a good compromise between these two aspects is critical for the overall performance of the matheuristic methods explained before. It is interesting that the problem of finding *efficient* values of α might be seen as a multiobjective optimisation problem.

In next section we explain the experiments that we perform to study how the choice of α hits local-search-based matheuristics performance. Based on the results, we draft some guidelines to set value of parameter α at the end of next section.

3. Computational Experiments

This section starts briefly introducing the problem we consider in this paper. Then, the experiments performed here are presented and their results are discussed.

3.1. The Capacitated Facility Location Problem. The capacitated facility location problem (CFLP) is a well-known problem in combinatorial optimisation. The CFLP has been shown to be NP-hard [28]. The problem consists of selecting specific sites at which to install plants, warehouses, and distribution centres while assigning customers to service facilities and interconnecting facilities using flow assignment decisions [23]. In this study we consider the CFLP problem to evaluate the performance of a very simple matheuristic algorithm. We consider a two-level supply chain in which a single plant serves a set of warehouses, which in turn serve a set of end customers or retailers. Figure 2 shows the basic configuration of our supply chain. Thus, the goal is to find a set of locations that serves the entire set of customers in an optimal way. As Figure 2 shows, each customer (or cluster) is served only by one warehouse.

The optimisation model considers the *installation* cost (i.e., the cost associated with opening a specific warehouse) and transportation or *allocation* cost (i.e., the cost of transporting one item from a warehouse to a customer). The mathematical model for the CFLP is

$$\text{CFLP}(x, y): \min_{x,y} \quad \sum_{i=1}^{n} (f_i y_i) + \sum_{i=1}^{n}\sum_{j=1}^{m} (c_{ij} d_j x_{ij}) \tag{4a}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} d_j x_{ij} \le I_i^{\text{cap}} y_i \quad \forall i = 1, \ldots, n, \tag{4b}$$

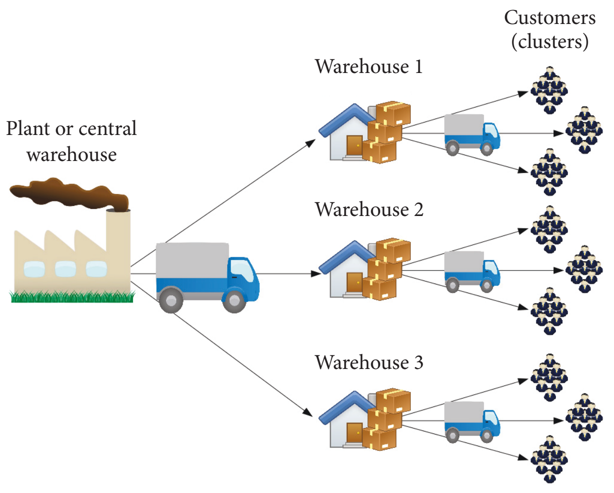


Figure 2: Distribution network structure considered in this study. It consists of one central plant, a set of potential warehouses, and a set of customers or retailers [23].

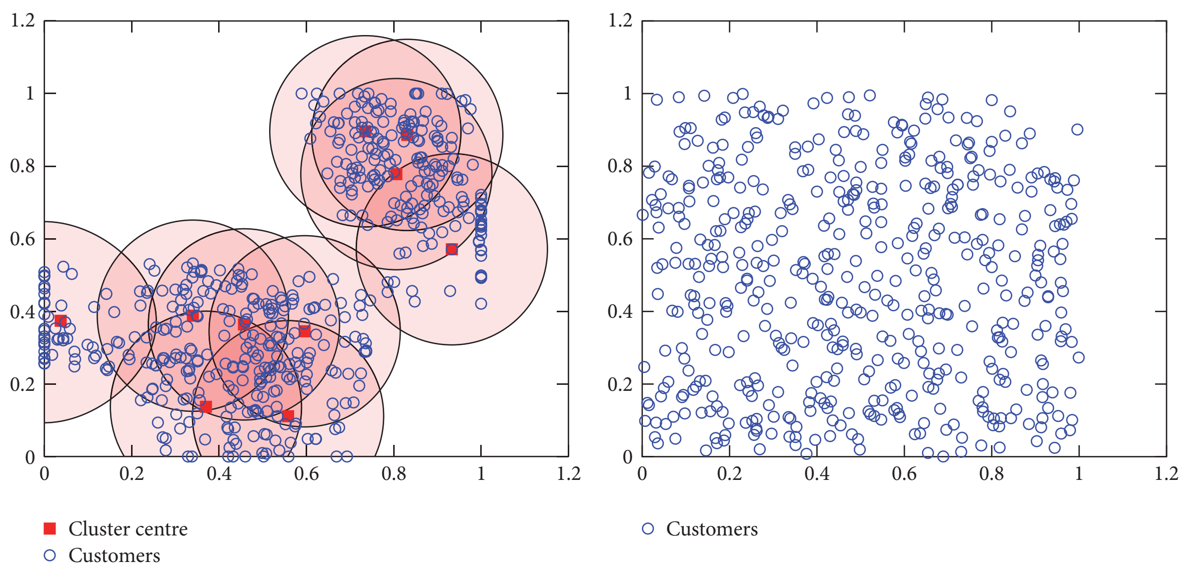
$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j = 1, \ldots, m, \tag{4c}$$

$$x_{ij} \le y_i \quad \forall i = 1, \ldots, n; \ \forall j = 1, \ldots, m, \tag{4d}$$

$$y_i, x_{ij} \in \{0, 1\} \quad \forall i = 1, \ldots, n; \ \forall j = 1, \ldots, m. \tag{4e}$$

Equation (4a) is the total system cost. The first term is the fixed setup and operating cost when opening warehouses. The second term is the daily transport cost between warehouse and customers which depends on the customer demand d and distance c_{ij} between warehouse i and customer j. Inequality (4b) ensures that total demand of warehouse i will never be greater than its capacity I_i^{cap}. Equation (4c) ensures that customers are served by only one warehouse. Equation (4d) makes sure that customers are only allocated to available warehouses. Finally, (4e) states integrality (0–1) for the binary variables x_{ij} and y_i. Other versions of the CFLP relax this constraints by making $x \in \mathbb{R}^{n \times m}$ with $0 \le x_{ij} \le 1$. In that case, the CFLP would be just as the MIP(x, y) problem.

3.2. Experiments. In this paper three benchmarks for the CFLP are considered. The first benchmark corresponds to problem sets $\mathscr{A}$, $\mathscr{B}$, and $\mathscr{C}$ from the OR Library [29]. Instances in problem sets $\mathscr{A}$, $\mathscr{B}$, and $\mathscr{C}$ consider 1,000 clients and 100 warehouses. Warehouses capacity for instances $\mathscr{A}_1$, $\mathscr{A}_2$, $\mathscr{A}_3$, and $\mathscr{A}_4$ are equal to 8,000; 10,000; 12,000; and 14,000, respectively. Warehouses capacity for instances $\mathscr{B}_1$, $\mathscr{B}_2$, $\mathscr{B}_3$, and $\mathscr{B}_4$ are equal to 5,000; 6,000; 7,000; and 8,000, respectively. Warehouses capacity for instances $\mathscr{C}_1$, $\mathscr{C}_2$, $\mathscr{C}_3$, and $\mathscr{C}_4$ are equal to 5,000; 5,750; 6,500; and 7,250, respectively. Table 1 shows these instances and their corresponding optimal values obtained by the MILP solver. Column t shows



(a) Clients distributed in clusters

(b) Clients uniformly distributed

Figure 3: Example of the instances considered in this study.

Table 1: Instances for the first benchmark used in this study (OR Library).

		OR Library		
Instances	$\|I\| \times \|J\|$	I^{cap}	$f(x^*)$	t (sec)
$\mathscr{A}_1$	$1{,}000 \times 100$	8,000	19,240,822.45	1,745
$\mathscr{A}_2$	$1{,}000 \times 100$	10,000	18,438,046.54	1,778
$\mathscr{A}_3$	$1{,}000 \times 100$	12,000	17,765,201.95	199
$\mathscr{A}_4$	$1{,}000 \times 100$	14,000	17,160,439.01	8
$\mathscr{B}_1$	$1{,}000 \times 100$	5,000	13,656,379.58	136
$\mathscr{B}_2$	$1{,}000 \times 100$	6,000	13,361,927.45	151
$\mathscr{B}_3$	$1{,}000 \times 100$	7,000	13,198,556.43	256
$\mathscr{B}_4$	$1{,}000 \times 100$	8,000	13,082,516.50	27
$\mathscr{C}_1$	$1{,}000 \times 100$	5,000	11,646,596.97	155
$\mathscr{C}_2$	$1{,}000 \times 100$	5,750	11,570,340.29	70
$\mathscr{C}_3$	$1{,}000 \times 100$	6,500	11,518,743.74	31
$\mathscr{C}_4$	$1{,}000 \times 100$	7,250	11,505,767.39	24

the time needed by the MILP solver to reach the optimal solution.

The second benchmark is a set of instances where clients and warehouses are uniformly distributed over an imaginary square of 100×100[distance units]^2 (see Figure 3(a)). We call this set DS1$_U$. The number of clients considered in instances belonging to set DS1$_U$ ranges from 500 to 1000 (500, 600, 700, 800, 900, and 1,000) while the number of warehouses considered varies between 500 and 1,000. Thus, set DS1$_U$ consists of 12 problem classes ($500 \times 500, 500 \times 600, \ldots, 500 \times 900, 500 \times 1000, 1000 \times 500, \ldots, 1000 \times 1000$). For each problem class, 10 instances are randomly generated using the procedure proposed in [30] and that was also used in [5, 23]. We do this in order to minimise any instance dependant effect. Table 2 shows the average values for each class of problems.

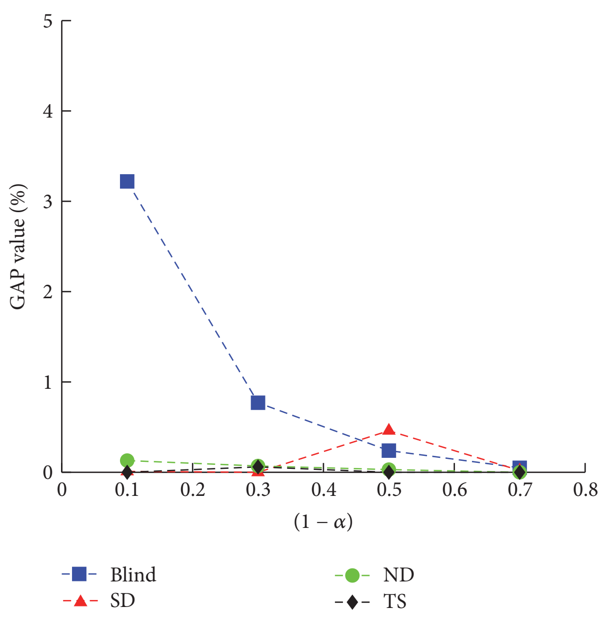
Finally, a third benchmark consisting on clients that are organised in clusters is considered. We call this benchmark DS1$_C$ (see Figure 3(b)). Instances in DS1$_C$ are generated very similar to the ones in DS1$_U$. The only difference is that clients in DS1$_C$ are not uniformly distributed and, thus, those warehouses that are within (or very close to) a cluster have an installation cost slightly higher than those warehouses that are far away from the clusters.

Table 2 shows the results obtained by the MIP solver from Gurobi when solving each instance of DS1$_C$. As we can see, the MIP solver is able to solve all the instances to optimality. Further, the solver finds the optimal solution for almost all the instances in less than 1,800 secs. Columns t_{avg}, t_{min}, and t_{max} in Table 2 show the average time and both minimum and maximum times, respectively. This is because we solve 10 different instances for each instance class. As mentioned before, we do this to avoid any instance dependent effect.

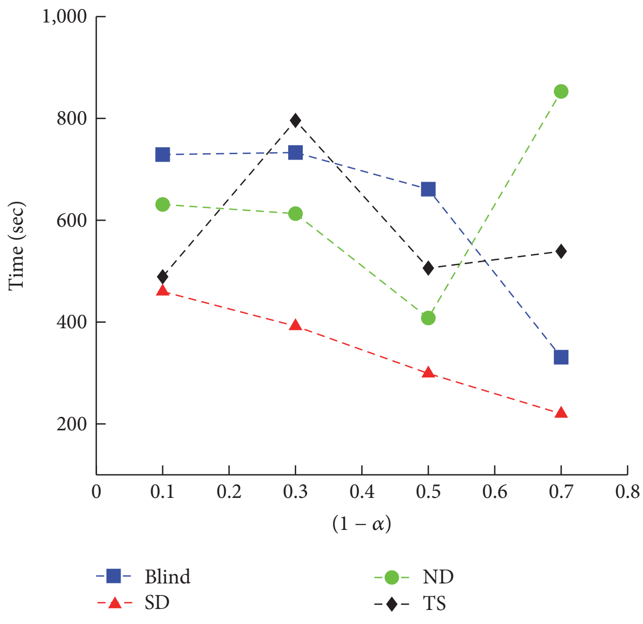
After we have solved the problem using the MIP solver, we apply the all three local-search-based matheuristics and

Table 2: Instances for the $DS1_U$ and $DS1_C$ benchmarks.

Instances		DS1_*U*				DS1_*C*			
	$\|I\| \times \|J\|$	$f(x^*)$	t_{avg}	t_{min}	t_{max}	$f(x^*)$	t_{avg}	t_{min}	t_{max}
(1)	500 × 500	120670.3	170	95	287	123900.7	191	99	410
(2)	500 × 1000	110534.5	267	194	356	113527.1	262	201	410
(3)	600 × 500	149539.9	261	105	424	149343.2	287	120	522
(4)	600 × 1000	141847.7	399	309	568	137451.4	331	255	501
(5)	700 × 500	182434.9	427	128	674	181149.3	292	96	645
(6)	700 × 1000	161882.7	726	263	1515	161872.9	596	306	1673
(7)	800 × 500	209319.5	455	227	748	207036.9	343	131	648
(8)	800 × 1000	193647.7	761	399	1679	188859.7	1040	590	1828
(9)	900 × 500	243403.7	391	179	681	236287.7	428	178	1024
(10)	900 × 1000	211302.4	1228	672	1787	215410.7	1158	507	1564
(11)	1000 × 500	270449.9	448	189	1484	265805.9	407	154	642
(12)	1000 × 1000	243625.0	1187	670	1791	239519.8	1419	588	2653



(a) Evolution of the GAP for *ORLib* $\mathscr{A}$ instances



(b) Evolution of the time for *ORLib* $\mathscr{A}$ instances

Figure 4: Average results obtained by the local search algorithms for each value of $(1 - \alpha)$ for *ORLib* $\mathscr{A}$ instances.

the blind algorithm to each instance and allow them to run for 2,000 secs. The proposed algorithms solve each instance 10 times for each value of $(1 - \alpha)$. Table 3 shows the results obtained by the local-search-based matheuristic methods for instances $\mathscr{A}$, $\mathscr{B}$, and $\mathscr{C}$ for four different values of $(1 - \alpha)$. As expected, the blind algorithm consistently obtains GAP values much higher than the other three methods. Further, local-search-based algorithms are able to find the optimal solution for the majority of the instances. While the average GAP for the blind algorithm is 1.65%, the average GAP for the SD, ND, and TS methods is 0.33%, 0.36%, and 0.15%, respectively. As expected, the best average value is obtained by the TS algorithm while difference between SD and ND algorithms is negligible. Regarding the time needed by the algorithms to converge, the blind algorithm is the one that takes longer (770 secs). As mentioned before, the ND algorithm is, in average, the fastest one with 299 seconds, while the SD algorithm almost doubles this time with 547 seconds, in average. TS algorithm takes 371 seconds, in average, before convergence.

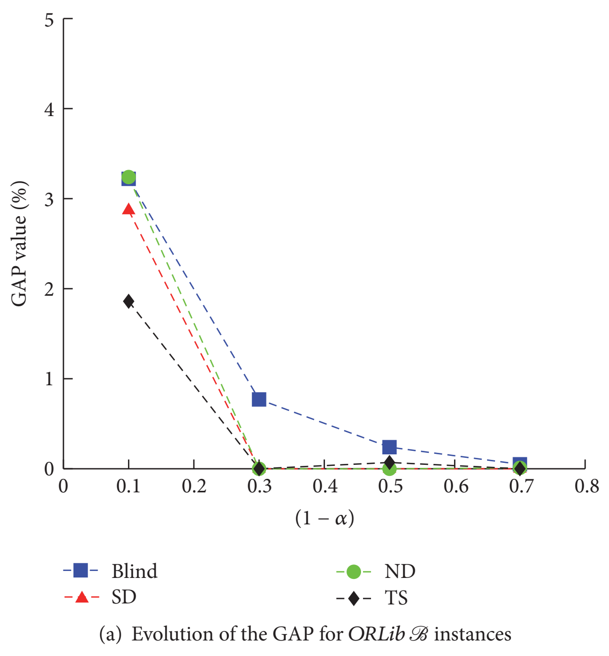
Figures 4(a), 5(a), and 6(a) show the evolution of the GAP, as parameter $(1 - \alpha)$ increases. As we expected, the larger the value of $(1 - \alpha)$ is set to, the smaller the GAP. Also, it is clear that the blind algorithms obtain higher GAP values than the other three algorithms considered in this study. As we mentioned before, the blind algorithm works as our baseline algorithm. It is interesting to note that, for all algorithms, the worst performance is obtained for $(1 - \alpha) = 0.1$. This is mainly because for this value not enough facilities are available and the algorithms open a set

Table 3: Results obtained by the local search methods for the OR Library's instances.

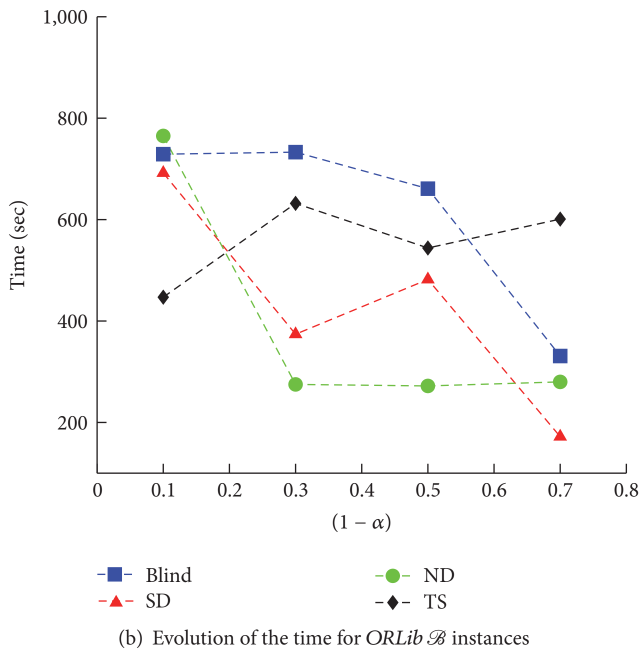
Inst	$(1-\alpha)$	Blind		SD		ND		TS	
		GAP	Time	GAP	Time	GAP	Time	GAP	Time
$\mathscr{A}_1$	0.1	5,22%	622	9,65%	248	9,45%	1424	6,73%	785
	0.3	1,63%	1124	0,00%	1240	0,00%	489	0,00%	736
	0.5	0,32%	664	0,00%	1095	0,00%	274	0,24%	700
	0.7	0,10%	745	0,00%	366	0,00%	138	0,00%	1476
$\mathscr{A}_2$	0.1	4,31%	463	0,00%	833	0,00%	1415	0,00%	866
	0.3	1,22%	893	0,00%	366	0,19%	1290	0,19%	615
	0.5	0,52%	987	1,83%	52	0,09%	956	0,00%	745
	0.7	0,11%	282	0,09%	378	0,00%	1562	0,00%	743
$\mathscr{A}_3$	0.1	1,81%	841	0,00%	341	0,31%	65	0,00%	686
	0.3	0,23%	503	0,00%	602	0,00%	693	0,00%	1083
	0.5	0,12%	880	0,00%	431	0,00%	442	0,00%	659
	0.7	0,00%	272	0,00%	151	0,00%	211	0,00%	211
$\mathscr{A}_4$	0.1	1,56%	990	0,00%	25	0,00%	84	0,00%	84
	0.3	0,00%	411	0,00%	8	0,00%	31	0,00%	31
	0.5	0,00%	111	0,00%	2	0,00%	14	0,00%	14
	0.7	0,00%	23	0,00%	1	0,00%	7	0,00%	7
$\mathscr{B}_1$	0.1	5,22%	622	0,03%	639	0,23%	960	0,01%	319
	0.3	1,63%	1124	0,01%	588	0,08%	436	0,03%	1452
	0.5	0,32%	664	0,00%	707	0,03%	216	0,00%	605
	0.7	0,10%	745	0,01%	347	0,00%	1630	0,00%	1192
$\mathscr{B}_2$	0.1	4,31%	463	1,44%	806	2,75%	222	0,09%	714
	0.3	1,22%	893	0,00%	135	0,00%	69	0,00%	1165
	0.5	0,52%	987	0,00%	186	0,00%	196	0,05%	530
	0.7	0,11%	282	0,00%	112	0,05%	315	0,00%	261
$\mathscr{B}_3$	0.1	1,81%	841	0,40%	403	0,43%	1270	0,30%	142
	0.3	0,23%	503	0,00%	96	0,00%	181	0,00%	267
	0.5	0,12%	880	0,00%	588	0,00%	236	0,00%	564
	0.7	0,00%	272	0,00%	179	0,00%	316	0,00%	316
$\mathscr{B}_4$	0.1	1,56%	990	0,00%	1309	0,32%	144	0,32%	144
	0.3	0,00%	411	0,00%	24	0,00%	360	0,00%	360
	0.5	0,00%	111	0,00%	57	0,00%	381	0,00%	381
	0.7	0,00%	23	0,00%	28	0,00%	349	0,00%	349
$\mathscr{C}_1$	0.1	8,09%	712	9,70%	1220	8,14%	92	10,27%	133
	0.3	3,20%	919	0,00%	657	0,00%	486	0,00%	394
	0.5	1,99%	1105	0,00%	173	0,00%	496	0,00%	1469
	0.7	0,48%	863	0,00%	239	0,00%	355	0,00%	247

Table 3: Continued.

Inst	$(1-\alpha)$	Blind		SD		ND		TS	
		GAP	Time	GAP	Time	GAP	Time	GAP	Time
$\mathscr{C}_2$	0.1	5,74%	937	1,42%	1254	2,32%	1397	0,70%	84
	0.3	1,64%	934	0,00%	136	0,00%	107	0,00%	172
	0.5	0,35%	562	0,00%	1225	0,00%	248	0,00%	537
	0.7	0,02%	766	0,00%	1294	0,00%	164	0,00%	169
$\mathscr{C}_3$	0.1	5,17%	979	1,11%	1254	1,54%	819	0,20%	585
	0.3	1,04%	993	0,00%	73	0,00%	38	0,00%	272
	0.5	0,22%	906	0,00%	587	0,00%	64	0,00%	1387
	0.7	0,04%	740	0,00%	551	0,00%	149	0,00%	779
$\mathscr{C}_4$	0.1	4,86%	663	1,35%	153	0,34%	523	0,85%	313
	0.3	0,65%	711	0,03%	7	0,03%	35	0,03%	107
	0.5	0,08%	821	0,00%	18	0,03%	11	0,03%	13
	0.7	0,02%	228	0,03%	8	0,03%	22	0,00%	29

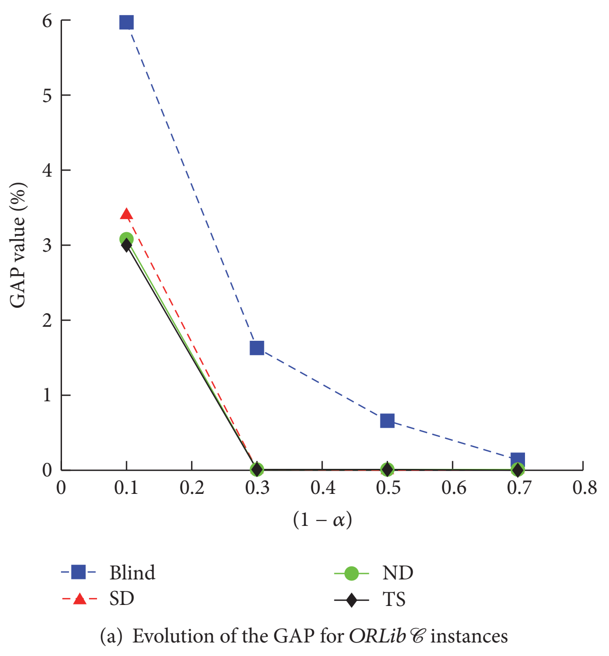


(a) Evolution of the GAP for *ORLib* $\mathscr{B}$ instances

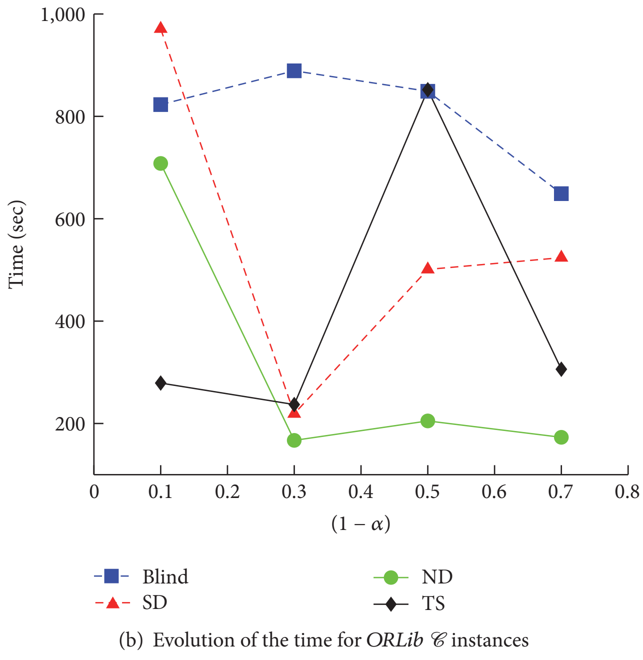


(b) Evolution of the time for *ORLib* $\mathscr{B}$ instances

FIGURE 5: Average results obtained by the local search algorithms for each value of $(1 - \alpha)$ for *ORLib* $\mathscr{B}$ instances.



(a) Evolution of the GAP for *ORLib* $\mathscr{C}$ instances



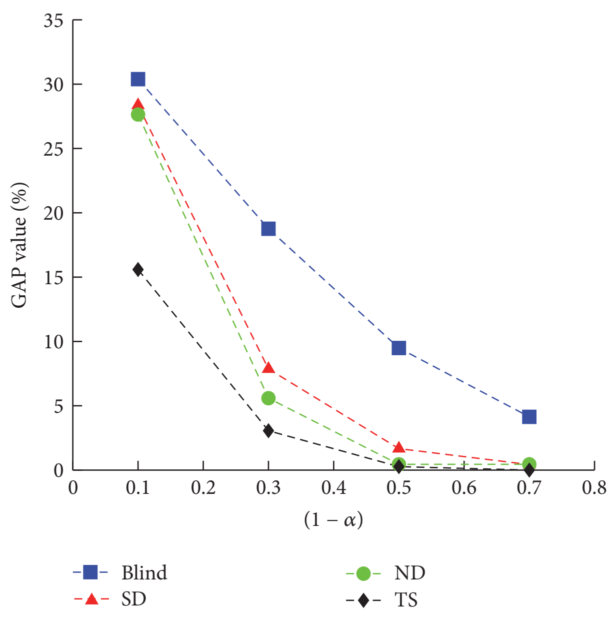
(b) Evolution of the time for *ORLib* $\mathscr{C}$ instances

FIGURE 6: Average results obtained by the local search algorithms for each value of $(1 - \alpha)$ for *ORLib* $\mathscr{C}$ instances.

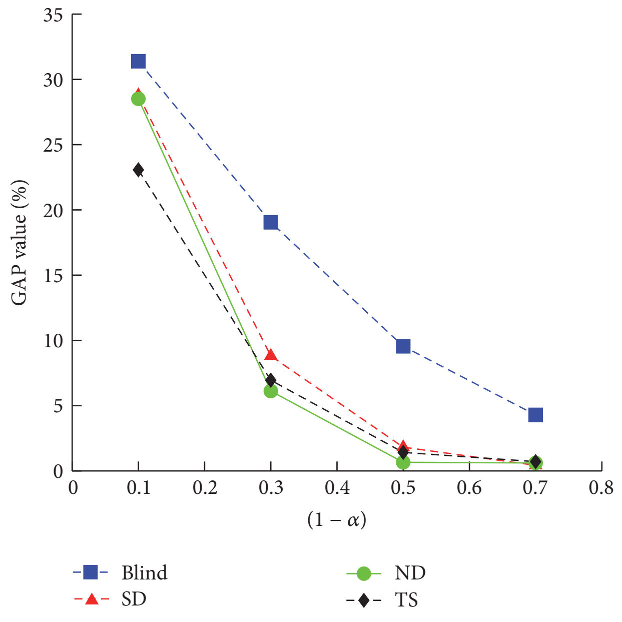
of facilities randomly so the problem has a feasible solution. However, this repairing process is completely random and, thus, algorithms performance is impaired.

Since the problems from the OR Library are only medium size instances, local-search-based matheuristics consistently find the optimal solution for almost all instances for $(1-\alpha) = \{0.3, 0.5, 0.7\}$. In fact, even the blind algorithm finds solutions that are very close to the optimal ones when $(1 - \alpha) = 0.7$.

Figures 4(b), 5(b), and 6(b) show the time needed by our algorithms to find its best solution, as parameter $(1 - \alpha)$ increases. Unlike we expected, the time needed to find the best solution does not increase as the parameter $(1 - \alpha)$ gets larger. In fact, both algorithms, SD and blind, converge faster when $(1 - \alpha = 0.7)$, the larger value we tried in our experiments. This can be explained because of the problems features. We note that optimal solutions for all the



(a) Evolution of the GAP for $DS1_U$ instances



(b) Evolution of the GAP for $DS1_C$ instances

Figure 7: Average GAP values obtained by the local search algorithms for each value of $(1 - \alpha)$ for DS1 instances.

problems in the benchmark of the OR Library need not too many warehouses to be open. Thus, when a large portion of the potential warehouses are available (as for the $(1 - \alpha) = 0.7$ case) the algorithm is likely to find such optimal solutions in early iterations. In fact, for the vast majority of the experiments, optimal solution is found within the first 2 to 3 iterations.

We now move on the $DS1_U$ and the $DS1_C$ benchmarks. As we noted before, these sets of instances are much larger and harder to solve than the problems in the OR Library we discussed above. Further, as the optimal solutions of these instances do not require too many warehouses to be open, the repairing procedure applied to the OR Library instances for $(1 - \alpha) = 0.1$ does not have a great impact on the results. Thus, local-search-based matheuristics outperform the results obtained by the blind algorithm for all values of parameter α. Figures 7(a) and 7(b) show the GAP values obtained by the all four algorithms for instances $DS1_U$ and $DS1_C$, respectively.

Just as in the OR Library instances, as the parameter $(1-\alpha)$ gets larger, the GAP approximates to 0 for the all three local-search-based matheuristic proposed in this paper. While for the $DS1_U$ instances both the TS and the ND algorithms reach GAP values very close to 0 for $(1 - \alpha) = 0.5$ and $(1 - \alpha) = 0.7$, for the $DS1_C$ instances the best value obtained by the TS and the ND algorithms is 0.71 and 0.61, respectively, for $(1 - \alpha) = 0.7$. Notably, the ND algorithm performs slightly better than the TS for the $DS1_C$ instance when $(1 - \alpha) = 0.5$, obtaining a best average value of 0.66 against the 1.42 obtained by the TS algorithm. In spite of that, we can note that the differences between the GAP obtained by both the ND and the TS algorithms when using $(1 - \alpha) = 0.5$ and $(1-\alpha) = 0.7$ are negligible, just as in the OR Library instances.

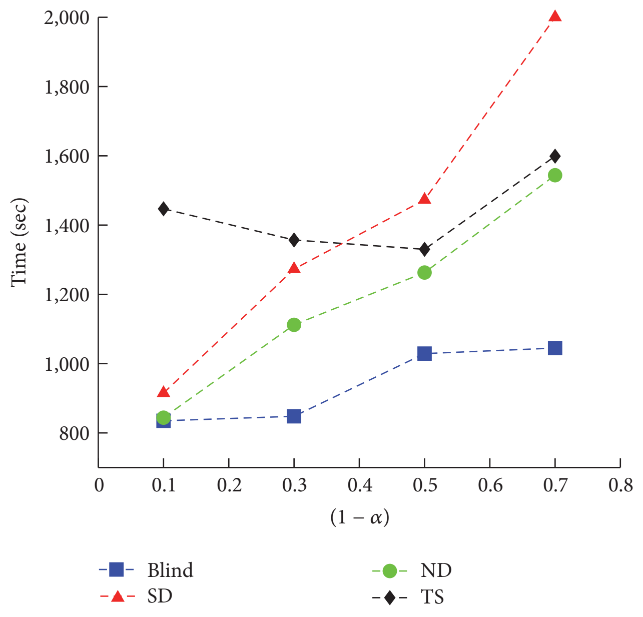
Figures 8(a) and 8(b) shows the time needed by the all four algorithms to find their best solution for instances $DS1_U$ and $DS1_C$, respectively. As expected, as the parameter $(1 - \alpha)$ gets larger, longer times are needed by the algorithms to converge. This is important as one would like to use a value for $(1-\alpha)$ such that minimum GAP values are reached within few minutes.

Results in Figures 7 and 8 show, clearly, that there is a compromise between the quality of the solution found by the algorithms and the time they need to do so. Further, we know that this compromise can be managed by adjusting the value of parameter $(1-\alpha)$. For the case of instances $DS1_U$ and $DS1_C$, such a value should be between 0.4 and 0.6. Moreover, for the case of the instances of the OR Library, parameter $(1 - \alpha)$ should be set around 0.3. It is interesting to note that while the size of instances $DS1_U$ and $DS1_C$ is equivalent, instances of the OR Library are much smaller.

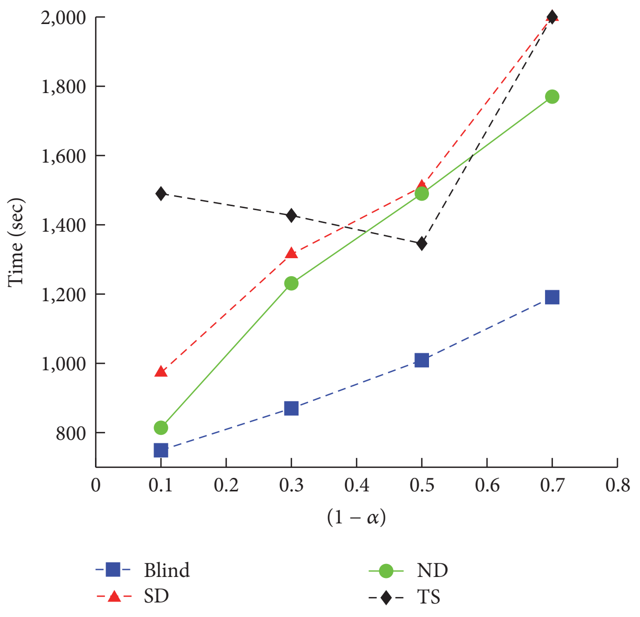
We can also note that instances that include clusters tend to take longer to converge. This is especially true as parameter $(1 - \alpha)$ gets larger. Further, for these instances, as the parameter $(1-\alpha)$ gets larger, less iterations are performed by algorithms although each of these iterations takes much longer. As mentioned before, these should be taken into account when using population-based algorithms within the framework presented in this study as such kind of heuristic algorithms needs to perform several iterations before to converge to good quality solutions.

4. Conclusions and Future Work

In this paper we show the impact of parameter tuning on a local-search-based matheuristic framework for solving



(a) Evolution of the time for $DS1_U$ instances



(b) Evolution of the time for $DS1_C$ instances

Figure 8: Average times needed by the local search algorithms for each value of $(1 - \alpha)$ for DS1 instances.

mixed integer (non)linear programming problems. In particular, matheuristics that combine local search methods and a MIP solver are tested. In this study, we focus on the size of the subproblem generated by the local search method that is passed on to the MIP solver. As expected, the size of the subproblems that are solved in turn by the matheuristic method has a big impact on the behaviour of the matheuristic and, consequently, on its obtained results: as the size of the subproblem increases (i.e., more integer/binary decision variables are considered) the results obtained by the MIP solver are closer to the optimal solution. The time required by the algorithms tested in this paper to find its best solution also increases as the subproblem gets larger. Further, as the subproblem gets larger, fewer iterations can be performed within the allowed time. This is important as other heuristics such as evolutionary algorithms and swarm intelligence, where many iterations are needed before converging to a good quality solution, might be not able to deal with large subproblems. We also note that the improvement in the GAP values after certain value of parameter $(1 - \alpha)$ is negligible and that this value depends to some extent on the size of the problem: for medium size instances such as the ones in the OR Library, parameter $(1-\alpha)$ should be set to a value around 0.3 while for larger instances ($DS1_U$ and $DS1_C$) it should be set to a value between 0.4 and 0.6. The specific values will depend on both the accuracy level requested by users and the time available to perform the algorithm. Therefore, the challenge when designing matheuristic frameworks as the one presented in this paper is to find a value for parameter $(1 - \alpha)$ that allows the heuristic algorithm to perform as many iterations as needed and that provides a good compromise between solution quality and run times. Although the values provided here are only valid for the problem and the algorithms considered in this paper, we think that the obtained results can be used as a guide by other researchers using similar frameworks and/or dealing with similar problems.

As a future work, strategies such as evolutionary algorithms and swarm intelligence will be tested within the matheuristic framework considering the results obtained in this study. We expect that intelligent methods such as the ones named before greatly improve the results obtained by the local search methods considered in this study. Moreover, the matheuristic framework used in this paper might also be applied to other MILP and MINLP problems such as, for instance, the beam angle optimisation problem in radiation therapy for cancer treatment.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

Guillermo Cabrera-Guerrero wishes to acknowledge *Ingeniería 2030*, DI 039.416/2017 and CONICYT/FONDECYT/INICIACION/11170456 projects for partially supporting this research.

References

[1] F. Neri and C. Cotta, “Memetic algorithms and memetic computing optimization: a literature review,” *Swarm and Evolutionary Computation*, vol. 2, pp. 1–14, 2012.

[2] Y.-s. Ong, M. H. Lim, and X. Chen, "Memetic Computation—Past, Present & Future," *IEEE Computational Intelligence Magazine*, vol. 5, no. 2, pp. 24–31, 2010.

[3] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 5, pp. 591–607, 2011.

[4] V. Maniezzo, T. Stützle, and S. Voß, Eds., *Matheuristics - Hybridizing Metaheuristics and Mathematical Programming*, vol. 10 of *Annals of Information Systems*, Springer, 2010.

[5] G. Guillermo Cabrera, E. Cabrera, R. Soto, L. J. M. Rubio, B. Crawford, and F. Paredes, "A hybrid approach using an artificial bee algorithm with mixed integer programming applied to a large-scale capacitated facility location problem," *Mathematical Problems in Engineering*, vol. 2012, Article ID 954249, 14 pages, 2012.

[6] M. Caserta and S. Voß, "A math-heuristic Dantzig-Wolfe algorithm for capacitated lot sizing," *Annals of Mathematics and Artificial Intelligence*, vol. 69, no. 2, pp. 207–224, 2013.

[7] L. C. Coelho, J.-F. Cordeau, and G. Laporte, "Heuristics for dynamic and stochastic inventory-routing," *Computers & Operations Research*, vol. A, no. 0, pp. 55–67, 2014.

[8] C. Lagos, F. Paredes, S. Niklander, and E. Cabrera, "Solving a distribution network design problem by combining ant colony systems and lagrangian relaxation," *Studies in Informatics and Control*, vol. 24, no. 3, pp. 251–260, 2015.

[9] B. Raa, W. Dullaert, and E.-H. Aghezzaf, "A matheuristic for aggregate production-distribution planning with mould sharing," *International Journal of Production Economics*, vol. 145, no. 1, pp. 29–37, 2013.

[10] H. Allaoua, S. Borne, L. Létocart, and R. Wolfler Calvo, "A matheuristic approach for solving a home health care problem," *Electronic Notes in Discrete Mathematics*, vol. 41, pp. 471–478, 2013.

[11] G. Cabrera-Guerrero, M. Ehrgott, A. J. Mason, and A. Raith, "A matheuristic approach to solve the multiobjective beam angle optimization problem in intensity-modulated radiation therapy," *International Transactions in Operational Research*, 2016.

[12] Y. Li, D. Yao, J. Zheng, and J. Yao, "A modified genetic algorithm for the beam angle optimization problem in intensity-modulated radiotherapy planning," in *Artificial Evolution*, E.-G. Talbi, P. Liardet, P. Collet, E. Lutton, and M. Schoenauer, Eds., vol. 3871 of *Lecture Notes in Computer Science*, pp. 97–106, Springer, Berlin, Germany, 2006.

[13] Y. Li, D. Yao, W. Chen, J. Zheng, and J. Yao, "Ant colony system for the beam angle optimization problem in radiotherapy planning: a preliminary study," in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 2, pp. 1532–1538, IEEE, Edinburgh, Scotland, UK, September 2005.

[14] S. Hanafi, J. Lazić, N. Mladenović, C. Wilbaut, and I. Crévits, "New hybrid matheuristics for solving the multidimensional knapsack problem," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 6373, pp. 118–132, 2010.

[15] E.-G. Talbi, "Combining metaheuristics with mathematical programming, constraint programming and machine learning," *4OR*, vol. 11, no. 2, pp. 101–150, 2013.

[16] C. Archetti and M. G. Speranza, "A survey on matheuristics for routing problems," *EURO J. in Computer Optimization*, vol. 2, no. 4, pp. 223–246, 2014.

[17] M. A. Boschetti, V. Maniezzo, M. Roffilli, and A. Bolufé Röhler, "Matheuristics: Optimization, simulation and control," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 5818, pp. 171–177, 2009.

[18] C.-H. Chen and C.-J. Ting, "Combining lagrangian heuristic and ant colony system to solve the single source capacitated facility location problem," *Transportation Research Part E: Logistics and Transportation Review*, vol. 44, no. 6, pp. 1099–1122, 2008.

[19] Y. Li, J. Yao, and D. Yao, "Automatic beam angle selection in IMRT planning using genetic algorithm," *Physics in Medicine and Biology*, vol. 49, no. 10, pp. 1915–1932, 2004.

[20] Y. Li, D. Yao, J. Yao, and W. Chen, "A particle swarm optimization algorithm for beam angle selection in intensity-modulated radiotherapy planning," *Physics in Medicine and Biology*, vol. 50, no. 15, pp. 3491–3514, 2005.

[21] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[22] M. Pirlot, "General local search methods," *European Journal of Operational Research*, vol. 92, no. 3, pp. 493–511, 1996.

[23] C. Lagos, G. Guerrero, E. Cabrera et al., "A Matheuristic Approach Combining Local Search and Mathematical Programming," *Scientific Programming*, vol. 2016, Article ID 1506084, 2016.

[24] V. Vails, M. A. Perez, and M. S. Quintanilla, "A tabu search approach to machine scheduling," *European Journal of Operational Research*, vol. 106, no. 2-3, pp. 277–300, 1998.

[25] S. Hanafi and A. Freville, "An efficient tabu search approach for the 0-1 multidimensional knapsack problem," *European Journal of Operational Research*, vol. 106, no. 2-3, pp. 659–675, 1998.

[26] J. Brandão and A. Mercer, "A tabu search algorithm for the multi-trip vehicle routing and scheduling problem," *European Journal of Operational Research*, vol. 100, no. 1, pp. 180–191, 1997.

[27] K. S. Al-Sultan and M. A. Al-Fawzan, "A tabu search approach to the uncapacitated facility location problem," *Annals of Operations Research*, vol. 86, pp. 91–103, 1999.

[28] P. B. Mirchandani and R. L. Francis, Eds., *Discrete Location Theory*, Wiley-Interscience, New York, NY, USA, 1990.

[29] J. E. Beasley, "OR-Library: distributing test problems by electronic mail," *Journal of the Operational Research Society*, vol. 41, no. 11, pp. 1069–1072, 1990.

[30] J. G. Villegas, F. Palacios, and A. . Medaglia, "Solution methods for the bi-objective (cost-coverage) unconstrained facility location problem with an illustrative example," *Annals of Operations Research*, vol. 147, pp. 109–141, 2006.