Comparing New Heuristics for the Pure Integer Capacitated Plant Location Problem*

Hugues Delmaire[1] *Juan A. Díaz*[1] *Elena Fernández*[1]
Maruja Ortega[2]

[1]Departament d'Estadística i Investigació Operativa
Universitat Politècnica de Catalunya
Pau Gargallo, 5; 08028 Barcelona, Spain
{hugues, jadiaz, elena}@eio.upc.es

[2]Departamento de Computación y T.I.
Universidad Simón Bolívar
Caracas, Venezuela
mof@ldc.usb.ve

Abstract
This paper considers the Pure Integer Capacitated Plant Location Problem (PI-CPLP). It is an NP-hard integer programming problem closely related to the Mixed Integer Capacitated Plant Location Problem (MI-CPLP). The difference between the two lays in the integrality constraints on the assignment variables. As opposed to MI-CPLP, in PI-CPLP, for a given set of open plants, the assignment subproblem remains NP-hard (specifically a Generalized Assignment Problem).
Several heuristics are proposed, based on the following approaches: Evolutive Algorithms (EA), Greedy Randomized Adaptive Search Procedure (GRASP), Simulated Annealing (SA) and Tabu Search (TS). All the algorithms proposed share three characteristics. First, they make direct use of the problem structure. The main problem is divided in two subproblems, one for the selection of plants and another one for the assignment of clients to plants. Second, they search for feasible solutions to the PI-CPLP using the relaxation based on the aggregation of the capacity constraints. Third, they basically explore the same neighborhood structures. These structures and the strategies used to explore them are also presented.
All the proposed algorithms have been tested computationally. Their performance is reported and their results are compared.

*The first author was supported by a Fellowship grant from Generalitat de Catalunya. The second author was partially supported by grant 110598/110666 from Consejo Nacional de Ciencia y Tecnología, México. The forth author was partially supported by grant SAB-95-0474 from Dir. Gral. de Enseñanza Superior, España.

1 Introduction

The Capacitated Plant Location Problems (CPLP's) constitute a classical area of research in optimization. They consider the case where a decision maker has to choose the location of new plants from a given set of alternatives in order to satisfy the demand of a set of clients at minimum cost. To do so, the minimization of a two member objective function subject to a set of constraints is used. The first member regroups fixed costs for the opening of individual plants. The second member regroups assignment costs of clients to plants. Constraints then insure that plants capacities are respected, and clients are correctly assigned. When the clients have to be assigned to only one plant (cannot be splitted between two or more), the problem becomes the Pure Integer Capacitated Plant Location Problem (PI-CPLP) that is studied in this paper. It can be modeled as:

$$(P) \qquad min\ z = \sum_{j\in J} f_j y_j + \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij}$$

s.t.

$$\begin{array}{lllr} \sum_{j\in J} x_{ij} & = 1 & \forall i \in I & (1) \\ \sum_{i\in I} d_i x_{ij} & \leq b_j y_j & \forall j \in J & (2) \\ x_{ij}, y_j & \in \{0,1\} & \forall i \in I, j \in J & (3) \end{array}$$

where: I is the set of clients, J is the set of plants, f_j is the fixed cost of opening plant j, c_{ij} is the cost of assigning client i to plant j, d_i is the demand of client i, and b_j is the capacity of plant j. The problem variables are y_j with value 1 if plant j is opened, 0 otherwise; and x_{ij} with value 1 if client i is assigned to plant j, 0 otherwise. The objective function adds the fixed costs of the open plants to the assignment costs of the clients. Constraints (1), together with constraints (3), insure that each client is assigned to one and only one plant, constraints (2) insure that the capacity of each plant is respected and that clients are assigned to open plants, and finally constraints (3) are the integer requirements on the variables.

The objective function of the previous model can be expressed in a different way by the introduction of the following notation: $cmin_i = min\ \{c_{ij},\ j \in J\}$, and $\Delta_{ij} = c_{ij} - cmin_i$. This notation leads to an equivalent formulation for the objective function:

$$\sum_{i\in I} cmin_i + min\ (\sum_{j\in J} f_j y_j + \sum_{i\in I}\sum_{j\in J} \Delta_{ij}x_{ij})$$

The interest of this reformulation rests on the interpretation of the Δ_{ij}'s. They give the relative cost of the assignment of client i to plant j with respect to its minimum assignment cost. Some of the heuristic algorithms presented here use this added information.

The PI-CPLP is a NP-hard integer programming problem related to other well-known discrete location problems. For instance, the Uncapacitated Plant Location

Problem (UPLP) is a NP-hard pure integer problem that can be seen as a particular case of the PI-CPLP where the capacities of the plants are greater that the sum of the clients' demands. Hence, the model for the UPLP needs not to consider constraints (2). The UPLP is a classical problem that has been widely studied both from a polyhedral and an algorithmic point of view (see, for instance [10] and [21]). Also the Mixed Integer Capacitated Plant Location Problem (MI-CPLP) is closely related to the PI-CPLP. The difference between the two problems lays in the definition of the x_{ij} variables. In the MI-CPLP those variables are continuous, allowing individual clients to be assigned to multiple plants. It also means that for a given set of open plants, the assignment sub-problem corresponds to a transportation problem. In the PI-CPLP those variables are required to be integers, thus each client has to be assigned to only one plant. A direct consequence is that, for a given set of open plants, the assignment sub-problem remains a Generalized Assignment Problem (GAP) which is a NP-hard integer problem, and for which even the feasibility question is NP-complete. Based on the previous remarks, a two stage decision is thus necessary to solve the PI-CPLP. The optimal set of locations has to be selected first. Then, the optimal allocation of clients within that set of plants has to be found, such that each client is allocated to only one plant.

A large literature exists on the MI-CPLP and a recent review can be found in [22]. Heuristic approaches are presented in [16], and an algorithm for large instances is given in [4]. Much less attention has been devoted to the PI-CPLP. To the best of our knowledge, no specialized exact method for solving the PI-CPLP can be found in the literature. Several Lagrangean Relaxation based approaches have been proposed (see [1], [2], [3], [4], [12], and [19]). In most of the cases, the iterative method used to solve the Lagrangean Dual is combined with some simple heuristic to obtain feasible solutions. However, no heuristic methods focused on obtaining good quality solutions in reduced computational times have been proposed for the PI-CPLP. In particular, no metaheuristic based method can be found in the literature. Yet, such approaches are fully justified in this case where feasibility is difficult to attain. Specially, taking into account that metaheuristic based methods have proven to be very effective for other classes of combinatorial optimization problems where feasibility is also difficult to achieve as (e.g. Routing Problems).

In this context, the motivation for this work is a) to propose heuristic approaches based on different metaheuristic methodologies for the PI-CPLP, b) to evaluate them by comparing the results obtained with the various approaches, and c) to draw conclusions about the adequacy of each of the selected methodologies for solving the PI-CPLP.

The proposed heuristics are of the following types: Evolutive Algorithm (EA), Greedy Randomized Adaptive Search Procedure (GRASP), Simulated Annealing (SA), and Tabu Search (TS). All the presented methods share the following characteristics:

a) Due to the nested decisions that have to be taken for solving the PI-CPLP, and given the difficulty of the subproblems to be solved at both levels, exploiting the hierarchical configuration of the problem plays a central role. Within each of the selected methodologies, the specific heuristic that is proposed has been designed to respect this structure, and, thus, contains a fist level focused on the plants' selection subproblem as well as an embedded second level focused on the allocation subproblem for a given set of open plants.

At the exception of the EA, each approach uses variations of the same constructive method for the plants' selection problem and of a specific methodology for the allocation subproblem. On the contrary, the EA uses that methodology at both levels. This is the approach that proved to be a better alternative to the constructive method for smaller problems. Unfortunately, this tendency was not confirmed for larger instances.

b) They explore the infeasible solutions' space. When feasibility is difficult to achieve, confining the search to the feasible space in heuristic methods often makes the search terminate after a reduced number of iterations. Thus, the quality of the obtained local optimum is usually not very good. For this reason, allowing the heuristic to operate in the infeasible solutions' space provides a higher level of flexibility to the search and allows reaching solutions that otherwise could not be found. Violation of feasibility can be taken into account by means of a penalty term in the objective function. All the methods presented in this paper operate on the same relaxation of the original problem and, hence, explore the same solutions' space.

c) They try to improve solutions by searching in the same neighborhood structures in the assignment subproblem. These are the classical shift and swap neighborhoods. The only special case being the GRASP whose speed allows to explore additional neighborhood structures during the second phase of the algorithm.

The above three characteristics can be seen as a template to which all the proposed methods have been adjusted. The use of such a template guarantees to a great extent that the comparisons to be made between the different approaches are objective.

The proposed heuristics have been tested over the set of test problems used in [3]. To evaluate the quality of the obtained solutions, the results have been compared to the best known values reported in the above reference and to the solutions generated by CPLEX in a time-limit of two hours of CPU. The obtained results confirm the interest of the proposed approaches. In particular, all the methods improve the previous best known solutions for all the test problems (with the exception of one single problem for one of the heuristics). Moreover, the deviation of the best solution generated with the proposed approaches from the optimal (or best-known) solution generated by CPLEX within the given time bounds never exceeds 0.63%, although on the average this deviation takes a value of 0.10%. Also, with the exception of EA, the proposed methods have proven to be robust since the average deviation from the optimal (or best-known) solution never exceeds 1.6% in the worst case. The behavior of EA is very good for smaller problems but it decreases as the size of the problems grows. Finally, excepting the Evolutive Algorithm all the other approaches are also efficient in terms of the required computational time.

The paper is organized in the following way: in Section 2, basic definitions and notation are introduced; in Section 3, the heuristics tested on the problem are presented: an Evolution type heuristics, a GRASP, a Simulated Annealing, and a Tabu Search algorithm; in Section 4, the results of computational experiences are given; finally, Section 5 concludes the work and identifies further research directions.

2 Definitions and Notation

Capacity constraints (2) are usually difficult to fulfill and reduce considerably the solutions' space. By considering constraints (1), constraints (2) can be aggregated in the surrogate constraint:

$$\sum_{j\in J} b_j y_j \geq D = \sum_{i\in I} d_i \quad (3')$$

Constraint (3') insures that the total capacity of the open plants is enough to satisfy the clients' demands. In the case of the PI-CPLP this constraint is a necessary, but not sufficient, condition for the existence of feasible solutions. The substitution of (2) by (3') in (P) leads to the following relaxed formulation:

$$\begin{array}{llll}
\text{(RP)} & \multicolumn{3}{l}{min\ z' = \sum_{j\in J} f_j y_j + \sum_{i\in I}\sum_{j\in J} \Delta_{ij} x_{ij}} \\
\text{s.t.} & & & \\
\sum_{j\in J} x_{ij} & = 1 & \forall i \in I & (1') \\
x_{ij} & \leq y_j & \forall i \in I, j \in J & (2') \\
\sum_{j\in J} b_j y_j & \geq D & & (3') \\
x_{ij}, y_j & \in \{0,1\} & \forall i \in I, j \in J & (4')
\end{array}$$

where constraints (2') are now needed to limit the assignment of clients to open plants. All the proposed approaches operate on (RP). This provides the search processes with a higher level of flexibility, since some feasible solutions to (P) that otherwise could not have been reached, can now be obtained via paths of feasible solutions to (RP).

Each solution σ to (P) or (RP) is associated to a pair (O, A) where $O \subseteq J$ represents the set of open plants and A the set of assignments for the clients, i.e.:

$$O = \{j : y_j = 1\} \subseteq J$$

$$A = \{p_i\}_{i\in I} \text{ where } p_i \in O, \forall i \in I \text{ and } p_i = j \Longleftrightarrow x_{ij} = 1$$

Note that the first element of the pair, set O, is redundant for the description of a solution since it can be directly obtained from set A. It has been introduced to ease the presentation of the proposed methods.

The following definition of slack variables s_j for the plants is also used:

$$s_j = b_j - \sum_{i/p_i=j} d_i$$

Each of these slack variables gives the remaining capacity for the corresponding plant in a solution at hand. They provide a useful infeasibility measure when dealing with solutions to (RP) that are not feasible solutions to (P). This measure is used by the different heuristics for guiding the search towards feasibility. This is achieved by incorporating a penalty term to the objective function to evaluate the overall infeasibility with respect to (P) of a given solution to (RP)

The neighborhood structures explored by the algorithms presented are the following:

Let $\sigma = (O, A)$ be a solution (to (RP) or (P)), then σ' is a neighbor of σ if and only if σ' is also a solution and σ' belongs to any of the two following sets

$N_{shift}(\sigma)$: Client reassignment neighborhood. This neighborhood can be obtained by maintaining the same set of open plants and reassigning one single client within this set.

$$\begin{aligned} N_{shift}(\sigma) &= \{\sigma' = (O', A') :\ O' = O,\ A' = \{p'_i\}_{i \in I}, \\ &\quad \exists! i^* \in I\ s.t.\ p'_{i^*} \neq p_{i^*},\ p'_{i^*} \in O'\} \end{aligned}$$

$N_{swap}(\sigma)$: Client interchange neighborhood. This neighborhood can be obtained by maintaining the same set of open plants and interchanging the assignments of two clients.

$$\begin{aligned} N_{swap}(\sigma) &= \{\sigma' = (O', A') :\ O' = O,\ A' = \{p'_i\}_{i \in I}, \\ &\quad \exists i_1, i_2 \in I\ s.t.\ p'_{i_1} = p_{i_2},\ p'_{i_2} = p_{i_1},\ p'_i = p_i\ \forall i \neq i_1, i_2\} \end{aligned}$$

3 Heuristics

In this section, the implementations of the heuristics designed to solve the PI-CPLP are described in detail. The sub-sections present the following implementations: in 3.1 an Embedded Evolutive Algorithm, in 3.2 a GRASP, in 3.3 a Simulated Annealing, and finally in 3.4 a Tabu Search.

3.1 Embedded Evolutive Algorithm

The concept of evolutive algorithm was proposed in [20], these algorithms differ from the classical binary encoded genetic algorithms (see [15]) as they allow any data structure to be processed by a set of operators implementing a survival of the fittest philosophy. The general structure of a classical genetic algorithm is:

```
Algorithm 1:
    Construct a fixed size initial population $P(0)$.
    While (stopping criterion not satisfied) do
        Select intermediate population $P'$ from $P(n-1)$
        Apply crossover and mutation to $P'$ to obtain $P(n)$
        Evaluate $P(n)$
```

Evolution type heuristics follow the same major steps. Encoded strings correspond to potential solution representatives. A given number of those strings form a population. Operators (crossover, mutation, evaluation, selection, and shuffle) are applied to this population in order to emulate the natural evolution process. Going through each step once is called a generation.

An important first step when using evolutive algorithms, is the definition of an adequate representation of the potential solutions into coded strings. In the algorithm presented in this section each potential solution $\sigma = (O, A)$ is represented by a set of two vectors:

- a binary vector of size $|J|$.
- an integer vector of size $|I|$.

The first vector determines the set O of open plants and the second one, the assignment A of clients to the plants. Figure 1 gives the vector representation for a case with O={1,3,5,6,8}. In this example, client 1 is assigned to plant 3, client 2 to plant 5, etc.

Open plants	Client assignment
1 0 1 0 1 1 0 1	3 5 6 3 8 6 1 3 8 1 6 1

Fig. 1: Vector representation of a solution.

The approach proposed in this section, called *embedded approach* (EEA), has been devised considering a decomposition of the solution set into smaller subsets (see for example: [7], [8]).

It can be seen as a hierarchical construction where at the first level the values of some of the problem variables are fixed by a heuristic. These values define a second level of search where another heuristic looks for the best possible solutions in the current subset.

In the case of the PI-CPLP, the first level is naturally defined as the opening plants problem. For each combination of open plants that satisfies the aggregated demand constraint, a subset of potential solutions is created (the second level of search). In this subset, the assignment of the clients to the open plants is considered. More precisely, the process applies a first evolutive algorithm to a population of candidate sets of open plants, and then a second or embedded evolutive algorithm is used to find the assignment of clients for each of these sets of open plants.

The overview of the method is given in Figure 2. The description of operators valid for both levels is now done.

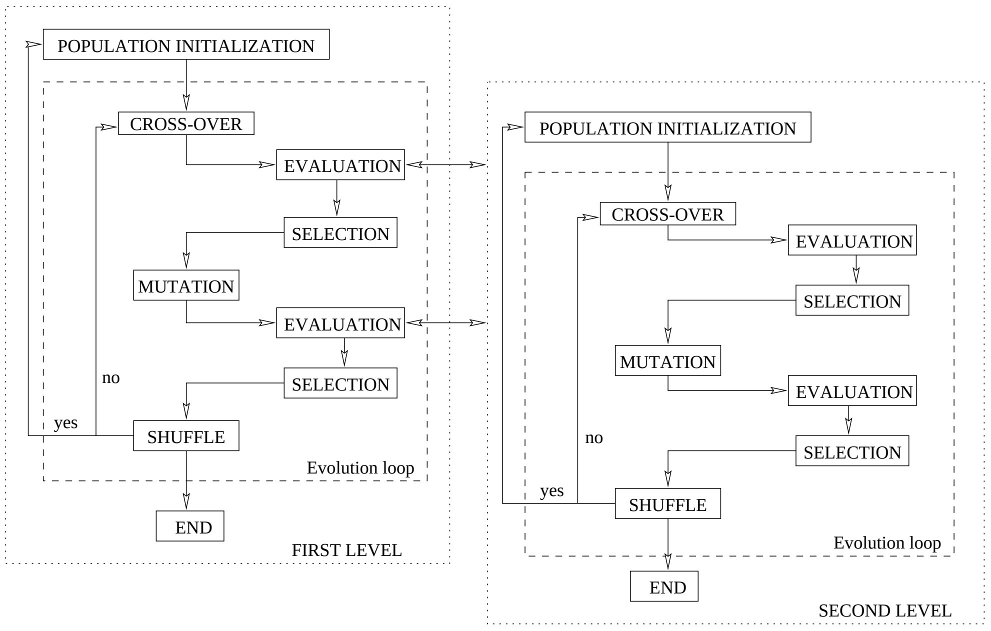


Fig. 2: The embedded approach using evolution type heuristics.

The selection operator is purely elitist with respect to the fitness function and the feasible domain. It selects the best string at each step of the process. This implementation is a purely feasible one since once the feasible domain is entered, the strings are not allowed to go out of it. When a string is selected for reproduction (crossover), sub-strings of equal length are taken on both halves of it and shifted (see Figure 3).

This crossover is a mono-parental operator. Each level of the search is now being addressed through a description of the characteristics of the operators that compose it.

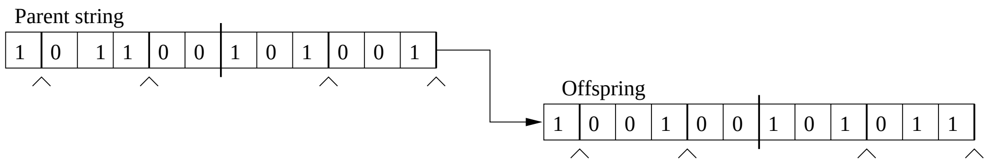


Fig. 3: Example of mono-parental crossover.

First level:

Initial population: All strings are randomly generated.

Mutation: When a member (plant) of a string is selected for mutation, the plant is closed if it was opened and vice-versa.

Evaluation: This is the most important operator since it does the link with the second level of search. Two cases must be taken into account: binary strings that are feasible or infeasible to (RP). In the first case (the aggregated demand constraint is verified), the second level of search is called, and the fitness (solution quality) of the string is defined by the result of the second level search. In the second case (the capacity of the open plants is not sufficient to satisfy constraint $3'$), a fitness function that simply looks for strings that satisfy (3') is used. The reason being that any combination of open plants regardless of their fixed costs can be a part of the best possible solution for the problem. The function used is the distance to feasibility:

$$f_1(O) \; = \; D \; - \sum_{j \in O} b_j$$

being D the minimum capacity needed by the clients, and the summation being done on the string's open plants. $f_1(O)$ thus gives an indication of the capacity missing in the set of open plants to satisfy the overall demand of the clients.

Shuffle: The shuffle operator (see [6]) controls the life cycles of the process. The operator consists of two separate mechanisms. The first controls the total number of generations allowed during the search; as soon as this number is reached the shuffle stops the current search. The second controls the life span of a given population: if the current best solution is not improved during a certain number of generations, the population is eliminated and the search continues with a brand new one.

Second level:
At this level, a string of open plants with sufficient global capacity is received. The decision to be assessed is the assignment of the clients to the plants.

Initial population: Required assignment strings are randomly generated from the list of open plants.

Mutation: When a client i is selected for mutation, the operator does the distinction between two possible states of the string: non-feasible, and feasible to (P).
In the first case, mutation is oriented to reduce infeasibility and again two subcases are considered. The first subcase occurs when the plant p_i to which client i is assigned has a negative slack (violated capacity) and is focused to reduce the capacity violation of this plant. Then two lists are sequentially explored. The first one contains all reassignments of i in N_{shift} to plants with enough available capacity. The second one is associated to interchanges in N_{swap}. Note that after selecting the client i to be interchanged, two factors condition the infeasibility reduction for the solutions generated in N_{swap}. The first is the demand d_{i_2} of the client i_2 to be interchanged with i, and the second is the slack value of the plant j_2 to which i_2 is currently assigned. The list then contains all interchanges in N_{swap} with clients that have a smaller demand than d_i and associated to plants that have a better slack value than p_i.

The second subcase happens when the plant p_i to which i is assigned has a positive slack. Now mutation is oriented to reduce the capacity violation of some violated plant. To do so, a list containing the interchanges in N_{swap} with clients that have a greater demand than d_i, which are assigned to plants with a negative slack value, and that do not violate the capacity of plant p_i are considered.

When the string is feasible, two lists containing all feasible movements in N_{shift} and N_{swap} respectively are considered.

When an allowed move is encountered in one of the lists, it is performed and evaluated. In the special case when all lists are empty, a random 3-way exchange (exchange between 3 randomly selected clients) is performed. This operator together with the selection process insures that the search moves towards the feasible space, and once there stays in it.

Evaluation: At this level, a two state evaluation function is also used. When the string is infeasible to (P), its score is considered to be the distance to feasibility:

$$f_2(\sigma) = \sum_{j \in O : s_j < 0} |s_j|$$

On the other hand, when the string is feasible, its fitness is directly given by the objective function:

$$f_2(\sigma) \;=\; \sum_{j\in J} f_j y_j \;+\; \sum_{i\in I}\sum_{j\in J} \Delta_{ij} x_{ij}$$

Shuffle: It works the same way as at the first level with the following added functionality: before eliminating the population, a final search in N_{swap} is done on the best string.

3.2 GRASP

The GRASP metaheuristic was originally proposed in [9]. It is an iterative method, each iteration containing two phases: the first phase builds a solution that is improved by a local search in the second phase. The general outline of a GRASP algorithm is:

```
Algorithm 2:
     While (stopping criterion not satisfied) do
        Construct a Greedy Randomized Solution
        Apply local Search
```

The construction phase of each iteration seeks a compromise between quality and variety of the solutions which is achieved by partially randomizing a greedy procedure. Each solution is iteratively constructed one element at a time. The choice of the next element to be added is decided by random selection from a Restricted Candidate List (RCL) containing the best candidates with respect to the greedy function (as opposed to standard greedy procedures in which the choice of the next element is determined by the top candidate).

An implementation for the PI-CPLP is now described.

Each of the two phases that take place at each iteration of the GRASP is basically devoted to one of the decision problems of the PI-CPLP. The constructive phase is focused on generating a set of plants; it also provides an initial assignment of clients within the obtained set of plants. The local search is mainly devoted to the assignment subproblem, however the initial set of plants may change during that phase.

Constructive phase:
At each step, a plant is opened and clients are assigned to it in such a way that its capacity constraint is not violated. Once a client is assigned to a plant it is never reassigned during the construction phase. It is further assumed that within each plant the clients are ordered by increasing values of the Δ_{ij}'s (ties are broken by decreasing values of the demands).

For a given partial solution, each closed plant j is evaluated by the greedy function:

$$\phi_j = \frac{f_j + \sum_{i\in Clients_j} \Delta_{ij}}{|Clients_j|}$$

where $Clients_j$ is the set of unassigned clients that fit into plant j, according to the

above mentioned ordering. The numerator is the cost derived from opening plant j and assigning to it all the clients indexed in $Clients_j$. Therefore ϕ_j corresponds to the overall cost per client assigned to plant j, if this plant were to be opened.
After evaluating ϕ_j for each closed plant, the RCL is built with all the elements whose greedy function value is below some threshold value. This value depends on the best (lowest) greedy function value for the current iteration.

Let O denote the set of open plants and C the set of assigned clients, the randomized greedy algorithm is the following:

```
Algorithm 2.1:
    Construct a Greedy Randomized Solution
    Initialize: $O := \emptyset$, $C := \emptyset$
    While $C \neq I$ and $O \neq J$ do
      $\forall j \notin O$
        Define $Clients_j$
        Evaluate $\phi_j$
      Evaluate $value = \min\{\phi_j : j \notin O\}$
      Define $RCL := \{j \notin O : \phi_j \leq (1+\alpha) \times value\}$
      Select randomly $j^* \in RCL$
      Update $O := O \cup \{j^*\}$
      Update $C := C \cup Clients_j$
```

The above procedure does not guarantee obtaining a feasible solution, however it never failed during the computational experiences. Nevertheless, a failure would not cause trouble to the overall procedure since the local search phase is designed to handle solutions that may violate the capacity constraints.

Local search:
The local search phase works on problem (RP), and a penalty term is added to the assignment costs. In particular, the assignment cost of client i to plant j is $\Delta_{ij} + \rho_{ij}$ where ρ_{ij} measures the level of infeasibility of the current solution due to the assignment of client i to plant j.

$$\rho_{ij} = \begin{cases} \dfrac{f_j}{b_j} \times |s_j| \times \dfrac{d_i}{\sum_{i:p_i=j} d_i} & \text{if } i \text{ is assigned to } j \text{ and } s_j < 0 \\ 0 & \text{otherwise} \end{cases}$$

$d_i / \sum_{i:p_i=j} d_i$ is the ratio of the total demand assigned to plant j "attributed" to client i, and $(f_j/b_j) \times |s_j|$ is the cost for having $|s_j|$ extra units of capacity in plant j. Then the overall term ρ_{ij} is the cost charged to client i for increasing the capacity of plant j in order to make feasible the current set of assignments to the plant.

Strategies for exploring the neighborhoods:
As mentioned in the introduction, GRASP allows the exploration of other neighborhood structures apart from N_{shift} and N_{swap} in its Local Search phase. The extremely

reduced requirements of computational time of the method and the need to improve its overall performance justify the introduction of the following new neighborhoods:

$N_{close}(\sigma)$: Closing plant neighborhood. This neighborhood is obtained by closing one single plant and reassigning all its clients within the plants that remain open.

$$\begin{aligned} N_{close}(\sigma) &= \{\sigma' = (O', A') : \ O' = O \setminus \{j^*\}, \ A' = \{p'_i\}_{i \in I}, \\ &\quad p'_i = p_i \ \forall i \ s.t. \ p_i \neq j^*, \ p'_i \in O' \ \forall i \ s.t. \ p_i = j^*\} \end{aligned}$$

$N_{inter1}(\sigma)$: Open plants interchange neighborhood. This neighborhood can be obtained by maintaining the same set of open plants and reassigning all clients of a given plant to another open plant (and vice versa).

$$\begin{aligned} N_{inter1}(\sigma) &= \{\sigma' = (O', A') : \ O' = O, A' = \{p'_i\}_{i \in I}, \\ &\quad \exists j_1, j_2 \in J \ s.t. \ p'_i = j_2 \ \forall i \ s.t. \ p_i = j_1, \ p'_i = j_1 \ \forall i \ s.t. \ p_i = j_2, \\ &\quad p'_i = p_i \ \forall i \ s.t. \ p_i \neq j_1, j_2\} \end{aligned}$$

$N_{inter2}(\sigma)$: Open-closed plants interchange neighborhood. This neighborhood can be obtained by interchanging one open plant (j_1) with a closed one (j_2). All the clients from j_1 are reassigned to j_2.

$$\begin{aligned} N_{inter2}(\sigma) &= \{\sigma' = (O', A') : \ A' = \{p'_i\}_{i \in I}, \exists j_1 \in O, j_2 \in J \setminus O \\ &\quad s.t. \ O' = O \setminus \{j_1\} \cup \{j_2\} \ p'_i = j_2 \ \forall i \ s.t. \ p_i = j_1 \\ &\quad and \ p'_i = p_i \ \forall i \ s.t. \ p_i \neq j_1\} \end{aligned}$$

The construction phase previously described provides good starting solutions for the local search. The following strategies then guide the search of the neighborhoods.

$N_{close}(\sigma)$: If an open plant has a single client assigned to it, and the remaining open plants satisfy the aggregated demand constraint, the move is performed if it improves the objective function value.

$N_{shift}(\sigma)$: For a given solution, clients with strictly positive assignment costs ($\Delta_{ij} + \rho_{ij}$) are ordered by decreasing values. This list is sequentially explored. The current element is reassigned to the first open plant that improves the cost function (if it exists). The plants are processed in the order they were opened in the construction phase. When a client is reassigned, the elements in the list are reordered and the next client selected. If the client is not reassigned it is removed from the list. The search continues until the list is empty.

$N_{swap}(\sigma)$: The previous list is explored in a similar way. When a client is chosen, it is interchanged with the first subsequent element of the list that improves the cost function (if it exists).

$N_{inter1}(\sigma)$: For each pair of open plants the cost of reassigning all the clients from one plant to the other (and vice-versa) is evaluated. The best interchange, among those improving the cost value, is performed.

$N_{inter2}(\sigma)$: For each pair of plants (j_1, j_2), j_1 open and j_2 closed, the cost of reassigning all clients from j_1 to j_2, closing j_1 and opening j_2, is evaluated. The best interchange, among those improving the cost value and satisfying the aggregated demand constraint, is performed.

Reducing infeasibility:
A final local search tries to reach feasibility from the solution at hand. In this stage, client reassignments (N_{shift}) and client interchanges (N_{swap}) are performed provided that the overall infeasibility is reduced. The objective function value is only taken into account if several moves reduce the infeasibility.

The complete local search phase can then be described as follows:

```
Algorithm 2.2:
     While ($end = false$) do
        Explore $N_{close}$
        Explore $N_{shift}$
        Explore $N_{swap}$
        If ($solution\ not\ updated\ in\ N_{close}, N_{shift}\ and\ N_{swap}$) then
           Explore $N_{inter1}$
        If ($solution\ not\ updated\ in\ N_{close}, N_{shift}, N_{swap}\ and\ N_{inter1}$) then
           Explore $N_{inter2}$
        If($solution\ not\ updated$) $end := true$
     Final local search stage:  Search for feasible solutions
        Explore $N_{shift}$ and $N_{swap}$ (Perform movements only if infeasibility is reduced)
```

3.3 Simulated Annealing

Simulated Annealing, introduced in [18] and [5] is based on a strong analogy with the *Metropolis Algorithm* -a technique used for modeling the physical annealing process of solids. The method allows (in a limited way) transitions from one solution to the other even when the value of the cost function increases. There is thus less possibility of being trapped in a local optimum. The general algorithm is the following:

```
Algorithm 3:
     Choose initial solution $\sigma$ and temperature $T$
     While (stopping criterion not satisfied) do:
        Randomly choose a neighbor $\alpha$ of $\sigma$
        $\Delta = cost(\alpha) - cost(\sigma)$
        If $\Delta \leq 0$, then $\sigma$ := $\alpha$
```

```
    If $\Delta > 0$, then $\sigma$ := $\alpha$ with probability $exp(-\frac{\Delta}{T})$
    $T := temp\_factor \times T$
```

The implementation details of this approach for the PI-CPLP are now described.

General algorithm:
As in the other approaches, the algorithm consists of two phases, one for each of the subproblems of PI-CPLP. First, a set of K solutions is constructed using a variation of the greedy randomized algorithm. Plants are opened until the aggregated demand is satisfied. The remaining clients (if there are any) are processed in descending order of their demand and assigned to the plant with the greatest capacity slack. Then the best R solutions with different plant sets are selected. The second phase is focused on the assignment problem. For each of the R initial solutions generated in the first phase a simulated annealing algorithm as proposed in [17] is applied. In this phase a solution is skipped if its fixed cost is greater than or equal to the best cost obtained so far.

Strategies for exploring the neighborhoods:
For a given solution, neighborhoods N_{shift} and N_{swap} are jointly explored. The reason is the following: if only N_{swap} is considered, all solutions generated have the same number of clients in each location. On the other hand, if only N_{shift} is considered, there is a great number of movements rejected because they seem very infeasible.

Generation mechanism:
To generate a neighbor solution two plants, j_1 and j_2, and a client i_1 assigned to plant j_1 are selected. If there is enough capacity to reassign the client i_1 to plant j_2 then the neighbor in N_{shift} is obtained by moving client i_1 to plant j_2. If there is not enough capacity, then a client i_2 assigned to plant j_2 is selected and a neighbor in N_{swap} is obtained by interchanging clients i_1 and i_2.
Both feasible and infeasible solutions to (P) are accepted. The procedure for selecting neighbor solutions generates from one up to L neighbors. The first generated neighbor that decreases or maintains the total exceeded capacity is returned. If none exists, the one with the lowest increase in capacity excess is returned. Ties are broken using the cost change evaluation.

Acceptance rule for a move:
In addition to the standard rule of move acceptance used by Simulated Annealing, a fraction of moves that increase the total violated capacity are accepted if they improve the cost of the current solution. In particular, if a move improves the cost but increases the total exceeded capacity, it is accepted with some probability. On the other hand, moves that increase both the cost and the total exceeded capacity are not allowed.
The *Initial_Solution*, and *Neighbor_Generation* procedures follow in order to complete the algorithm description:

```
Algorithm 3.1:  Initial_Solution
    $O := \emptyset$, $C := \emptyset$
    While ($available\_capacity \leq D$) do
        Apply greedy randomized algorithm to select the next
        plant to open and update $O$ and $C$
    While ($C \neq I$) do
        Select the client $i \in I \setminus C$ with greatest $d_i$
        Select the plant $j \in O$ with greatest $s_j$
        $C := C \cup \{i\}$
        $p_i := j$
```

```
Algorithm 3.2: Neighbor_Generation
    Given a solution $\sigma = (O, A)$
    Select two plants $j_1, j_2 \in O$ $^{(1)}$
    Select a client $i_1$ from plant $j_1$
    if ($d_{i_1} \leq s_{j_2}$) then
        $\sigma' := (O', A') \in N_{shift}(\sigma)$ where $O' = O$, $p_{i_1} = j_2$, $p'_i = p_i \; \forall i \neq i_1$
    else
        Select a client $i_2$ from plant $j_2$
        $\sigma' := (O', A') \in N_{swap}(\sigma)$ where $O' = O$, $p_{i_1} = j_2$, $p_{i_2} = j_1$, $p'_i = p_i \; \forall i \neq i_1, i_2$
```

$^{(1)}$ When σ is not feasible j_1 is selected among the violated plants with a probability according to its violated capacity and j_2 is selected among the non violated plants with a probability according to its available capacity. When σ is feasible, j_1 and j_2 are chosen at random

3.4 Tabu Search

Tabu Search, introduced in [13] and [14], exploits a collection of principles for intelligent problem solving. It uses: (1) flexible memory structures; (2) an associated mechanism of control to be used with the memory structures to define tabu restrictions and aspiration criteria; (3) records of historical information for different time spans which permit the use of strategies for intensification and, in some cases, diversification.

The structure of a simple Tabu Search algorithm is:

```
Algorithm 4:
    Given an initial solution $\sigma$
    While (stopping criterion not satisfied) do:
        Create a candidate list of moves ($\equiv \{\sigma' | \sigma' \in N(\sigma)\}$)
        Choose the best admissible candidate
            (based on Tabu restrictions and aspiration criteria)
        Update admissibility conditions
            (Tabu restrictions and aspiration criteria)
```

The implementation details of the Tabu Search for the PI-CPLP are now described:

General algorithm:
Similarly to the other approaches presented in this paper, the algorithm is divided in two phases. In the first phase a set of R initial solutions are generated using Algorithm 3.1, in the same way it is done with the Simulated Annealing approach. In the second phase, the assignment problem is solved for each of the R initial solutions using Tabu Search. An initial solution is skipped if its fixed cost is greater than or equal to the best cost obtained so far.

Objective function:
Since infeasible solutions for (P) are allowed during the search, the objective function to be minimized is defined by:

$$f_1 = z' + \rho \times \sum_{j \in O : s_j < 0} |s_j|$$

where ρ is a penalty parameter, that is adjusted as in [11]. It increases or decreases depending on the number of infeasible or feasible solutions obtained in the last iterations.

Move:
Given a solution $\sigma = (O, A)$, moves into N_{shift} and N_{swap} are considered during the search process. Assignments of clients to plants with $\Delta_{ij} > 0$ are considered elite since they may have an opportunity to improve. Candidate moves are those associated to elite clients. When exploring candidate moves, if a client i fits in a plant j ($p_i \neq p_j$), only the reassignment of this client to plant j is considered. On the contrary, if client i does not fit in plant j ($p_i \neq p_j$), only the interchanges between client i and clients assigned to plant j are considered. Clients are processed in descending order of Δ_{ij}. Candidate moves are explored until one improves the cost. If there is no such candidate, the one with the minimum increase is selected.

Tabu list:
The tabu list is a circular list of ordered pairs (i, j) with a fixed size ($tabu\_size$). It is generated at the beginning of the search process and updated at each iteration. If a client i is moved from plant j_1 to plant j_2 during the iteration l, any candidate move that tries to assign the client i to the plant j_1 will be forbidden during the next $(l + tabu\_size)$ iterations.

Aspiration criteria:
At any iteration of the search process, the tabu restriction for a move can be bypassed if its objective function value is better than the best produced so far and it does not increase infeasibility.

Diversification:
During the search process some movements seem very unattractive, one consequence being that some regions of the search space are never explored. For this reason, at

each iteration a frequency matrix is updated. Any element fr_{ij} of this matrix records the number of times that the client i has been assigned to plant j during the search. When the search is completed the process is restarted replacing the cost matrix c by the frequency matrix fr for a few iterations. Thus the assignments with very low frequency are selected. The search process then continues using matrix c. This is used as a strategy for diversifying the tabu process.

4 Computational Experience

All heuristic approaches presented here have been tested on the set of 33 problems used in [2]. The set is divided into four classes, C_1, C_2, C_3 and C_4, according to the dimension of the problems, $(m \times n)$, where m is the number of clients and n is the number of plants. C_1 contains six (20×10) problems, C_2 contains eleven (30×15) problems, C_3 contains eight (40×20) problems and C_4 contains eight (50×20) problems.

The parameters used in the different algorithms are the following:

EEA: The initial populations are generated randomly. The population size consists of 3 members. At the first level, the crossover is applied to 95% of the population, and the mutation to 10% of the strings components. At the second level, the crossover is applied to 10% of the population, and the mutation to 95% of the strings components. At both levels, the shuffle reinitiates the search after 10 generations without improvement. At the first level, the search is stopped after 500 generations, and after 200 at the second level of search.

GRASP: The total number of iterations is 50, and the value of parameter α is 0.4.

SA: The parameter K has value 300, R has value 5, and L has a value equal to $m/2$. A description of the remaining parameters can be found in [17]. They are:

freez_limit $= 20$, *size_factor* $= 3$, *cutoff* $= 2$, *initial_temp* $= 30$,

temp_factor $= 0.97$, and *min_percent* $= 0.05$.

TS: The parameter K has value 300, and R has value 5. The *tabu_size* is $m/4$, which is also the number of iterations during diversification. The stopping criteria corresponds to 350 iterations without improvement.

Experiments with the different heuristic approaches have been performed on a VAX-2000 system. Results correspond to average values over 25 runs.

To evaluate the quality of the solutions generated with the proposed methods all the test problems have also been solved using the Version 4.0 of the CPLEX software package, in a time-limit of two hours of CPU (7200 seconds). The experiments with CPLEX have been performed on a Sun Sparc Station 10/30 with 4 HyperSparc at 100 Mhz using only one processor. After some experimentation to find an appropriate strategy to solve the problems, only problems in the C1 class could be solved optimally from scratch within the given time-limit. For the other classes of test problems, the value of the best solution found with the proposed metaheuristic methods was used as an incumbent value to speed up the tree search. With this additional information, CPLEX could optimally solve all the problems in the C2 class and most of the problems in C3 and C4. However, the optimality of the obtained solutions could not be proved for test problems 18, 20 and 21 in C3 and 26, 30, 31 and 33 in C4. The results obtained with CPLEX are depicted in Figure 7. The cells marked with an asterisk correspond to values for which optimality could not be proved. Although a rigorous comparison of times is not possible since the experiments were carried out in different computers, these results illustrate the difficulty to obtain good quality solutions in a reasonable amount of computation time for the considered set of test problems.

Figure 4 depicts bar graphs for each class of problems with the average of the mean deviation from the best known value. For each class k, the average, A_k, has been obtained with the expression:

$$A_k = \frac{\sum\limits_{j \in C_k} \mu_j}{|C_k|} \qquad\qquad k = 1, ..., 4.$$

where μ_j is the mean deviation from the best known value, $bestj$, for problem j obtained with CPLEX as explained above:

$$\mu_j = \frac{\dfrac{\sum\limits_{i=1}^{25}(solution_{ij} - best_j)}{best_j}}{25}$$

and $solution_{ij}$ is the value of the best solution obtained for problem j in run i.

For the C1 class of problems, EEA and TS provide the best results. For the other classes, TS provides the best results, followed by SA and GRASP which behave similarly. The behavior of TS is very satisfactory since its average deviation ranges from 0.37% to 0.81% for the different classes of problems. Excepting EEA, approaches have proven to be robust, since their performance does not vary significantly as the problem dimension increases. In the case of EEA, although for small dimensions the quality of the solutions are equivalent to those of TS, its performance decreases as dimension grows and for the classes of larger problems, the average deviation is 3.4%. We have not found a satisfactory explanation for the decrease in the algorithm performance. In the case of GRASP the improvement with respect to the initial solutions generated with the Greedy Randomized Heuristic has proven to be significant.

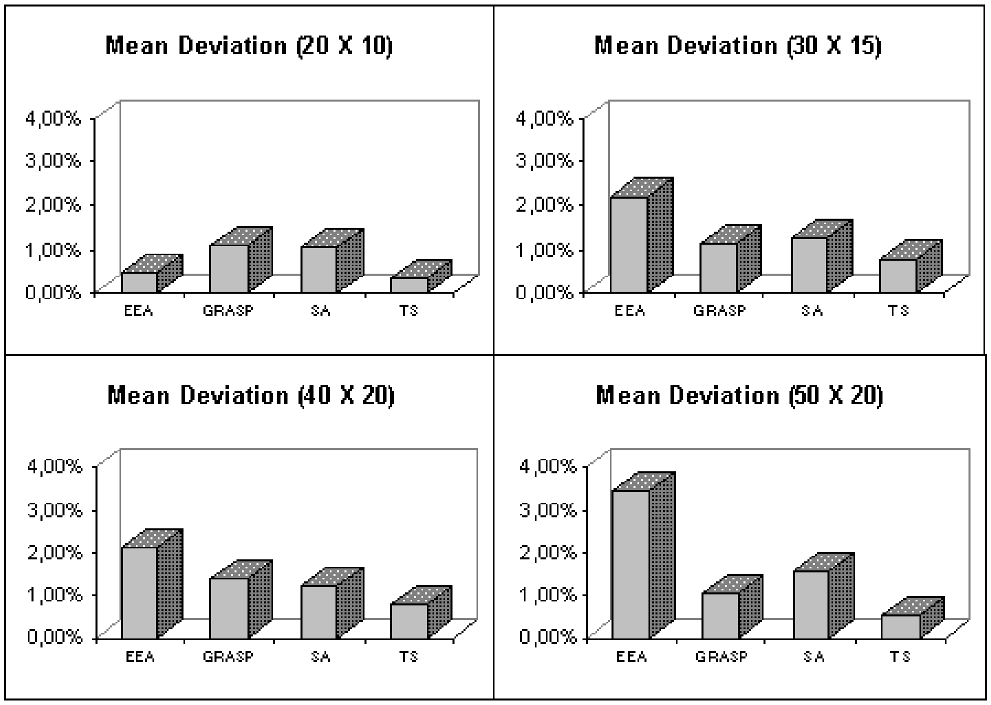


Fig. 4: Bar graphs for each class of problem.

The average CPU times (in seconds) both to obtain the best solution, T_{1k}, and to terminate the procedure, T_{2k}, were computed for each class of problems. These values are shown in Figure 5. They have been obtained with the expressions:

$$T_{1k} = \frac{\sum\limits_{j \in C_k} \tau_{1j}}{|C_k|}, \qquad T_{2k} = \frac{\sum\limits_{j \in C_k} \tau_{2j}}{|C_k|}, \qquad k = 1, ..., 4.$$

where τ_{1j} and τ_{2j} are the mean times over the 25 runs for problem j, i.e.

$$\tau_{1j} = \frac{\sum\limits_{i=1}^{25} t_{1j}^i}{25}, \qquad \tau_{2j} = \frac{\sum\limits_{i=1}^{25} t_{2j}^i}{25}.$$

where t_{1j}^i and t_{2j}^i denote, respectively, the best solution and global times required by the i-th execution of problem j.

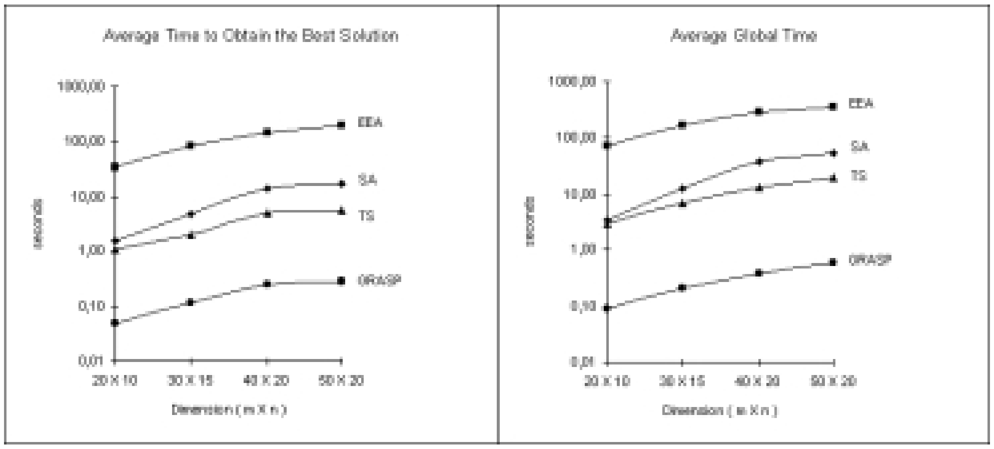
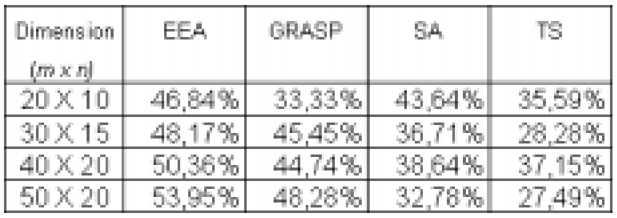


Fig. 5: Average CPU times.

Note that the vertical axis is a logarithmic scale. Hence, it is clear from this figure that the GRASP requires several orders of magnitude less time than the second best approach (which is TS) both for terminating and for generating its best solution. On the other extreme, EEA requires at least one order of magnitude more time than all the other approaches. Figure 6 shows the average proportions of the overall time required by each of the heuristics to generate its best solution for each class of problems. As can be seen, it is very unlikely that an increase in the overall time of any of the heuristics could provide any improvement on the best generated solutions.



Dimension (*m x n*)	EEA	GRASP	SA	TS
20 X 10	46,84%	33,33%	43,64%	35,59%
30 X 15	48,17%	45,45%	36,71%	28,28%
40 X 20	50,36%	44,74%	38,64%	37,15%
50 X 20	53,95%	48,28%	32,78%	27,49%

Fig. 6: Required average proportions of the overall times.

The times consumed by the experiments with CPLEX cannot be compared to those shown in Figure 5. On the one hand, excepting for the C1 class, the tree search started with the incumbent value provided by the best of the heuristic approaches. On the other hand, the required times present a tremendous variability. Thus, the average time for each class is, to a great extent, misleading. Finally, in spite of corresponding to experiments in a different computer, in many cases they can safely be considered far beyond the time-limits of Figure 5 even for the logarithmic scale. However, these times are shown in Figure 7, since they permit to appreciate the efficiency of the proposed methods.

Figure 7 also shows the best results obtained with each of the algorithms presented. The shadowed cells correspond to the best overall value obtained with the proposed methods and the cells in bold to the optimal values (or best known values when optimal values are not known). The last column shows the percent deviations between these two values. The table also shows the best known values reported in [2]. The average improvements with respect to previous known value are depicted in column *Improvement from previous best-known.*

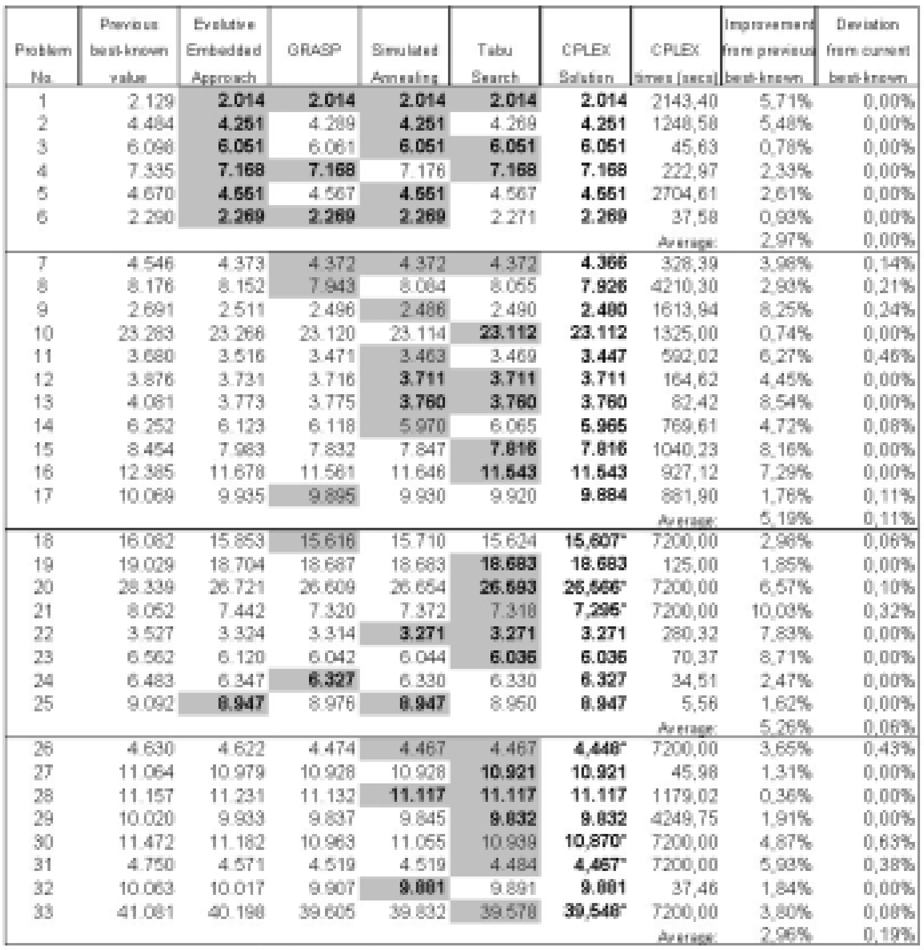
The results in Figure 7 show that, in general, the algorithm with the best average behavior also provided the overall best solution. Note that TS gives the best solution in 21 out of the 33 test problems. The exception is C_1 where EEA always provided the best solution but on the average did not perform as well as TS.

As can be seen, all the algorithms have outperformed the best known value for all the test problems (excepting EEA for problem 28). For each class of problems, the average improvement from the best known value ranges from 2.96% to 5.26% which in our opinion is quite significant. The percent deviation from the optimal (or best known) solutions is very low since it never exceeds 0.63% and on the average it takes a value of 0.10%. It can be observed that the different approaches generate, in general, very high-quality solutions. In particular, EA provides the optimal solutions to all problems in C1. With respect to the complete set of test problems, TS generates the optimal solution (or best-known when optimal is not known) for nearly 50% of the problems. Taking into account the required computational times, these results are really good, specially considering the difficulties of CPLEX not only to optimally solve the problems but also to prove optimality of the given solutions.

5 Conclusions

This paper considers different heuristic approaches for the PI-CPLP: An Evolutive Algorithm, a Greedy Randomized Adaptive Search Procedure, a Simulated Annealing and a Tabu Search.

All the algorithms proposed share three common characteristics. First, they con-



Problem No.	Previous best-known value	Evolutive Embedded Approach	GRASP	Simulated Annealing	Tabu Search	CPLEX Solution	CPLEX times [secs]	Improvement from previous best-known	Deviation from current best-known
1	2.129	**2.014**	**2.014**	**2.014**	**2.014**	**2.014**	2143,40	5,71%	0,00%
2	4.484	**4.251**	4.289	**4.251**	4.269	**4.251**	1248,58	5,48%	0,00%
3	6.098	**6.051**	6.061	**6.051**	**6.051**	**6.051**	45,63	0,78%	0,00%
4	7.335	**7.168**	**7.168**	7.176	**7.168**	**7.168**	222,97	2,33%	0,00%
5	4.670	**4.551**	4.567	**4.551**	4.567	**4.551**	2704,61	2,61%	0,00%
6	2.290	**2.269**	**2.269**	**2.269**	2.271	**2.269**	37,58	0,93%	0,00%
							Average:	2,97%	0,00%
7	4.546	4.373	4.372	4.372	4.372	**4.366**	328,39	3,98%	0,14%
8	8.176	8.152	7.943	8.084	8.065	**7.926**	4210,30	2,93%	0,21%
9	2.691	2.511	2.496	2.486	2.490	**2.480**	1613,94	8,25%	0,24%
10	23.283	23.266	23.120	23.114	**23.112**	**23.112**	1325,00	0,74%	0,00%
11	3.680	3.516	3.471	3.463	3.469	**3.447**	592,02	6,27%	0,46%
12	3.876	3.731	3.716	**3.711**	**3.711**	**3.711**	164,62	4,45%	0,00%
13	4.081	3.773	3.775	**3.760**	**3.760**	**3.760**	82,42	8,54%	0,00%
14	6.252	6.123	6.118	5.970	6.065	**5.965**	769,61	4,72%	0,08%
15	8.454	7.983	7.832	7.847	**7.816**	**7.816**	1040,23	8,16%	0,00%
16	12.385	11.678	11.561	11.646	**11.543**	**11.543**	927,12	7,29%	0,00%
17	10.069	9.935	9.895	9.930	9.920	**9.884**	881,90	1,76%	0,11%
							Average:	5,19%	0,11%
18	16.082	15.853	15.616	15.710	15.624	**15,607***	7200,00	2,98%	0,06%
19	19.029	18.704	18.687	18.683	**18.683**	**18.683**	125,00	1,85%	0,00%
20	28.339	26.721	26.609	26.654	**26.593**	**26,566***	7200,00	6,57%	0,10%
21	8.052	7.442	7.320	7.372	7.318	**7,296***	7200,00	10,03%	0,32%
22	3.527	3.324	3.314	**3.271**	**3.271**	**3.271**	280,32	7,83%	0,00%
23	6.562	6.120	6.042	6.044	**6.036**	**6.036**	70,37	8,71%	0,00%
24	6.483	6.347	**6.327**	6.330	6.330	**6.327**	34,51	2,47%	0,00%
25	9.092	**8.947**	8.976	**8.947**	8.950	**8.947**	5,56	1,62%	0,00%
							Average:	5,26%	0,06%
26	4.630	4.622	4.474	4.467	4.467	**4,448***	7200,00	3,65%	0,43%
27	11.064	10.979	10.928	10.928	**10.921**	**10.921**	45,98	1,31%	0,00%
28	11.157	11.231	11.132	**11.117**	**11.117**	**11.117**	1179,02	0,36%	0,00%
29	10.020	9.933	9.837	9.845	**9.832**	**9.832**	4249,75	1,91%	0,00%
30	11.472	11.182	10.963	11.055	10.939	**10,870***	7200,00	4,87%	0,63%
31	4.750	4.571	4.519	4.519	4.484	**4,467***	7200,00	5,93%	0,38%
32	10.063	10.017	9.907	**9.881**	9.891	**9.881**	37,46	1,84%	0,00%
33	41.081	40.198	39.605	39.832	39.578	**39,546***	7200,00	3,80%	0,08%
							Average:	2,96%	0,19%

Fig. 7: Best results for the algorithms.

sider the structure of the problem by defining two phases, one for the plants' selection problem and another one for the allocation subproblem. Second, they search for feasible solutions to the original problem using the relaxation obtained by aggregation of the capacity constraints. Third, they explore the same neighborhood structures.

These three characteristics can be seen as a template to which the proposed methods have been adjusted. The different components of all the algorithms are presented. Also the neighborhood structures and the strategies used to explore them are described.

Computational experiments have been performed on a set of test problems from the literature and the behavior of the different algorithms has been compared both to the previous best known values and to the best solutions obtained with CPLEX in two hours of CPU time. The obtained results prove that the template used for the proposed algorithms is successful for solving the PI-CPLP.

With respect to the methodologies that have been used, all have proven to be effective, since for all the test problems the previous best known value has been improved with all of them (with the exception of EEA for one single problem). Also the deviations from the best solutions generated by CPLEX prove the quality of the proposed approaches.
EEA has provided the optimal solutions for all the small-sized problems. However, its performance decreased considerably for larger instances. We have not found an adequate explanation for the decrease on its performance for larger problems. This invites to further investigate on that point.

SA and GRASP have proven to be good alternatives, in terms of the quality of the generated solutions and of the average deviation from the best know values. Additionally, GRASP behaved remarkably well in terms of the required computational time.

However, TS has proven to be a really effective method both in terms of the quality of the generated solutions and in terms of its deviation from the best known value for problems of all sizes.

The results obtained suggest that a hybrid approach combining the speed of GRASP for finding good initial solutions with other more sophisticated search techniques such as TS is extremely promising.

References

[1] J. Barceló, and J. Casanovas, "A heuristic lagrangean algorithm for the capacitated plant location problem", *European J. Oper. Res.*, **15**(1984)212-226.

[2] J. Barceló, A. Hallefjord, E. Fernández, and K. Jörnsten, "Lagrangean relaxation and constraint generation procedures for capacitated plant location problems with single sourcing", *Oper. Res. Spekt.*, **12**(1990)79-88.

[3] J. Barceló, E. Fernández, and K. Jörnsten, "Computational results from a new lagrangean relaxation algorithm for the capacitated plant location problem", *European J. Oper. Res.*, **53**(1991)38-45.

[4] J.E. Beasley, "An algorithm for solving large capacitated warehouse location problems", *European J. Oper. Res.*, **33**(1988)314-325.

[5] V. Cerny, "A thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm", *J. Optim. Theory Appl.*, **45**(1985)41-51.

[6] H. Delmaire, A. Langevin and D. Riopel, "Evolution systems and the Quadratic Assignment Problem", *Working Paper G-95-24*, GERAD, Montreal (Quebec), Canada.

[7] H. Delmaire, A. Langevin and D. Riopel, "A decomposition framework using genetic algorithms for global optimization", *Working Paper G-96-08*, GERAD, Montreal (Quebec), Canada.

[8] H. Delmaire, A. Langevin and D. Riopel, "Skeleton-based facility layout design using genetic algorithms", *Ann. Oper. Res.*, **69**(1997)85-104.

[9] T.A. Feo and M.G.C. Resende, "Greedy randomized adaptive search procedures", *J. Global Optim.*, **6**(1995)109-133.

[10] R.D. Galvão, L.R. Raggi, "A Method for Solving to Optimality Uncapacitated Location Problems", *Annals of Operations Research*, **18** (1989)225-244.

[11] M. Gendreau, G. Laporte and R. Séguin, "A tabu search heuristic for the vehicle routing problem with stochastic demands and customers", *Oper. Res.*, n^o3, **44**(1996)469-477.

[12] M. Guignard and K. Opaswongkarn, "Lagrangean dual ascent algorithms for computing bounds in capacitated plant location problems", *European J. Oper. Res.*, **46**(1990)73-83.

[13] F. Glover, "Tabu Search-Part I", *ORSA J. Comput.*, n^o3, **1**(1989)190-206.

[14] F. Glover, "Tabu Search-Part II", *ORSA J. Comput.*, n^o1, **2**(1990)4-32.

[15] J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[16] S.K. Jacobsen, "Heuristics for the capacitated plant location model", *European J. Oper. Res.*, **12**(1983)253-261.

[17] D.S. Johnson, C.R. Aragon, L.A. Mcgeoch, and C. Schevon, "Optimization by simulated annealing: an experimental evaluation; Part II, Graph coloring and number partitioning", *Oper. Res.*, **39**(1990)370-406.

[18] S. Kirpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by simulated annealing", *Science*, **220**(1993)671-680.

[19] J.G. Klincewic and H. Luss, "A lagrangean relaxation heuristic for capacitated facitily location with single-source constraints", *J. Operational Res. Soc.*, **37**(1986)495-500.

[20] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, 1992.

[21] P.R. Mirchandani, R.L. Francis (eds.), *Discrete Location Theory*, Wiley-Interscience, 1990.

[22] R. Sridharan, "The capacitated plant location problem", *European J. Oper. Res.*, **87**(1995)203-213.