

A CHAIN-INTERCHANGE TABU SEARCH
METHOD
(Tabu search in solving p -facility location-allocation
problems)

Nenad MLADENović
GERAD and Ecole des Hautes Etudes Commerciales
5255, avenue Decelles, Montreal, Canada H3T1V6

José A. MORENO
Departamento de Estadística, I.O. y Computación.
Universidad de La Laguna, Spain.

J. Marcos MORENO-VEGA
Departamento de Estadística, I.O. y Computación.
Universidad de La Laguna, Spain.

7 de junio de 2003

Resumen

A new category of Tabu search, Chain-interchange heuristic method is proposed in this paper. It can be viewed as a Tabu search method adopted for a large class of problems, as well as extended interchange heuristic method. Instead of interchanging two solution attributes in order to generate a set of neighbourhood of a current solution, we interchange positions of four attributes. This simple extension of the well known descent local search *1-interchange* method allows us to obtain an efficient descent-ascent local search heuristic. Thus, our move could be easily augmented in any problem where Interchange heuristic has been used, now with property of escaping from local optima trap. Some location-allocation problems that could be solved by the same *chain-interchange* algorithm only by changing objective function are listed. Moreover, most of the problems listed are for the first time suggested to be solved by Tabu Search. Computer results are reported.

Keywords: tabu search, chain-interchange, location-allocation.

1. Introduction

A new category of Tabu search heuristics, **Chain-interchange** method for solving combinatorial optimization problems is proposed. It could be seen as a Tabu search method adopted for a large class of problems, as well as extended interchange heuristic.

Tabu search (TS) was introduced by Glover (1986) and independently by Hansen (1986) (under the name *Steepest Ascent Mildest Descent*) as a meta-heuristic designed to achieve a global optimum to combinatorial optimization problems. TS belongs to a class of *Local Search* Heuristic Methods, or more specifically to a class of *Descent - Ascent* Methods [14]. The major components include a *short - term memory* process which is the core component of the search procedure, an *intermediate memory* process for regionally intensifying the search, and a *long - term memory* process for globally diversifying the search. The short - term memory process is implemented through a set of *tabu conditions* and the associated *aspiration criteria*. The idea is to prevent cycling in descent-ascent process by avoiding reversal or repetition of previously visited solutions. Intermediate and long-term memory processes are performed by restricting the search to a special subregion and by inducing the search to a new subregion of the solution space. TS has already been applied to an increasingly wide spectrum of problems. For a survey of a variety of successful applications, see [9], [10] and [11] .

Tabu Search application papers have usually contained the authors suggestions on how to answer the following problem specific questions: What neighbourhood structure is used in the search? How short, intermediate and long-term memory processes for a given optimization problem are defined? What TS rules among many could be used?, etc. Hence, on one side there are metaheuristic rules (strategical and tactical), but on another there are problems with their specific features. As there are so many possible answers on all these questions, no two authors would make use of the same TS method in solving the same problem. As a result, several TS algorithms could be designed in solving the same combinatorial optimization problem. In our opinion, the preceding facts suggest a possible new approach in the classification of TS methods into a few groups according to specific criteria.

It can be seen, according to TS application papers, that there are many different problems that use the same move (or neighbourhood structure) or the same

combination of moves in the local search process. For example, the *1-interchange* (or swap) move has successfully been applied in solving general fixed charge problems ([32]), pipe-line design problems ([12]), *k*-cardinality tree problems ([18]), *k*-cardinality subgraph problems ([24]), etc. These examples show that local search logic can bring together problems that do not have similar mathematical programming models. They all have a set of neighbourhood points obtained in the same way: Any element removed from the set that represents a solution can be replaced by each another element not in the current solution. A similar conclusion holds for *insertion* and for *add/drop* moves, for example. The combination of *add/drop* and *1-interchange* moves has been applied in solving *p*-median and simple plant location problems ([16]), multicommodity location-allocation with balancing requirements ([4]), etc. Thus, a neighbourhood structure is suggested as a key for the classification of TS methods. In this paper we have two objectives:

- Firstly, to initiate a different approach in applied Tabu Search by *finding a class of optimization problems that could be solved by using the same TS memory processes*, i.e., the same data structure. Our approach is based on the fact that there are more optimization problems that could be solved by TS than data structures developed for solving them in an efficient way. This approach allows us to solve a large class of optimization problems by changing the subroutine that evaluates objective function value only.
- Secondly, to introduce the *chain-interchange* move, which is an extension of the well-known *1 – interchange* move. Instead of interchanging one solution attribute that is in the solution with one that is not, we interchange four attributes. In that way TS *recency-based memory* is easily obtained, i.e., the possibility of getting out from the local optima trap can be achieved without additional efforts. Moreover, the chain-interchange move connects the descent Interchange heuristic method and the descent-ascent TS method. In other words, numerous problems that have been solved by Interchange heuristic before, now can be solved by basic TS, using almost the same code.

The **Chain-Interchange TS method** (CITS) is the proposed new category of TS methods that use the chain-interchange move. It satisfies both our objectives above and has the following basic properties: (i) it makes a move at each iteration to the best neighbouring solution, even if it is not better than the current solution

in order to avoid local optima trap; (ii) two waiting (tabu) lists are introduced to prevent cycling. The lists consist of elements waiting to go out of the solution and to go into the solution respectively; (iii) bring into the solution an element which has been out of the solution for the largest amount of iterations. As a result, searches can be induced to a new regions of the solution space.

In the next section, some problems that could be solved by 1-interchange and thus by Chain-Interchange TS method (CITS), are given. Our list of applications is meant to be illustrative, not exhaustive. In the same section the formulation of p -facility location-allocation problem is given because of the following reasons: a detailed pseudo-code of algorithm (in Section 3) is explained in terms of location-allocation; computer experience will be reported (in Section 4) for p -median problem only; previously, TS was not recommended as a means of solving p -facility location-allocation problems. Section 5 concludes the paper.

2. Applications of interchange neighbourhood

The list of problems that could be solved by TS using neighbourhood structure obtained by 1-interchange move is extensive. But different data structures could be developed for solving them, depending how the solution space is defined. In some mathematical programming problems, the feasible solution is represented as a subset of *variables*, in applied graph theory solutions are defined as a subset of *edges* or subset of *vertices* of some graph associated to the problem, etc. Then the variables, edges, vertices etc., are interchanged in order to get a set of neighbourhood solutions. In other words, several different data structures could be designed to support 1-interchange and chain-interchange (or k -interchange and k -chain-interchange in general) moves, but each of them still could contain a large number of optimization problems.

Mathematical Programming examples

Even the move used in Simplex method for *Linear Programming* (LP) is a 1-interchange move: a variable removed from the set of basic variables is replaced by a nonbasic variable according to the simplex pivoting rule. This is equivalent to moving from the current corner point solution to an adjacent one. In the regular (non degen-

erate) case, the complete neighbourhood consists of n adjacent corner points, since the feasible domain is an n -dimensional simplex (polyhedron). Of course, we need not use heuristics in solving LP because there exist pivoting rules (Bland's rule for example) that guarantee no cycling, and since the local minimum obtained is a global one as well. But *Concave minimization* problems are NP-hard and therefore heuristic methods are welcome. In Concave minimization the optimal solution is again in the extreme point of the polyhedron and the move from one corner solution point to another is obtained by interchange of variables as in the simplex method. Interchange of variables has already been proposed in solving some *Integer Programming* problems by TS as well (see [32]).

Applied graph theory examples

The *1-interchange* and thus *chain-interchange* move could be applied to a combinatorial optimization problem where the solution could be easily calculated if the optimal spanning tree (arborescence) of that graph is known. Consider two (nondirected or directed) spanning trees $T_1 = (V, E_1)$ and $T_2 = (V, E_2)$ of the graph G . The *distance* between these two trees is denoted by $d(T_1, T_2)$ and is defined to be the number of edges in T_1 not in T_2 (or equivalently the number of edges in T_2 not in T_1 since both T_1 and T_2 have $n - 1$ edges). If the distance $d(T_1, T_2) = 1$, i.e. if

$$E_1 \cup E_2 - E_1 \cap E_2 = \{e_1, e_2\},$$

where $e_1 \in E_1$ and $e_2 \in E_2$, then T_2 could be derived from T_1 by removing the edge e_1 from T_1 and introducing the edge e_2 . Such a transformation of T_1 into T_2 is called an *elementary tree transformation* (recently, a simpler move of interchange type has been proposed in [28]). In spanning tree problems the $c_{ij} = c(e)$ terms could represent the distances as well as the costs, and the applications include the design of transportation systems (the vertices are terminals and the edges are highways, pipelines, air routes, and so on) as well as communication systems. The vertices could also represent interfacing satellite computer terminals located at various distances from one another. Let us mention some of the known hard optimization problems whose solution is a spanning tree: the *optimal pipeline design problem* ([20]); *1-terminal TELPAK problem* ([29]); *minimum spanning tree problem with time window constraints* ([31]), etc. In all of these problems, $n - 1$ edges out of $m = |E_G|$ have to be found. The same data structure could be used in solving *k-cardinality tree* ([7], [18]) and *k-cardinality subgraph* ([5], [24]) problems, where k out of m edges of G

should be detected. Note that interchanges of r edges of some graph is usually called r -opt strategy ([21]).

The p -facility location-allocation problem

The formulation of p -facility location-allocation problems is given in more detail because of the following reasons: a detailed pseudo-code of algorithm in the next section is explained in terms of location-allocation; computer experience will be reported for p -median problem only, i.e. on one well known discrete location-allocation representative; previously, TS was not recommended as a means of solving p -facility location-allocation problems.

A Location-Allocation (LA) problem consists in finding the best selection of points to open facilities at them and the way for serving the users. In order to formalise the discrete LA models, we consider a set L of m location points, a set D of n demand points and a $m \times n$ matrix with the costs $c(x, u)$ of attending a user at u from a facility at x , $\forall x \in L, \forall u \in D$. Every alternative solution is given by the set X of location points for the facilities and the allocation of every demand point $u \in U$ to the facility point $A(u) \in X$ that serves every user at u . The objective function to be minimized depends on the locations X and the allocations $A(\cdot)$.

For directly-allocated models the objective cost function is monotone and every user at u is allocated to the facility point $A(u)$ such that $c(A(u), u) \leq c(x, u)$, $\forall x \in X$. Then the solution space for these problems is $\mathcal{S} = \{S = X/X \subseteq L\}$. In the other hand and for any LA problem, given the allocation $A(\cdot)$ of the users, the location of the facilities is given by $X = A(D)$. So the solution space can be considered as $\mathcal{S} = \{S = A/A : D \rightarrow L\}$. Another way for giving the allocations consists in providing the partition of the demand point set in the points allocated to every facility points, i.e., by giving $U(x) = \{u \in D : A(u) = x\}$, $\forall x \in X$.

However, for describing the search procedures for these problems, it is convenient to use all the items related with the solution; i.e., the locations given by X , the allocations given by $A(\cdot)$ and the partition given by $U(\cdot)$. In any implementation we can use any of them for performing any step since $X = A(D)$ and $U(x) = A^{-1}(x)$, $\forall x$. Therefore, we can use the solution space:

$$\mathcal{S} = \{S = (X, A, U) / X \subseteq L, A : D \rightarrow X, U : X \rightarrow 2^D \text{ with } A(u) = x \text{ iff } u \in U(x)\}.$$

The p -facility problem is a directly-allocated LA problem with solution space

$$\mathcal{S} = \{S = X/X \subseteq L : |X| = p\}.$$

Then the objective function characterizes the problem (*the p -median problem, the p -center problem, the balanced p -centdian problem, the median p -center problem, the p -capture problem, the p -covering problem, the p -equity problem, the p -antcenter problem, the p -mean problem, the p -log-median problem*). A list of these problems is given in [1]. Among them the p -median problem is most known. Let

$$c(X, u) = \min_{x \in X} c(x, u).$$

The p -median problem. The objective function is the average of the allocation costs.

$$f(X) = \frac{1}{|U|} \sum_{u \in U} c(X, u).$$

Note that a list above can be enlarged with four p -dispersion problems: *MaxMinMin*, *MaxSumMin*, *MaxMinSum* and *MaxSumSum*, (see [6]).

3. A Chain-Interchange TS method

Different neighbourhood structures have been suggested in the literature for solving LA problems by local search heuristics (see [3], [12], [17], [19], [22], [26], etc.). We shall pay attention to two among them which are of interchange type: *reallocation* (or *2-opt*) and *substitution*. Then we shall show how to extend those moves to *chain-reallocation* and *chain-substitution*. Generalization to r -chain-reallocation (or r -chain-opt) and r -chain-substitution (or r -chain-interchange mechanism [27]) is quite obvious.

Let (\mathcal{S}, f) be the solution space and the objective function of a combinatorial problem. For a location-allocation problem, let $S = (X, A, U)$ be the current solution given by the locations X , the allocations A and the partition U . The usual elemental move for the location-allocation problems is the following:

- **Reallocation** of demand point u to facility point x .
 - Take $x \in X$ and $u \in D$ with $A(u) \neq x$ and $U(A(u)) \neq \{u\}$.
 - Do $A(u) \leftarrow x$ and update X and U accordingly.

The set of feasible solutions of a p -facility location-allocation problem is $\mathcal{S} = \{S = X/X \subseteq L : |X| = p\}$. Then the usual elemental move for these problems is

- **Substitution.**

- Take $x \in X$ and $y \notin X$ and do $X \leftarrow X - \{x\} + \{y\}$.

Chain-interchange moves

A common property of *Reallocation*, on one side and *Substitution* moves on the other is that they all interchange positions of some entities (solution attributes) that belong to the solution with some that are not in the solution. If the solution is represented by pairs $(x, u), x \in L, u \in D$ (or by edges of associated graph), then a reallocation move means that one edge goes out from the solution and another one goes into it. On the other hand if the solution is represented as a subset of L ($X \subseteq L$) then the interchange move is a substitutional move: one facility (vertex of associated graph) goes in X , another goes out (in [27] λ -substitution move is called λ -interchange mechanism).

We shall now introduce a new *chain-interchange* move. This move could be obtained as an extension of reallocation and substitution moves: if elements that should be inter-changed from one step to another are the edges (x, u) of associated graph, we have a *chain-reallocation* move; if such elements are facilities $x \in X$, then a *chain-substitution* move is obtained. We shall describe in detail only the chain-substitution move and give a detailed pseudo-code for it in the next subsection. Analog results hold for chain-reallocation.

Let us divide a set of facility points L into four disjoint subsets X_1, T_1, X_2 and T_2 , such that:

$$X = X_1 \cup T_1, X_1 \cap T_1 = \emptyset, |X| = p$$

$$L \setminus X = X_2 \cup T_2, X_2 \cap T_2 = \emptyset, |L \setminus X| = m - p,$$

with $|X_1| = p - t_1, |T_1| = t_1, |X_2| = m - p - t_2$ and $|T_2| = t_2$. Sets T_1 and T_2 are two tabu lists with cardinalities t_1 and t_2 respectively. Then the chain-substitution move is as follows:

- **Chain-substitution move.**

- Take $x_{out} \in X_1, x_{in} \in X_2, x' \in T_1$ and $x'' \in T_2$. Do the following:

$$\begin{aligned}
X_1 &\leftarrow X_1 - \{x_{out}\} + \{x'\}; \\
T_1 &\leftarrow T_1 - \{x'\} + \{x_{in}\}; \\
X_2 &\leftarrow X_2 - \{x_{in}\} + \{x''\}; \\
T_2 &\leftarrow T_2 - \{x''\} + \{x_{out}\}.
\end{aligned}$$

As the result of above four moves we obtain one substitution move, i.e., we take $x_{out} \in X$ and $x_{in} \notin X$ and get $X \leftarrow X - \{x_{out}\} + \{x_{in}\}$. The question is why four (instead of only one) substitution moves are needed? It is done because now not only descent but also ascent moves are allowed in the search process. By taking out facility point x_{out} from $X_1 \subseteq X$, not from X , some facility points are forced to be in the solution (i.e., in T_1) for a certain number of steps, even if excluding them from the solution would produce a better solution. In other words, before $x' \in T_1$ was chosen to leave the waiting list T_1 (in First In First Out - FIFO discipline), it had been kept in the solution for $t_1 = |T_1|$ iterations. Therefore a maximum of $t_1 + 1$ ascent moves are allowed in the search process (if the current solution is local minimum). The same holds true for facility point x'' , i.e., before it went out from T_2 , it had been forced out of the solution for t_2 iterations. Let us emphasize this fact in following Property.

Property 1 *The sequence of objective function values $f_i, i = 1, 2, \dots$ obtained by successive implementation of the chain-substitution move above (under the FIFO rule in T_1 and T_2), has a period of length $\alpha \geq |T_1| + |T_2| + 2$.*

Proof. The sequence $f_i, i = 1, 2, \dots$ could 'get caughtup in a loop' (a cycle of numbers that is repeated endlessly) only if the same solution X is obtained after α steps. We shall show that $\alpha \geq t_1 + t_2 + 2$. Actually, the minimum number of iterations of facility $x_{out}^{(i)} \in X_1$ needed in order to belong to X_1 again is $t_1 + t_2 + 2$: one step to go out from X_1 , t_2 iterations being in T_2 because of FIFO rule, at least one step in X_2 and t_1 steps in T_1 . It is easy to see that all other facilities that went out from the solution after $x_{out}^{(i)}$ could be in T_1 , thus, in the solution. \diamond

To change the positions of two elements with indices *out* and *in* in some ordered set $x(\cdot)$, it is necessary to perform 3 operations: $xx := x(out); x(out) := x(in); x(in) := xx$. Note that for our four changes we need only 5 operations (k and l are indices of entities in tabu lists T_1 and T_2 respectively):

Chain-substitution-move (out, k, in, l, x)

$xx := x(out);$
 $x(out) := x(k);$
 $x(k) := x(in);$
 $x(in) := x(l);$
 $x(l) := xx.$

Note also that two important TS steps are performed with only these five statements: (i) move to a new current solution; (ii) update tabu list.

Algorithm

In the algorithm below we select some TS rules to include them, together with the chain-substitution move, in the proposed Chain-Substitution Tabu Search (CSTS). Although the questions as to what rules among many to use in the search still exist, following our approach we reduced the number of possible answers.

The number of demand (fixed) points n , the number of facility points m and the number of new facilities p are given. Let us assume that we are able to calculate the objective function value $f(X)$ if solution $X \subseteq L$ is known. These are all the input data needed. The output values of procedure *chain-substitution* are: optimal solution $X_{opt} = x_{opt}(j), j = 1, \dots, p$ and $f_{opt} = f(X_{opt})$.

Procedure Chain-Substitution TS ($m, n, p, f_{opt}, X_{opt}$)

Step 1. Parameters of the method:

- t_1, t_2 - lengths of the tabu lists T_1 and T_2 ;
- $nbmax$ - the maximum number of iterations between 2 improvements;
- $nlong$ - number of times a long term memory process will be used, i.e.,
how many times the procedure will be restarted in an unexplored area of solution space.

Step 2. Initialization

Let us denote with $x(j), j = 1, \dots, m$ a set of facility location points L in some order, where its first p elements represent the solution.

- Find an initial ordering of array $x(j), j = 1, \dots, m$ at random, i.e., let $x(j)$ be any permutation of numbers from 1 to m . (This permutation could be obtained for example by setting $x(j) = j$ first, and then by changing the position of first p elements with some with index randomly chosen from interval $[1, m]$).
- put $x_{opt}(j) := x(j), j = 1, \dots, m$, where $x_{opt}(j), j = 1, \dots, p$ represents the

incumbent solution (best solution found thus far); let $f_{opt} := f(x_{opt})$

- $nbiter := 0$, (* current iteration *);
- $bestiter := 0$ (* iteration when the best solution has been found *);
- Give initial values for counters in two tabu lists: $cnt1 := p$ and $cnt2 := m$;
- Let long term memory array $d(j), j = 1, \dots, m$, represents iteration number when variable j went out from the solution last time; initialize that array as $d(j) := 0$;

Step 3. Diversification loop

For $k = 1$ to $nlong$

Step 4. Descent-ascent search

While $(nbiter - bestiter < k \cdot nbmax)$ **do**

$nbiter := nbiter + 1$;

Step 4.1 Find the best solution in the neighbourhood

(* Generate a set of neighbourhood solutions $N(x)$; choose non tabu solution x^* optimizing f over $N(x)$ ($x^*(j)$ differs from $x(j)$ in only two indices out and in); check if there is a tabu solution better than f_{opt} (aspiration level); in that case, $x_{opt} := x^*$)

Around $(m, n, p, t_1, t_2, x, f^*, x_{opt}, f_{opt}, out, in)$;

Step 4.2 (* Keep the best solution *)

if $(f^*$ better than $f_{opt})$ **then**

$f_{opt} := f^*$; $x_{opt}(out) := x(in)$; $bestiter := nbiter$

Step 4.3 $d(x(out)) := nbiter$ (* Update long term memory array *);

Step 4.4 (* Move to a new solution and update tabu lists *)

Chain-substitution-move $(out, cnt1, in, cnt2, x)$;

$cnt1 := cnt1 - 1$; **if** $(cnt1 = p - t_1)$ **then** $cnt1 := p$;

$cnt2 := cnt2 - 1$; **if** $(cnt2 = m - t_2)$ **then** $cnt2 := m$;

EndWhile

Step 5. Diversification

(* Bring into the solution variable j^* that has been out for longest time, i.e., find minimum of $d(j), j = p + 1, \dots, m$, and exchange its position with one from T_1 *)

$xx := x(p)$; $x(p) := x(j^*)$; $x(j^*) := xx$;

EndFor

Obviously, the complexity of one iteration of descent-ascent search (*Step 4*) depends on the complexity of the procedure **Around** (*Step 4.1*) since **chain-substitution-move** is $O(1)$. We shall now give a pseudo-code for *best-improvement* (or *steepest descent mildest ascent* [13]) version of the algorithm, i.e., version that explores the complete neighbourhood consisting $p(m - p)$ solutions. Then modifications in order to get different versions are obvious: (i) *partial best-improvement* - take one facility

out from the solution at random and exchange it with all $m - p$ facilities that do not belong to the current solution; (ii) *first-improvement* - enumerate all neighbourhood solutions until first improvement; (iii) *random search* - take v (a parameter) neighbourhood solutions at random and move to the best among them; (iv) *random walk* - take one neighbourhood solution at random etc. Note that for a random walk algorithm tabu lists are not needed ($t_1 = t_2 = 0$), since the probability of cycling is almost zero. The similar conclusion holds for random search. Note also that a *1-substitution* move (with all versions mentioned above) can in fact be viewed as a special case of our *chain-interchange* move, where the parameters $t_1 = |T_1|$ and $t_2 = |T_2|$ are set to zero.

Procedure Around ($m, n, p, t_1, t_2, x, f^*, f_{opt}, x_{opt}, out, in$)

```

 $f^* := big$  (for minimization);  $f^* := -big$  (for maximization);
For  $i = 1$  to  $p$  (* Enumerate candidates that go out from the solution *)
     $jj := x(i)$ 
    For  $k = p + 1$  to  $m$  (* Enumerate candidates that go into the solution *)
         $x(i) := x(k)$  (* Interchange facilities *);
         $f := f(x)$  (* Find objective function in  $x$  *);
        if ( $i \leq p - t_1$  and  $k \leq m - t_2$ ) then
            (* Keep the best non tabu solution *)
            if ( $f$  better than  $f^*$ )  $f^* := f$ ;  $out := i$ ;  $in := k$ ;
        else (* Aspiration criterion *);
            if ( $f$  better than  $f_{opt}$ ) then
                 $f_{opt} := f$ ;  $out := i$ ;  $in := k$ ;  $x_{opt}(out) := x(in)$ ;
            Endif;
        Endif;
    Endif;
     $x(i) := jj$ ;
EndFor;

```

The procedure *Around* employs a simple type of aspiration criterion, consisting of removing a tabu status from a trial move when the move yields a solution better than the best obtained so far (f_{opt}). Then it becomes both, a new current and a new incumbent solution.

In the framework of TS, diversification of the search is performed by using the so called long term memory function. It has been noted that a search with the

memory has a greater chance of visiting unexplored regions of the solution space than a random multistart search. Moreover, multistart implementation suffers from 'central-limit catastrophe' when the problem size grows large. In [2] it is shown that usually in combinatorial problems, very good solutions are located near other good solutions. That result may explain why simulated annealing, tabu search and other descent-ascent local search heuristics have been so successful in practice. But these also explain why TS has been successful even without using the long term memory function (see [11] & 3.4 for such applications).

However, there are easy ways of implementation of TS long term memories in our algorithm. One possibility of doing this has already been explained in steps 4.3 and 5 of the algorithm: d_j represents the last iteration number when facility point j (attribute of the solution) has been in the solution ([32]). Note that only one operation in each iteration is needed to update sequence d_j (Step 4.3).

Another possibility is in introducing *frequency-based memory* ([11]) in our method. Let r_j represent counts of the number of occurrences of a facility point j in the solution in previous iterations. When improvement with local search is no longer possible, intensification and diversification could be performed using array r_j (in Step 5 of the algorithm) in some of the following ways: (i) *Intensification by Selection*: take the p largest elements from r_j and put the corresponding facilities into the new solution. Among them, put facilities that correspond to the t_1 largest members of r_j into T_1 ; (ii) *Diversification by Negative Selection*: bring into the solution p facilities with the corresponding p smallest values of array r_j (or d_j), etc.

4. Computational experience

Three sets of experiments were conducted for p – *median* problem only. The first one examined road distances between 47 important European cities ([25]); the second one used the Ruspini data [30] for 75 fixed points, which is widely referenced in classification or clustering theory; the third one used n randomly generated points from square in the plain $[0,100] \times [0,100]$ ($n = 100, 200, 300$).

Without loss of generality, in all testing it is assumed that a set of potential facilities is equal to a set of customers ($L = D$, $m = n$). We compare results of our

p	z_{opt}	Objective function value			% deviation			CPU Time (sec.)		
		RW	1-INT	CSTS	RW	1-INT	CSTS	RW	1-INT	CSTS
5	28711	28711	29938	28711	0.00	4.27	0.00	9.00	0.36	2.47
10	17677	17845	18474	17677	0.95	4.51	0.00	5.54	0.42	1.55
15	12749	12918	12749	12749	1.33	0.00	0.00	4.47	0.73	1.29
20	9328	9762	9657	9328	4.65	3.53	0.00	3.96	0.75	1.15
25	6561	6908	6920	6561	5.29	5.47	0.00	3.68	0.53	1.06
30	4455	4705	4530	4455	5.61	1.68	0.00	3.50	0.59	0.98

Cuadro 1: Results for the 47-European cities: $nfun = 10000$, $nlong = 1$, $t_1 = 1$, $t_2 = 3$

Chain-Substitution TS method (CSTS) with two well-known methods: Random Walk (RW) algorithm (with local improvement of the solution) and descent 1-Interchange (1-INT) algorithm. In the Random Walk algorithm, both the facility that goes out of the solution and the facility that goes into the solution are chosen at random. The new solution determines a partition of a set of customers into p groups. Local improvement consists in solving separately p 1-facility problems, i.e., the best center for each given set of customers is found. We obtain the 1-Interchange method as a special case of Chain-Interchange method, by setting parameters t_1 and t_2 to zero (tabu lists are empty). RW and CSTS terminate when a given maximum number of function evaluations $nfun$ is reached, while 1-INT stops naturally (when there is no better solution in the neighbourhood of a current solution). The basic version of Chain-substitution method is used in comparison: complete neighbourhood of $(p - t_1)(m - p - t_2)$ solutions is considered, thus aspiration criterium as well as long term memory processes have not been used. The results for the 47-European cities are given in **Table 1**. p represents the number of new facilities and z_{opt} the exact minimum value. The other columns contain the objective function values, % deviation calculated relative to z_{opt} and CPU times for all methods compared. The results for Ruspini data are summarized in **Table 2** using the same format as Table 1. It appears that Chain-Substitution (CSTS) algorithm out-performs both 1-INT and RW. The results for random test problems are given in **Table 3**. We compare CSTS and 1-INT only. Average results for ten runs for each n and p are reported ($n = 100, 200, 300$; $p = 10, 20, \dots, \lfloor \frac{n}{2} \rfloor$). We denoted by v the number of function evaluations 1-INT method had used before it stopped. Then the stopping criterium we used for CSTS method was set to $nfun = 3v$. Hence, CSTS had three times more

p	z_{opt}	Objective function value			% deviation			CPU Time (sec.)		
		RW	1-INT	CSTS	RW	1-INT	CSTS	RW	1-INT	CSTS
5	779.68	779.68	796.44	779.68	0.00	2.15	0.00	49.57	1.68	14.09
10	512.81	515.14	512.81	512.81	0.45	0.00	0.00	29.51	3.52	8.67
15	389.98	390.28	393.83	393.83	0.08	0.99	0.99	22.75	5.45	7.12
20	314.10	318.66	315.05	315.05	1.45	0.30	0.30	19.20	6.90	6.83
25	252.43	263.50	257.45	255.80	4.39	1.99	1.33	17.12	6.11	5.97
30	199.41	214.28	208.56	207.37	7.45	4.59	3.99	15.79	5.68	5.40
35	159.90	169.88	166.48	163.10	6.24	4.11	2.00	14.93	5.07	4.98
40	127.63	139.56	132.46	128.62	9.35	3.78	0.77	14.30	4.74	4.62

Cuadro 2: Results for Ruspini data: $n = 75$, $nfun = 25000$, $nlong = 1$, $t_1 = 1$, $t_2 = 3$

steps than 1-INT.

5. Conclusions

The 1-Interchange move was successfully applied to a large class of combinatorial problems, but it did not allow escape from local optima traps. We suggested a simple extension of this move, the *chain-interchange* move, which would allow us to obtain an efficient descent-ascent local search heuristic of Tabu search type.

The list of implementation of 1-interchange, and thus chain-interchange moves is very large. Even the simplex method for linear programming and some concave minimization methods are of that type: one variable goes out from the basis, another goes in. On the other hand, for the same problem, different interchange strategies can be developed in the solution process. For example, in directly-allocated LA problems the same solution can be represented as a subset of vertices (locations) as well as a subset of edges (allocations) of some graph. Obviously, the set of neighbourhood solutions obtained by all possible interchanges in both cases are not the same. This fact could limit a general approach in which problem specific knowledge is ignored. In this paper, a **middle approach** is suggested: recognition of a class of problems with similar properties, such that the same data structure (the same algorithm) may be applied. We addressed our attention to location-allocation problems which have

n	p	Objective	function	% Improvem-	CPU	Time(s.)
		1-INT	CSTS			
100	10	104.042	103.920	0.12	7.01	18.62
100	20	63.238	63.026	0.34	10.81	29.13
100	30	43.908	43.858	0.11	13.59	36.57
100	40	31.994	31.908	0.27	13.99	38.37
100	50	22.963	22.896	0.29	14.31	39.49
200	10	225.037	223.862	0.52	27.80	75.32
200	20	145.766	144.530	0.85	45.78	125.41
200	30	108.394	108.107	0.26	69.39	192.40
200	40	86.655	86.087	0.66	73.38	203.80
200	50	73.222	71.276	2.66	66.83	184.58
200	60	62.885	60.267	4.16	61.24	172.11
200	70	54.381	51.332	5.61	57.03	161.57
200	80	47.016	43.866	6.70	54.73	155.65
200	90	40.341	37.367	7.37	52.27	149.14
200	100	34.525	31.557	8.59	50.35	144.94
300	10	345.635	344.549	0.31	68.02	185.86
300	20	230.303	228.614	0.73	114.06	314.81
300	30	176.323	175.513	0.46	163.02	451.39
300	40	145.228	143.851	1.67	142.96	387.71
300	50	125.903	120.618	4.20	129.59	348.56
300	60	112.727	104.581	7.23	115.93	315.63
300	70	101.474	91.300	10.03	107.96	295.23
300	80	91.892	81.141	11.70	103.40	282.38
300	90	83.869	72.862	13.12	97.10	266.64
300	100	76.740	65.612	14.50	93.07	257.55
300	110	70.120	59.335	15.38	89.70	250.33
300	120	64.201	53.538	16.61	86.13	242.12
300	130	58.418	48.308	17.31	82.77	235.23
300	140	52.661	43.309	17.76	80.20	233.92
300	150	47.554	38.679	18.66	78.16	233.45

Cuadro 3: Average results for ten runs: $nfun = 3v$, $nlong = 1$, $t_1 = 1$, $t_2 = 3$

similar problem specific knowledge. For solving them we developed one version of the chain-interchange method, *chain-substitution* heuristic.

A future work on this “middle” approach with chain-interchange method could be developed in four directions: (i) *enlarge* the class of problems that could be solved efficiently with algorithm *chain-substitution* proposed in § 3 of this paper (by only changing the subroutine that evaluates objective functions), and compare results with other good heuristics (for each particular problem). That could be for example p -cluster problems, p -dispersion problems, simple plant location problem, quadratic knapsack problems, etc.; (ii) for solving large instances of LA problems *extend* basic version of the *chain-substitution* algorithm using some hybrids or other known ideas from TS; (iii) develop data structures that allow implementation of another version of the chain-interchange method such as *chain-reallocation*, and compare results with the *chain-substitution* algorithm; (iv) in this paper, the 1-chain-interchange method is in fact proposed. Obviously it could be generalized to an *r-chain-interchange* algorithm. Then it becomes a hybrid of a Variable Neighbourhood Algorithm (VNA) [23] and TS.

Acknowledgement. We thank Dr. A. Klose from St Gallen University, Switzerland, for providing us with the exact solutions found in Tables 1 and 2. We also thank B. Milosević for helpful comments in making the final version of a paper.

Referencias

- [1] **Brandeau, M.L., Chiu, S.S.** An overview of representative problems in Location Research. *Management Science* 35 (6) (1989), 645-674.
- [2] **Boese, D.K., Kahng B.A., and Muddu S.** A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters* 16 (1994), 101-113.
- [3] **Cooper, L.** Heuristic methods for location-allocation problems. *SIAM Review* 6 (1) (1964), 37-53.
- [4] **Crainic, G., M. Gendreau, P. Soriano and M. Toulouse.** A tabu search procedure for multicommodity location/allocation with balancing requirements. *Annals of Operations Research* 41 (1993), 359-383.

- [5] **Ehrgott M.** *K*-cardinality subgraphs. *OR-Proceedings 94*, pp. 86-91. Springer: Berlin, Heidelberg, New-York, 1995.
- [6] **Erkut, E. and Neuman S.** Comparison of four models for dispersing facilities. *INFOR* vol. 29, No. 2, (1990), 68-85.
- [7] **Fischetti, M., Hamacher H.W., Jornsten K. and Maffioli F.** Weighted *k*-cardinality trees: complexity and polyhedral structure. *Networks* 24 (1994), 11-21.
- [8] **Glover F.** Future paths for integer programming and links to artificial intelligence. *Comput. & Oper. Res.* 5(1986) 533-549.
- [9] **Glover, F.** Tabu search. Part I. *ORSA J. Computing* 1 (1989) 190-206.
- [10] **Glover, F.** Tabu search. Part II. *ORSA J. Computing* 2 (1990) 4-32.
- [11] **Glover, F., and M. Laguna.** Tabu Search, in C. Reeves ed., *Modern heuristic techniques for combinatorial problems* (Chapter 3), Oxford, Blackwell, 1994.
- [12] **Handler, G.Y., Mirchandani, P.B.** *Location on Networks*. MIT Press. 1979.
- [13] **Hansen, P.** The steepest ascent mildest descent heuristic for combinatorial programming. *Congress on numerical methods in combinatorial optimization*, Capri, Italy (1986).
- [14] **Hansen, P., and B. Jaumard.** Algorithms for the maximum satisfiability problem. *Computing* 44 (1990), 279-303.
- [15] **Hansen, P. and Mladenovic, N.** Educational Software: Tabu Search in Optimal Pipeline Design Problem. *Optimization Days 1993*, May 12-14, Montreal, Canada.
- [16] **Hansen, P., Peeters, D. and Thisse F.** Public facility location models: A selective survey. In *Location analysis of public facilities*, eds. J.F. Thisse and H.G. Zoller. Amsterdam: Nort Holland.
- [17] **Jacobsen, S.K.** Heuristic for the capacitated plant location model. *E.J.O.R.* 12 (1983), 253-261.
- [18] **Jornsten, K. and Lokketangen A.** Tabu Search for weighted *K*-cardinality trees. *APJOR - Avio-Pacific J. of Oper. Res.* (1995) (to appear).
- [19] **Kuehn, A.A., and Hamburger M.J.** A heuristic program for locating warehouses. *Management Science* 9 (4) (1963), 643-666.

- [20] **Lih, K-W., M. Breton., Hansen, P., Mladenovic, N.** A Mathematical Programming Approach to Oil Pipeline Design, *TIMS/ORSA National Meeting*, April 26-29, 1992, Orlando, Florida, U.S.A.
- [21] **Lin, S. and Kernighan B.W.** An effective heuristic algorithm for the travelling-salesman problem. *Oper. Res.* 21 (1973), 498-516.
- [22] **Maranzana, F.E.** On the location of supply points to minimize transportation costs. *Operations Research Quarterly* 12 (1964), 138-139.
- [23] **Mladenović, N.** A variable neighbourhood algorithm - a new meta-heuristic for combinatorial optimization. Optimization days, Montreal, May 10-12 (1995) 22.
- [24] **Mladenović, N.** A chain-interchange heuristic in solving K-cardinality subgraph problem. *3rd Balkan conference on Oper. Res.*, 16-19 October 1995, Thessaloniki, Greece.
- [25] **Moreno, J.A., Garcia, R.L. and Moreno-Vega, J.** A parallel genetic algorithm for the discrete p-median problem. *Studies in Locational Analysis*, Issue 7, 1994.
- [26] **Moreno, J.A., Rodríguez, C. and Jiménez N.** Heuristic Cluster Algorithm for the Multiple Facility Location-Allocation Problems. *Revue de Automatique, Informatique y de Recherche Operationelle.* 25 (1) (1991). 97-107.
- [27] **Osman, I. and Christofides N.** Capacitated clustering problems by hybrid simulated annealing and tabu search. *Int. Trans. Oper. Res.* Vol. 1. No. 3 (1994), 317-336.
- [28] **Pruhs, K.** Using local adaptations to reconfigure a spanning tree of a network. *Discrete Applied Mathematics* 57 (1995) 67-74.
- [29] **Rothfarb, B., and Goldstein, M.** The One-terminal Telpak Problem. *Operations Research*, 19, (1971), 156-169.
- [30] **Ruspini, E. H.** Numerical methods for fuzzy clustering. *Information Sciences* 2 (1970). 319-350.
- [31] **Solomon, M.** The Minimum Spanning Tree Problem with Time Windows Constraints, *American Journal of Mathematical and Management Sciences*, 6 (3&4), (1986), 399-421.
- [32] **Sum, M. and McKeown P.G.** Tabu Search applied to the general fixed charge problem. *Annals of Operations Research* 41 (1993), 405-420.