

Variable Neighborhood Tabu Search and its Application to the Median Cycle Problem ^{*}

José A. Moreno Pérez, J. Marcos Moreno-Vega,
Inmaculada Rodríguez Martín
Dpto. de Estadística, I.O. y Computación
Universidad de La Laguna

Resumen

The Variable Neighborhood Tabu Search consists of a constructive phase and a series of local searches that use tabu tools. When the local search stops at a nontabu local minimum, a shake procedure starts a new local search. The local searches and the shake procedure are based on a series of standard moves. The local search consists of applying the best possible move until no such move exists. The shake procedure consists of applying a number of random moves. Since the shake could provide an infeasible solution, the local search considers as possible moves those providing a feasible solution or reducing the infeasibility. A location/allocation problem consists of selecting the location for some facilities and the allocation of the users to them. Two functions to be minimized are considered: the length of the solution, as a measure of the set of locations, and the total cost of the allocations. The standard moves for these problems are the add, drop and add/drop moves. The heuristic procedure is tested on the Median Cycle Problem.

1. Introduction

A great variety of logistic problems can be considered as location-allocation problems where a set of points is selected to hold facilities that must serve

^{*}This research has been partially supported by Gobierno de Canarias through project PI1999/116.

to all the users. In this situation there are two typical functions to be minimized: a measure of the connectivity structure among the opened facilities, and the cost of allocating the users to the facilities. These two functions play different roles in the optimization models. In some problems the connectivity structure usually implies a unique investment, while the assignment or allocation of users to facilities may imply daily costs. In other problems, the connectivity cost is assumed by the server and the allocation costs by the users.

In the classical p -median and p -center problems the measure of the connection structure is only the number of facilities, that is fixed to p , and the allocation cost to be minimized is respectively the sum and the maximum of the distances between the facilities and the users assigned to them. The Uncapacitated Facility Location Problem consists of minimizing the sum of costs of the opened facilities and the allocation cost given by the sum of the distances between the users and the nearest facility. In other classical optimization problems, the connection structure is a cycle or a tree and the measure is its length. Let us mention, for example, the Shortest Covering Path Problem (Current, Cohon and ReVelle [1]), the Maximal Direct Covering Tree Problem (Hutson and ReVelle [8]), and the Covering Tour Problem (Gendreau, Laporte and Semet [2]).

The allocation cost can also include a measure of a structure connecting the users with the facilities. These connection structure usually have a tree, cycle or a star shape for each facility. With the star structure each user is connected by a path to a facility, and the allocation cost is the sum of the distances between the users and their nearest facilities.

One of these problems is the Median Cycle Problem, in which the connection structure is a cycle visiting a fixed depot and the allocation structure is of star type. One version of this problem consists of finding the cycle that visits the depot and minimizes the sum of connection and allocation costs. Another version consists of minimizing the length of a tour passing through a depot while imposing an upper bound on the sum of the distances from all the users to the cycle.

In this paper we introduce a new metaheuristic, the Variable Neighborhood Tabu Search (VNTS), consisting of a combination of the Variable Neighborhood Search (VNS) and Tabu Search (TS), and consider its application for solving generic location-allocation problems. The details of implementation for the Median Cycle Problem are provided and the corresponding computational experience is reported. Some solutions better than the best

known till now are obtained.

2. The VNTS Heuristic

Local Search methods for combinatorial and global optimization proceed by performing a sequence of local changes in an initial solution which improve each time the value of the objective function until a local optimum is found. That is, at each iteration an improved solution x' in the neighbourhood $\mathcal{N}(x)$ of the current solution x is obtained, until no further improvement is found. In recent years, several metaheuristics have been proposed which extend in various ways this scheme and avoid being trapped in a local optimum. The most famous of these metaheuristics are Simulated Annealing (Kirkpatrick, Gelatt and Vecchi [9]), Genetic Search (Holland [7]), and Tabu Search (Glover [3] and [4], Glover and Laguna [5]).

To avoid termination at a local minimum, metaheuristics allow nonimproving moves. However, this presents the risk of cycling. In Tabu Search the use of a short term memory helps to forbid moves that might lead to recently visited solutions.

Variable Neighbourhood Search (Mladenović and Hansen [12]) is a relatively new technique whose originality consists of trying to escape from local optima by changing the neighbourhood structure. In its basic form, VNS explores a set of neighbourhoods of the current solution, makes a local search from a neighbour solution to a local optimum, and moves to it only if there has been an improvement.

The metaheuristic method we propose, the Variable Neighbourhood Tabu Search (VNTS), was originally inspired in VNS. However, our conception of the neighbourhoods is somehow different. In VNTS the different neighbourhoods used are nested and are defined from an original one \mathcal{N} in the following way:

$$\mathcal{N}_1(x) := \mathcal{N}(x), \mathcal{N}_2(x) := \bigcup_{y \in \mathcal{N}(x)} \mathcal{N}_1(y), \dots, \mathcal{N}_k(x) := \bigcup_{y \in \mathcal{N}(x)} \mathcal{N}_{k-1}(y).$$

That is, $\mathcal{N}_i(x)$ are the neighbours of x at distance from 1 to i (see Figure ??).

The VNTS comprises the following steps:

VNTS algorithm

1. Initialization:
 - a) Select an initial neighbourhood \mathcal{N} , from which \mathcal{N}_k , $k = 1, \dots, k_{\text{máx}}$ are defined as described before.
 - b) Find an initial solution x .
 - c) Set $best_x := x$.
 - d) Choose a stopping condition.

2. Repeat the following until the stopping condition is met:
 - a) Local Search: Apply a greedy tabu local search in $\mathcal{N}(x)$ until a local minimum x' is found.
 - b) If x' is better than $best_x$, do $best_x := x'$.
 - c) Shaking: Generate at random a solution x'' in $\mathcal{N}_s(x')$.
 - d) Set $x := x''$.

Tabu search tools are incorporated to both local search and shaking procedures to modulate the intensification and diversification of the search. The local search tests all the nontabu moves, and that providing the best feasible solution is made. The shake consists of applying a number s of random moves (feasible or not). This number of random moves is the size of the shake. The shake could provide an infeasible solution without feasible neighbour. In those cases the local search is allowed to make the best nontabu move among those that reduce the infeasibility. Therefore, the local search consists of choosing the best feasible nontabu move, if there is one, or the best nontabu move that reduces the infeasibility. As stopping criteria usual conditions can be applied: the total CPU time, the number of local searches, the number of moves, the relative number of moves without improving the best solution found.

The tabu tool proposed is a list with the last elements that have been dropped from the solution. The elements that are in the tabu list are not allowed to go immediately inside the solution again. This tabu tool is more powerful than the usual list of tabu moves, since with a short list of tabu elements many moves are forbidden.

To show the wide applicability of the VNTS metaheuristic consider the problems where each solution is given by a selection of a set of elements from

an universe. Many of relevant combinatorial optimization problems can be formulated in this way. The standard moves for those problems are the add, drop and the interchange moves. These moves consist of adding a new member to the solution, dropping a member from the solution, or interchanging a member of the solution for another one out of the solution. The selection can be subject to some constraints. In those cases, the moves that provide a feasible solution are called feasible moves. In next section we study with more detail the application of VNTS to a general Location-Allocation problem.

3. General application of VNTS to Location-Allocation problems

We consider Location-Allocation problems where the solution consists of selecting a set of points S to hold the facilities. The solution is evaluated by two functions: the length of the solution, given by $Len(S)$, and the cost of the solution, given by $Cos(S)$. The length of the solution S can represent the total length of a cycle, of a tree, the number of points selected, or any other measure of the structure connecting the points in the solution. The cost of the solution S is a measure of the structure connecting the users to the set of selected points. An usual cost function is the sum or the maximum of the distances from the users to the points in the solution.

The most common approaches used to solve this kind of bicriteria problems are to fix an upper bound on one of the two functions and to minimize the other one, or to minimize a convex combination of them.

3.1. Coding the solution

The selection of a solution coding that provides an efficient way of implementing the moves and evaluating the solutions is essential for the success of any search method.

We propose the following coding of the solutions. Let n be the size of the universe. A solution S is represented by an array $[v_i : i = 1, \dots, n]$ where v_i is the i -th element of the solution, for $i = 1, 2, \dots, m$, and the $(i - m)$ -th element outside the solution, for $i = m + 1, \dots, n$.

3.2. Evaluating the solutions

For some problems, the evaluation of a solution could mean to solve another optimization problem. For instance, when looking for routes the solutions are ordered sets of visited points. Therefore, the evaluation of the length of the shortest route requires solving a TSP. The efficiency when evaluating the modification produced in the objective function by a move is very important, because a large number of moves are tested. Therefore, having an efficient procedure to (may be heuristically) evaluate the modifications of the solution is very useful. The information given by the actual solution can be used to design an efficient way of evaluating a move, by considering only the local modifications that it would produce. So, when a solution is modified by adding and/or dropping a point, the length of the new solution is obtained by extracting the leaving point and/or looking for the best position to insert the new one. In addition, to calculate the cost of the new solution we need to know the new optimal allocation of the users. Usually we only need to know if the entering point can improve the allocation of the users and if the users allocated to the leaving point need to be reallocated.

3.3. The moves

The usual moves considered in this kind of problems are the add, drop and interchange moves. Given a solution S and an element v_j not in the solution, an add move consists of adding v_j to S . A drop move consists of eliminating an element v_i from S . Given a solution S , an element v_i in the solution and an element v_j not in the solution, an interchange or add/drop move consists of dropping v_i from S and adding v_j to S .

If any constraint is imposed, not every move is feasible. The moves that provide a feasible solution are called feasible moves. If the constraint consists in an upper bound on a function of the solution then the moves that reduce the infeasibility can be used to get feasible solutions.

3.4. Initial solution

Usually a constructive heuristic is used to generate an initial solution. Standard constructive heuristics consists of a series of add moves starting from an empty solution. During the building process the length of the candidate solutions increases while their allocation cost decreases. The procedure

stops when the solution obtained is feasible. The most common strategies for adding elements to the solutions are to use simple, random, and good add moves. The good properties that a constructive method should have in order to be used in MultiStart or GRASP metaheuristic are:

Efficacy: The degree of optimality of the solution proposed must be high.

Efficiency: The amount of resources (time) used must be small.

Randomness: The ability to cover the whole search space.

The following procedures are standard constructive methods:

1. **Random Insert method.** Select a new member at random.
2. **Best Insert method.** Select a new member in order to minimize the increment in the length of the solution.
3. **Cheapest Insert method.** Select a new member in order to maximize the decrement in the allocation cost of the new solution.
4. **Lambda Insert method.** Take a new member minimizing a linear combination of the increment in the length and (minus) the decrement in the assignment cost.

4. The Median Cycle Problem (MCP)

The proposed heuristic has been tested on the Median Cycle Problem (MCP). The MCP consists of finding an optimal cycle visiting a given vertex, the depot, taking into account the connection and allocation costs. The allocation cost of each vertex is the distance to the cycle. A first version of the problem, MCP1, calls for finding the cycle that minimizes the sum of both types of costs. A second version, MCP2, consists of minimizing the length of the cycle passing through the depot while the total allocation cost is bounded by a given threshold.

4.1. Formulation of the problem

Let $G = (V, E)$ be a complete undirected graph where $V = \{v_1, \dots, v_n\}$ is the vertex set and v_1 denotes the depot. Let $L[v_i, v_j]$ denote the nonnegative length associated to each edge $e = [v_i, v_j] \in E$, and $D[v_i, v_j]$ be the nonnegative allocation cost for each pair $(v_i, v_j) \in V \times V$.

A solution is a tour or cycle C including the depot. Every non visited vertex is allocated to its nearest vertex in the cycle, being the allocation cost equal to the distance between them. The length of the cycle is given by:

$$Len(C) = \sum_{(v_i, v_j) \in C} L[v_i, v_j].$$

The total allocation cost is given by:

$$Cos(C) = \sum_{v_i \in V} \min_{v_j \in V(C)} D[v_i, v_j],$$

where $V(C)$ is the set of vertices of the cycle C .

The MCP1 consists of finding the cycle C visiting the depot that minimizes $Len(C) + Cos(C)$, while the MCP2 consists of finding the cycle C minimizing $Len(C)$ subject to $Cos(C) \leq d_0$.

5. The VNTS for MCP

In this section we describe with more details the implementation of a VNTS heuristic for MCP.

5.1. Coding of the routes

The solution given by a cycle C visiting m vertices is represented by an array $S = [v_i : i = 1, \dots, n]$ where v_1 is the depot, and v_i is:

- the i -th vertex of the cycle, for $i = 2, \dots, m$, and
- the $(i - m)$ -th vertex not belonging to the cycle, for $i = m + 1, \dots, n$.

Note that the solution gives not only the points visited but also their order in the cycle, starting from the depot.

The assignment or allocation costs are stored in a matrix D , and the lengths or routing costs are stored in a matrix L . The length of the solution cycle represented by S is given by:

$$Len(S) := \sum_{i=2}^m L[v_{i-1}, v_i] + L[v_1, v_m].$$

And its allocation cost is given by:

$$Cos(S) := \sum_{i=m+1}^n \min_{j=1..m} D[v_i, v_j].$$

An optimal solution S^* of MCP1 is a solution that minimizes $Len(S) + Cos(S)$. The solution S is feasible for MCP2 if $Cos(S) \leq d_0$. An optimal solution S^* for MCP2 is a feasible solution that minimizes $Len(S)$, that is, the solution S^* such that $Cos(S^*) \leq d_0$ and

$$Len(S^*) := \min\{Len(S) : Cos(S) \leq d_0\}.$$

5.2. The moves

In this section we show how to update the length and cost of a solution after a move has been done.

5.2.1. Inserting a vertex in the cycle

The insertion of a new vertex v_j in the cycle implies to know in which position it is to be inserted. The assignment cost of the new solution does not depend on the insertion position. So we will always insert the new vertices in the best possible position, i.e., the position that provides a shortest cycle.

The length of the new solution S_{kj} , obtained by the insertion of a new vertex v_j in the cycle just after the vertex v_k , is given by:

- for $k = 1, \dots, m - 1$:

$$Len(S_{kj}) := Len(S) + L[v_k, v_j] + L[v_j, v_{k+1}] - L[v_k, v_{k+1}],$$

- for $k = m$:

$$Len(S_{kj}) := Len(S) + L[v_m, v_j] + L[v_j, v_1] - L[v_m, v_1],$$

Every vertex is inserted at the position k where the minimum

$$\min_{k=1..m} Len(S_{kj})$$

is reached. Therefore, to evaluate the length of the new cycle takes $O(m)$ time.

After the insertion, the new total cost is given by:

$$Cos(S_{kj}) := Cos(S) - \sum_{i=m+1}^n \max \left\{ 0, \min_{h=1..m} D[v_i, v_h] - D[v_i, v_j] \right\}.$$

Note that the decrement in the cost depends only on the vertex inserted, and not on the position k and it takes $O(mn)$ time.

Improving the updating of the costs The computation of the cost of the new solution after each adding move can be simplified by storing the individual allocation costs in a vector named $Cos1[.]$, defined as:

$$Cos1[i] := \min_{j=1..m} D[v_i, v_j], \quad i = 1, \dots, n.$$

Therefore, the total cost is:

$$Cos(S) := \sum_{i=m+1}^n Cos1[i].$$

The new individual allocation costs obtained when the vertex v_j is inserted can be calculated in the following way:

$$Cos1[i] := \min \{ Cos1[i], D[v_i, v_j] \}.$$

Then, the updating of the new allocation costs takes $O(n)$ time.

5.2.2. Deleting a vertex from the cycle

Let S_k denote the solution obtained if the k -th vertex of the cycle, v_k , is removed from S , for $k = 2, \dots, m$. The new length, depending on k , is given by:

- for $k = 2, \dots, m - 1$:

$$Len(S_k) := Len(S) + L[v_{k-1}, v_{k+1}] - L[v_{k-1}, v_k] - L[v_k, v_{k+1}]$$

- for $k = m$:

$$Len(S_k) := Len(S) + L[v_{m-1}, v_1] - L[v_{m-1}, v_m] - L[v_m, v_1],$$

The individual allocation costs have to be updated only for the vertices v_i such that $Cos1[i] = D[v_i, v_k]$. For those vertices,

$$Cos1[i] := \min_{\substack{j=1, \dots, m \\ j \neq k}} D[v_i, v_j].$$

It takes $O(mn)$ operations.

Second Improvement in the updating of the costs The computation of the cost of the new solution obtained when a vertex is removed from the cycle can be improved by using the second best allocation. Consider a solution S in which the individual allocation cost for every vertex v_i , $i = 1, \dots, n$, is

$$Cos1[i] := \min_{j=1, \dots, m} D[v_i, v_j] = D[v_i, v_{r(i)}].$$

Then let the second best allocation cost of v_i , $i = 1, \dots, n$, be

$$Cos2[i] := \min_{\substack{j=1, \dots, m \\ j \neq r(i)}} D[v_i, v_j].$$

Let V_k be the set of vertices allocated to vertex v_k , that is, $V_k := \{v_i \in V : Cos1[i] = D[v_i, v_k]\}$. Then, if vertex v_k is removed from the cycle, the new total allocation cost is:

$$Cos(S_k) := \sum_{v_i \in V_k} Cos2[i] + \sum_{v_i \in V \setminus V_k} Cos1[i].$$

So, the new allocation cost can be computed in $O(n)$ time.

When removing v_k , the first and second individual allocation costs have to be recomputed only for the vertices v_i such that $D[v_i, v_k] \leq Cos2[i]$. For those vertices v_i , the first allocation costs are updated using the following rule: if $Cos1[i] = D[v_i, v_k]$, then $Cos1[i] := Cos2[i]$. And the second allocation costs are updated by

$$Cos2[i] := \min_{\substack{j=1, \dots, m \\ j \neq r(i)}} D[v_i, v_j],$$

where $r(i)$ is such that $Cos1[i] = D[v_i, v_{r(i)}]$.

The second best allocation cost is a tool similar to the one used in the VNDS by Hansen, Mladenović, and Pérez-Brito [6].

5.2.3. Interchanging two vertices

The interchange of two vertices consists of a combination of an add and a drop move. A vertex is removed from the cycle, and a vertex is inserted in the best possible position in the cycle. The length and cost of the new solution are computed as follows. Let S_{ij} , $1 < i \leq m$ and $m < j \leq n$, be the new solution consisting in interchanging v_i and v_j , by dropping v_i and adding v_j in the best possible position, say after v_k . Then the new length is obtained by $Len((S_i)_{kj})$ using the former formula for adding and dropping a vertex. So it takes $O(m)$ time to evaluate the new cycle length.

The new allocation cost is given by:

$$\begin{aligned} Cos(S_{ij}) & : = \min\{D[v_i, v_j], \min_{\substack{l=1, \dots, m \\ l \neq i}} D[v_i, v_l]\} \\ & + \sum_{k=m+1}^n \min\{D[v_k, v_j], \min_{\substack{l=1, \dots, m \\ l \neq i}} D[v_k, v_l]\}, \end{aligned}$$

which would imply $O(mn)$ operations. However, using the values in $Cos1$ and $Cos2$, an improved procedure can be applied. We first test if the going in vertex is the best or the second best allocation for the users. This is done for every $k > m$, in the following way. If $D[v_k, v_j] \leq Cos1[k]$, then $Cos2[k] := Cos1[k]$ and $Cost1[k] := D[v_k, v_j]$. Otherwise, if $D[v_k, v_j] \leq Cost2[k]$ then $Cos2[k] := D[v_k, v_j]$.

Then we need to update the individual costs of the users assigned to the dropped vertex, i.e., for the users v_k , $k > m$, such that $Cos1[k] = D[v_k, v_i]$. For those v_k we do $Cos1[k] := Cos2[k]$ and

$$Cos2[k] := \min_{\substack{l=1, \dots, m \\ l \neq r(k)}} D[v_k, v_l],$$

where $r(k)$ is such that $Cos1[k] = D[v_k, v_{r(k)}]$.

The updating of the individual allocation costs, and therefore, of the total allocation cost, takes $O(mn_i + n)$ time, n_i being the number of vertices assigned to vertex v_i .

5.3. The initial solution

To obtain an initial cycle we use a constructive method. We start only with the depot. At each iteration a new vertex is chosen at random at it is

inserted in the best position, i.e., the position that gives a smallest increment in the value of the objective function. At each iteration the computation of the length takes $O(m)$ time and the computation of the cost takes $O(mn)$. The procedure stops when there are at least three vertices in the cycle and:

- For MCP1: the objective function cannot be decreased anymore.
- For MCP2: the cycle becomes feasible.

5.4. Tabu list

We use a tabu list to improve the efficiency of the search by avoiding moves leading to recently visited solutions. In order to do so we put in a FIFO tabu list the vertices dropped from the cycle. The tabu list is kept at the end of the array S that codes the solution. In fact, if the size of the tabu list is t , a solution consisting of a cycle visiting m vertices is represented by an array $S = [v_i : i = 1, \dots, n]$ where v_1 is the depot, and v_i is:

- the i -th vertex of the cycle, for $i = 2, \dots, m$,
- a vertex outside the cycle, for $i = m + 1, \dots, n$, and
- a vertex in the tabu list, for $i = n - t + 1, \dots, n$.

When a vertex is removed from the cycle it is first placed in a position k , being k the position $m + 1$ if the performed move was a drop move, and the position previously occupied by the added vertex if an add-drop was performed. In a second step, the vertex in position k and the first vertex in the tabu list are interchanged (see Mladenović, Moreno-Pérez and Moreno-Vega [13]).

Only vertices v_j , with j from $m + t + 1$ to n , are tested for being included in the cycle. So, a leaving vertex will be out of the cycle in at least t iterations (the size of the tabu list). Note that this tabu tool is more powerful than the usual list of tabu moves, since with a short list of tabu elements many moves are forbidden.

5.5. Shake procedure

The shake procedure (used by Mladenović and Hansen [12] in their VNS heuristic) allows to escape from local minimum without destroying completely the good properties of the current solution. Given a size s for the shake

procedure, we choose s times two vertices at random, v_i and v_j . If the vertex v_i is in the cycle and v_j is outside the cycle we do the corresponding add/drop move, if both vertices are in the cycle we drop v_i , and if both vertices are outside the cycle we add v_j . The going-in vertex v_j is always chosen taking into account the tabu list, and it is inserted in the best possible position. The going-out vertex v_i , if dropped, goes to the tabu list, that is updated.

5.6. Local search

Given a solution, the greedy local search is implemented by choosing each time the best possible move among all the add, drop or add/drop moves. For MCP1 the best move is that minimizing the sum of length and allocation costs. For MCP2, the best possible move is that with minimum length among those that provide a feasible cost or a less infeasible cost (the latest only if there are not feasible moves). This is done to allow the process to move through feasibility if this property has been lost when making the shake.

Let S_{ij} denote the solution obtained from S by interchanging v_i and v_j , for $i = 2, \dots, m$ and $j = m + 1, \dots, n$. Let S_k denote the solution obtained from S by dropping v_k from the cycle, if $k = 2, \dots, m$, or adding v_k to it, if $k = m + 1, \dots, n$. The pseudocodes of the local searches are:

Local Search for MCP1

Start with S^n , $n = 0$.

Repeat

$$S' \leftarrow \arg \min_{k=2, \dots, n} \{Len(S_k^n) + Cos(S_k^n)\}$$

$$S'' \leftarrow \arg \min_{\substack{i=2, \dots, m \\ j=m+1, \dots, n}} \{Len(S_{ij}^n) + Cos(S_{ij}^n)\}$$

$$S^{n+1} \leftarrow \arg \min \{Len(S') + Cos(S'), Len(S'') + Cos(S'')\}$$

$$n \leftarrow n + 1$$

Until $Len(S^{n-1}) + Cos(S^{n-1}) = Len(S^n) + Cos(S^n)$.

Local Search for MCP2

Start with S^n , $n = 0$.

Repeat

$$S' \leftarrow \arg \min_{k=2, \dots, n} \{Len(S_k^n) : Cos(S_k^n) \leq \max\{d_0, Cos(S^n)\}\}$$

$$S'' \leftarrow \arg \min_{\substack{i=2, \dots, m \\ j=m+1, \dots, n}} \{Len(S_{ij}^n) : Cos(S_{ij}^n) \leq \max\{d_0, Cos(S^n)\}\}$$

$$S^{n+1} \leftarrow \arg \min\{Len(S'), Len(S'')\}$$

$$n \leftarrow n + 1$$

Until $Len(S^{n-1}) = Len(S^n)$.

After the set of vertices in the solution has been determined this way, the length of the cycle is improved by performing classical edge interchange moves such as 2-opt, 3-opt, Lin and Kernighan interchange, or some combination of them (see Lawler et al.[11]).

5.7. The strategy for the sizes

The size t of the tabu list is initially 0. Each time a vertex is removed from the cycle it goes to the tabu list and t augments one unit. When t reaches a given bound (ten per cent of n in our case) it is fixed, and the tabu list starts to work as a FIFO list. Just before performing the shake, the tabu list is emptied again.

The parameter s that controls the shake procedure is initialized to 2. Each time a local minimum is reached the shake procedure is applied s times. If the current local minimum is equal to the one found just before, s is augmented by one unit. If the current local minimum is better than the best known solution, s is set again to 2. We do not allow s to become larger than m .

5.8. Stopping criterion

In our computational experiments we use as stopping criterion the total number of iterations performed. Let it be the current iteration, then we stop when $it > Max_it$ for a given constant Max_it .

6. Computational results

The application of the VNTS to the two versions of MCP described in the previous sections were implemented in C++ and run on a SUN Ultra-60 computer running at 300 MHz. They were tested on a set of instances from the TSPLIB involving between 50 and 200 vertices and having EUC2D format (Euclidean distances).

For MCP1 the connection and allocation costs were symmetric and proportional to the Euclidean distance. To obtain optimal solutions visiting

approximately 75 %, 50 % and 25 % of the total number of vertices in the instances, we set $L[v_i, v_j] = \alpha d_{ij}$ and $D[v_i, v_j] = (10 - \alpha)d_{ij}$ for $\alpha \in \{5, 7, 9\}$, being d_{ij} the Euclidean distance between vertices v_i and v_j . For MCP2 the allocation cost was identical to the routing cost, and the threshold d_0 was computed as a proportion of the assignment cost c_0 of a shortest triangle including the depot. This was done to yield optimal routes visiting approximately 75 %, 50 % and 25 % of the total number of vertices in the instances. We then set $d_0 = \lceil \beta c_0 \rceil$ for $\beta \in \{0,08, 0,22, 0,42\}$.

Tables 1 to 3 and Tables 4 to 6 show the results obtained for MCP1 and MCP2 respectively. The column headings are defined as follows:

Name: instance name as it appears in the TSPLIB.

Exact-t: computing time for the exact procedure, in hh:mm:ss format.

Heur-t: computing time for the VNTS, in seconds.

%Error: the relative deviation of the heuristic solution from the optimal one.

The maximum number of iterations *Max_it* was set to 200. Five runs of the algorithm were performed for each instance, and we show in columns **Heur-t** and **%Error** the average, minimum and maximum values obtained.

The error is computed as $100(opt - heur)/opt$, being *opt* the solution given by the branch-and-cut algorithm described in Labbé et al. [10], and *heur* the heuristic solution. The solution given by the branch-and-cut algorithm is not really the optimal one in all the cases, since there was a time limit of two hours in the execution time for each instance. The negative values that appear in the error columns correspond to instances in which the heuristic solution is better than the best solution found by the branch-and-cut algorithm in two hours.

For each problem, different kinds of edge interchange moves were tested within the local search phase. We show in this section the tables corresponding to the best selection of moves, which were the combination of 2-opt and 3-opt for MCP1 with $\alpha = 5$ (Table 1), by 3-opt for MCP1 with $\alpha = 7$ and MCP2 with $\beta = 0,08$ (Tables 2 and 4 respectively), by 2-opt for MCP1 with $\alpha = 9$ (Table 3), and by the combination of 2-opt and Lin-Kernighan interchange for MCP2 with $\beta = 0,22$ and $\beta = 0,42$ (Tables 5 and 6 respectively).

We can conclude from the results that, in general, VNTS gives better results for MCP1 than for MCP2. Regarding the solution cycle size, the performance of VNTS is better when the solution is a small cycle. In those cases the average error is always smaller than 0.49 for MCP1 and smaller

than 0.60 for MCP2 (see Tables 3 and 6). For medium and big cycles, visiting approximately 50 % and 75 % of the total number of vertices in the instances, the error percentages increase.

7. Conclusions

The VNTS metaheuristic proposed in this paper is a very general tool that can be used to look for good solutions to optimization problems. We show how to apply it to any optimization problem consisting of the selection of an optimal set of items, considering the standard add, drop and add/drop moves. The consideration of constraints in the formulation does not increase the complexity of the procedures, as is shown for the location/allocation problems. The detailed implementation for the Median Cycle Problem got acceptable results in very few seconds when tested on the TSPLIB instances. Furthermore, for some identified instances, the VNTS got solutions better than the best known until the moment.

Future research will be focus on applying the metaheuristic to other location/allocation instances and to more general optimization problems.

8. Acknowledgments

We wish to thank professors Nenad Mladenović, Alain Hertz and Marino Widmer for their valuable comments and suggestions.

Referencias

- [1] J. Current, J.L. Cohon, C.S. ReVelle. The shortest covering path problem: an application of locational constraints to network design, *Journal of Regional Science* 24 (1984), 161-183.
- [2] M. Gendreau, G. Laporte, F. Semet. The Covering Tour Problem, *Operations Research* 45 (1997), 568-576.
- [3] F. Glover. Tabu Search - Part I. *ORSA Journal on Computing* 1 (1989), 190-206.

Name	Exact-t	Heur-t	%Error	Name	Exact-t	Heur-t	%Error
eil51.tsp	0:00:02	4,13	0,92	eil101.tsp	0:00:57	35,61	1,49
		3,90	0,40			33,50	0,34
		4,35	1,20			38,28	2,93
st70.tsp	0:00:46	11,20	0,12	pr107.tsp	0:01:16	45,31	0,13
		10,73	0,00			42,42	0,00
		12,40	0,32			46,81	0,22
eil76.tsp	0:00:32	14,56	0,99	pr136.tsp	0:06:48	108,66	1,51
		13,67	0,20			102,96	0,04
		14,98	1,83			113,98	2,46
pr76.tsp	0:05:39	15,87	3,53	pr144.tsp	1:25:37	126,96	1,93
		15,20	1,93			120,18	0,69
		16,28	4,47			138,53	4,37
rat99.tsp	0:00:45	36,73	1,64	kroA150.tsp	0:08:22	142,11	2,40
		33,41	1,10			137,53	0,43
		42,22	1,95			145,11	3,50
kroA100.tsp	0:02:33	38,51	0,12	kroB150.tsp	+2:00:00	148,74	-3,09
		36,87	0,00			134,19	-3,67
		40,72	0,24			157,15	-1,64
kroB100.tsp	0:01:19	39,22	1,04	pr152.tsp	+2:00:00	144,73	-2,13
		36,18	0,26			140,94	-2,63
		41,31	2,03			151,25	-1,49
kroC100.tsp	0:01:04	36,60	0,53	rat195.tsp	0:34:14	301,61	3,29
		35,90	0,34			297,00	2,39
		37,53	0,88			308,34	3,83
kroD100.tsp	0:01:07	37,35	2,08	kroA200.tsp	+2:00:00	366,99	-4,71
		36,18	0,08			350,83	-5,59
		38,31	4,26			384,00	-4,11
kroE100.tsp	0:04:06	37,44	0,59	kroB200.tsp	0:49:08	347,40	2,73
		36,47	0,41			339,40	1,78
		38,35	0,95			364,12	3,73

Cuadro 1: VNTS results for MCP1 with $\alpha = 5$. 2-opt + 3-opt

Name	Exact-t	Heur-t	%Error	Name	Exact-t	Heur-t	%Error
eil51.tsp	0:00:14	5,91	0,00	eil101.tsp	0:06:53	33,57	0,64
		5,01	0,00			22,79	0,00
		6,55	0,00			41,04	1,16
st70.tsp	0:00:31	20,70	0,00	pr107.tsp	0:04:07	43,67	0,00
		15,59	0,00			34,51	0,00
		27,97	0,00			58,71	0,01
eil76.tsp	0:00:58	21,52	0,00	pr136.tsp	0:10:08	86,98	1,31
		13,84	0,00			74,85	0,05
		27,13	0,00			97,07	2,74
pr76.tsp	0:06:23	13,58	0,29	pr144.tsp	0:21:12	140,21	0,72
		10,92	0,00			125,99	0,00
		15,70	0,61			147,00	1,56
rat99.tsp	0:02:23	29,34	0,62	kroA150.tsp	0:23:21	122,23	0,37
		23,32	0,26			110,13	0,00
		36,95	1,34			131,82	1,58
kroA100.tsp	0:02:27	44,55	1,30	kroB150.tsp	0:13:10	131,96	0,90
		39,55	0,00			111,20	0,00
		55,14	4,52			147,89	1,85
kroB100.tsp	0:02:09	31,38	0,23	pr152.tsp	+2:00:00	136,39	-1,15
		26,36	0,19			130,46	-1,58
		36,31	0,33			140,22	-0,48
kroC100.tsp	0:02:38	43,33	0,00	rat195.tsp	1:14:16	237,71	0,63
		38,76	0,00			200,60	0,11
		52,59	0,00			259,92	1,23
kroD100.tsp	0:03:13	36,43	0,54	kroA200.tsp	+2:00:00	306,36	-2,94
		31,83	0,06			250,96	-3,26
		39,42	1,67			357,11	-2,17
kroE100.tsp	0:01:54	54,39	0,16	kroB200.tsp	0:54:44	323,49	2,47
		40,75	0,00			266,08	1,79
		66,75	0,68			354,47	3,16

Cuadro 2: VNTS results for MCP1 with $\alpha = 7$. 3-opt

Name	Exact-t	Heur-t	%Error	Name	Exact-t	Heur-t	%Error
eil51.tsp	0:00:11	2,68	0,00	eil101.tsp	0:13:41	49,50	0,00
		2,04	0,00			37,72	0,00
		3,09	0,00			65,94	0,00
st70.tsp	0:01:47	23,60	0,00	pr107.tsp	0:16:06	100,38	0,00
		18,56	0,00			54,01	0,00
		27,58	0,00			128,07	0,00
eil76.tsp	0:03:46	20,84	0,00	pr136.tsp	0:51:10	120,72	0,00
		17,87	0,00			99,44	0,00
		23,37	0,00			151,24	0,00
pr76.tsp	0:04:46	10,71	0,05	pr144.tsp	0:51:08	174,38	0,00
		6,56	0,00			126,92	0,00
		22,86	0,18			222,85	0,00
rat99.tsp	0:12:01	27,37	0,00	kroA150.tsp	1:27:46	142,28	0,00
		23,02	0,00			120,82	0,00
		34,47	0,00			196,87	0,00
kroA100.tsp	0:12:42	27,17	0,14	kroB150.tsp	1:28:37	91,07	0,05
		19,75	0,00			53,80	0,00
		36,95	0,28			127,84	0,14
kroB100.tsp	0:13:19	32,05	0,03	pr152.tsp	1:30:10	116,90	0,49
		21,25	0,00			61,10	0,00
		50,60	0,09			196,28	1,61
kroC100.tsp	0:11:40	85,35	0,00	rat195.tsp	+2:00:00	177,39	-4,35
		71,01	0,00			107,52	-4,45
		92,33	0,00			240,32	-4,16
kroD100.tsp	0:15:16	67,72	0,00	kroA200.tsp	+2:00:00	261,71	-1,49
		49,72	0,00			160,70	-1,67
		90,26	0,00			373,66	-1,32
kroE100.tsp	0:13:06	26,93	0,00	kroB200.tsp	+2:00:00	254,82	-2,91
		19,62	0,00			131,02	-2,94
		36,46	0,01			469,27	-2,82

Cuadro 3: VNTS results for MCP1 with $\alpha = 9$. 2-opt

Name	Exact-t	Heur-t	%Error	Name	Exact-t	Heur-t	%Error
eil51.tsp	0:00:39	4,71	2,25	eil101.tsp	0:05:05	41,57	4,58
		4,42	0,87			40,90	3,02
		4,94	3,18			42,11	6,26
st70.tsp	0:01:17	12,80	1,74	pr107.tsp	+2:00:00	43,34	0,09
		12,57	0,61			41,90	-0,01
		13,07	3,64			44,17	0,37
eil76.tsp	0:02:59	17,04	5,70	pr136.tsp	+2:00:00	128,88	4,26
		16,56	4,91			118,75	3,21
		17,27	6,39			133,95	4,84
pr76.tsp	0:50:15	15,29	5,39	pr144.tsp	+2:00:00	113,15	1,65
		14,97	1,06			89,78	0,71
		15,54	10,32			129,51	3,73
rat99.tsp	0:21:57	36,96	1,71	kroA150.tsp	+2:00:00	153,32	1,44
		36,15	1,21			150,61	-0,28
		37,55	2,41			157,78	3,78
kroA100.tsp	0:19:33	39,95	2,39	kroB150.tsp	1:02:26	150,39	1,70
		39,03	0,87			144,78	0,73
		40,54	8,04			154,53	2,39
kroB100.tsp	0:17:59	40,61	1,94	pr152.tsp	+2:00:00	114,66	-1,74
		39,37	0,94			111,42	-2,41
		41,41	4,28			119,07	-1,24
kroC100.tsp	0:22:23	38,31	1,12	rat195.tsp	+2:00:00	339,93	-3,58
		37,02	0,34			334,40	-4,21
		39,62	1,96			345,02	-2,77
kroD100.tsp	0:16:59	41,18	1,74	kroA200.tsp	+2:00:00	396,81	4,01
		39,09	1,45			387,63	3,30
		42,87	1,98			405,35	5,03
kroE100.tsp	0:19:39	39,80	1,70	kroB200.tsp	+2:00:00	409,43	4,28
		39,15	1,06			392,54	2,46
		40,39	2,48			432,73	6,91

Cuadro 4: VNTS results for MCP2 with $\beta = 0,08$. 3-opt

Name	Exact-t	Heur-t	%Error	Name	Exact-t	Heur-t	%Error
eil51.tsp	0:00:19	2,36	2,44	eil101.tsp	0:11:25	22,34	1,53
		2,09	0,81			21,20	1,05
		2,62	4,07			23,07	2,79
st70.tsp	0:00:56	8,63	0,59	pr107.tsp	0:21:40	29,55	0,13
		7,90	0,00			23,13	0,00
		9,28	2,36			42,57	0,64
eil76.tsp	0:01:51	9,16	1,87	pr136.tsp	0:45:57	72,87	3,86
		8,23	0,75			64,88	0,46
		9,81	4,12			95,11	7,04
pr76.tsp	1:33:17	10,91	1,04	pr144.tsp	1:45:03	78,44	0,63
		10,12	0,53			64,47	0,19
		11,43	1,58			95,62	1,50
rat99.tsp	0:06:31	28,57	0,27	kroA150.tsp	1:41:36	87,85	3,82
		25,50	0,00			77,39	0,36
		29,89	0,67			96,71	5,83
kroA100.tsp	0:22:24	23,84	4,30	kroB150.tsp	1:03:36	91,39	0,36
		21,43	0,03			72,80	0,12
		25,63	7,22			105,78	0,57
kroB100.tsp	0:14:00	25,45	2,50	pr152.tsp	+2:00:00	90,97	-5,25
		24,22	0,14			66,97	-5,84
		26,54	6,31			139,59	-4,22
kroC100.tsp	0:08:58	29,44	0,25	rat195.tsp	+2:00:00	192,23	0,39
		22,19	0,00			164,29	0,23
		36,64	0,52			223,79	0,58
kroD100.tsp	0:10:24	26,64	1,36	kroA200.tsp	+2:00:00	231,10	-11,35
		23,19	0,40			203,91	-12,47
		32,13	2,53			276,24	-10,51
kroE100.tsp	0:17:40	27,03	0,22	kroB200.tsp	+2:00:00	221,38	-3,88
		23,27	0,16			196,87	-5,56
		29,38	0,30			267,48	-1,36

Cuadro 5: VNTS results for MCP2 with $\beta = 0,22$. 2-opt+LK

Name	Exact-t	Heur-t	%Error	Name	Exact-t	Heur-t	%Error
eil51.tsp	0:01:11	2,85	0,12	eil101.tsp	0:13:46	20,48	0,60
		2,16	0,00			16,99	0,60
		3,43	0,58			27,29	0,60
st70.tsp	0:02:20	8,24	0,17	pr107.tsp	1:21:46	16,97	0,15
		6,29	0,00			13,10	0,00
		11,12	0,84			23,29	0,52
eil76.tsp	0:03:10	9,42	0,00	pr136.tsp	1:57:07	49,63	0,29
		7,53	0,00			43,41	0,25
		11,28	0,00			59,51	0,36
pr76.tsp	0:08:22	9,43	0,29	pr144.tsp	+2:00:00	43,68	-1,93
		8,38	0,01			25,18	-2,75
		11,14	0,59			56,65	-0,72
rat99.tsp	0:14:41	21,06	0,29	kroA150.tsp	+2:00:00	62,59	-4,97
		12,30	0,00			56,03	-7,06
		24,65	0,97			68,71	-1,80
kroA100.tsp	0:19:19	20,98	0,47	kroB150.tsp	+2:00:00	50,24	-1,41
		16,47	0,00			41,74	-1,85
		24,74	0,65			62,88	-1,07
kroB100.tsp	0:15:52	24,85	0,24	pr152.tsp	+2:00:00	78,65	-6,73
		17,15	0,00			59,71	-6,73
		28,82	0,64			89,21	-6,71
kroC100.tsp	0:18:43	24,17	0,11	rat195.tsp	+2:00:00	133,67	-6,21
		19,92	0,11			108,82	-6,31
		28,47	0,11			186,20	-6,15
kroD100.tsp	0:27:05	23,01	0,38	kroA200.tsp	+2:00:00	139,95	-5,75
		18,31	0,00			89,08	-5,89
		27,41	0,64			177,77	-5,65
kroE100.tsp	0:15:58	22,67	0,10	kroB200.tsp	+2:00:00	140,39	-5,34
		17,75	0,00			103,46	-5,58
		29,78	0,13			197,85	-4,78

Cuadro 6: VNTS results for MCP2 with $\beta = 0,42$. 2-opt+LK

- [4] F. Glover. Tabu Search - Part II. *ORSA Journal on Computing* 2 (1990), 4-32.
- [5] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston 1997.
- [6] P. Hansen, N. Mladenović and D. Pérez-Brito. Variable Neighborhood Decomposition Search. To appear in *Journal of Heuristics* (1999).
- [7] J.H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, Michigan 1975.
- [8] V.A. Hutson, C.S. ReVelle. Maximal direct covering tree problems, *Transportation Science* 23 (1989), 188-299.
- [9] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi. Optimization by simulated annealing, *Science* 220 (1983), 671-666.
- [10] M. Labbé, G. Laporte, I. Rodríguez Martín and J.J. Salazar González. The Median Cycle Problem. Working paper, CRT, Université de Montréal, 1999.
- [11] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys. *The Traveling Salesman Problem*. John Wiley & sons, New York 1985.
- [12] N. Mladenović and P. Hansen. Variable Neighborhood Search. *Computers Oper. Res.* 24 (1997), 1097-1100.
- [13] N. Mladenović, J.A. Moreno-Pérez, and J. Marcos Moreno-Vega. A Chain-Interchange Heuristic Method, *Yugoslav J. Oper. Res.* 6 (1996), 41-54.