Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 173 (2006) 540–555
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Discrete Optimization

A new tabu search algorithm for the vehicle routing problem with backhauls

José Brandão

Departamento de Gestão, EEG, Universidade do Minho, Largo do Paço, 4704-553 Braga, Portugal

Received 26 January 2004; accepted 10 January 2005
Available online 1 April 2005

Abstract

In the distribution of goods from a central depot to geographically dispersed customers happens quite frequently that some customers, called linehauls, receive goods from that depot while others, named backhauls, send goods to it. This situation is described and studied by the vehicle routing problem with backhauls. In this paper we present a new tabu search algorithm that starting from pseudo-lower bounds was able to match almost all the best published solutions and to find many new best solutions, for a large set of benchmark problems.

Keywords: Vehicle routing; Backhauls; Tabu search; Lower bound

1. Introduction

A company that daily delivers (or collects) goods to a set of customers geographically spread, with a given demand, using a fleet of vehicles that start from a central depot and return to it after the service, faces a problem that is most usually called the *vehicle routing problem* (VRP). The resolution of this problem consists in defining the best assignment of the customers to the vehicles and the sequence in which they are served in order minimise the total travelling cost. When both delivery and collection of goods are required we have a particular case that is named the *vehicle routing problem with backhauls* (VRPB). The delivery and the pickup, or backhaul, customers could be taken separately originating two distinct VRPs, but if both types of customers are serviced in the same route significant savings can be obtained.

In the literature there are many descriptions of applications of the VRPB as, for example, in the article written by Casco et al. (1988). Among other examples, these authors describe the following two real-world cases: a company that distributes coffee, juice, etc. to its customers and brings back the raw materials to its factory; another company that sells fire extinguishers, delivers to its customers new

E-mail address: sbrandao@eeg.uminho.pt

0377-2217/$ - see front matter
doi:10.1016/j.ejor.2005.01.042

extinguishers and picks up old unusable extinguishers or those that require service at the depot.

In practice we may find VRPBs with many different characteristics like the percentage of backhaul customers, if a vehicle travels multiple trips per day, if a vehicle can service backhauls only, etc. However, usually, the VRPBs are divided into three categories according to the order of the deliveries and pickups:

(i) The pickups can be made before the deliveries.
(ii) The pickups can be made before the deliveries and each customer can be simultaneously a delivery and a pickup point.
(iii) In a route, first are served the linehaul customers and then the backhauls. No routes are allowed containing only backhauls, but a route can contain linehaul customers only.

All these types of VRPBs can appear in practice, but the first two are less frequent because it is usually difficult to re-arrange the goods in a partially loaded vehicle. Due to this reason, they have been also less studied. Some authors that have investigated the first two types are Golden et al. (1985), Casco et al. (1988) and Salhi and Nagy (1999).

This research is about the third type of VRPB (although the same algorithms, with a few modifications, can be applied to the other two types and even to the case when routes containing only backhauls are allowed) which has been studied by several authors like the following ones: Deif and Bodin (1984), Goetschalckx and Jacobs-Blecha (1989, 1993), Toth and Vigo (1997), Toth and Vigo (1999), Mingozzi et al. (1999) and Osman and Wassan (2002). Deif and Bodin (1984) used the classical Clarke and Wright savings algorithm with a modification in the definition of the savings. Goetschalckx and Jacobs-Blecha (1989), applied the concept of spacefilling curves to develop heuristics for the VRPB. Goetschalckx and Jacobs-Blecha (1993) introduced a new heuristic based on the generalised assignment problem for the formation of the clusters of customers that originate the routes. They have shown that this new algorithm performs better than those published before. Toth and Vigo (1997) and Mingozzi et al. (1999) created two different mathematical formulations of the VRPB and they have been able to solve exactly problems with up to 100 customers. Toth and Vigo (1997) tackled both the symmetric and asymmetric VRPB, while Mingozzi et al. (1999) solved only the symmetric case. In spite of the quality of these exact methods, the VRPB is NP-hard and therefore approximate algorithms are still required for solving most of the VRPBs encountered in practice. So far, the best published heuristic for the VRPB is due to Toth and Vigo (1999), which takes advantage of the information given by a Lagrangian relaxation, proposed by the same authors to obtain lower bounds for the exact resolution, in the formation of clusters of customers. The clusters are then converted into feasible routes and, in the end, intra-route and inter-route post-optimisation procedures are applied to improve the quality of the solution.

More recently, Osman and Wassan (2002) published a tabu search metaheuristic which, on average, produces even better solutions than Toth and Vigo's (1999) algorithm, but requires much more computing time. They use two heuristics for generating the initial solutions: one that combines savings and insertion and another that combines savings and assignment. In their tabu method, the neighbourhood is defined by the interchange of one or two consecutive customers between two routes. On the other hand, the tabu tenure is defined dynamically during the search by a reactive procedure.

The VRPB studied in this research is defined as follows. Let $G = (V, A)$ be a complete undirected network where $V = \{0\} \cup L \cup B$ is a set of vertices and $A = \{(i,j) : i, j \in V\}$ is the set of arcs, and to each arc (i,j) is associated a nonnegative cost (distance) c_{ij}, with $c_{ij} = c_{ji}$ for each $i, j \in V$, such that $i \neq j$ and $c_{ii} = +\infty$ for each $i \in V$. The subsets $L = \{1, 2, \ldots, n\}$ and $B = \{n+1, n+2, n+m\}$, represent, respectively, the linehaul and the backhaul customers, and 0 represents the depot. The total number of customers is represented by N. In the depot there are K identical vehicles with a capacity Q. Each customer $i \in L \cup B$ requires a given quantity q_i to be delivered ($i \in L$) or collected ($i \in B$). The number of vehicles is defined

as $K \geqslant \max\{K_L, K_B\}$, where K_L and K_B is the minimum number of vehicles needed to serve all the linehaul and backhaul customers, respectively. The resolution of the VRPB consists in finding a set of routes, starting and ending at the depot, with a minimum total cost, given by the sum of the arcs belonging to the routes, such that

(i) Each vehicle travels exactly one route.
(ii) In a route, the linehaul are served before the backhaul customers (these are called the precedence constraints) and no route can contain only backhaul customers. Because of these conditions, we consider that the cost $c_{ji} = c_{0j} = R$, for each $j \in B$ and $i \in L$, where R is a very large constant (this cost plays an important role in the tabu search method for discovering feasible solutions).
(iii) The total demand of all the customers, considering the linehauls and the backhauls separately, on a route must not exceed the vehicle capacity.
(iv) Each customer is visited just once by one of the vehicles, and its requirements must be completely fulfilled.

This paper is organised as follows. In Section 2 we describe how the initial solutions have been obtained. In Section 3 we present the tabu search algorithm, starting with the definition of its main features and then showing its global structure. In Section 4 we present the computational experiments, used to test the quality of the algorithm, and in the last section we draw the conclusions.

2. The initial solutions

In this paper we use two methods for obtaining the initial solutions, both of which explore the affinities between the VRP and the VRPB

(i) A method based on the fact that a VRPB route, with linehauls and backhauls, is constituted by two Hamiltonian paths linked together. Since, as explained below, this corresponds to the resolution of two separate open VRPs we will call it *open initial solution*.
(ii) A method based on a lower bound for the VRP, which will be called *K-tree initial solution*, because we use K-trees for the calculation of that lower bound.

2.1. The open initial solution

In a VRPB we have two distinct sets of customers: linehauls and backhauls. Therefore, we can consider each set of customers separately and solve two VRPs and, in the end, join together pairs of routes, one from each VRP, in order to obtain a VRPB solution. However, there is another kind of VRP with a closer structure to the VRPB, which is called *open* VRP (OVRP), because in the end of the route the vehicles are not required to return to the depot. This way, a better alternative is to solve two separate OVRPs and, in the end, link each Hamiltonian path of the linehaul solution with another Hamiltonian path of the backhaul solution, until no backhaul path remains unlinked.

In the following we describe in detail the algorithm used to obtain the open initial solution.

Step 1. Solve two distinct OVRPs, one for each set of customers. The OVRP resolution is composed of two phases: *initial phase* (construction of the routes of the initial solution) and *improvement phase*.

In the *initial phase* a nearest neighbour heuristic, that creates a set of routes sequentially, is used. Each route is started with the unrouted customer nearest from the depot, then is added the nearest admissible customer from this one, and so on. A customer is said to be admissible if its insertion in the route does not exceed the capacity of the vehicle. When no customer is admissible in the route under construction a new route is started or the process stops if all the customers are already routed.

In the *improvement phase*, the tabu search algorithm (*phase I*) described in Section 3 is applied. In this phase, for the linehaul OVRP, the objective is to minimise the total distance travelled by the vehicles, considering a fixed number of vehicles (routes), while for the backhaul OVRP, the first objective is to minimise the number of routes

and the second one is to minimise the total distance of the routes.

Step 2. Each backhaul path is linked to the end of one of the linehaul paths. For each group of two paths (one linehaul and one backhaul) there are four different ways of connecting their extremes. All these combinations are considered and, in the end, the least cost link is chosen, and these two paths (one linehaul and another backhaul) form a route of the solution and they are removed from the sets of paths available. Repeat this step until either the set of linehaul or the set of backhaul paths is empty. In the end, each linehaul (or backhaul) path remaining is connected to the depot forming a close route. If any route contains only backhaul customers the solution is infeasible, but it can be easily repaired by the tabu search method.

2.2. The K-tree initial solution

The other method used to obtain the initial solution is based on a pseudo-lower bound as described in the following.

Let us define K as the number of vehicles used to service all the customers. As stated before, K can be the minimum necessary or higher.

A *K-tree* is defined as a set of $N + K$ edges that span a graph with $N + 1$ nodes. The 0-tree is the usual spanning tree. The minimum cost *K-tree* can be determined by a polynomial algorithm provided by Fisher (1994b). Very briefly, this algorithm consists of the following: first the minimum spanning tree (MST) of the graph is determined and then K least cost edges not belonging to the MST are added to it. After this, appropriate interchanges of edges are made, until the degree of the depot is $2K$.

Fisher (1994a) formulated the VRP as a minimum cost *K-tree*, with degree $2K$ on the depot, with two other types of constraints: the degree two on each customer and the capacity constraints. Making a Lagrangian relaxation of these two types of constraints, the resulting problem (LRP) still corresponds to a minimum *K-tree* problem, dependent on two sets of dual variables associated with those constraints. The solution value of the LRP, which is solved by the subgradient method, is a lower bound for the VRP. Each iteration of the subgradient method gives a different lower bound, with a general trend of improvement as more iterations are performed. We executed 1200 iterations and chose the ten best lower bounds. In the application of the subgradient method, we followed Fisher's (1994a) research, but we have introduced a few modifications that are explained in Brandão (2001). The lower bound value is the sum of the cost of a minimum cost *K-tree* (it is from this tree that is obtained the initial solution) with the values of the capacity multipliers. Since there is no guarantee that this is a lower bound for the VRPB we call it pseudo-lower bound. However, its structure should have many points in common with a true lower bound, due to the similarities between the VRP and the VRPB. If we have used a lower bound, like the one defined by Toth and Vigo (1997), the final results could be better, but the computer programmes to calculate it were not available. On the contrary, we followed this procedure, because the purpose of this research is not the definition of a tight lower bound for the VRPB, and we could benefit from our previous work with *K-trees*, namely computer programmes, for the VRP (Brandão (2001)).

In order to apply this method to the VRPB, we made the following assumptions:

(i) No distinction is made between linehaul and backhaul customers and, therefore, no precedence constraints are considered.
(ii) The (virtual) capacity of the vehicles is increased proportionally to the sum of the demand of the backhaul customers. For example, if $K = 5$, Q (the real capacity) $=$ 4000 and the sum of the backhauls' demand is 10,000, in the determination of the lower bound it is assumed that the capacity of the vehicle is $Q' = 4000 + 10{,}000/5 = 6000$.

The procedure that we used to obtain the initial solution from a minimum cost *K-tree* is explained in the following paragraphs.

In the graph defined by the edges of the minimum cost *K-tree*, the minimum cost spanning tree (MST) is calculated. Then, deleting in this MST the edges incident on the depot, a given number

of connected components is obtained, each will give rise to one or more routes, according to the procedure described below.

For each connected component, the first chain starts at the depot and follows the sequence of customers (defined by the component) until it appears one with degree greater than two (if the degree of the customer is one or two the chain is well defined: it ends in the first case and, in the second case, it goes on unbroken even if the demand exceeds the capacity of the vehicle). At this point we decide to include this customer in the chain if its demand does not exceed the capacity (for delivery or for collection, considered separately) already available in the vehicle; otherwise, the chain ends and a new chain is started with it. In both cases, the next customer to enter the chain is the nearest, in the component, not yet in any chain, with degree greater than one (if one exists). When this chain finishes, because of a customer with degree one or greater than two, a new one is started at the last customer with degree greater than two that is linked to the customers that are not yet in any chain. Each chain generated in this way is a VRPB route.

An example of a *K-tree* is represented in Fig. 1. In this problem, there are 12 customers (the first nine are linehauls and the other three are backhauls), the depot (0) and three vehicles. Applying the procedure above, the edges represented by dashed lines are removed (because they do not belong to the MST) and the following five routes are obtained: 0_11_8_12_5_9_0; 0_7_10_0; 0_1_2_3_0; 0_4_0; 0_6_0.

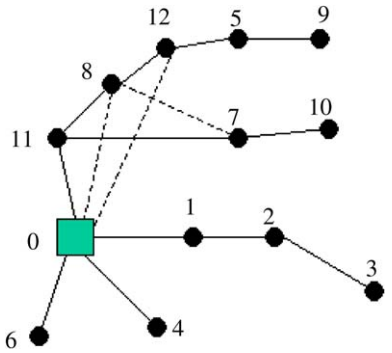

Fig. 1. Example of a *K-tree*.

In general, the initial VRPB solution generated from a *K-tree* is infeasible either in terms of precedence constraints or capacity constraints, and the number of routes is usually greater than K. In order to try to overcome all these sources of infeasibility and, at the same time, to improve the initial solution, the tabu search procedure (*phase I*) described in Section 3 is applied. In the end of *phase I* we always get a feasible solution in terms of precedence constraints.

The procedures for determining the initial solution from a *pseudo-lower bound* are summarized below:

Step 1. Consider the linehauls plus the backhauls simply as customers. Consider the capacity of the vehicle as Q' instead of Q. Therefore, assume the VRPB as being a VRP.
Step 2. Following Fisher (1994a), formulate the VRP as a minimum cost *K-tree*, with degree *2K* on the depot, with the additional constraints of degree two on each customer and of capacity.
Step 3. Make the Lagrangian relaxation (*LR*) of the degree and capacity constraints.
Step 4. Solve the *LR* problem by the subgradient method and execute 1200 iterations. Select the best 10 lower bounds and keep the corresponding *K-trees*.
Step 5. Using each of the 10 *K-trees* and the procedure described previously create 10 initial solutions.

3. The tabu search algorithm

The tabu search theory and many of its applications can be find in Glover and Laguna (1997) and, therefore, our unique concern in is section is to describe how the tabu search principles have been applied to solve the VRPB. A good tabu search method requires, besides the correct application of the tabu search principles, good local search heuristics, and additional great benefits can be obtained if the tabu search is combined with

information given by exact methods. This paper has justly been produced in this vein, by carrying in the initial solution the information given by pseudo-lower bounds to the VRPB, as was described in the previous section.

The tabu search algorithm (TSA) starts with an initial solution, feasible or infeasible, generated by one of the methods described in the previous section. The TSA is composed of three phases that are applied sequentially: *phase I* that makes a first improvement of the initial solution and whose main goal is to create a feasible VRPB solution; *phase II* that takes most of the computing time and gives the main contribution to improve the solution; *phase III* that tries a further improvement of the solution, obtained in *phase II*, using a frequency-based memory. The main structure of these three phases, which is defined by the items described below, is the same; they only differ in some parameter values, type of moves allowed and methods of improving individual routes.

3.1. Neighbourhood structure

The neighbourhood of the current solution is defined through the candidate set and the types of moves. In this research, the set of candidates is constituted by all the customers, except in *phase III*, when only $[N/2]$ of the customers belong to that set.

Three types of trial moves have been used: *insert*, *swap* and *intrachange*. An *insert* move consists in taking a customer from one route and inserting it into another route or it can also be inserted in a new route if the it is a linehaul customer. A *swap* move consists in exchanging two customers belonging to two different routes. In the *intrachange* a linehaul changes position with a backhaul customer inside the same route. The *intrachange* is only applied in *phase I* and to those routes that do not respect the precedence constraints. In a route the set of candidates for the intrachange is constituted by the customers that are out of position. A linehaul is out of position if its index in the route is higher than the number of linehauls in the route, and a backhaul is out of position if its index is lower or equal to that number (we are assuming that the depot has index 0 in the route). The customer that goes to a different route must be inserted between customers of the same type (linehaul or backhaul) or near the depot or between the extremes of the linehaul and backhaul paths. Besides, the customer is inserted where its insertion cost is lower. On the other hand, the insert move originates a solution with one route less if the customer is removed from a route with one customer only. Since the first objective is to use a fixed number of vehicles, once this number has been attained, the algorithm neither allows the creation of new routes nor the elimination of existing ones, except while no feasible solution is known. If a solution is feasible and contains only one route none of these types of moves is performed.

The computing time of each iteration of the TSA depends, to a great extent, on the number of trial moves. The computational complexity, resulting from the trial moves, of each iteration, is $O(N^2 + N^3)$, where the first and the second terms are, respectively, for the insert and swap moves. The computing time can be reduced substantially if in each iteration the cost of the trial moves is only calculated for those two routes that have been transformed in the preceding iteration. This has been done by keeping in memory the cost of the trial moves that remain unchanged from one iteration to another.

Since the computing time is strongly determined by the trial swap moves one way of reducing it is to reduce the frequency of the trial swap moves. We decided to execute the trial swap moves in every ten iterations, because this frequency gives a good trade-off between the quality of the final solution and the computing time.

3.2. Tabu tenure and aspiration criteria

The tabu status of a move is defined in the following way: a customer that leaves a route cannot return to it during a given number of iterations, θ. The tabu tenure of a move, θ, is a random number in the interval $\theta \in [N/6, N/2]$, where the extremes of this interval are rounded up to the nearest integer. However, we have also experimented with a tabu tenure of fixed size in the middle of that interval, i.e., equal to $\lceil N/3 \rceil$ in order to make some

comparisons. Besides, this last value is always used in *phases I* and *III* of the TSA. The definition of the tabu tenure was based on our previous research with the OVRP (Brandão, 2004), but without any preliminary experiments with the VRPB.

The tabu status of a move can be overruled if a solution is feasible and is better than any feasible solution known so far, or if it is infeasible and its cost is lower than the cost of any infeasible solution already known. In these two cases, not only the tabu status of the move is overruled but also the move is immediately performed without executing the remaining trial moves. This first best strategy has the objective of accelerating the search, but obviously it may impede the discovery of a better local optimum of the current neighbourhood.

3.3. Evaluation of the moves

An important feature of the TSA is the *strategic oscillation* that results from crossing the boundary of the feasibility. Therefore, a move can be performed even if the resulting solution is infeasible. Each potential move is evaluated according to the cost of the solution (C) resulting from the move, which is given by the following equation:

$$C = \sum_{i=1}^{r} [c(i) + Pl(i)], \tag{1}$$

where $c(i)$ is the cost of route i, r is the total number of routes in the solution, P is a penalty, $l(i)$ is the excess of load (in the linehaul plus in the backhaul parts of the route) in route i. If a solution is feasible, $l(i)$ is equal to zero for all the routes. P is initially equal to 1 and is multiplied by 2 if during the last 10 consecutive iterations all the solutions have been infeasible. Conversely, P is divided by 2 if all the solutions have been feasible during the last 10 consecutive iterations. We used this value of P based on our previous experience with the VRP and the OVRP, but it is possible that other values could give better results.

In each iteration is chosen the trial move, not tabu, that generates the minimum cost solution as defined by Eq. (1).

3.4. Improving procedures of the routes

Given a VRPB solution its cost may be reduced by optimising each of the individual routes. However, to find an optimal route is a travelling salesman problem (TSP), if the route contains only linehaul customers, or a TSP with precedence constraints if backhaul customers are also present in the route. In both cases we have a NP-hard problem and, therefore, we should rely on approximate methods as the two described in the following:

(i) The *nearest neighbour* (NN) method. Considering that each part of the route (the linehaul and the backhaul paths) is a set of customers (S) plus the depot, this method works as follows: starting at the depot, the next customer of the sequence is the customer of S not yet in the sequence and nearest to the last that entered in the sequence. This method is very simple and very fast, but gives good results with the OVRP (Brandão, 2004). This is the reason why it has been used in the improvement phase of the open initial solution (*phase I* of the TSA). However, it should not perform so well with VRPB routes, because the link to the depot (for routes with linehaul customers only) or the link to the first backhaul customer of the backhaul path is not considered. This is one of the reasons for using the next procedure, which is more powerful, but is also more complex and requires more computing time.

(ii) The *unstringing* and *stringing* (US) procedure created by Gendreau et al. (1992).

When a route contains only linehaul customers the US is applied exactly as it was originally conceived by their authors. However, if a route is constituted by two Hamiltonian paths, the US requires some modifications. First of all, the algorithm must deal with a path and not a cycle. In second place, it is important to consider, in the cost of the path, the link between the last linehaul and the first backhaul, or vice versa if we are optimising a backhaul path.

The first operation, before applying the US to a VRPB route, is to try the other three ways of linking the two paths, and choose the best one. Then the US is applied to the linehaul path. If there is an improvement of this path, this may originate

a new linking customer (the last customer of the path), that is taken into account in the next application of the US to the backhaul path. Then, if there is an improvement in the backhaul path, the whole process is repeated until there are no more improvements.

The parameter of the US that defines the neighbourhood among the customers, p, has been defined equal to five, in this research. The US is only applied in *phases II* and *III*, and, since it is rather time consuming, we imposed some restrictions on its application:

(i) in the end of each iteration to each of the routes just modified, if it has not been applied to that route for at least 10 iterations;
(ii) to all the routes of the current solution after every N iterations without improvement of the best known feasible solution.

In *phase I*, when applied to the *K-tree* initial solution, no improving procedure was used (neither the NN nor the US methods), because a route may not respect the precedence constraints.

3.5. Execution control and restart

In the TSA there are three ways of controlling the execution time: the maximum total number of iterations (N_1), the maximum number of iterations without improvement of the best known feasible solution (N_2) and the maximum number of iterations without improvement of the best known feasible or infeasible solution (N_3). The execution of the programme is stopped when the number of iterations N_1 and N_2 are both attained, or when the number of iterations doubles N_1, or when the value N_3 is achieved. Therefore, the total number of iterations is not known in advance, depending on the evolution of the search.

With the aim of intensifying the search around good local optima we apply a process that has proved to be very successful. This consists in restarting the search from the best known feasible solution (or best infeasible if none is feasible), if during a given number of iterations (M) there is no improvement neither in the best known infeasible solution nor in the best known feasible one. With this restart, the tabu list is emptied.

The values of all these parameters have been established as a function of the problem size (N) and in a way that allows them to adapt to the search process. For example, the combination of N_1 and N_2 as stopping criterion allows the search to go on if a new best solution as been discovered recently, which indicates that the algorithm is exploring a promising region. A similar reasoning is behind the use of N_3. On the other hand, N_2 should be greater than M to give time for improvement after the restart. When all the parameters are fixed, including the tabu tenure and the candidate set, there is no reason for using N_3 greater than the double of M, because after two restarts without improvement the algorithm starts cycling.

3.6. Reduction of the number of routes

Sometimes the sole application of the insert move is not enough for finding a solution with a predefined value K of routes. For these cases, there is a mechanism which consists in joining the two routes with the lowest linehaul demand. This mechanism is only applied if the following two conditions are verified simultaneously:

(i) the number of routes in the current solution is greater than K;
(ii) the current iteration is greater than $50\sqrt{N}$ and at least 200 iterations have been executed since the last application of this procedure, or at each restart.

3.7. Long-term memory

The strategic oscillation and the restart already described are two forms of long-term memory, but besides these we used a frequency-based memory. In order to do this, during *phase II* the number of moves of each customer is stored. After finishing this phase, is started *phase III* whose only differences from *phase II* are the set of candidates and the use of the insert type of move only. In the beginning of *phase III* the set of candidates is constituted by half (more precisely, $\lceil N/2 \rceil$) of the

customers with the lowest frequency of moves. After the restart, the set of candidates is the complementary set of customers, i.e., the other half with the highest frequency. A new restart implies the use of the first set and so on. The main objective of *phase III* is to intensify the search inside small regions around good solutions already known. However, since they are already good, no large improvements are expected, but the experience shows that sometimes small improvements occur that hardly can be achieved by other means.

3.8. Global description of the algorithm

In this research, we implemented three versions of the TSA that have been called *TSA open*, *TSA-K-tree* and *TSA-K-tree_r* (for short, in Tables 1 and 2 we write simply *open*, *K-tree* and *K-tree_r*). Although the main structure of the TSA is the same they differ in a few details, as follows:

(i) *Phase I* is identical for the *TSA-K-tree* and *TSA-K-tree_r*, but in the *TSA-open* it is applied separately to the linehaul and backhaul sets. Besides, in the first two versions *phase I* is applied 10 times because 10 initial solutions are used.

(ii) *Phase II* is identical for the three versions except that in the *TSA-K-tree_r* the tabu tenure is random. On the other hand, *phase II* is applied to the solutions coming from *phase I* which implies that it is applied once in the *TSA-open*, 10 times in the *TSA-K-tree* and 20 times in the *TSA-K-tree_r* because in this case it is applied twice to each initial solution. In the *TSA-K-tree_r* the best three *different* solutions (if, for example, all the 20 solutions are identical only one can be chosen) are selected and *phase II* is repeated with these three solutions.

(iii) *Phase III* is identical for the versions *TSA-K-tree* and *TSA-open*, but in the *TSA-K-tree_r* it is not applied. Besides, in the *TSA-K-tree* it is applied three times—once to each of the best three different solutions chosen in the end of *phase II*. In the *TSA-K-tree*, after finishing this phase, *phase II* is applied again to these three solutions, with different values of N_1, N_2, and M.

Some other details of the TSA not yet defined, especially parameter values (the nonintegers are rounded to the nearest integer), can be found below.

Phase I

(i) For the *TSA-open*: $N_1 = 400\sqrt{N'}$, $N_2 = 0$, $N_3 = 20N'$, $M = 10N'$ (where $N' = n$ and $N' = m$, for the linehauls and backhauls, respectively). No swap moves are performed. The two paths modified in each iteration are improved by the NN procedure. In the end of this phase the linehaul paths are linked with the backhaul paths as described in Section 2.1 (step 2) originating a VRPB solution.

(ii) For the *TSA-K-tree* and the *TSA-K-tree_r*: $N_1 = 400\sqrt{N}$, $N_2 = 0$, $N_3 = 7N$, $M = 3.5N$. Only insert and intrachange moves are allowed. The trial intrachange is only applied every five iterations.

Phase II

(i) For all versions: $N_1 = 1000\sqrt{N}$, $N_2 = 18N$, $N_3 = 28N$, $M = 14N$.

(ii) For the *TSA-K-tree* after applying *phase III*: $N_1 = 1300\sqrt{N}$, $N_2 = 21N$, $N_3 = 40N$, $M = 20N$. Both insert and swap moves are allowed, but the trial swap moves are only performed every 10 iterations.

Phase III

$N_1 = 400\sqrt{N}$, $N_2 = 0$, $N_3 = 20N$, $M = 10N$. Only insert moves are allowed.

4. Computational experiments

All the programmes of this article have been written in C language and executed on a Pentium III HP Vectra VEi8 at 500 MHz. The performance of our algorithm was tested using two sets of benchmark problems, kindly provided to us by Wassan, that have been used by most of the researchers that studied the VRPB. To our knowledge the best published solutions for these test problems have been given by the exact algorithms of Toth and Vigo (1997) and Mingozzi et al. (1999), and the heuristic algorithm of Toth and Vigo (1999) and Osman and Wassan's (2002) metaheuristic. So, we are going to use their results for assessing the performance of our algorithm.

Table 1
Results for the first set of problems

No.	Name	N	n	m	Q	K	Best published	Our best		TV99		OW		Open		*K-tree*		*K-tree_r*	
								Cost	Ratio	Cost	CPU (seconds)	Cost	CPU (seconds)	Cost	CPU (seconds)	Cost	CPU (seconds)	Cost	CPU (seconds)
1	A1	25	20	5	1550	8	229,886	229,886	100.0	229,886	6.0	229,886	31	230,548	3	229,886	24	229,886	40
2	A2	25	20	5	2550	5	180,119	180,119	100.0	180,119	4.2	180,119	36	180,119	3	180,119	21	180,119	25
3	A3	25	20	5	4050	4	163,405	163,405	100.0	163,405	1.8	163,405	42	163,405	2	163,405	17	163,405	25
4	A4	25	20	5	4050	3	155,796	155,796	100.0	155,796	2.0	155,796	55	155,796	1	155,796	8	155,796	14
5	B1	30	20	10	1600	7	239,080	239,080	100.0	239,080	11.9	240,286	54	239,080	3	239,080	32	239,080	53
6	B2	30	20	10	2600	5	198,048	198,048	100.0	198,048	6.3	198,048	61	198,048	3	198,048	23	198,048	24
7	B3	30	20	10	4000	3	169,372	169,372	100.0	169,372	0.2	169,372	105	169,372	1	169,372	11	169,372	18
8	C1	40	20	20	1800	7	249,448	250,557	100.4	250,557	20.1	251,503	127	250,557	4	250,557	47	250,557	75
9	C2	40	20	20	2600	5	215,020	215,020	100.0	215,020	10.8	215,020	153	215,020	5	215,020	46	215,020	80
10	C3	40	20	20	4150	5	199,346	199,346	100.0	199,346	4.1	199,346	161	200,195	2	199,346	23	199,346	37
11	C4	40	20	20	4150	4	195,366	195,366	100.0	195,366	4.9	195,367	202	195,366	2	195,366	19	195,366	37
12	D1	38	30	8	1700	12	322,530	322,530	100.0	322,707	18.3	322,530	103	322,530	6	322,530	72	322,530	115
13	D2	38	30	8	1700	11	316,709	316,709	100.0	316,711	20.1	318,938	99	317,851	6	316,709	70	316,709	113
14	D3	38	30	8	2750	7	239,479	239,479	100.0	239,482	13.9	239,479	125	239,479	7	239,479	75	239,479	136
15	D4	38	30	8	4075	5	205,832	205,832	100.0	205,832	10.3	205,882	141	205,832	7	205,832	74	205,832	99
16	E1	45	30	15	2650	7	238,880	238,880	100.0	238,880	22.1	238,880	23	238,880	9	238,880	84	238,880	134
17	E2	45	30	15	4300	4	212,263	212,263	100.0	212,802	13.3	212,263	288	212,263	12	212,263	111	212,263	172
18	E3	45	30	15	5225	4	206,659	206,659	100.0	206,659	10.7	206,659	302	208,837	3	206,659	50	206,659	123
19	F1	60	30	30	3000	6	263,173	263,173	100.0	263,174	30.1	265,570	582	264,565	13	263,173	162	263,173	249
20	F2	60	30	30	3000	7	265,213	265,213	100.0	266,643	30.1	265,943	555	266,921	14	265,213	110	265,493	210
21	F3	60	30	30	4400	5	241,120	241,120	100.0	241,487	22.5	241,941	725	254,736	15	241,120	60	241,120	138
22	F4	60	30	30	5500	4	233,861	233,861	100.0	234,377	18.4	235,709	822	234,341	16	233,861	93	233,861	201
23	G1	57	45	12	2700	10	306,305	306,305	100.0	307,274	30.2	306,305	350	306,308	18	306,306	207	306,306	342
24	G2	57	45	12	4300	6	245,441	245,441	100.0	245,441	30.1	245,441	503	251,334	19	245,441	200	245,441	371
25	G3	57	45	12	5300	5	229,507	229,507	100.0	231,202	27.5	230,119	574	231,203	6	231,203	117	229,507	196
26	G4	57	45	12	5300	6	232,521*	232,521	100.0	232,646	30.0	235,251	513	235,410	6	232,521	177	232,521	183
27	G5	57	45	12	6400	5	221,730	221,730	100.0	223,068	27.3	221,730	587	228,833	15	221,730	151	221,730	242
28	G6	57	45	12	8000	4	213,457	213,457	100.0	213,457	15.4	213,457	740	213,644	16	213,457	128	213,457	213
29	H1	68	45	23	4000	6	268,933	268,933	100.0	271,185	35.1	268,933	988	271,300	13	268,933	261	268,933	363
30	H2	68	45	23	5100	5	253,365	253,365	100.0	253,365	29.8	253,365	1140	253,365	18	253,365	323	253,365	398
31	H3	68	45	23	6100	4	247,449	247,449	100.0	247,536	27.9	247,449	1319	250,824	23	247,449	233	247,449	345
32	H4	68	45	23	6100	5	250,221	250,221	100.0	252,370	33.7	250,221	1160	264,133	9	250,221	150	250,221	167
33	H5	68	45	23	7100	4	246,121	246,121	100.0	246,121	17.0	246,121	1326	246,121	12	246,121	126	246,121	188

(*continued on next page*)

Table 1 (*continued*)

No.	Name	*N*	*n*	*m*	*Q*	*K*	Best published	Our best		TV99		OW		Open		*K-tree*		*K-tree_r*	
								Cost	Ratio	Cost	CPU (seconds)	Cost	CPU (seconds)	Cost	CPU (seconds)	Cost	CPU (seconds)	Cost	CPU (seconds)
34	H6	68	45	23	7100	5	249,135	249,135	100.0	249,135	17.0	253,248	1221	249,618	10	249,135	116	249,135	161
35	I1	90	45	45	3000	10	351,607*	**350,246**	100.1	358,625	45.5	351,905	1930	351,179	35	350,437	449	350,435	648
36	I2	90	45	45	4000	7	309,943	309,944	100.0	313,917	45.1	311,649	2445	309,944	44	309,944	456	309,944	542
37	I3	90	45	45	5700	5	294,507*	294,507	101.7	301,186	45.1	294,507	2970	306,523	43	294,507	482	294,507	574
38	I4	90	45	45	5700	6	295,988*	295,988	100.3	297,709	45.1	296,904	2703	299,144	40	295,988	450	295,988	630
39	I5	90	45	45	5700	7	301,226*	301,236	100.0	304,603	45.1	302,529	2468	306,554	29	301,731	335	301,236	504
40	J1	94	75	19	4400	10	335,006*	335,007	101.1	343,659	50.4	336,165	2321	337,588	54	335,007	735	335,007	965
41	J2	94	75	19	5600	8	310,800*	**310,417**	103.3	318,984	50.1	311,077	2617	321,764	55	311,028	685	310,793	946
42	J3	94	75	19	8200	6	279,219*	279,219	102.2	285,483	50.1	281,458	3159	300,776	30	282,095	598	279,306	764
43	J4	94	75	19	6600	7	296,773*	**296,533**	103.5	302,422	50.1	296,773	2916	303,227	48	298,188	618	296,860	859
44	K1	113	75	38	4100	10	394,637*	**394,376**	101.7	407,939	88.0	395,546	4470	402,273	71	394,988	866	394,974	1334
45	K2	113	75	38	5200	8	362,360*	**362,130**	100.8	372,906	88.1	363,512	5331	368,506	72	365,065	870	363,829	1302
46	K3	113	75	38	5200	9	365,693*	365694	100.9	380,632	87.8	367,706	5261	369,733	76	366,734	867	366,246	1204
47	K4	113	75	38	6200	7	349,039*	**348,950**	102.0	361,775	87.9	349,039	5957	351,418	47	351,940	712	351,345	1150
48	L1	150	75	75	4400	10	426,018*	**425,772**	105.6	454,130	112.7	426,660	13,045	428,277	124	425,865	1543	426,401	2366
49	L2	150	75	75	5000	8	402,245*	**401,228**	104.0	431,840	112.7	405,760	15,436	405,931	149	403,910	1533	402,152	2477
50	L3	150	75	75	5000	9	403,886*	**402,720**	102.9	423,930	112.9	403,886	13,830	403,981	124	403,807	1588	404,391	2333
51	L4	150	75	75	6000	7	384,844*	**384,637**	102.3	402,219	112.7	385,133	16,559	407,127	124	388,510	1531	384,999	2287
52	L5	150	75	75	6000	8	388,062*	**387,928**	101.8	406,470	112.9	388,544	16,304	399,644	138	393,899	1508	389,044	2258
53	M1	125	100	25	5200	11	400,861*	**398,593**	110.5	411,204	95.3	402,947	6193	405,649	104	400,429	1235	400,384	1858
54	M2	125	100	25	5200	10	398,909*	**396,917**	112.5	417,882	95.8	398,909	6330	405,678	106	399,625	1232	398,924	1913
55	M3	125	100	25	6200	9	377,353*	**376,309**	110.0	386,249	95.7	379,994	7264	381,708	103	378,464	1226	377,433	1878
56	M4	125	100	25	8000	7	348,624*	**34,8418**	106.3	362,117	95.4	348,624	8830	354,378	110	349,980	1045	349,091	1858
57	N1	150	100	50	5700	11	408,926*	**408,101**	106.4	428,328	113.5	408,926	12,800	411,370	148	408,724	1575	409,531	2468
58	N2	150	100	50	5700	10	409,280*	**408,066**	108.4	430,688	112.7	410,982	130,68	413,768	132	410,909	1686	408,287	2430
59	N3	150	100	50	6600	9	396,168*	**394,338**	106.8	414,785	113.0	396,168	14,211	401,369	131	399,701	1605	394,338	2441
60	N4	150	100	50	6600	10	397,754*	**396,055**	105.3	416,321	113.6	398,207	14,230	410,424	131	398,824	1541	399,029	2298
61	N5	150	100	50	8500	7	376,432*	**373,477**	105.0	389,349	113.1	377,091	16,868	382,050	139	374,985	1464	376,522	2494
62	N6	150	100	50	8500	8	377,665*	**374,691**	104.0	388,976	113.4	377,665	14,864	383,535	130	382,751	1338	374,774	2468
Average							291,107	290,764	101.8	297,288	48.5	291,804	4026	294,990	46	291,704	535	291,160	815
Number of optimal solutions							34	32	–	18	–	23	–	15		30	–	30	–
Difference to our best (%)							0.1	–	–	2.2	–	0.4	–	1.5		0.3	–	0.1	–

Table 2
Results for the second set of problems

No	Name	N	n	m	Q	K	Best published	Our best	Ratio	TV99		OW		Open		*K-tree*		*K-tree_r*	
										Cost	CPU (seconds)	Cost	CPU (seconds)	Cost	CPU (seconds)	Cost	CPU (seconds)	Cost	CPU ((seconds)
1	Eil22_50	21	11	10	6000	3	371	371	100.0	371	5.1	371	28	371	0.2	371	9	371	11
2	Eil22_66	21	14	7	6000	3	366	366	100.0	366	4.8	366	26	366	0.4	366	4	366	6
3	Eil22_80	21	17	4	6000	3	375	375	100.0	375	7.0	375	26	375	0.8	375	7	375	11
4	Eil23_50	22	11	11	4500	2	682	682	100.0	682	2.6	729	32	682	0.8	682	19	682	25
5	Eil23_66	22	15	7	4500	2	649	649	100.0	649	5.5	658	36	649	0.8	649	8	649	10
6	Eil23_80	22	18	4	4500	2	623	623	100.0	623	3.9	675	40	623	1.5	623	14	623	13
7	Eil30_50	29	15	14	4500	2	501	501	100.0	501	3.3	503	96	501	2.3	501	15	501	22
8	Eil30_66	29	20	9	4500	3	537	537	100.0	539	7.4	537	80	537	3.2	537	22	537	44
9	Eil30_80	29	24	5	4500	3	514	514	100.0	522	7.5	514	84	519	1.1	514	44	514	55
10	Eil33_50	32	16	16	8000	3	738	738	100.0	742	16.4	738	107	738	5.3	738	59	738	96
11	Eil33_66	32	22	10	8000	3	750	750	100.0	753	15.8	750	103	750	1.9	750	18	750	27
12	Eil33_80	32	26	6	8000	3	736	736	100.0	761	15.9	755	100	736	1.7	736	30	736	54
13	Eil51_50	50	25	25	8000	3	559	559	100.0	562	40.8	559	271	560	7.6	559	72	559	117
14	Eil51_66	50	34	16	8000	4	548	548	100.0	553	48.5	548	455	551	10.5	548	102	548	148
15	Eil51_80	50	40	10	8000	4	565	565	100.0	574	53.1	570	393	565	13.9	565	107	565	163
16	EilA76_50	75	37	38	140	6	739	739	100.0	756	164.3	739	1422	749	12.4	739	135	739	237
17	EilA76_66	75	50	25	140	7	768	768	100.0	780	148.3	768	1148	772	12.5	768	182	768	259
18	EilA76_80	75	60	15	140	8	781*	781	101.2	833	238.2	811	1490	787	18.6	784	267	781	491
19	EilB76_50	75	37	38	100	8	801	801	100.0	825	240.5	809	1144	806	9.5	804	148	801	245
20	EilB76_66	75	50	25	100	10	873	873	100.0	891	241.0	873	997	877	23.6	873	291	873	412
21	EilB76_80	75	60	15	100	12	919	919	100.0	948	240.7	931	915	927	27.9	919	368	919	609
22	EilC76_50	75	37	38	180	5	713	713	100.0	715	110.5	715	1620	719	23.9	714	151	714	239
23	EilC76_66	75	50	25	180	6	734	734	100.0	745	148.7	734	1352	747	14.6	735	243	734	297
24	EilC76_80	75	60	15	180	7	733*	733	101.2	759	219.4	736	841	736	18.8	737	354	737	451
25	EilD76_50	75	37	38	220	4	690	690	100.0	691	93.7	690	1377	694	21.1	690	183	690	285
26	EilD76_66	75	50	25	220	5	715*	715	100.6	717	89.7	715	2296	736	12.5	715	205	715	291
27	EilD76_80	75	60	15	220	6	694*	694	100.4	710	190.6	694	3056	700	31.2	694	296	694	523
28	EilA101_50	100	50	50	200	4	842*	**834**	102.2	852	213.5	842	5508	858	27.2	844	385	841	679
29	EilA101_66	100	67	33	200	6	846	846	100.0	868	240.6	852	3227	868	23.0	854	284	850	563
30	EilA101_80	100	80	20	200	6	875*	**862**	103.2	900	241.0	875	4610	894	40.5	873	622	866	914
31	EilB101_50	100	50	50	112	7	933*	**925**	102.8	962	241.6	936	9280	944	52.1	934	451	932	740
32	EilB101_66	100	67	33	112	9	998*	**987**	104.4	1040	241.3	998	7789	1020	68.6	1001	583	993	961
33	EilB101_80	100	80	20	112	11	1021*	**1008**	101.2	1060	241.6	1021	2580	1033	47.5	1016	463	1010	819
Average							702.7	701.1	100.5	715.9	114.6	708.7	1592	708.8	16.3	703.3	186	702.2	297
Number of optimal solutions							24	24	–	7	–	14		12	–	20	–	22	–
Difference to our best (%)								–	–	2.1	–	1.1		1.1	–	0.3		0.2	–

The first set of problems was originally proposed by Goetschalckx and Jacobs-Blecha (1989) and is constituted by 62 randomly generated Euclidean problems. Customers' coordinates (x, y) have a uniform distribution, whose values are contained, respectively, in the intervals $[0, 24{,}000]$ and $[0, 32{,}000]$. The depot is centrally located at $(12{,}000, 16{,}000)$. Customers' demand was generated from a normal distribution with a mean of 500 units and a standard deviation of 200 units. The other characteristics of the test problems like the number of customers, the capacity of each vehicle and the number of vehicles used, was also established by those authors. In the calculation of the cost (or distance) c_{ij} between vertex (i,j) we used the same assumptions taken by Toth and Vigo (1997) and Mingozzi et al. (1999) in their exact methods: the Euclidean distance is calculated in double precision, then is multiplied by 10, and after rounded to the nearest integer. The value of final solution is divided by 10 and then rounded to the nearest integer. Toth and Vigo (1999) present their results using an integer matrix and, therefore, their solution value is sometimes lower than the optimum calculated, as stated above, with a higher precision. In these cases, we put in our Table 1 not the original value given by Toth and Vigo (1999), but the optimum value; for all the other problems we reproduce the original value.

The second set of test problems is made up of 33 problems that were constructed by Toth and Vigo (1997) from a set of 11 VRP problems, originally proposed by Eilon et al. (1971). Each of these VRP problems originated three different VRPB problems, each containing an approximate percentage of linehaul customers of 50%, 66% and 80%, respectively. Considering the original number of the customers, the backhaul customer is the last of every two $(2, 4, 6, \ldots)$, three and five for an approximate percentage of linehaul customers of 50%, 66% and 80%, respectively. Customers' demand and the capacity of the vehicles are the same as in the original VRP, and the number of vehicles was established as $K = \max\{K_L, K_B\}$, where K_L and K_B is the minimum number of vehicles required by the sets of linehaul and backhaul customers, respectively. When $K_L < K_B$ the sets of linehaul and backhaul customers are swapped because no VRPB route can contain backhaul customers only. The cost between each pair of customers is defined as the Euclidean distance between them, rounded to the nearest integer.

For a correct evaluation and comparison of the quality of two algorithms the computing time must be taken into account. However, this task is usually very difficult due to the enormous variety of computers available and used by different researchers. A very rough measure of computers performance can be obtained using Dongarra's (2001) tables that present the number (in millions) of floating-point operations per second (Mflop/seconds) executed by each computer, when solving standard linear equations, with LINPACK program. From Dongarra's (2001) report we take that the Pentium III at 500 MHz has a speed of 72.5 Mflop/seconds (assuming a linear increase between the Pentium III at 450 MHz and the Pentium III at 550 MHz, because the Pentium III at 500 MHz is not in the tables), that the Sun Sparc1000 at 50 MHz (used by Osman and Wassan, 2002) has a speed of 10 Mflop/seconds and that the PC 486/33 (used by Toth and Vigo, 1999) has a speed of 1.3 Mflop/seconds. Therefore, we will assume in our comparisons that the computer used by us is 50 times faster than the used by Toth and Vigo (1999) and seven times faster than the used by Osman and Wassan (2002).

In Tables 1 and 2 we present the results obtained by our algorithms for the first and second sets of problems, respectively. All the solutions in these tables have been obtained using the same parameter values in all the algorithms and in both sets of problems (except the columns with our best solutions, as explained later), since our objective was to compare the performance of our algorithms with a given set of parameters. However, we know that some parameter values may perform better with some particular problems or set of problems and worse with others. Therefore we could obtain better average solution values, using the same or lower computing time, by choosing different combinations of parameters, but this is not a proper way of testing the performance of the algorithms, especially for practical purposes. On the other hand, in order to study the variability of the

solutions given by the *TSA-Ktree_r*, due to the use of a random tabu tenure, it was executed five times with both sets of problems, and the best run (and the respective computing time), in terms of average cost, is in Tables 1 and 2, respectively.

The meaning of the head of the columns in Tables 1 and 2, not yet explained, is the following:

- *Best published* is the solution cost taken from Toth and Vigo (1997) or Mingozzi et al. (1999) or Toth and Vigo (1999) or Osman and Wassan (2002). The solutions that are not proven to be optimal are marked with an asterisk.
- *Our best* is the best solution cost that we found, in our experiments, mainly from different runs of the *TSA-K-tree_r*, and also from using different frequencies of the swap moves, but not by allowing a larger number of iterations.
- *Ratio* is our best solution divided by the optimal solution, when it is known, or by the lower bound taken from Mingozzi et al. (1999) or Toth and Vigo (1999), whichever is better, and multiplied by 100.
- *TV99* is the solution cost and the computing time taken from Toth and Vigo (1999).
- *OW* is the solution cost and the computing time given by Osman and Wassan's (2002) algorithm (we use in our comparison their best version, SAH + RTS).

From Table 1 we can conclude that all the three versions of the TSA give better solutions than the heuristic of Toth and Vigo (1999), both in terms of average cost and number of optimal solutions (except the *TSA-open*). However, the average computing time taken by this heuristic is much lower, even than the *TSA-open*, which is very fast.

The *TSA-K-tree* is better than Osman and Wassan's (2002) algorithm in terms of average cost, number of optimal solutions found and also computing time. The *TSA-K-tree_r* is even better but takes more computing time. The best solutions found by Osman and Wassan's (2002) algorithm have an average cost of 291261.8 and 31 of them are optimal. Again, their algorithm is worse than ours in relation to both criteria, and none of their best solutions is better than ours, but 27 are worse.

In relation to the best solutions published, our algorithm found 21 new best solutions (those in bold) and matched all the other solutions except five, but for three of them the difference is only a unit (about 0.0003%), which might be due to some rounding discrepancy. On the other hand, the average ratio of 101.8, that measures the average percentage gap between the lower bounds (or optimal solutions) and our best solutions, shows that our best solutions are close to the optimum.

Table 2 shows that any version of the TSA algorithm performs better than the TV99, but the computing time is substantially higher even for the version *TSA-open* (about seven times). The performance of the *TSA-K-tree_r* is particularly good because it almost matches the minimum average cost and finds 22 optimal solutions.

The average results produced by Osman and Wassan's (2002) algorithm are almost identical to those produced by the *TSA-open*, but taking much more computing time (about 14 times). The other two versions of the TSA produce substantially better results than their algorithm, but the *TSA-K-tree_r* needs a little more computation time. Their best solutions have an average cost of 708.4, and none of them is better than ours, but 16 are worse.

The TSA found five new best solutions and all the known optimal solutions. Considering the average ratio of 100.5, we can say that our solutions values are very close to the optimum.

Considering both Tables 1 and 2, there is no doubt that the *TSA-K-tree* produces much better solutions than the *TSA-open*. This is due to two reasons: the information contained in the pseudo-lower bounds and the diversity created by the ten different initial solutions. In the *TSA-K-tree_r* the search is further diversified, in relation to the *TSA-K-tree*, by the use of a random tabu tenure and due to the exploration of each initial solution twice, but this increases the computing time. However, this time can be reduced by executing the *TSA-K-tree_r* without trial swap moves, which produced (just one run) the following: an average cost of 291278.5, 31 optimal solutions, and the average computing time was 617 seconds, for the first set of problems; an average cost of 704.8, 21 optimal solutions, and the average computing time was 258 seconds, for the second set.

The execution of the *TSA-K-tree_r* five times gave the following results (the computing times are very similar in every run):

(i) for the first set, the average cost has ranged from 291160.5 to 291408.8, and the number of optimal solutions from 30 to 31, and the global average has been 291305.7 for the cost and 30.6 for the number of optimal solutions;
(ii) for the second set, the average cost varied between 702.2 and 702.7, and the number of optimal solutions between 19 and 22, and the global average has been 702.5 for the cost and 20.2 for the number of optimal solutions.

These results show two things: the algorithm is very robust, because the range of variation is low; as expected, the *TSA-K-tree_r* performs better than the *TSA-K-tree*, on average.

Since the *TSA-K-tree* gives much better results than the *TSA-open*, but taking much more computing time, we executed the *TSA-open* during the same time taken by the *TSA-K-tree*, for each test problem (the time given in Tables 1 and 2). For doing this, we used unlimited values of N_1, N_2 and N_3, and we set $M = 70N$, which is a value that still allows several restarts and, in general, avoids cycling. This experiment produced the following results: for the first set of test problems—average cost of 292331.3 and 23 optimal solutions; for the second set of test problems—average cost of 705.3 and 17 optimal solutions. These results show that the *TSA-open* has a good performance, but still substantially inferior, in both criteria, to the *TSA-K-tree*. This proofs that the initial solutions used by the *TSA-K-tree* have a very positive effect in the final solution.

In order to determine the influence of the trial swap moves in the performance of the TSA (*phase II*), the versions *TSA-open* and *TSA-K-tree* have been executed with the following frequencies: no swap moves and every ten, every five, every three, every two and every iteration. The main conclusions are the following: the computing time always increases with the frequency; in general there is a trend of improvement of the average solution cost when the frequency increases, but this does not happen always; the largest gain in the average cost usually occurs between the no execution of trial swap moves and their execution every ten iterations.

The average percentage improvement obtained in *phase III* was 0.16% and 0.07%, for the first set of problems, with the *TSA-open* and *TSA-K-tree*, respectively. For the second set of problems, that improvement was 0.06% and 0.02%, respectively. For most of the individual problems, there is no gain in applying *phase III*, but for a problem, solved by the *TSA-open*, the gain has achieved 6.7%. The computing time taken by this phase has been about 15% and 4% of the total with the *TSA-open* and *TSA-K-tree*, respectively. Although the influence of *phase III* is small on average, it may contribute to find very good solutions for some particular problems.

5. Conclusions

In this paper, we present a new tabu search algorithm for the VRPB that performs better than any other known algorithm. The best performance is achieved by the version that obtains the initial solutions from pseudo-lower bounds, which is an innovative feature used for the first time with the VRPB. This good behaviour is due to the information contained in the pseudo-lower bounds and to the diversity created by the use of several initial solutions. A further diversification was attained by the use of a random tabu tenure with consequent better results.

We found that the US algorithm of Gendreau et al. (1992), adapted by us to the VRPB, plays an important role in the discovery of good solutions. However, since it is rather time consuming, the overall computing time was reduced by using it with a given frequency (not every iteration), but even improving the average quality of the solutions by increasing the number of iterations.

Acknowledgements

This research was supported by Fundação para a Ciência e Tecnologia under the program

PRAXIS XXI, project no. 3/3.1/CEG/ 2661/95. The author is indebted to the anonymous referees for their valuable comments on this article.

References

Brandão, J., 2001. A lower bound based meta-heuristic for the vehicle routing problem. In: Ribeiro, C., Hansen, P. (Eds.), Essays and Surveys in Metaheuristics. Kluwer Academic Publishers, Dordrecht, pp. 151–168.

Brandão, J., 2004. A tabu search algorithm for the open vehicle routing problem. European Journal of Operational Research 157, 552–564.

Casco, D., Golden, B., Wasil, E., 1988. Vehicle routing with backhauls: Models, algorithms and case studies. In: Golden, A., Assad, A. (Eds.), Vehicle Routing: Methods and Studies. Elsevier Science Publishers, Amsterdam, pp. 127–147.

Deif, I., Bodin, L., 1984. Extension of the Clarke and Wright algorithm for solving the vehicle routing with backhauling. In: Kidder, A. (Ed.), Proceedings of the Babson Conference on Software Uses in Transportation and Logistics Management, Babson Park, pp. 75–96.

Dongarra, J., 2001. Performance of various computer using standard linear equations software. Report CS-89-85, University of Tennessee.

Eilon, S., Watson-Gandy, C., Christofides, N., 1971. Vehicle scheduling. In: Distribution Management: Mathematical Modelling and Practical Analysis. Griffin, London (chapter 9).

Fisher, M., 1994a. Optimal solution of vehicle routing problems using minimum k-trees. Operations Research 42 (4), 626–642.

Fisher, M., 1994b. A polynomial algorithm for the degree-constrained minimum k-tree problem. Operations Research 42 (4), 775–779.

Gendreau, M., Hertz, A., Laporte, G., 1992. New insertion and post-optimization procedures for the traveling salesman problem. Operations Research 40 (6), 1086–1094.

Glover, F., Laguna, M., 1997. Tabu Search. Kluwer Academic Publishers, Dordrecht.

Goetschalckx, M., Jacobs-Blecha, C., 1989. The vehicle routing problem with backhauls. European Journal of Operational Research 42, 39–51.

Goetschalckx, M., Jacobs-Blecha, C., 1993. The vehicle routing problem with backhauls: Properties and solution algorithms. Technical Report MHRC-TR.88-13, Georgia Institute of Technology.

Golden, B., Baker, E., Alfaro, J., Schaffer, J., 1985. The vehicle routing problem with backhauling: Two approaches. In: Hammesfahr, R., (Ed.), Proceedings of the Twenty-first Annual Meeting of S.E. TIMS, pp. 90–92.

Mingozzi, A., Giorgi, S., Baldacci, R., 1999. An exact method for the vehicle routing problem with backhauls. Transportation Science 33 (3), 315–329.

Osman, I., Wassan, N., 2002. A reactive tabu search meta-heuristic for the vehicle routing problem with back-hauls. Journal of Scheduling 5 (4), 263–285.

Salhi, S., Nagy, G., 1999. A cluster insertion heuristic for single and multiple depot vehicle routing problems with back-hauling. Journal of the Operational Research Society 50 (10), 1034–1042.

Toth, P., Vigo, D., 1997. An exact algorithm for the vehicle routing problem with backhauls. Transportation Science 31 (4), 372–385.

Toth, P., Vigo, D., 1999. A heuristic algorithm for the symmetric and asymmetric vehicle routing problems with backhauls. European Journal of Operational Research 113, 528–543.