Computers & Operations Research 36 (2009) 2983–2993

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

The double traveling salesman problem with multiple stacks: A variable neighborhood search approach

Ángel Felipe, M. Teresa Ortuño, Gregorio Tirado*

Department of Statistics and Operations Research, Facultad de Matematicas, Universidad Complutense de Madrid, UCM, Plaza de Ciencias 3, 28040 Madrid, Spain

ARTICLE INFO

Available online 8 February 2009

Keywords:
Traveling salesman problem
Variable neighborhood search
LIFO loading

ABSTRACT

The double traveling salesman problem with multiple stacks (DTSPMS) is a vehicle routing problem that consists on finding the minimum total length tours in two separated networks, one for pickups and one for deliveries. A set of orders is given, each one consisting of a pickup location and a delivery location, and it is required to send an item from the former location to the latter one. Repacking is not allowed, but collected items can be packed in several rows in such a way that each row must obey the LIFO principle. In this paper, a variable neighborhood search approach using four new neighborhood structures is presented to solve the problem.

1. Introduction

The double traveling salesman problem with multiple stacks (DTSPMS) was introduced in Petersen and Madsen [22] as one of the main problems of a joint project with a computer software company. This company provides solutions for the transportation and fleet management business and hence it has to deal with very different routing problems that appear in real life. They were interested in finding the best way to handle orders consisting on picking-up items from several locations belonging to one region and delivering them in the corresponding locations of another region that is far apart from the first one. The vehicles used by the company, that are rear-loaded, have a 40-foot pallet container and the items to be delivered are usually standardized Euro Pallets, which fit 3 by 11 on the vehicle container. The driver is not allowed to touch the goods due to security reasons, and as a consequence repacking is not permitted during the whole process. Then the first items to be picked up must be the last ones to be delivered, and since the items fit 3 by 11 on the rear-loaded container, there are three independent loading rows, each of which behaves as an independent last-in-first-out (LIFO) system. In this situation, the company had to decide the picking-up route in the first region, the way to load the items in the vehicle and the delivery route in the second region, keeping in mind that, due to the three independent loading rows of the rear-loaded vehicle, the order in which items are picked-up and the way they are stored restrict the order in which deliveries can be performed. The objective of the company, as usual, was to minimize the final total travel cost, that is proportional to the length of the routes.

This real life problem is formulated as a vehicle routing problem that is called DTSPMS. It adds new elements to the classical vehicle routing problems which have been so much studied during the last decades, but this new problem has not been treated in detail in the literature yet. As previously stated, in the DTSPMS there is a single vehicle in which repacking is not allowed, but the load can be packed in several rows that must obey the LIFO principle, while there are no mutual constraints between two different rows. From now on the rows of the container will be referred as *stacks*. Two independent regions that are supposed to be very far apart are considered, and one item must be picked up in every location of the first region and delivered to the corresponding location of the second region.

Hence the elements of the problem are *two separated networks*, each of them represented by a complete graph, and a set of *orders*, each of which consists of a pickup location in the first network and a delivery location in the second network. Each network has a distinguished node called *depot* that represents the location from which the vehicle departs and arrives after covering its route.

Every location of one of the regions is represented by a node in its corresponding network (pickup or delivery) and every connection between two locations is represented by an edge, and thus every order is associated to one node in the pickup network and another node in the delivery network. The two regions of the problem are supposed to be far apart, and thus some long-haul transport is required to carry the picked-up load from the depot of the first region to the depot of the second one. This transportation is not part of the problem, and thus the only costs are the ones originated by travelled distances in the pickup route and the delivery route.

* Corresponding author. Tel.: +34 913944535; fax: +34 913944606.
E-mail address: gregoriotd@mat.ucm.es (G. Tirado).

0305-0548/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2009.01.015

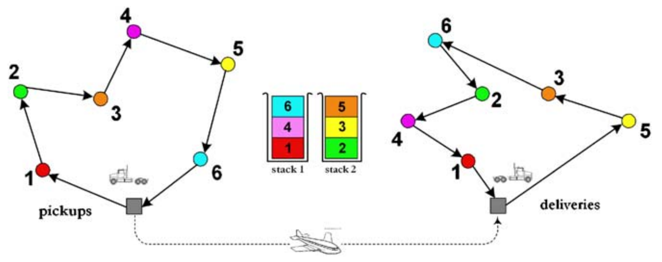

Fig. 1. A feasible solution for the DTSPMS.

For simplicity we will suppose that all stacks have the same capacity, all items are uniform and every order consists of one single item. The number of available stacks in the container and their capacity are parameters of the problem.

Fig. 1 represents the pickup route (left hand side) and the delivery route (right hand side) of a solution of a DTSPMS with six orders. A stack assignment that is compatible with the previous routes is shown as well in this figure, assuming that there are two available stacks with capacity 3.

The objective of the problem is to find a pair of hamiltonian cycles, one for the pickup network and one for the delivery network, such that the sum of travelled distances in both networks is minimized and capacity restrictions and precedence conditions are satisfied.

Hence a feasible solution for an instance of the DTSPMS consists of a *pickup route*, a *delivery route* and a *stack assignment*. Each route determines an ordering of the corresponding set of nodes (pickup and delivery nodes), and the stack assignment gives the stack in which each item must be stored. A stack assignment, on its own, cannot determine what position will occupy every item on its corresponding stack, since that depends on the routes of the solution. Given a stack assignment together with one route (pickup or delivery) the *loading plan* can be designed, providing the exact position of each item in the container.

The DTSPMS generalizes the classical TSP, since any TSP instance can be transformed into a DTSPMS instance in which all delivery locations are placed at the same point, and thus the DTSPMS is an NP-hard problem as well. The precedence constraints associated to the LIFO stacks impose very strong restrictions on both routes of the DTSPMS, making it very hard to characterize feasible solutions in an easy and operative way to solve the problem. This is the main reason why the DTSPMS is such a hard problem to solve, being in fact quite more difficult than the TSP. The largest DTSPMS instances that could be solved to optimality have 12 orders, while TSP instances with thousands of customers could be solved in a reasonable running time. Note also that, given a pickup route and a delivery route found by considering two independent TSPs, the probability of finding a feasible loading plan is extremely low, making it necessary to approach the problem globally.

As previously stated, only very small instances can be solved to optimality, so then heuristics are needed to find good solutions for instances with a realistic size. In Petersen and Madsen [22] two neighborhood structures for the problem are proposed and four heuristic approaches using iterated local search (ILS), tabu search (TS), simulated annealing (SA) and large neighborhood search (LNS) are presented, being the LNS approach the algorithm with the best overall performance. In this paper, we propose a new hybridized variable neighborhood search (VNS) approach to solve the problem using the two existing neighborhood structures and four new ones.

The DTSPMS combines pickup and delivery problems (Cordeau et al. [7], Desaulniers et al. [8]) with the TSP. The traveling salesman problem with pickup and delivery (TSPPD) has been treated during the last years using heuristics (Bianchessi and Righini [1], Hernández-Pérez and Salazar-González [17]) and exact methods (Dumitrescu et al. [10], Hernández-Pérez and Salazar-González [18]); imposing LIFO loading conditions a new problem called the TSPPD with LIFO loading (TSPPDL) is obtained, which presents more similarities with the DTSPMS (with only one stack). Exact approaches (Carrabs et al. [2], Cordeau et al. [6]) and heuristics based on VNS (Carrabs et al. [3], Cassani and Righini [4]) and column generation for the multi-vehicle version (Xu et al. [24]) have been recently proposed to solve the TSPPDL. More complex problems including different loading conditions in two or three dimensions and allowing items to be heterogeneous have also been approached in Doerner et al. [9], Gendreau et al. [11,12] and Iori et al. [19].

The remainder of this paper is organized as follows. Section 2 presents a mathematical model for the problem and Section 3 describes the neighborhood structures that will be used. In Section 4 two methods to generate initial solutions for the problem are presented. Some heuristics to solve the problem using VNS methodology are proposed in Section 5 and detailed computational experience is presented in Section 6. Finally, Section 7 draws some conclusions from this work.

2. Mathematical programming model

Following Petersen and Madsen [22], let $G^1=(V^1,A^1)$, $G^2=(V^2,A^2)$ be two complete graphs that represent, respectively, the pickup network and the delivery network of the problem. For every $\delta \in \{1,2\}$, each edge $(i,j) \in A^\delta$ belonging to graph G^δ has an associated weight c_{ij}^δ that represents the distance or travel cost between nodes i and j. If the number of orders is denoted by n, the node sets are $V^\delta = \{v_0^\delta, v_1^\delta, \ldots, v_n^\delta\}$, $\delta \in \{1,2\}$, where v_0^δ represents the depot and $v_1^\delta, \ldots, v_n^\delta$ are associated to the n orders. Finally, let m be the number of available stacks in the container of the vehicle and Q their maximum capacity.

Let us define set $V_*^\delta = V^\delta \backslash \{v_0^\delta\}$, that is the set of nodes that are associated to orders (the depot is removed) in each graph $\delta \in \{1,2\}$, and set $P = \{1,\ldots,m\}$ containing the m available stacks. A set $D = \{1,\ldots,n\}$ of orders is considered, and thus the item associated to every order $i \in D$ must be picked up at location $v_i^1 \in V_*^1$ of G^1, packed into a stack $p \in P$ and delivered at location $v_i^2 \in V_*^2$ of G^2.

The sets of variables used to model the DTSPMS as a *binary integer programming problem* are the following:

$$x_{ij}^\delta = \begin{cases} 1 & \text{if } j \text{ is visited immediately after } i \text{ in network } \delta \\ 0 & \text{otherwise} \end{cases} \qquad \forall i,j \in V^\delta$$

$$y_{ij}^\delta = \begin{cases} 1 & \text{if } j \text{ is visited after } i \text{ in network } \delta \\ 0 & \text{otherwise} \end{cases} \qquad \forall i,j \in V_*^\delta$$

$$z_{ip} = \begin{cases} 1 & \text{if order } i \text{ is assigned to stack } p \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in D,\ \forall p \in P$$

where $\delta \in \{1,2\}$.

Variables $\{x_{ij}^\delta\}$ define the routes of the solution, variables $\{y_{ij}^\delta\}$ indicate the precedence relations between pickups and deliveries in both networks and are used to ensure that the LIFO principle is obeyed by every stack, and variables $\{z_{ip}\}$ determine the stack assignment.

Hence the model for the DTSPMS is as follows:

$$\min \sum_{\substack{i,j \in V^\delta \\ \delta \in \{1,2\}}} c_{ij}^\delta \cdot x_{ij}^\delta$$

$$\sum_{i \in V^\delta} x_{ij}^\delta = 1 \quad \forall j \in V^\delta,\ \forall \delta \tag{1}$$

$$\sum_{j\in V^\delta} x_{ij}^\delta = 1 \quad \forall i \in V^\delta,\ \forall\delta \tag{2}$$

$$y_{ij}^\delta + y_{ji}^\delta = 1 \quad \forall i,j \in V_*^\delta,\ i\neq j,\ \forall\delta \tag{3}$$

$$y_{ik}^\delta + y_{kj}^\delta \leqslant y_{ij}^\delta + 1 \qquad \forall i,j,k \in V_*^\delta,\ \forall\delta \tag{4}$$

$$x_{ij}^\delta \leqslant y_{ij}^\delta \quad \forall i,j \in V_*^\delta,\ \forall\delta \tag{5}$$

$$y_{ij}^1 + z_{ip} + z_{jp} \leqslant 3 - y_{ij}^2 \quad \forall i,j \in V_*^\delta,\ \forall p \in P \tag{6}$$

$$\sum_{p\in P} z_{ip} = 1 \quad \forall i \in D \tag{7}$$

$$\sum_{i\in D} z_{ip} \leqslant Q \quad \forall p \in P \tag{8}$$

$$x,y,z \in \{0,1\} \tag{9}$$

The cost of a solution is the sum of total travelled distances in both graphs, and the objective of the problem is to minimize this cost. Constraints (1) and (2) are the *flow conservation constraints* and state that each node is visited exactly once in each route. Constraints (3) and (4) ensure that precedence variables $\{y_{ij}^\delta\}$ are defined correctly: for each pair of nodes i,j and for each graph δ, i is visited before j or j is visited before i in route δ, and for each node k if i is visited before k and k is visited before j in route δ, i is necessarily visited before j in route δ (transitivity constraints). Constraints (5) express that variables x_{ij}^δ and y_{ij}^δ must be compatible: if node j is visited immediately after node i in route δ (i.e. $x_{ij}^\delta = 1$), node i precedes node j in route δ (i.e. $y_{ji}^\delta = 1$).

Constraints (6) state that the LIFO principle must be followed in every stack: if two orders i and j are assigned to the same stack and item i is picked up *before* item j in route 1, then item j must be delivered *before* item i in route 2. Constraints (7) and (8) ensure, respectively, that each order is assigned to exactly one stack and that the maximum capacity of the stacks is not exceeded. Finally, constraints (9) express that the variables used in the model are binary.

3. Neighborhood structures

In this section, the two neighborhood structures (route swap, RS and complete swap, CS) introduced in Petersen and Madsen [22] are extended with four new ones, to be used jointly in a VNS approach. In what follows a solution S for the DTSPMS will be represented by (π_1, π_2, λ), where π_1 is associated to route 1 (pickups), π_2 to route 2 (deliveries) and λ to the stack assignment. The set of all feasible solutions will be denoted by X, π_1, π_2 are permutations of D and λ is an application $\lambda : D \longrightarrow P$.

3.1. Route swap

The RS neighborhood of a solution of the DTSPMS contains the solutions that can be obtained by swapping the positions of two consecutive orders in one of the routes of the solution. The stack assignment is not changed, but the other route of the solution (in which the swapping was not performed) may be modified in order to keep the feasibility of the solution.

A RS operation on one route is determined by the first order to be swapped, and thus there are $n-1$ different RS operations on each route. Then the size of RS neighborhood is $O(2(n-1)) = O(n)$.

3.2. Complete swap

The CS neighborhood of a solution of the DTSPMS contains the solutions that can be obtained by swapping the stack positions of two orders that are assigned to *different* stacks. To keep the feasibility of the solution the positions of the two chosen orders in both routes of the solution must be swapped as well.

A CS operation is determined by a pair of orders assigned to different stacks, and then the size of CS neighborhood is $O(n^2)$.

3.3. In-stack swap (ISS)

The *In-stack swap* (ISS) neighborhood of a solution of the DTSPMS contains the solutions that can be obtained by swapping the stack positions of two orders that are assigned to *the same* stack. To do this and keep the solution feasible the positions of these two orders must be swapped in both routes of the solution. The loading plan is modified without changing the stack assignment, since both orders remain assigned to the same stack.

An ISS operation on a solution $S = (\pi_1, \pi_2, \lambda)$ is determined by a pair of orders $u, v \in D$ assigned to the same stack. For every $k \in P$ let $n_k = |\lambda^{-1}(k)|$ be the number or orders assigned to stack k. There are $\sum_{k=1}^m \binom{n_k}{2}$ pairs assigned to the same stack, and thus the number of different ISS operations on S is $O(\sum_{k=1}^m \binom{n_k}{2})$. Since $n_k \approx n/m$, the size of ISS(S) is $O(\sum_{k=1}^m \binom{n_k}{2}) = O(m\binom{n/m}{2}) = O(m(n/m)^2) = O(n^2/m)$.

3.4. Reinsertion

The *reinsertion* (R) neighborhood of a solution of the DTSPMS contains the solutions that can be obtained by moving one order to a different position in both routes of the solution and reassigning it to a different stack.

When a reinsertion operation is performed, the routes and the stack assignment are modified at the same time. One operation of this type if defined by

1. The order $u \in D$ to reinsert.
2. The stack $k \in P$ that will receive order u.
3. The position i of route 1 in which order u will be picked-up.
4. The position j of route 2 in which order u will be delivered.

The stack k into which order u will be loaded must have room for new items and precedence constraints must remain satisfied after moving order u to positions i^* and j^* in both routes of the solution. Once u, k and one of the reinsertion positions have been fixed, the only way to enforce precedence constraints to be satisfied is restricting the possible values of the other reinsertion position: it must belong to an interval Υ whose ends are determined by those orders that are assigned to stack k and visited immediately before and after the fixed reinsertion position.

Fig. 2 shows how to perform a reinsertion operation in which order u is moved from position i_1^u to position i^* in route 1 and reassigned from stack k_1 to stack k_2. Order v occupies position i^* in π_1 and the new route that is obtained after performing the operation is denoted by $\hat{\pi}_1$. Two different cases are considered, depending on the relative order of u and v in π_1.

A reinsertion operation on a solution S is univocally determined by a 4-vector (u, k, i^*, j^*), where $u \in D, k \in P, i^* \in I, j^* \in \Upsilon$, $I = \{1, \ldots, n\}$. We have that $|D| = n$, $|P| = m$ and $|I| = n$, but the size of Υ depends on i^* and S. It is the length of an interval whose ends are the positions of two orders assigned consecutively to the same stack, which coincides, in average, with the averageseparation between two orders assigned consecutively to the same stack, $m-1$. Hence the size of $R(S)$ is $|\{(u,k,i^*,j^*)|\, u \in D, k \in P, i \in I, j^* \in \Upsilon\}| = |D| \cdot |P| \cdot |I| \cdot |\Upsilon| = O(n \cdot m \cdot n(m-1)) = O(n^2m^2)$.

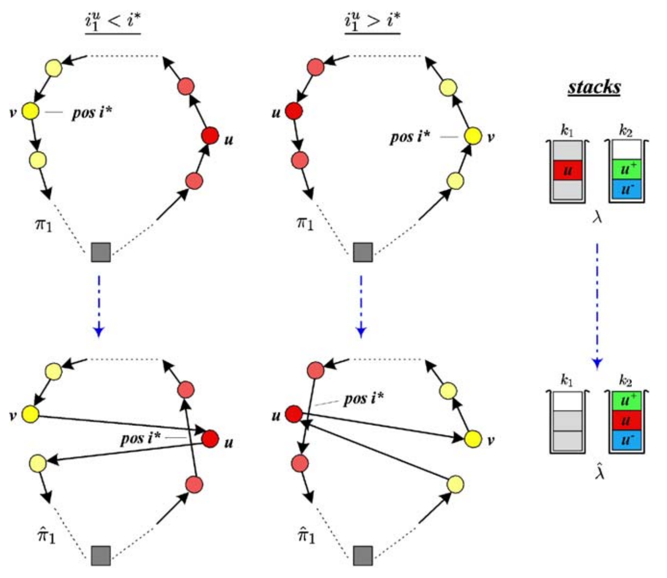

Fig. 2. A reinsertion operation.

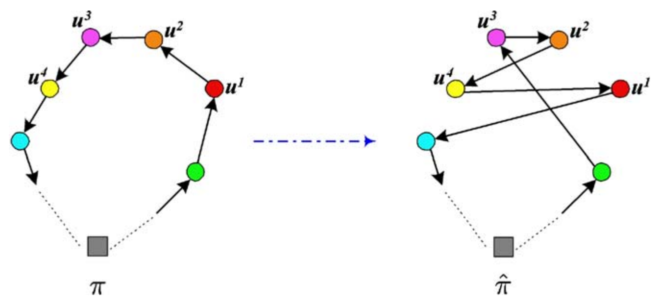

Fig. 3. An r-route permutation operation.

In most real life instances the maximum capacity of the rows of the container of the vehicle is tight ($mQ = n$), causing all stacks to be full in every feasible solution. In that case the loading part of a solution cannot be modified by any reinsertion operation, since none of the stacks have room for new items, making the operator much less powerful. To avoid this problem some extra capacity may be used to let the operator work properly if the maximum capacity is tight. The use of extra capacity introduces intermediate non-feasible solutions in the search, that are fixed by using a simple recolocation heuristic to move infeasible orders to stacks with room for new items.

3.5. *r*-Route permutation

The *r-route permutation* (r-RP) neighborhood of a solution of the DTSPMS contains the solutions that can be obtained by permuting r orders that are assigned to different stacks ($r \leqslant m$) and visited consecutively in one route, while the other route and the stack assignment remain unchanged. The new solution is feasible since the r orders to be permuted are assigned to different stacks and thus there are no precedence constraints between them.

Fig. 3 shows how an r-RP operation involving orders $\{u^1, u^2, u^3, u^4\}$ is performed on a route π.

An r-RP operation on route $\delta \in \{1, 2\}$ is determined by the first order u^1 to be permuted, since the rest are the $r-1$ orders that follow u^1 in route δ. There are $r!$ possible permutations and at most $n-r+1$ candidates for u^1, and thus the size of r-RP(S) is $O(2r!(n-r+1)) = O(nr!)$.

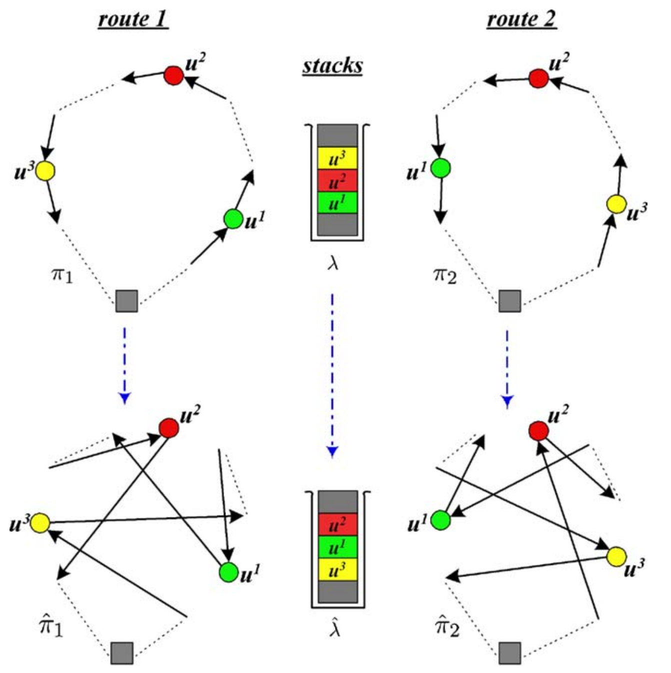

Fig. 4. An r-stack permutation operation.

3.6. Stack permutation

The *r-stack permutation* (r-SP) neighborhood of a solution of the DTSPMS contains the solutions that can be obtained by permuting r orders that are loaded consecutively into the same stack. The loading plan is modified but the stack assignment remains untouched, since chosen orders remain assigned to the same stack.

Fig. 4 shows how an r-SP operation involving orders $\{u^1, u^2, u^3\}$ is performed.

An r-SP operation is determined by a permutation and a set of r orders assigned consecutively to one stack. The number of sets of r orders assigned consecutively to stack k is $n_k - r + 1$, where n_k is the size of stack k. The average size of a stack is n/m and thus the size of r-SP(S) is $O(m \cdot (n/m - r + 1) \cdot r!) = O((n - mr + m)r!) = O(nr!)$.

The size of r-SP neighborhood grows exponentially with r and, thus, only small values of r can be considered, but the possibility of dividing the corresponding stack in disjoint pieces of r consecutive orders and performing an r-SP operation independently in each of the pieces arises naturally. The *r-complete stack permutation* (r-CSP) neighborhood of a solution of the DTSPMS contains the solutions that can be obtained by choosing a stack k and performing an r-SP operation in every set of r consecutive orders assigned to k. An r-CSP operation is determined by a stack k and s permutations, where $s = \lfloor t/r \rfloor$ and t is the size of stack k, and the average size of a stack is $O(n/m)$. Hence the size of $r-\text{CSP}(S)$ is $O(m(r!)^s) = O(m(r!)^{t/r}) = O(m(r!)^{n/mr})$.

4. Generation of initial solutions

An initial solution for the DTSPMS can be generated in a constructive way, following some operations than may be deterministic or random, or transforming a solution of a particular case of the problem that can be solved easily. Two methods to generate initial solutions based on these two ideas are presented next.

4.1. Solving a DTSPMS with a single stack

Definition 1. DTSPSS is a particular case of the DTSPMS in which the number or available stacks is $m = 1$. Let (P) be an instance of the

DTSPMS. The instance (P') that equals (P) but has only one available stack is called the DTSPSS *instance associated to* (P).

Let *SS* be a feasible solution for the DTSPSS. The pickup and delivery routes of *SS* are exactly opposites, since every order must be assigned to the only available stack, and thus the whole load of the vehicle must verify the LIFO principle. Hence, one of the routes univocally determines the other, and the stack assignment is no longer needed as part of the solution. Then, a solution for the DTSPSS consists of a pickup route π_1 and a delivery route π_2 that are opposites, and it is determined by one of them alone.

Let (P) be an instance of the DTSPMS as defined in Section 2 and let (P') be the DTSPSS instance associated to (P). Instance (P') reduces to a TSP instance with set of nodes D in which, for every $i,j \in D$, the distance between node i and node j is the sum of the distances between the localizations of orders i and j in the pickup and delivery graphs that (P) defines. That is, the cost matrix of the TSP instance is defined as

$$d_{TSP}(i,j) = c^1_{ij} + c^2_{ij} \quad \forall i,j \in D, \;\; i \neq j.$$

The solution of this TSP determines one route of the solution of (P'), and the other route is exactly opposite. The cost of this solution in (P') and in the associated TSP instance is the same, since the sum of distances on both graphs is considered in both problems.

A solution $SS = (\pi_1, \pi_2)$ of (P') can be easily transformed in a solution $S = (\pi_1, \pi_2, \lambda)$ of (P) adding any stack assignment λ that does not exceed the maximum capacity of the stacks, since precedence relations will always hold due to the fact that π_2 is exactly opposite to π_1. Thus, solving a standard TSP a solution *SS* for the DTSPSS can be obtained, and a feasible solution *S* for the DTSPMS can be constructed from *SS* without extra cost.

Hence we conclude that an initial solution for the DTSPMS can be obtained just solving a TSP instance, that is still NP-hard but easier to solve than the DTSPMS instance. In this work, we used the savings algorithm proposed in Clarke and Wright [5] followed by a 2-opt interchange algorithm to solve the associated TSP and obtain initial solutions for the DTSPMS.

4.2. Randomized directed generation

An initial solution $S = (\pi_1, \pi_2, \lambda)$ for the DTSPMS can be randomly generated easily, directing the construction process to guarantee the feasibility of the obtained solution as follows.

1. The pickup route π_1 is generated randomly, choosing a permutation of the n orders that define the problem at random.
2. Once π_1 is constructed, the stack assignment λ can be generated in different ways:
   (a) The stack that every order is assigned to is chosen at random, discarding those stacks that have reached their maximum capacity due to previous assignments of orders.
   (b) Following the ordering of orders determined by route π_1, assign alternatively every order to a different stack: the order picked up in position i is assigned to stack $(i \bmod m) + 1$ and, thus, $\lambda(\pi_1(i)) = (i \bmod m) + 1$.
   (c) Also following the ordering determined by route π_1, orders are assigned to the same stack until its maximum capacity is reached; when the stack is full the next stack is considered and the assignments continue using the new stack. Then the order that is picked up in position i is assigned to stack $\lfloor i/Q \rfloor + 1$ and, thus, $\lambda(\pi_1(i)) = \lfloor i/Q \rfloor + 1$.
3. Assuming that π_1 and λ are constructed, a delivery route π_2 that is compatible with them, meaning that solution (π_1, π_2, λ) is feasible, can be generated in different ways:
   (a) Route π_2 is exactly opposite to π_1. If this is the case, π_1 and π_2 are compatible regardless of which stack assignment is considered.
   (b) In route π_2 the orders that are assigned to the same stack are delivered consecutively: first the orders assigned to stack 1 are delivered following the opposite picking order, after that the orders assigned to stack 2 are delivered, and so on until the last stack is considered.
   (c) Route π_2 is constructed step by step, choosing at every stage which order (placed on the head of the stacks) is the next order to be delivered. At every step there will be at most m available orders to deliver, and the choice among them can be performed at random or following a *greedy* criterion: choose the nearest order, the farthest one, the order assigned to the largest stack, etc.

5. VNS algorithms

VNS is a metaheuristic that was proposed in Hansen and Mladenovic [13], Mladenovic [20] and Mladenovic and Hansen [21]. It is based in the use of different neighborhood structures to perform the local search, changing from one to another every time a local optimum is found. A local optimum with respect to one neighborhood structure is not necessarily a local optimum with respect to another one, but a global optimum is always a local optimum with respect to any neighborhood structure. Hence, using different neighborhood structures it is more likely to find the global optimum or to get better local optima, and that is the reason why this systematic change of neighborhood structure used in VNS works so well.

The main idea behind VNS is extremely simple and it has produced very good results, causing a very fast development of this metaheuristic and being applied to very different problems in the literature. Depending on the size and nature of the problem to be solved and on the available resources, new features have been introduced to adapt VNS to match new requirements (see Hansen and Mladenovic [14–16]), trying to keep the simplicity of the basic structure.

As far as we are concerned, the best results for real sized instances of the DTSPMS have been obtained in Petersen and Madsen [22] using a LNS (see Saw [23]) approach. This heuristic is based on the removal of a large subset of orders from the current solution and their iterative reinsertion to obtain a new feasible solution. Different criteria are used to choose the orders to be removed and the strategy to follow for their reinsertion. This remove-reinsert procedure modifies at the same time the routing part and the loading part of the solutions, being able to guide the search process without using any other operator. In our approach we use different neighborhood structures to allow the algorithm modify both loading and routing parts of the solutions, introducing diversification into the search process at the same time and being able to reach unexplored regions of the solution space.

In the next two sections the most important standard VNS algorithms (VND and GVNS) are applied to the DTSPMS, and in the third section a new algorithm introducing a restart mechanism, tabu lists and other features into the general variable neighborhood search (GVNS) standard strategy is presented.

5.1. Variable neighborhood descent (VND)

VND (Mladenovic and Hansen [21]) is the most simple VNS algorithm. A set of neighborhood structures is considered, and the algorithm moves from one to another when a local optimum is found, the search being restarted from the first structure every time a better solution is found. The algorithm stops when an improving solution cannot be found in any of the considered neighborhood structures, and thus the final solution is always a local optimum with respect to all of them. Algorithm 1 shows the application of VND to the DTSPMS.

Algorithm 1. VND.

```
Input
  • $S$: Initial solution.
  • $\Delta=\{\Delta_k, k=1,\dots,n_\Delta\}$: Set of neighborhood structures from
  {RS, CS, ISS, R, $r$-RP, $r$-CSP}.
Output
  • $S$: Best known solution.
Pseudocode
  1. Initialization: Do $k=1$.
  2. Search start: Do $\hat{S}=S$, $improve=F$.
  3. Local search: Find the best solution $\bar{S}\in\Delta_k(\hat{S})$ belonging
  to the $k$th neighborhood of $\hat{S}$.
  4. If $z(\bar{S})<z(\hat{S})$, do $\hat{S}=\bar{S}$, $improve=T$ and go back to step 3.
  5. Change of neighborhood structure:
    • If $improve$ do $k=1$.
    • Otherwise do $k=k+1$.
  6. Stopping condition:
    • If $k\leqslant n_\Delta$ go back to step 2.
    • If $k>n_\Delta$ and $improve$, do $k=1$ and go back to step 2.
    Otherwise, END: the best solution found is $S$.
```

Algorithm VND is extremely fast, but it gets trapped in a solution that, although it is a local optimum with respect to every neighborhood structure, it may be very far away from the global optimum. The GVNS algorithm presented next introduces a mechanism into VND to avoid this problem.

5.2. General variable neighborhood search (GVNS)

GVNS (Mladenovic and Hansen [21]) is a more complex algorithm based on VNS philosophy that uses VND as a subprocedure. It considers two sets of neighborhood structures, one to perform a local search using VND and another one to perturb the solution every time a local optimum is found. Perturbation is very important because it allows the consideration of deteriorating solutions that make the algorithm not to get trapped in the local optimum found during the last iterations.

Perturbation is performed choosing at random a solution belonging to the corresponding neighborhood of the current solution, and the search process is continued from that new solution. This perturbation operation may be performed more than once consecutively on the current solution to get farther from the last local optimum and diversify the search process, being this controlled by a parameter, called *shake*, that determines the number of perturbation operations to be made. Algorithm 2 shows the application of GVNS to the DTSPMS.

Algorithm 2. GVNS.

```
Input
    • $S$: Initial solution.
    • $\Delta=\{\Delta_k, k=1,\dots,n_\Delta\}$, $\Omega=\{\Omega_k, k=1,\dots,n_\Omega\}$: Two finite
    sets of neighborhood structures belonging to {RS, CS, ISS,
    R, $r$-RP, $r$-CSP}.
    • $n^*$: Number of iterations.
    • $shake$: Number of random operations that are performed
    for perturbation.
Output
    • $S^*$: Best solution found.
Algorithms used
    • VND.
Pseudocode
  1. Initialization: Do $S^*=S$, $n=0$.
  2. Restart $\Omega$: Do $k=1$ and $improve=F$.
  3. Perturbation with $\Omega$:
    3.1 Do $i=0$, $\bar{S}=S$.
    3.2 Choose $\hat{S}\in\Omega_k(\bar{S})$ at random.
    3.3 Do $i=i+1$, $\bar{S}=\hat{S}$.
    3.4 If $i<shake$ go back to step 3.2.
  4. Local search: Call $\bar{S}=\text{VND}(\hat{S},\Delta)$ to get a local optimum.
  5. Local update: If $z(\bar{S})<z(S)$, do $S=\bar{S}$, $improve=T$.
  6. Best solution update: If $z(\bar{S})<z(S^*)$, do $S^*=\bar{S}$.
  7. Change of neighborhood:
    • If $improve$ go back to step 2.
    • Otherwise do $k=k+1$.
  8. Next neighborhood:
    • If $k\leqslant n_\Omega$ go back to step 3.
    • Otherwise do $n=n+1$.
  9. Stopping condition:
    • If $n<n^*$ go back to step 2.
    • Otherwise: END. The best solution found is $S^*$.
```

GVNS is a standard VNS algorithm that performs quite well when applied to the DTSPMS. In the next section, a new hybridized variable neighborhood search (HVNS) algorithm that uses GVNS as a subprocedure is presented. This algorithm introduces some new features and outperforms standard VNS algorithms, as it will be shown in Section 6.

5.3. Hybridized variable neighborhood search (HVNS)

HVNS is a VNS algorithm designed specifically for the DTSPMS. It uses GVNS to improve the current solution, but other interesting features are added to the VNS basic scheme: some iterations of the algorithm are applied to different initial solutions and then the algorithm is restarted from the best of these improved solutions; this last phase of the algorithm performed on the best improved solution will be referred to as *intensification*; different orderings of the operators used in the VND may be considered, and they may be randomly chosen according to some given probabilities; when the current solution could not be improved after many iterations or it is too far from the best known solution the search process is restarted; and finally tabu lists associated to each operator are used in order not to repeat or undo moves performed in the last iterations.

A brief description of HVNS algorithm is presented next.

```
HVNS:

1. Generate $m_I$ random initial solutions $S_i$, $i=1,\dots,m_I$.
2. $S_i^* := \text{HS}(n_I^*, S_i)\ \forall i=1,\dots,m_I$.
3. $S^* := S_j$, **where** $cost(S_j)\leqslant cost(S_i)\forall i\neq j$.
4. $S^* := \text{HS}(n^*, S^*)$, **return** $S^*$.
HS($k$,$S$):

1. Perform one iteration of a standard GVNS on $S$ keeping a tabu
   list for each operator.
2. **If** the current solution was not improved during the last iterations
   or it is much worse than $\bar{S}$ **then** restart the search from $\bar{S}$ after
   performing some perturbation operations on it.
3. **If** maximum number of iterations $k$ is reached **then** return $\bar{S}$. **Else**
   go back to step 2.
```

Subroutine HS(k,S) improves the given solution S during k iterations using standard GVNS algorithm as part of the search process. This subroutine is called from HVNS algorithm for every considered initial solution, and then it is called again to perform a final search from the best solution found so far. Both algorithms are described in full detail next.

Algorithm 3. HVNS.

```
Input
  • $\Delta = \{\Delta_k, k = 1, \dots, n_\Delta\}$, $\Omega = \{\Omega_k, k = 1, \dots, n_\Omega\}$: Two finite sets of neighborhood structures belonging to {RS, CS, ISS, R, r-RP, r-CSP}.
  • shake: Number of random operations that are performed for perturbation.
  • $m_I$: Number of initial solutions to be generated.
  • $n_I^*$: Number of iterations to be performed on each initial solution.
  • $n^*$: Number of iterations to be performed on the best solution found in the initial phase.
  • $\theta$: Number of different orderings of $\{\Delta_k\}$ that are considered to perform the local search (VND).
  • $R = \{R_1, \dots, R_\theta\}$: Set containing $\theta$ permutations of $\{1, \dots, e_\Delta\}$.
  • $P = \{P_1, \dots, P_\theta\}$: Probability of choosing every ordering $\{R_1, \dots, R_\theta\}$.
  • itmax: Maximum number of consecutive iterations without improving the current solution.
  • pcost: Maximum percentage of cost deviation with respect to the best known solution.
Output
  • $S^*$: Best solution found.
Algorithms used
  • HS
  • GVNS
Pseudocode
  1. Do $i = 1$.
  2. Initial solution: Generate an initial solution $S$.
  3. If $i = 1$ do $S^* = S$.
  4. Search: $S = \mathrm{HS}(n_I^*, S)$.
  5. Update: If $z(S) < z(S^*)$ do $S^* = S$.
  6. Another initial solution: If $i < n_I$ do $i = i + 1$ and go back to step 2.
  7. Intensification: Do $S = S^*$ and $S^* = \mathrm{HS}(n^*, S)$.
  8. End: The best known solution is $S^*$.
```

Algorithm 4. HS.

```
Input
  • iter: Number of iterations to be performed.
  • S: Initial solution.
Output
  • $S^*$: Best solution found.
Pseudocode
  1. Do $it_m = 0$, $it = 0$.
  2. Initial improvement: Call $S = \mathrm{VND}(S, \Delta)$ to get a local optimum without perturbation. If $z(S) < z(S^*)$ do $S^* = S$.
  3. Ordering: Choose at random a permutation $R_j \in R$ according to the probabilities given by $P$. The neighborhood structures belonging to $\Delta$ will follow the ordering given by $R_j$.
  4. General Search: Call $S = \mathrm{GVNS}(S, \Delta, \Omega, 1, shake)$. Do $it = it + 1$.
  5. If $S$ was not improved during the last GVNS iteration, do $it_m = it_m + 1$.
  6. If $it_m > itmax$ or $100(z(S)/z(S^*) - 1) > pcost$, do $it_m = 0$, $S = S^*$ and perform shake random operations on $S$ using operators in $\Delta$.
  7. Stopping condition:
    • If $it < iter$, go back to step 3.
    • Otherwise: END. Best known solution is $S^*$.
```

6. Implementation details and computational results

The three proposed algorithms (VND, GVNS and HVNS) have been implemented in Fortran 95 and run on a 1600 MHz processor. Unless otherwise stated, the sets of neighborhood structures used are $\Delta = \{\mathrm{RS}, \mathrm{CS}, \mathrm{ISS}, \mathrm{R}, 3\text{-RP}, 4\text{-CSP}\}$ for local search and $\Delta = \{\mathrm{RS}, \mathrm{CS}, \mathrm{ISS}, \mathrm{R}\}$ for perturbation. The initial solution for algorithms VND and GVNS and the first initial solution for algorithm HVNS are obtained solving the associated DTSPSS, while the rest of initial solutions considered by HVNS are obtained by randomized directed generation.

Algorithm VND stops when the current solution is a local optimum with respect to every neighborhood structure, and the running time needed to solve any instance considered in this work is negligible. On the other hand, the stopping condition for algorithms GVNS and HVNS is the number of iterations; we will perform computational tests for a fixed running time (wall clock time), and thus for each particular case the number of iterations to be run is calculated to match the corresponding running time.

VND heuristic is deterministic, but that is not the case of GVNS and HVNS, in which randomness is introduced to diversify the search process. Because of that, all results presented in this section obtained by the latter algorithms are averages over three runs performed with different seeding.

The term *quality* of a solution to an instance will refer to the deviation of that solution from the best known solution, and it is calculated by dividing the cost of the given solution by the cost of the best known solution. The percentage deviation will be called *gap* and calculated as $100(quality - 1)$.

Three sets of 10 instances with 33, 66 and 132 orders, called *C33*, *C66* and *C132*, respectively, were randomly generated for parameter setting. For evaluation of proposed algorithms we used three sets of 20 instances with 12, 33 and 66 orders that were taken from Petersen and Madsen [22] and we also generated one more set of 20 larger instances with 132 orders called *T132*. Pick-up and delivery locations of all instances were randomly generated in a 100×100 square, while the depot is placed in the center of the square at (50,50) and the number of available stacks is 3. The travel cost to go from one node to another is the Euclidean distance between the two corresponding locations rounded to the nearest integer, in accordance with the conventions from TSPLIB. Test instances can be downloaded from Petersen's web, http://www.imm.dtu.dk/~hlp, and from http://www.espaciotd.jazztel.es/dtspmsEn.htm.

Next we present some computational tests performed on different sized instances (sets *C33*, *C66* and *C132*) to choose the final sets of parameters to be used, and in the following sections the results obtained for evaluation instances using these parameters are reported.

6.1. Parameter setting

The parameters that must be determined for the HVNS algorithm are the following: number of random operations for perturbation (*shake*), number of initial solutions (m_I), percentage of running time dedicated to intensification on the best initial solution, orderings of the operators, maximum number of iterations without improving current solution (*itmax*) and maximum deviation cost (*pcost*). Some figures showing the calibration results on sets *C33*, *C66* and *C132*, used to determine the best set of parameters, are presented next; in all these figures, running time (in seconds) is represented in the x-axis and average percentage of deviation from the best solution is represented in the y-axis. Note that, in order to suitably represent the data, the scales used for the y-axis are different for each figure.

Fig. 5 shows the results obtained for different values of $shake = 0, 1, 2, 3$ for the three sets of 10 instances considered for calibration. It can be observed that not allowing perturbation operations ($shake = 0$) leads to the worst results in all cases, and there are not significative differences between performing 1, 2 or 3 perturbation operations at each iteration. Hence we conclude that ascending moves must be allowed, and we decide to make an intermediate choice and use $shake = 2$.

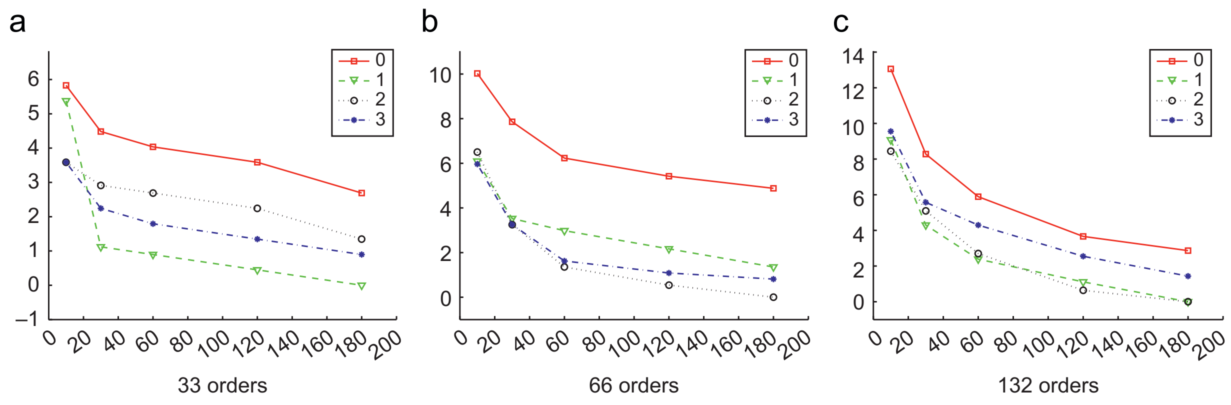

Fig. 5. Results with *shake* = 0, 1, 2, 3.

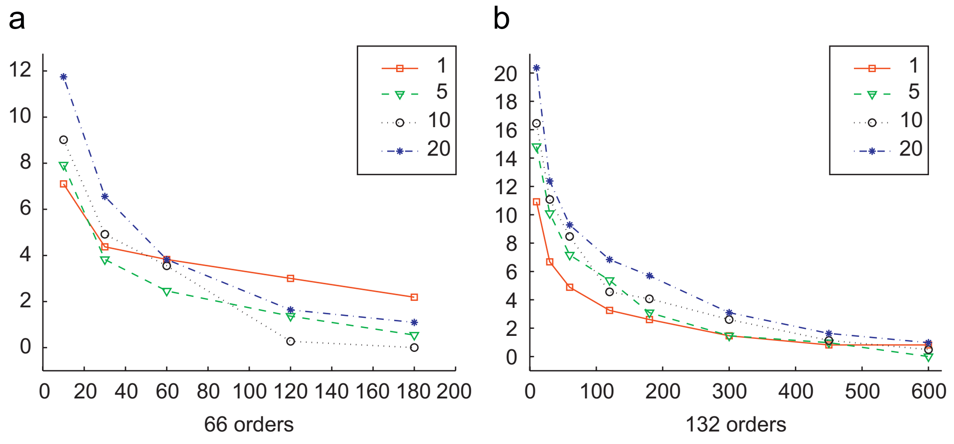

Fig. 6. Results with different numbers of initial solutions.

In Fig. 6 the results obtained for the 66-order sized set 6(a) and 132-order sized set 6(b) using 1, 5, 10 or 20 initial solutions are presented. For the first set, with 10 seconds of running time it can be observed that the use of one only initial solution is the best choice, and the solutions become worse every time the number of initial solutions is increased. If the available running time is increased a bit, the use of five initial solutions produces better results than using only one, and with about 2 minutes or more the use of 10 initial solutions is the best choice, then 5, 20 and finally 1, that yields the worst average results. Thus it can be concluded that the number of initial solutions must be increased with running time. The 132-order sized set shows a similar behavior but now the use of several initial solutions does not become useful until enough running time is considered, indicating that the number of initial solutions can be rapidly increased for not too large instances but it should be increased more slowly for larger ones. Then the decision we make is to determine the number of initial solutions independently for each case depending on the total running time available and on the size of the instances to be solved.

Fig. 7 shows the results obtained for the 66-order sized set using 0%, 10%, 20% or 50% of available running time for intensification on the best initial solution. The worst average results are obtained not using intensification and there are not significative differences between the other three choices. Similar results are obtained for the other instance sizes and then we conclude that intensification is useful. Hence, as in the case of the ascending moves, we make an intermediate choice and dedicate 20% of total running time to intensification.

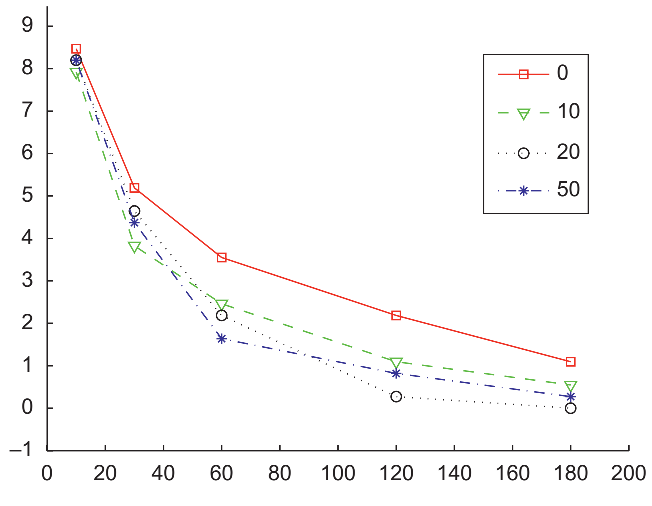

Fig. 7. Amount of intensification.

Several orderings for the operators, with different associated probabilities, were also tested for calibration instances and significative differences were not observed, so then we decide to choose the usual order for neighborhood structures, in increasing size order: 3-RP, RS, ISS, CS, R, 4-CS. The maximum number of iterations to be allowed without improving current solution was effective for the considered instances if it was between 5 and 20, so then we choose *itmax* = 10. And, finally, the algorithm showed also to be quite robust with respect to the maximum deviation cost allowed, that we set to *pcost* = 30%.

6.2. Instances with known optimal solution

The smallest test instances considered in this work have a size of 12 orders because these are the largest instances that could be solved to optimality using commercial packages, and thus the results obtained by the heuristics for these instances can be compared with the optimal solutions. A set of 20 instances of such size is generated from the 33-order set by dropping the last 21 orders of each instance.

The best heuristic results obtained so far for these instances were provided in Petersen and Madsen [22] using a LNS approach, that offered optimal solutions for all instances in 10 seconds.

The solutions obtained using VND are on average within 12–14% of the optimal solution, with a negligible running time. The optimal solution cannot be achieved for any instance.

Using algorithm HVNS the results are much better and the average gap of the solutions is around 0.2% with a running time of only 1 second per instance. For 10 out of the 20 instances the optimal solution is achieved in the 3 performed runs, while for 18 out of 20 instances the optimal solution is achieved in 2 of the 3 performed runs. Only for two instances the optimal solution is found in none of the runs, but in those cases the gap is always less than 1%. With a running time of 5 seconds the optimal solution for all instances is achieved in all runs. Finally, using GVNS similar results are obtained.

Consequently we conclude that for small-sized instances algorithms HVNS and GVNS behave similarly and provide the optimal solution for all instances.

6.3. Instances with 33 orders

Two sets of 10 instances with 33 orders, called Set0 and Set1, are considered now to test the heuristics. Set0 is formed by instances R00–R09 and Set1 by R10–R19, and the problem to be solved, as usual, is the 3-stack DTSPMS.

In Table 1 the results obtained using VND, GVNS and HVNS are compared to the best known results for these instances, that were obtained using a LNS approach (clearly outperforming TS, SA and ILS, see Petersen and Madsen [22]). The first column contains the names of the instances; in the second column a lower bound for each instance is given, calculated by solving the *n*-stack problem obtained by dropping the precedence constraints; the third column shows the cost of best known solutions, taken from Petersen and Madsen [22] and obtained running LNS algorithm for about 3 hours; in the fourth column the quality of results obtained using VND heuristic are reported; and in the next six columns the quality of results for LNS, GVNS and HVNS heuristics with a running time of 10 seconds and 3 minutes are presented. Finally, in the last two rows we indicate the average quality of results on both sets of instances.

Solutions offered by VND are quite far away from the best known (more than 50%) but note that the running time required by the algorithm is negligible. With 10 seconds of running time both VNS approaches outperform LNS, obtaining solutions that are less than 1% away from the best known using HVNS. With a longer running time of 3 minutes the three algorithms provide solutions that are very close to the best known, but HVNS approach is again the one with the best performance, reaching the best known solutions for 18 out of 20 instances in all runs.

Finally, it is very important to point out that the solutions obtained running HVNS for a few hours match exactly the best known solutions obtained by LNS for *all* 33-order instances; this fact suggests that all these best known solutions obtained independently by both algorithms are, in fact, optimal.

Table 1
Results on instances with 33 orders.

Problem	LB	Best	VND	10 seconds			3 minutes		
				LNS	GVNS	HVNS	LNS	GVNS	HVNS
R00	911	1063	1.493	1.04	1.008	1.008	1.01	1.000	1.000
R01	875	1032	1.712	1.04	1.010	1.008	1.01	1.000	1.000
R02	935	1065	1.482	1.04	1.012	1.008	1.01	1.003	1.000
R03	961	1100	1.665	1.06	1.009	1.000	1.01	1.000	1.000
R04	937	1052	1.594	1.05	1.017	1.005	1.02	1.005	1.000
R05	900	1008	1.480	1.03	1.035	1.022	1.01	1.011	1.000
R06	998	1110	1.543	1.06	1.029	1.000	1.02	1.001	1.000
R07	963	1105	1.540	1.05	1.010	1.004	1.01	1.000	1.000
R08	978	1109	1.573	1.04	1.014	1.000	1.01	1.000	1.000
R09	976	1091	1.492	1.04	1.011	1.000	1.01	1.003	1.000
R10	901	1016	1.594	1.05	1.003	1.000	1.00	1.000	1.000
R11	892	1001	1.469	1.06	1.023	1.000	1.01	1.000	1.000
R12	984	1109	1.549	1.04	1.010	1.002	1.01	1.000	1.000
R13	956	1084	1.493	1.04	1.018	1.000	1.01	1.003	1.000
R14	879	1034	1.551	1.03	1.015	1.017	1.00	1.000	1.000
R15	985	1142	1.587	1.04	1.013	1.014	1.01	1.001	1.000
R16	967	1093	1.551	1.02	1.014	1.000	1.00	1.000	1.000
R17	946	1073	1.542	1.04	1.023	1.009	1.00	1.001	1.000
R18	1008	1118	1.533	1.05	1.028	1.028	1.01	1.007	1.007
R19	938	1089	1.522	1.03	1.010	1.006	1.01	1.003	1.002
Avg. Set0			1.557	1.04	1.015	1.005	1.01	1.002	1.000
Avg. Set1			1.539	1.04	1.016	1.008	1.01	1.001	1.001

Table 2
Results on instances with 66 orders.

Problem	LB	Best	VND	10 seconds			3 minutes			
				LNS	GVNS	HVNS	LNS	GVNS	HVNS	Best*
R00–66	1237	1594	1.700	1.19	1.103	1.038	1.07	1.047	1.031	1597
R01–66	1257	1600	1.641	1.20	1.073	1.064	1.08	1.049	1.043	1600
R02–66	1295	1576	1.665	1.20	1.104	1.077	1.12	1.027	1.062	1576
R03–66	1290	1631	1.732	1.14	1.089	1.059	1.06	1.033	1.025	**1630**
R04–66	1295	1611	1.669	1.18	1.057	1.077	1.09	1.048	1.029	1626
R05–66	1204	1528	1.673	1.18	1.106	1.069	1.07	1.032	1.033	**1524**
R06–66	1294	1651	1.687	1.17	1.087	1.105	1.07	1.028	1.036	1651
R07–66	1307	1653	1.616	1.17	1.055	1.064	1.08	1.040	1.010	1655
R08–66	1297	1607	1.598	1.18	1.083	1.111	1.07	1.048	1.034	1613
R09–66	1276	1598	1.591	1.18	1.075	1.086	1.08	1.023	1.031	1601
R10–66	1339	1702	1.671	1.17	1.065	1.078	1.09	1.036	1.039	1702
R11–66	1268	1575	1.662	1.19	1.075	1.067	1.08	1.039	1.053	1575
R12–66	1295	1652	1.643	1.19	1.067	1.060	1.10	1.034	1.022	**1646**
R13–66	1275	1617	1.720	1.19	1.071	1.087	1.10	1.025	1.025	**1616**
R14–66	1245	1611	1.739	1.21	1.095	1.066	1.09	1.044	1.014	**1610**
R15–66	1228	1608	1.660	1.19	1.067	1.065	1.10	1.042	1.019	**1604**
R16–66	1356	1725	1.654	1.16	1.070	1.082	1.07	1.028	1.026	**1720**
R17–66	1274	1627	1.711	1.21	1.112	1.075	1.10	1.053	1.051	1646
R18–66	1328	1671	1.661	1.18	1.084	1.065	1.08	1.024	1.023	1676
R19–66	1256	1635	1.704	1.17	1.077	1.052	1.09	1.034	1.029	1646
Average			1.670	1.18	1.081	1.072	1.08	1.037	1.032	

6.4. Instances with 66 orders

Another set of 20 instances with 66 orders is now considered. These instances are taken from Petersen and Madsen [22] as well and they were randomly generated in the same way as the 33-order instances.

The information given on Table 2 is similar to the one given on Table 1 but referred to the new set of instances with size 66, including one more column labeled by Best* containing the best known solutions obtained running HVNS for a few hours. It can be seen that the results offered by VND are even farther away from the best known solutions, because the considered instances are larger. HVNS and GVNS outperform LNS, being the former again the one with the best performance in all instances. Now the differences between

HVNS and LNS are greater, being the average gap reduced from 18% to 7.2% with 10 seconds and from 8% to 3.2% with 3 minutes. Note also that the results obtained by HVNS in 10 seconds are in average better that the ones obtained by LNS in 3 minutes. Finally, in the last column it can be observed that the best known solutions of seven instances are improved using HVNS approach (in bold) and for other five instances the same best known solution is obtained (underlined).

6.5. *Instances with 132 orders*

With the purpose of testing the performance of proposed heuristics when applied to larger instances, we generated another set of 20 instances with 132 orders in the same way as the previously considered instances. To the best of our knowledge there are no results in the literature for instances of this size.

In Table 3 the results obtained by VND, GVNS and HVNS with different running times are compared to the best known results for these instances. The first column contains the names of the instances; the second column shows the cost of best known solutions, obtained running HVNS for 12 hours; in the third column the quality of results obtained using VND heuristic are reported; and in the next eight columns the gaps for GVNS and HVNS heuristics with a running time of 10 seconds and 3, 5 and 8 minutes are presented. Finally, in the last row we indicate the average quality of results. It can be observed that both algorithms have quite a similar performance, but for each fixed running time the average results provided by HVNS are around 1% closer to the best known than the ones provided by GVNS.

Table 3
Results on instances with 132 orders.

Problem	Best	VND	10 seconds		3 minutes		5 minutes		8 minutes	
			GVNS	HVNS	GVNS	HVNS	GVNS	HVNS	GVNS	HVNS
R00–132	2591	1.675	1.222	1.157	1.068	1.066	1.055	1.063	1.039	1.038
R01–132	2645	1.687	1.178	1.167	1.084	1.054	1.074	1.042	1.053	1.045
R02–132	2639	1.711	1.180	1.139	1.071	1.052	1.054	1.046	1.054	1.031
R03–132	2752	1.694	1.145	1.124	1.047	1.032	1.053	1.042	1.023	1.017
R04–132	2603	1.636	1.153	1.131	1.053	1.046	1.050	1.030	1.036	1.029
R05–132	2616	1.686	1.186	1.158	1.056	1.057	1.052	1.047	1.027	1.047
R06–132	2576	1.662	1.163	1.160	1.063	1.071	1.056	1.046	1.044	1.049
R07–132	2615	1.636	1.169	1.147	1.048	1.075	1.044	1.048	1.037	1.036
R08–132	2638	1.669	1.185	1.143	1.077	1.053	1.055	1.046	1.053	1.034
R09–132	2554	1.644	1.145	1.136	1.041	1.035	1.034	1.025	1.018	1.013
R10–132	2646	1.662	1.197	1.190	1.068	1.064	1.062	1.037	1.038	1.036
R11–132	2632	1.623	1.171	1.133	1.044	1.046	1.062	1.052	1.041	1.027
R12–132	2555	1.708	1.175	1.185	1.086	1.068	1.070	1.053	1.063	1.054
R13–132	2659	1.683	1.155	1.157	1.062	1.050	1.046	1.033	1.022	1.021
R14–132	2605	1.661	1.149	1.140	1.050	1.037	1.053	1.025	1.042	1.027
R15–132	2626	1.681	1.208	1.185	1.074	1.053	1.045	1.054	1.042	1.038
R16–132	2534	1.668	1.197	1.160	1.058	1.067	1.047	1.045	1.053	1.046
R17–132	2569	1.701	1.203	1.142	1.064	1.046	1.065	1.043	1.039	1.035
R18–132	2652	1.636	1.168	1.151	1.053	1.035	1.034	1.020	1.034	1.019
R19–132	2644	1.666	1.151	1.160	1.068	1.042	1.040	1.046	1.052	1.028
Avg.		1.669	1.175	**1.153**	1.062	**1.053**	1.053	**1.042**	1.041	**1.034**

6.6. *Evolution of performance with running time*

Fig. 8 shows how the solutions obtained by HVNS for all considered sets of instances improve when increasing the available running time for the algorithm. Fig. 8(a) represents the average gaps obtained for both sets of 33-order instances (Set0 and Set1) and the set of 66-order instances with running times of 10, 30, 60, 120, 180, 300, 450 and 600 seconds, and Fig. 8(b) shows the results corresponding to the 132-order instances.

In Figs. 8(a) and (b) it can be observed that not very large running times are needed to obtain good quality solutions for instances with different sizes.

7. Conclusions

This paper approaches a recently introduced variant of the TSP/PDP called DTSPMS in which little research has been carried out. We are dealing with a vehicle routing problem in which pickups and deliveries must be performed in two separated networks and some precedence constraints must be respected. We proposed a new hybridized variable neighborhood search heuristic that uses the two existing operators for the problem and four new ones, allowing the algorithm to modify both loading and routing parts of the solutions and introducing diversification into the search process at the same time. Computational results show that this new VNS heuristic outperforms previous heuristics such as iterated local search, tabu search, simulated annealing, large neighborhood search and standard general variable neighborhood search. Good solutions could be found in a short running time, and the best known solutions of some considered instances were improved.

The design of other heuristics or metaheuristics to obtain good quality solutions for large instances of the problem is open for further work. It would also be an interesting line of research for the future to try to design exact approaches that are able to solve to optimality instances closer to realistic size in a reasonable running time. Generalizing the problem allowing the use of multiple vehicles to serve

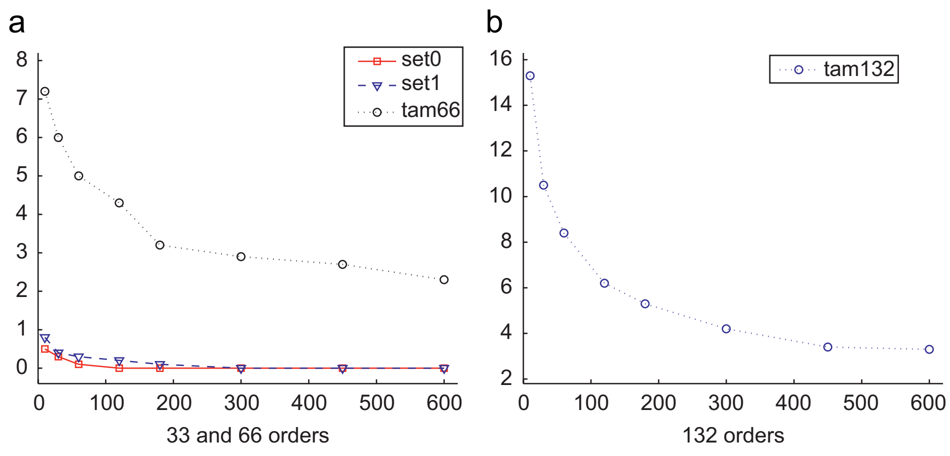

Fig. 8. Evolution with running time.

the customers, introducing multiple depots in both networks or including time windows would lead to other interesting and, in most cases, even more difficult problems. And also some particular cases in which some orders are picked up at the same location or sent to the same customer can have interesting practical applications and may be approached using particular techniques.

Acknowledgments

This work was partially supported by Project UCM2006-910707 from UCM, Project TIN2006-06190 from DGI and Project MTM2006-14961-C05-05 from MCYT.

The authors would like to thank the anonymous referees for their valuable comments and suggestions, that helped us to highly improve the paper.

References

[1] Bianchessi N, Righini G. Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery. Computers & Operations Research 2007;34(2):578–94.
[2] Carrabs F, Cerulli R, Cordeau J-F. An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with lifo loading. INFORMS Journal on Computing 2007;19:618–32.
[3] Carrabs F, Cordeau J-F, Laporte G. Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. INFORMS Journal on Computing 2007;19:618–32.
[4] Cassani L. Righini G. Heuristic algorithms for the TSP with rear-loading. In: 35th Annual Conference of the Italian Operational Research Society, AIRO XXXV, Lecce, Italy; 2004.
[5] Clarke G, Wright J. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 1964;12:568–81.
[6] Cordeau J-F, Iori M, Laporte G, Salazar-González J. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. Networks, to appear.
[7] Cordeau J-F, Laporte G, Potvin J-Y, Savelsvergh MW. Transportation on demand. In: Barnhart C, Laporte G, editors. Handbooks in operations research and management science, vol. 14. Amsterdam: Elsevier; 2006. [Transportation].
[8] Desaulniers G, Desrosiers J, Erdmann A, Solomon MM, Soumis F. VRP with pickup and delivery. In: Toth P, Vigo D, editors. The vehicle routing problem. Philadelphia, PA: SIAM; 2002. p. 225–42 [chapter 9].
[9] Doerner DF, Fuellerer G, Hartl RF, Gronalt M, Iori M. Metaheuristics for the vehicle routing problem with loading constraints. Networks 2007;49(4): 294–307.
[10] Dumitrescu I, Ropke S, Cordeau J-F, Laporte G. The traveling salesman problem with pickups and deliveries: polyhedral results and branch-and-cut algorithm. Mathematical Programming, Series A, 2008, Forthcoming. ISSN 1436-4646, doi:10.1007/s10107-008-0234-9.
[11] Gendreau M, Iori M, Laporte G, Martello S. A heuristic algorithm for a routing and container loading problem. Transportation Science 2006;40:342–50.
[12] Gendreau M, Iori M, Laporte G, Martello S. A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. Networks 2008;51(1):4–18.
[13] Hansen P, Mladenović N. Variable neighbourhood search: principles and applications. European Journal of Operational Research 2001;130:449–67.
[14] Hansen P, Mladenović N. Developments of variable neighbourhood search. In: Ribero C, Hansen P, editors. Essays and surveys in metaheuristics. Dordrecht: Kluwer Academic Publishers; 2001. p. 415–40.
[15] Hansen P, Mladenović N. Variable neighbourhood search. In: Pardalos P, Resende M, editors. Handbook of applied optimization. New York: Oxford University Press; 2002. p. 221–34.
[16] Hansen P, Mladenović N. Variable neighbourhood search. In: Glover F, Kochenberger GA, editors. Handbook of metaheuristics. Dordrecht: Kluwer Academic Publishers; 2003. [chapter 6].
[17] Hernández-Pérez H, Salazar-González JJ. Heuristics for the one-commodity pick-up and delivery traveling salesman problem. Transportation Science 2004;38:245–55.
[18] Hernández-Pérez H, Salazar-González JJ. The one-commodity pickup-and-delivery traveling salesman problem: inequalities and algorithms. Networks 2007;50(4):258–72.
[19] Iori M, Salazar-González J, Vigo D. An exact approach for the vehicle routing problem with two-dimensional loading constraints. Transportation Science 2007;41(2):253–64.
[20] Mladenović, N. A variable neighbourhood algorithm, a new metaheuristic for combinatorial optimization. Abstracts of papers presented at Optimization Days. Montréal; 1995. p. 112.
[21] Mladenović N, Hansen P. Variable neighbourhood search. Computers & Operations Research 1997;24:1097–100.
[22] Petersen HL, Madsen OBG. The double travelling salesman problem with multiple stacks—formulation and heuristic solution approaches. European Journal of Operational Research, in press, doi:10.1016/j.ejor.2008.08.009 (ISSN 0377-2217).
[23] Shaw P. Using constraint programming and local search methods to solve vehicle routing problems. In: Maher M, Puget J-F, editors. Principle and practice of constraint programming—CP9 8. Berlin: Springer; 1998.
[24] Xu H, Chen Z, Rajagopal S, Arunapuram S. Solving a practical pickup and delivery problem. Transportation Science 2003;37:347–64.