European Journal of Operational Research 198 (2009) 139–147

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Production, Manufacturing and Logistics

The double travelling salesman problem with multiple stacks – Formulation and heuristic solution approaches

Hanne L. Petersen *, Oli B.G. Madsen

DTU Transport, Technical University of Denmark, Bygningstorvet 1, 2800 Kgs. Lyngby, Denmark

ARTICLE INFO

Article history:
Received 20 July 2007
Accepted 3 August 2008
Available online 23 August 2008

Keywords:
Routing
Packing
Metaheuristics
TSP variants

ABSTRACT

This paper introduces the *double travelling salesman problem with multiple stacks* and presents four different metaheuristic approaches to its solution. The double TSP with multiple stacks is concerned with determining the shortest route performing pickups and deliveries in two separated networks (one for pickups and one for deliveries) using only one container. Repacking is not allowed, instead each item can be positioned in one of several *rows* in the container, such that each row can be considered a LIFO (last in, first out) stack, but no mutual constraints exist between the rows. Two different neighbourhood structures are developed for the problem and used with each of three local search metaheuristics. Additionally some simpler removal and reinsertion operators are used in a Large neighbourhood search framework. Finally some computational results are given along with lower bounds on the objective value.

1. Introduction

As congestion is an ever-growing problem on the roads all over the world, intermodality is playing an increasingly important role in the transportation of goods. Furthermore, the complexity of the resulting planning problems presents additional requirements to the tools available to planners.

The project that forms the basis of this paper was initiated in cooperation with a company producing computer software systems for operation and fleet management in small and medium-sized transportation companies. The software company encountered this problem at one of its prospective customers, and the problem is intriguing in that it does not seem to have been treated previously in the literature, at the same time as it is conceptually simple.

The *double travelling salesman problem with multiple stacks* (DTSPMS) is concerned with finding the shortest routes performing pickups and deliveries in two separated networks/regions. The problem permits neither repacking nor vertical stacking, instead the items can be packed in several *rows* (horizontal stacks) in the container, such that each row must obey the LIFO (last in, first out) principle, while there are no mutual constraints between the rows.

In the DTSPMS a set of orders is given, each one requiring transportation of one item from a customer in the pickup region to a customer in the delivery region, i.e. each order contains a pickup customer and a delivery customer for one item. The items are required to be boxes/pallets of identical dimensions and each region has a depot. The two regions are far apart, and thus some long-haul transportation is required between the depots. This long-haul transportation is not part of the problem considered here. All pickups and deliveries must be carried out using the same container, which cannot be repacked along the way, and items in the container can only be accessed from the opening in one end of the container. Hence the problem to be solved consists of determining the shortest Hamiltonian tour through each of the networks, in such a way that a feasible loading plan exists. No time windows are considered in this problem.

In practice this situation can occur when the container is loaded onto a truck to perform the pickup operations, then returned by that truck to a local depot/terminal where it is transferred onto a train, ship or another truck, which then performs the long-haul transportation. Upon arrival at the depot/terminal in the delivery region, the container is again transferred to a truck, which carries out the actual deliveries. The terminals only have facilities to perform container movements, and do not offer any opportunities for opening or repacking the container.

It is assumed that each order consists of exactly one item, thus if one pickup or delivery location is shared by several orders the corresponding node in the graph will be duplicated.

To state the problem more formally two weighted complete graphs $(V^G, E^G), G \in \{\mathrm{P}, \mathrm{D}\}$ are given for pickup (P) and delivery (D), respectively, and the purpose is to find a Hamiltonian tour through each graph, such that the sum of the weights of the edges used is minimised. Each graph G has a depot node v_0^G, $G \in \{\mathrm{P}, \mathrm{D}\}$, customer nodes $v_1^G, \ldots, v_n^G$, and symmetric edge costs c_{ij}^G. A set of n orders $\{1, \ldots, n\}$ is given, where order i must be picked up at

* Corresponding author. Tel.: +45 45251515.
E-mail address: hlp@transport.dtu.dk (H.L. Petersen).

0377-2217/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2008.08.009

$v_i^P \in V^P$ and delivered at $v_i^D \in V^D$. Finally, $V_C^G = V^G \setminus \{v_0^G\}, G \in \{P, D\}$ denotes the set of customer nodes in graph G. Whenever superscript G is used in the following it will indicate a distinction between pickup and delivery, $G \in \{P, D\}$.

The edge costs are assumed to be symmetric throughout this paper, but the presented algorithms do not rely on this property, and can also handle problems with asymmetric edge costs.

Throughout the paper it will be assumed that the number of orders n equals the number of loading positions. The loading container will have R rows, each of length L, and thus $n = R \cdot L$. The total number of nodes in the problem is $2n + 2$.

The connection between the two tours to be found is given by the loading of the container. Since no repacking is allowed, the only items that can be delivered "next" at any time during delivery are the ones that can be accessed from the opening of the container. This implies that the loading is subject to LIFO constraints.

However, in the DTSPMS there is no LIFO ordering for the container as a whole. Rather, it contains several loading rows, each of which can be considered a LIFO stack, but all rows are independently accessible.

In real life the items to be transported would typically be standardised Euro Pallets, which fit 3×11 on the floor area of a 40-foot pallet container, providing three independent loading rows.

A solution to a given problem consists of a *pickup route*, a *delivery route*, and a *row assignment*, which for each item indicates which loading row it must be placed in. A row assignment only gives the row that each item should be placed in, and does not indicate which position the item will occupy in that row. Given a route (pickup or delivery) and a row assignment, one can construct the *loading plan*, which gives the exact position of each item inside the loaded container.

The problem may at first glance seem purely theoretical, since the extra mileage incurred by not being able to repack may seem prohibitive. However the problem has been encountered in real-life applications, where this extra mileage is justified by the wages stemming from handling and requirements to comply with union restrictions (the driver is not allowed to handle the goods).

Special cases of the DTSPMS occur when the number of loading rows is equal to 1 or to the number of orders n. In both cases the problem of finding a row assignment for the solution becomes irrelevant.

In the single row case the pickup route will strictly dictate the delivery route (or vice versa), and the two routes will be exact opposites. In this case the problem can be solved by adding the transposed distance matrix of the delivery graph to the distance matrix of the pickup graph, and solving a regular TSP for the resulting distance matrix.

Conversely, when the number of loading rows equals the number of orders n, the two routes do not impose any restrictions on each other and the optimal solution to the problem consists of the optimal solutions to the two independent TSPs.

The DTSPMS as described here is a combination of the travelling salesman problem (TSP) and pickup and delivery problems (PDPs) and does not seem to have been treated previously in the literature, however early presentations of the work presented in this paper have inspired additional work presented in [9], which uses several new operators, along with the ones presented here, in a variable neighbourhood search (VNS) framework.

Although being concerned with pickups and deliveries, the DTSPMS differs significantly from the "regular" PDP as described in e.g. [6,7], and earlier in [15,21]. A number of variations of the PDP have been described in the more recent survey [18].

The main additional complication is the availability of multiple LIFO loading rows and thus the need to present a loading plan as part of the solution. The regular PDP with LIFO ordering (one stack only) has been treated using both heuristics [3] and exact methods [5,2].

In the regular PDP it is necessary to make sure that each pickup is performed before the corresponding delivery. This is automatically ensured in the DTSPMS, since all pickups are performed before all deliveries. Furthermore when capacity constraints are present in a PDP these must be checked for every node that is visited. In the DTSPMS all items will need to be kept in the container at the same time, and therefore it would suffice to check the capacity of the full vehicle if any kind of capacity constraints were present (which would not happen in the plain DTSPMS, but could occur with an extension to multiple vehicles, or if not all orders need to be served).

Apart from the regular PDP, another class of problems that show similarities with the DTSPMS is the TSP with Backhauls (TSPB) (cf. e.g. [23]). Here the property that "all pickups lie before all deliveries" is preserved, however there is no longer any constraints tying a pickup to its corresponding delivery.

The DTSPMS is a special case of the generalised pickup and delivery problem with loading constraints in 2 dimensions. Routing problems with more general loading constraint are described in [13,11,10].

The Multi-Pile VRP (MPVRP), is a problem somewhat similar to the DTSPMS, combining routing and loading, using several available stacks/piles. The paper [8] solves the MPVRP using tabu search and ant colony optimisation. The MPVRP is a generalisation of the DTSPMS, with varying dimensions of the transported items (leading to overlap between the stacks/piles).

The paper is organised as follows: First a mathematical formulation of the problem is presented in Section 2 and some comments are made on its implementation in GAMS/CPLEX. Next, four different heuristic solution approaches are presented in Section 3, with emphasis on the developed neighbourhood structure that is based on the structure of the problem, and is used for the first three approaches. Finally, Section 4 describes the implementations and gives some computational results, and Section 5 concludes on the described heuristic solution approaches and gives some suggestions for future work on the DTSPMS.

2. Mathematical formulation

The DTSPMS can be modelled as a binary integer programming problem with variables

$$x_{ij}^G = \begin{cases} 1 & \text{if edge } (i,j) \text{ is used in graph } G, \\ 0 & \text{otherwise} \end{cases} \quad \forall i,j \in V^G,$$

$$y_{ij}^G = \begin{cases} 1 & \text{if } v_i^G \text{ is visited before } v_j^G, \\ 0 & \text{otherwise} \end{cases} \quad \forall i,j \in V_C^G,$$

$$z_{ir} = \begin{cases} 1 & \text{if item } i \text{ is placed in row } r, \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in V_C^G \quad r = 1, \ldots, R.$$

Again $G \in \{P, D\}$.

The objective function can then be expressed as:

$$\min \sum_{\substack{i,j \in V^G \\ G \in \{P,D\}}} c_{ij}^G x_{ij}^G. \tag{1}$$

The constraints can be stated as follows:

$$\sum_i x_{ij}^G = 1 \quad \forall j \in V^G, \tag{2}$$

$$\sum_j x_{ij}^G = 1 \quad \forall i \in V^G, \tag{3}$$

$$y_{ij}^G + y_{ji}^G = 1 \quad \forall i,j,G, \ i \neq j, \tag{4}$$

$$y_{ik}^G + y_{kj}^G \leqslant y_{ij}^G + 1 \quad \forall i,j,k,G, \tag{5}$$

$$x_{ij}^G \leqslant y_{ij}^G \quad \forall i,j,G, \tag{6}$$

$$y_{ij}^{\mathrm{P}} + z_{ir} + z_{jr} \leqslant 3 - y_{ij}^{\mathrm{D}} \quad \forall i,j,r = 1,\ldots,R, \tag{7}$$

$$\sum_r z_{ir} = 1 \quad \forall i, \tag{8}$$

$$\sum_i z_{ir} = L \quad \forall r = 1,\ldots,R, \tag{9}$$

$$x,y,z \in \mathbb{B}, \tag{10}$$

where i,j and k are in V_C^G unless otherwise stated, and G is always in $\{\mathrm{P},\mathrm{D}\}$.

Constraints (2) and (3) are flow conservation constraints, stating that one unit of flow must enter and exit each node.

Constraints (4) ensure that for each pair of nodes (i,j) a precedence variable must be set, i.e. either i is visited before j or j before i. Constraints (5) express that if i is before k and k is before j, then i must necessarily be visited before j and constraints (6) ensure that if the edge (i,j) is used, then the according precedence variable is set (i is visited before j).

Constraints (7) express the LIFO constraints that are only relevant when two items are in the same row, i.e. if i and j are placed in the same row, and i is picked up before j, then i must be delivered after j (i may not be delivered before j).

Finally, (8) ensure that all items must be assigned to exactly one row, and (9) enforce the row capacity/length L.

The model has been implemented in GAMS 21.5 with CPLEX 9.1, on a UNIX system with 16 GB RAM/1200 MHz, which was able to solve problems for container sizes up to 2×5 or 3×4 within an hour of running time. Since the typical real-life instance is of size 3×11 and the time available for solving was limited, it was therefore decided to attempt to solve the problem heuristically.

3. Heuristic solution approaches

Since the mathematical model is unsolvable for problems of realistic size using a standard solver, a number of metaheuristic solution approaches have been considered for this problem. A survey of previous use of metaheuristics in vehicle routing problems can be found in [12].

Tabu search (TS) has previously presented good solutions to vehicle routing problems, which are similar in nature to the current problem. Many variations of tabu search exist, however this paper will only consider the simple version, which always uses the best neighbouring solution, and has constant tabu tenure.

Simulated annealing (SA) is another method that is well-known to provide good results to many different problems. Its advantage over TS is that it can move faster through a number of neighbourhoods, since it immediately chooses one neighbour at each iteration, rather than comparing several neighbours, and can thus complete a higher number of iterations in a given time. This comes at a cost of more randomised behaviour.

To conclude the traditional neighbourhood-based metaheuristics, a simple *Steepest Descent* approach has been implemented to determine how easy it is to locate good local optima from random starting points in the solution space. This has then been used in a simple iterated local search (ILS) with random restarts and steepest descent used as the local search strategy.

Finally a *large neighbourhood search* (LNS) algorithm has been implemented for the problem. LNS has in recent years showed good performance for VRP-like problems and seems promising when dealing with highly constrained problems, such as the one treated here.

The first three solution approaches all solve the problem by local search in a rather limited neighbourhood and are all based on an initial solution and some neighbourhood structure. Thus once one or more neighbourhoods have been developed, they can be reused for several approaches. Two different neighbourhoods have been developed here, both of which preserve feasibility of the solution, and, as it will be explained later, in combination the two can cover the entire feasible solution space. These two neighbourhoods have been implemented for use with each of the first three abovementioned heuristic approaches.

The LNS approach is based on a combination of simpler operators, performing either deletion or insertion of orders. Since these operators are less problem-specific, this approach allows for the operators to be inspired by heuristic solutions to more general problems, such as the TSP or VRP.

Since the company that introduced the problem were interested in the running times that would be experienced by the customers, wall clock times was chosen as the stopping criterion. This additionally ensured that the results from the different algorithms would be directly comparable. A 10 second interval was chosen to resemble online computations, while 3 minutes was expected to produce considerably better solutions within a duration that users would still be willing to wait for.

3.1. Initial solution

A feasible solution to the problem can be found by solving the single-stack version of the problem. In this case the pickup and delivery routes must be exact opposites, and the solution can be obtained by adding the two distance matrices and solving a regular TSP on the resulting distance matrix. This problem has been solved using a savings algorithm as introduced by Clarke and Wright in [4] to solve the TSP, and this initial solution has been used for the TS, SA and LNS implementations, since these only need one initial solution to the problem.

For the ILS, a number of initial solutions were constructed by randomly generating an ordering of the items to use for the pickup route, and reversing this for the delivery route, thus all of the generated solutions are still based on feasible solutions to the single-stack problem. Loading rows were then assigned randomly by partitioning all orders evenly among the available rows.

3.2. Feasible neighbourhood structures

In this section the two operators that form the basis of the three initial solution approaches are introduced.

3.2.1. Route-swap

The first operator only performs changes to the routing of the two tours, and leaves the row assignment untouched. The neighbourhood consists of all possible exchanges of two items A and B in a route where they immediately follow each other. To ensure feasibility of the resulting solution it is necessary during this operation to consider whether the two items are placed in the same loading row. These two cases can be seen in Fig. 1 (top resp. bottom). From left to right the figure show pickup route, loading plan and delivery route, before and after performing a *route-swap*.

If A and B are placed in the same row (top of Fig. 1), then they will be neighbours in this row and their positions in the loading plan of the container will be swapped by swapping their pickup order. Consequently their positions must also be swapped in the delivery route (even when A and B are not consecutive here).

If A and B are placed in different loading rows (bottom of Fig. 1), then, since they are consecutive in the pickup route, this means that when v_A^{P} is visited, v_B^{P} will be next in line and since A and B are in separate rows, then the loading position of B will also be accessible (and be the last accessible position in its row), and thus the positions of A and B in the final loading plan can remain unchanged when swapping the pickup order, and so can the delivery route.

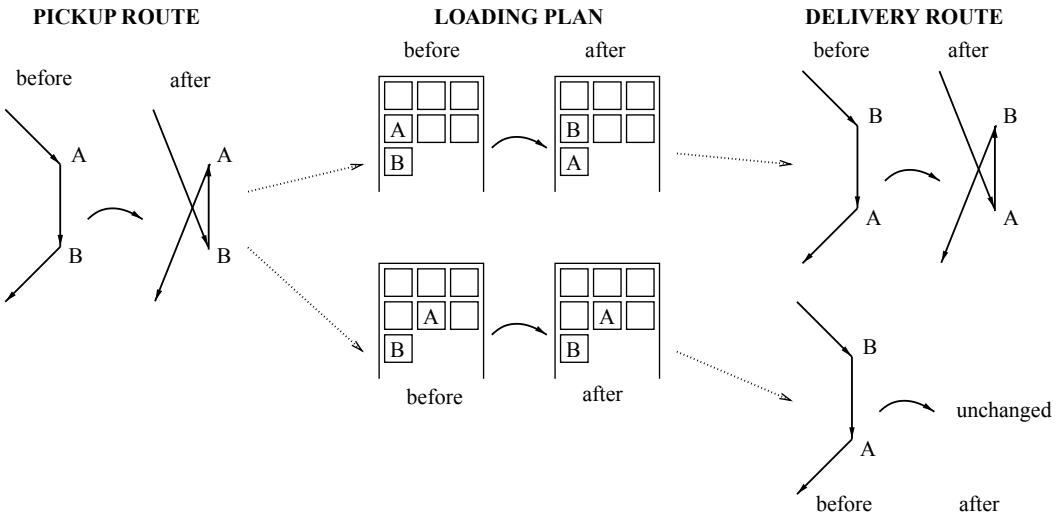

Fig. 1. An illustration of *route-swap*.

Both of these arguments can be reversed when the operator is used on the delivery route.

The entire neighbourhood can be traversed by considering all values of $i = 1, \ldots, n-1$ for both routes τ, thus the size of this neighbourhood is $2n-2$, i.e. $O(n)$, where n is the number of orders.

3.2.2. Complete-swap

The operator *complete-swap* is focusing on the row assignment, while only updating the routes to maintain feasibility. It considers any pair of items that are currently assigned to different rows (regardlessly of routing), and swaps their positions in the loading plan and in each of the routes. This is illustrated in Fig. 2. The top half shows (part of) the two graphs and loading plan *before* performing the *complete-swap*, while the bottom half shows them after. *A* and *B* are the two orders that are swapped, while *C* is some order that is visited between the two, and would be blocking if the routes were not updated as part of the swap.

To traverse the entire neighbourhood all pairs of orders must be examined (skipping pairs where both orders are assigned to the same row), and the size of the neighbourhood is therefore $O(n^2)$.

3.2.3. The operators in combination

Since *route-swap* does not affect the row assignment of a solution, it is obvious that one cannot reach the entire solution space by only using this operator. Similarly when performing *complete-swap* the mutual visiting orders of the two routes will never change (e.g. if an order is handled third in the pickup route and last in the delivery route, then there can never be an order visited third in the pickup route and not last in the delivery route, if only *complete-swap* is applied). However by using a combination of the two operators it becomes possible to cover the entire solution space.

The idea of combining several different neighbourhood structures is not unique - it is also the idea behind variable neighbourhood search (VNS, see [17]). However, the background for using this approach here is different. The purpose of VNS is to be able to reduce solution time, while the use of a combined neighbourhood in this paper is a necessity to ensure access to all feasible solutions.

Any solution can be constructed from any other solution by first performing a series of *complete-swaps* until the loading plan is correct, followed by a number of *route-swap* operations to make the routes match. When performing a *route-swap* only one of the routes will be affected, unless the swapped items are assigned to the same row. In this case the two items *must* be swapped in both routes as the solution would otherwise become infeasible.

3.3. Iterated local search

The idea behind iterated local search (cf. [16]) is to use a simple local search procedure a number of times in an intelligent way, to eventually produce good results. In the current implementation

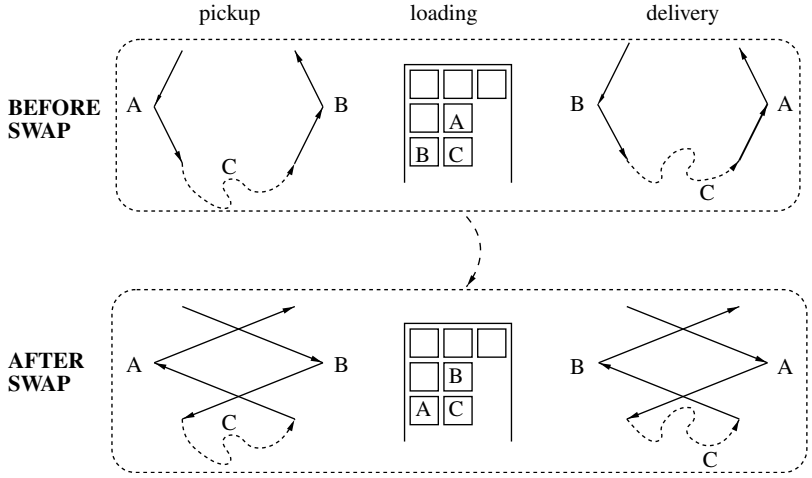

Fig. 2. An illustration of *complete-swap*.

this has been done by using steepest descent at the local search procedure, and using random restarts as a rather crude iteration control. The restart mechanism is usually at the heart of the algorithm, and thus the implementation here should not be seen as a representative of what ILS would be capable of for this problem. However it was implemented to examine the value of using a pure steepest descent approach, and the results should be judged as such.

3.4. Tabu search

In the tabu search implementation the fact that two different neighbourhood structures must be considered has been dealt with by systematically changing between the two types, following some pre-defined pattern. Two parameters are used to describe this pattern, namely the *length* or *period* of the pattern, `perLen`, and the *ratio* between the two operators, `ratio`. E.g. the parameter combination $\texttt{perLen} = 20, \texttt{ratio} = 0.3$ indicates that the first 6 iterations should use *route-swap*, the next 14 should use *complete-swap*, and from iteration 21 onwards this pattern is repeated.

In this way the deterministic nature of tabu search has been maintained, by not introducing randomness in the selection of the operator to use for each iteration.

For each operator it has additionally been necessary to decide on some attributes to register in the tabu list, and for both operators the choice has been to register which two orders were swapped.

For moves of type *route-swap* it is thus *not* registered on which route (pickup or delivery) the swap was performed.

Although moves of both types are marked by the same attributes in the tabu list, a separate list is kept for each move type. This is due to the differences in the two operators. Consecutively performing one move of each type on the same pair of orders does not imply that one move is reversing the other, and does not lead the algorithm to start cycling between a few solutions, and hence the two operators should not be kept in the same tabu list.

The TS algorithm implemented here is rather simple, with a constant tabu list length, fixed switching strategies between the operators, and a stopping criterion based on elapsed time. Using a different stopping criterion, such as a certain number of iterations without improvement, was decided against to ensure comparability between all of the implemented algorithms.

3.5. Simulated annealing

The SA implementation handles the two available operators by randomly selecting one of them for each iteration, and the probability for choosing each operator is expressed in the parameter *ratio*, indicating the probability of choosing *route-swap*.

Traditionally SA approaches take three parameters; an initial temperature T_s, a final temperature T_e, and a temperature reduction function.

The temperature reduction function is usually based on the reduction scheme

$$T_{i+1} = c \cdot T_i, \tag{11}$$

where $T_0 = T_s$, $c \in (0;1)$ is some reduction factor, and the algorithm terminates when the final temperature T_e is reached.

However for the purpose of comparability SA has here been implemented to take running time t as an input parameter instead of the temperature reduction factor/function.

Thus the temperature at time t has been calculated as:

$$T(t) = T_s \cdot \left(\frac{T_e}{T_s}\right)^{\frac{t}{t_{\max}}}, \tag{12}$$

where $t_{\max}$ is the total allowed running time. This ensures that $T(0) = T_s$, $T(t_{\max}) = T_e$ and the behaviour between these two points corresponds to that of (11) when temperature T_e is reached at time $t_{\max}$. In this implementation the temperature is updated at each iteration.

3.6. Large neighbourhood search

Large neighbourhood search (cf. [22]), is based on simply removing (larger) parts of the solution, and subsequently reinserting the affected customers. Thus it does not use neighbourhood operators in the same sense as the previously discussed local search based metaheuristics.

An implementation of LNS must consider such issues as strategies for removal and insertion, degree of destruction at each iteration and acceptance criterion for the generated solutions.

In this case both the removal and insertion strategies are based on a variety of simple strategies, where the strategy to apply at each operation is selected randomly with some probability (which is a parameter). As with the other algorithms used in this paper, the stopping criterion for LNS has been running time.

3.6.1. Operators

Two removal strategies have been implemented – one is based on a measure of *relatedness* similarly to the description in [22], thus attempting to remove orders similar to those already removed in the same iteration, while the other is based on removing the orders that are most expensive to cover in the current solution. In this context the relatedness of two orders has been expressed as the sum of the distances between the orders in each of the two graphs.

Reinsertion is performed using a range of four different insertion heuristics: Nearest, farthest, cheapest and most expensive (all based on [14]).

Additionally, a certain amount of noise has been added when choosing the order to insert, to increase diversification.

3.6.2. Parameters

In addition to the use of the available operators a number of parameters have been applied and tested for the LNS implementation.

The criterion for accepting new solutions has been based on simulated annealing acceptance, similar to the approach used in [20]. As in Section 3.5 the calculation of the temperature is based on a pre-determined stopping criterion, rather than using the temperature itself as the stopping criterion.

Additionally it has been decided to remove *orders* from the solution rather than *customers*. This is based on the structure of the DTSPMS, which strongly reduces the reinsertion possibilities, if e.g. a pickup customer has been removed, while the corresponding delivery customer remains in the solution.

Finally, the steepest descent used for ILS has been used to improve the solution after each iteration.

4. Computational results

Each of the four algorithms has been implemented in Java 1.5 and tests have been performed for running times 10 and 180 seconds (both are wall clock times).

All tests have been performed on a Dell D610 laptop with 1.5 GB RAM and a 1.60 GHz processor running Windows XP.

4.1. Test instances

The test instances that have been used for the evaluation of the solution approaches have been generated randomly, by finding

two sets of n random (real) points in a 100×100 square. The depot is placed in the centre of the square at (50, 50). All distances are Euclidean distances rounded to the nearest integer, in accordance with the conventions from TSPLIB. Note that this rounding implies that the triangle inequality is not preserved, however the algorithms described in this paper do not rely on this inequality to be valid.

Two test sets have been generated, each containing ten problems of size $n = 33$ orders (i.e. with 33 customers in each graph). The first set was used for parameter tuning for each of the heuristics, and both sets were used for the final results reported in Table 2.

The test instances generated for this problem can be obtained from http://www.imm.dtu.dk/~hlp.

In this paper all test problems have been solved for the 3-stack problem.

4.2. Bounds

Since the optimal solutions to the test instances are unknown, it is desirable to find good lower bounds to evaluate the quality of the solutions obtained during testing.

One lower bound can be found by relaxing the constraints that ensure loading feasibility between the graphs, i.e. solving the n-stack problem. This way the problem reduces to two individual TSPs, that can be solved to optimality with tools such as Concorde (cf. [1]). Obviously each TSP solution is a lower bound on the optimal tour through that graph, and the sum of the tours is then a lower bound on the sum of the tours in any feasible solution to the DTSPMS with any number of rows.

However, intuitively this bound must be expected to be quite weak, since one would expect many changes to be necessary before a feasible loading plan could be constructed for such a solution.

In addition to this problem-specific lower bound some common lower bounds have been calculated and compared, to demonstrate some properties of the problem. These numbers can be found in Table 1.

The first column gives the problem number. The second column (nS) gives the lower bound described above (the optimal solution to the n-stack problem), column LP gives the lower bound provided by solving the LP relaxation of the model (1)–(10) and root gives the objective value in the root node after CPLEX has added cuts. BB shows the lower bound obtained by CPLEX after an hour of running time with BestBound used as the node selection strategy. Best shows the value of the best known solution for each problem, while Best/LB shows the quality of the best known solution compared to the best lower bound (i.e. Best/nS). All best known solutions have been found by allowing the best metaheuristic (LNS) to use a longer running time. The last two columns contain upper bounds: SS gives the optimal solution to the single-stack problem, while the column *init* gives the heuristic solution to problem SS, which was used as the initial solution that was used for TS, SA and LNS.

As can be seen from the table the n-stack bound is always considerably stronger than the LP bound, even after CPLEX has been allowed some time for improving this bound. Comparisons show that the best known solutions are all within 11–18% of the best lower bound (nS). Given that this lower bound is found by completely ignoring all sequencing constraints, one may assume that these lower bounds are quite weak. This suggests that the best found solutions could be reasonably good.

The best known solutions have all been found by performing a number of runs of length around 2 hours. The best approach for obtaining good solutions has turned out to be running several runs of a somewhat shorter length, rather than performing one run with a very long running time. Performing one run of 15 minutes length gives solutions that are on average 0.5% above the best known, and when performing four runs of each 90 minutes (i.e. 6 hours total time), the best known solution was matched at least once for all 20 instances.

Table 1
Bounds

	nS	LP	root	BB	Best	$\frac{\text{Best}}{\text{LB}}$	init	SS
R00	911	793	802	812	1063	1.17	1886	1685
R01	875	820	824	831	1032	1.18	1690	1581
R02	935	826	844	845	1065	1.14	1672	1563
R03	961	887	905	905	1100	1.14	1836	1745
R04	937	859	860	863	1052	1.12	1671	1628
R05	900	811	814	816	1008	1.12	1548	1439
R06	998	941	944	944	1110	1.11	1739	1644
R07	963	894	900	900	1105	1.15	1867	1695
R08	978	899	911	922	1109	1.13	1761	1636
R09	976	889	909	910	1091	1.12	1610	1553
R10	901	822	833	839	1016	1.13	1697	1575
R11	892	810	820	823	1001	1.12	1494	1429
R12	984	934	946	950	1109	1.13	1778	1673
R13	956	887	895	897	1084	1.13	1707	1613
R14	879	794	803	803	1034	1.18	1704	1565
R15	985	903	916	917	1142	1.16	1943	1783
R16	967	857	887	894	1093	1.13	1767	1647
R17	946	847	882	884	1073	1.13	1716	1620
R18	1008	876	920	921	1118	1.11	1796	1673
R19	938	839	855	864	1089	1.16	1725	1633

4.3. Results

A number of test runs have been performed for each of the four algorithms implemented. The details of these implementations will first be presented individually before presenting the final results and comparisons in Table 2. It is worth noting that the first three implementations all use the two improvement operators that were designed specifically for the DTSPMS and presented in Section 3.2, while the LNS implementation instead uses more general removal and insertion operators as described in Section 3.6.1.

All of the algorithms required considerations regarding the ratio between the different operators, and apart from the ILS algorithm each includes a number of parameters that additionally needed to be calibrated.

Table 2
Result summary

	10 seconds				3 minutes			
	LNS	SA	TS	ILS	LNS	SA	TS	ILS
R00	1.04	1.26	1.42	1.66	1.01	1.13	1.23	1.58
R01	1.04	1.17	1.34	1.64	1.01	1.08	1.21	1.61
R02	1.04	1.19	1.38	1.57	1.01	1.10	1.24	1.53
R03	1.06	1.22	1.44	1.66	1.01	1.12	1.24	1.62
R04	1.05	1.25	1.38	1.59	1.02	1.11	1.23	1.56
R05	1.03	1.22	1.25	1.53	1.01	1.15	1.21	1.47
R06	1.06	1.19	1.37	1.56	1.02	1.11	1.26	1.51
R07	1.05	1.23	1.39	1.66	1.01	1.11	1.26	1.58
R08	1.04	1.21	1.36	1.56	1.01	1.11	1.27	1.51
R09	1.04	1.15	1.29	1.47	1.01	1.08	1.19	1.46
R10	1.05	1.24	1.45	1.67	1.00	1.15	1.25	1.64
R11	1.06	1.24	1.24	1.49	1.01	1.10	1.22	1.48
R12	1.04	1.21	1.44	1.60	1.01	1.13	1.22	1.55
R13	1.04	1.23	1.37	1.55	1.01	1.08	1.22	1.53
R14	1.03	1.22	1.47	1.63	1.00	1.11	1.25	1.58
R15	1.04	1.21	1.37	1.62	1.01	1.10	1.26	1.56
R16	1.02	1.20	1.35	1.61	1.00	1.10	1.18	1.55
R17	1.04	1.24	1.48	1.60	1.00	1.12	1.28	1.58
R18	1.05	1.20	1.33	1.59	1.01	1.13	1.21	1.53
R19	1.03	1.16	1.31	1.57	1.01	1.11	1.25	1.54
Avg. set 0	1.04	1.21	1.36	1.59	1.01	1.11	1.23	1.54
Avg. set 1	1.04	1.22	1.38	1.59	1.01	1.11	1.23	1.56

All calibration runs were performed on the problems from the first data set to determine a good set of parameters, and these parameters were then applied to the problems from the both of the two data sets to produce the final results presented in Table 2. More elaborate results of the calibration runs can be obtained from [19], or by contacting the author.

In the remainder of this paper the term *solution quality* will be used to refer to the deviation from the best known solution, i.e. the objective value of the solution to an instance is divided by the value of the best known solution for that instance.

4.3.1. Iterated local search

For each iteration of the local search algorithm the operator to use has been chosen randomly with some probability *ratio*. The best results were obtained with value 0.4 for both of the running times considered, meaning that 40% of the moves were of type *route-swap*.

The algorithm could typically complete around 2600 iterations within the allotted 3 minutes.

4.3.2. Tabu search

Next the impact of the length of the tabu list, *tabuLength*, was examined. Based on the results tabu lengths of 7 and 11 were used for the remaining runs, for 10 seconds and 3 minutes, respectively.

Finally the combination of the operators was examined, given by the parameters *ratio* and *perLen* as described in Section 3.4. Again it proved beneficial to use a low value for *perLen* (tested values: 10, 30, 50, 70, 90), namely 10 for both running times. It was assessed that 10 was so close to 0 that using an even lower value would not produce any significant gain. The best value of *ratio* turned out to be higher for tabu search than for iterated local search, with values of 0.9 for 10 seconds and 0.7 for 3 minutes runs, thus using the less time-consuming operator more frequently, especially for the shorter TS runs (tested values: 0.1, 0.3, 0.5, 0.7, 0.9).

The algorithm was able to complete around 137,000 iterations within the allotted 3 minutes (ratio 0.7).

This strong preference for using the faster operator also gives an indication, that the number of iterations that can be completed is important to the result (number of iterations completed is approximately 54,000 for ratio 0.1 and 88,000 for ratio 0.5). This in turn indicates that the use of one combined neighbourhood might not pay off, since the number of iterations would then be reduced by around 50%.

4.3.3. Simulated annealing

The *ratio* parameter turned out to have a rather small impact on the quality of the solutions obtained by simulated annealing, and the chosen values were 0.4 for 10 seconds and 0.5 for 3 minute runs. These tests were furthermore performed with ratios 0.0 and 1.0, to demonstrate the effect of using only one operator. As expected this approach produced considerably worse solutions, and for running time 3 minutes the solution quality was 1.47 (*route-swap* only) and 1.35 (*complete-swap* only) when using one operator exclusively, while the corresponding numbers were around 1.1 for all tested values of *ratio* in the interval [0.1; 0.9].

Finally several combinations of start and end temperatures were examined, to determine the final set of parameters.

With these parameters the algorithm could complete around 15 million iterations in 3 minutes.

4.3.4. Large neighbourhood search

For both running times it showed beneficial to use a relatedness-based removal strategy in most iterations, with some uses of the most-expensive strategy for increased diversification. The final setting used relatedness in 80% of the iterations for 3 minute runs, and 60% for 10 second runs.

The best strategy for insertion in all cases was to randomly select one of the four available insertion strategies.

The number of orders to remove was selected randomly at each iteration – in the interval [4,17] for 3 minutes and [3,15] for 10 seconds. This indicates that the longer runs could benefit from a higher degree of diversification than the shorter runs.

Additionally it was tested to use the steepest descent algorithm from the ILS for re-optimisation at each iteration. The longer 3 minute running times would benefit from this refinement, while the shorter 10 second runs performed better when using less refinement and spending the time on additional iterations.

The acceptance of new solutions was performed using the criterion from simulated annealing – there was no variation in temperature interval between the two running times.

Finally an amount of noise was added to the solution when determining which order to reinsert next. Here the 10 second runs showed an overall best performance when these values were modified with ±40%, while the corresponding number for 3 minutes was ±20%.

The LNS algorithm could complete around 12,500 iterations within 3 minutes.

4.3.5. Summary of the results

After calibrating each of the three heuristics, Table 2 summarises the results obtained by applying these parameters to the test problems from both sets.

The table shows the quality of the solutions that have been found with each of the four heuristic approaches for the two different running times. All values are found using the "best" set of parameters found in the preceding sections.

The table shows that among the four implementations presented LNS consistently produces the best results, and gives results around 4% above the best known for a running time of 10 seconds, and within 1–2% when allowed to run for 3 minutes.

Additionally the table shows a very clear ranking of the implementations. For all but the ILS a significant improvement can be observed by increasing the running time.

The results also show that using the ILS algorithm with the short running time produces solutions that are only slightly better than the initial solutions used for the two other approaches. This demonstrates that a clear benefit can be obtained by using something more sophisticated than the current ILS, which is basically a series of steepest descents with random restart.

The top half of the table (problems R00–R09) shows the results obtained for set 0 which was used for parameter tuning, while the bottom half (problems R10–R19) shows the results for set 1 which has been used exclusively for the final evaluation of the algorithms. Averages for each set are reported at the bottom of the table. This shows that the solution quality is not significantly higher for the problems that have been used for calibration, i.e. the choice of calibration problems does not seem to have influenced the final parameter values.

4.4. Instances with known optimal solution

The biggest instance size which can currently be solved to optimality with the mathematical model presented in Section 2 contains 12 customers in each graph. A series of such instances has been obtained by considering the first 12 orders of each of the above-mentioned instances. Table 3 gives the optimal solution to each of these reduced instances, along with the average gap obtained by running the LNS, SA, and TS algorithms on these. The gap is averaged over three runs for LNS and SA, and in parentheses is given the number of times (out of three) where the optimal solution was found.

Table 3
Results for 12 order instances

	opt	$\frac{nS}{opt}$	$\frac{root}{opt}$	$\frac{LP}{opt}$	10 seconds			3 minutes		
					LNS	SA	TS	LNS	SA	TS
R00-12	694	0.98	0.96	0.96	1.00 (3)	1.02 (0)	1.04 (0)	1.00 (3)	1.01 (1)	1.04 (0)
R01-12	710	1.00	0.89	0.86	1.00 (3)	1.03 (0)	1.06 (0)	1.00 (3)	1.02 (0)	1.06 (0)
R02-12	606	0.98	0.91	0.89	1.00 (3)	1.02 (0)	1.08 (0)	1.00 (3)	1.01 (0)	1.08 (0)
R03-12	680	0.99	0.96	0.96	1.00 (3)	1.01 (0)	1.06 (0)	1.00 (3)	1.00 (3)	1.06 (0)
R04-12	607	0.99	0.95	0.93	1.00 (3)	1.03 (0)	1.09 (0)	1.00 (3)	1.00 (3)	1.06 (0)
R05-12	567	0.99	0.89	0.87	1.00 (3)	1.00 (2)	1.12 (0)	1.00 (3)	1.00 (3)	1.13 (0)
R06-12	747	0.99	0.95	0.90	1.00 (3)	1.03 (1)	1.08 (0)	1.00 (3)	1.00 (2)	1.08 (0)
R07-12	557	0.96	0.90	0.85	1.00 (3)	1.02 (0)	1.03 (0)	1.00 (3)	1.00 (3)	1.03 (0)
R08-12	690	0.98	0.98	0.98	1.00 (3)	1.04 (0)	1.13 (0)	1.00 (3)	1.00 (3)	1.12 (0)
R09-12	669	1.00	0.97	0.92	1.00 (3)	1.02 (0)	1.08 (0)	1.00 (3)	1.00 (1)	1.07 (0)
R10-12	633	0.95	0.93	0.91	1.00 (3)	1.01 (0)	1.11 (0)	1.00 (3)	1.00 (2)	1.08 (0)
R11-12	591	0.96	0.95	0.93	1.00 (3)	1.04 (0)	1.07 (0)	1.00 (3)	1.00 (3)	1.07 (0)
R12-12	722	0.99	0.92	0.91	1.00 (3)	1.01 (0)	1.09 (0)	1.00 (3)	1.00 (2)	1.08 (0)
R13-12	664	0.97	0.96	0.95	1.00 (3)	1.02 (2)	1.09 (0)	1.00 (3)	1.00 (3)	1.10 (0)
R14-12	650	0.98	0.85	0.84	1.00 (3)	1.03 (0)	1.07 (0)	1.00 (3)	1.01 (2)	1.07 (0)
R15-12	595	0.97	0.95	0.93	1.00 (3)	1.01 (2)	1.01 (0)	1.00 (3)	1.00 (2)	1.03 (0)
R16-12	577	0.99	0.98	0.98	1.00 (3)	1.02 (0)	1.05 (0)	1.00 (3)	1.00 (3)	1.05 (0)
R17-12	737	0.99	0.97	0.95	1.00 (3)	1.01 (1)	1.07 (0)	1.00 (3)	1.00 (1)	1.07 (0)
R18-12	724	0.98	0.98	0.97	1.00 (3)	1.01 (0)	1.07 (0)	1.00 (3)	1.00 (2)	1.07 (0)
R19-12	753	0.99	0.95	0.92	1.00 (3)	1.02 (0)	1.11 (0)	1.00 (3)	1.00 (2)	1.11 (0)
Avg.		0.98	0.94	0.92	1.00 $(\frac{60}{60})$	1.02 $(\frac{8}{60})$	1.08 (0)	1.00 $(\frac{60}{60})$	1.00 $(\frac{41}{60})$	1.07 (0)

The first column of Table 3 shows the optimal value, and the next two columns show the quality of the lower bounds obtained from the n-stack problem, and by letting CPLEX add cuts to the root node (cf. the bounds presented in Section 4.2). Solution times for obtaining the optimal solution ranges from 14 to 2850 seconds, with an average of 450.

Again the nS bound turns out to be superior in all cases, and in this case provides a quite good bound – probably due to the size of the problems. It should still be expected that the quality of this bound will deteriorate with increased values of R, since the nS bound is exactly the optimal solution for $R = 1$. A few tests that could be completed with $R = 3, L = 5$ confirm this expectation.

These results confirm the superiority of the LNS implementation over the three other implementations, by consistently finding the optimal solution, and also confirm that SA performs better than TS.

4.5. Larger problem instances

A real-life application containing 66 orders can also be imagined if the transported items are all vertically stackable and have a height that is less than half of the height of the container. In this case the top item of such a vertical stack must obviously be removed first, and additionally all items from one position must be removed before the top item on the next position becomes available. Thus solving an instance using such stacking of height 2 is identical to solving an instance with rows of double length.

The previously used test instances have all been increased to size 66 by letting the original random generation procedure continue until the desired number of orders has been generated (these larger instances are also available at http://www.imm.dtu.dk/~hlp), and a series of test runs have been performed on these instances with the parameter settings found earlier. The results hereof can be found in Table 4.

Again it can be seen that LNS outperforms the other heuristics.

Additionally it can be noticed that the gap between the lower bound and the best known solution has increased notably (avg. 30%). This probably indicates that these instances are harder and the best known solution could therefore be further away from the optimum, but most likely also reflects that when the number of items in each loading row increases, the quality of the n-stack lower bound decreases.

Table 4
Results for 66 order instances

	Best	nS	10 seconds			3 minutes		
			LNS	SA	TS	LNS	SA	TS
R00-66	1594	1237	1.19	1.38	1.59	1.07	1.21	1.45
R01-66	1600	1257	1.20	1.39	1.60	1.08	1.22	1.57
R02-66	1576	1295	1.20	1.38	1.63	1.12	1.28	1.61
R03-66	1631	1290	1.14	1.35	1.58	1.06	1.23	1.53
R04-66	1611	1295	1.18	1.46	1.52	1.09	1.28	1.43
R05-66	1528	1204	1.18	1.40	1.53	1.07	1.21	1.49
R06-66	1651	1294	1.17	1.38	1.56	1.07	1.21	1.51
R07-66	1653	1307	1.17	1.44	1.56	1.08	1.22	1.41
R08-66	1607	1297	1.18	1.41	1.54	1.07	1.27	1.52
R09-66	1598	1276	1.18	1.35	1.59	1.08	1.25	1.53
R10-66	1702	1339	1.17	1.35	1.60	1.09	1.23	1.49
R11-66	1575	1268	1.19	1.40	1.55	1.08	1.25	1.49
R12-66	1652	1295	1.19	1.38	1.55	1.10	1.24	1.41
R13-66	1617	1275	1.19	1.39	1.60	1.10	1.24	1.53
R14-66	1611	1245	1.21	1.38	1.65	1.09	1.24	1.54
R15-66	1608	1228	1.19	1.38	1.56	1.10	1.22	1.48
R16-66	1725	1356	1.16	1.34	1.51	1.07	1.17	1.41
R17-66	1627	1274	1.21	1.40	1.70	1.10	1.29	1.59
R18-66	1671	1328	1.18	1.40	1.58	1.08	1.23	1.52
R19-66	1635	1256	1.17	1.41	1.57	1.09	1.22	1.48
Avg.			1.18	1.39	1.58	1.08	1.24	1.50

The best known solutions to these instances have again been found by performing a series of runs of a few hours, however a lot more runs were required for these instances than for the smaller instances of 33 customers. Performing two runs of 4 hours each (thus 8 hours total time), will generate a solution that is around 1% above the best known, which reduces to 0.53% above the best known for 8×4 hours (i.e. 32 hours total), to 0.35% above for 16×4 hours, and to 0.25% for 25×4 hours (total 100 hours). Different combinations of running time/number of runs have been tested, and a series for 4 hour runs generally gave the best results.

5. Conclusion and future work

This paper has introduced the DTSPMS which is a new variant of the TSP/PDP, presented a mathematical formulation of the

problem, and demonstrated the behaviour of different heuristic solution approaches on the problem.

Four different implementations have been tested, and comparisons have shown that large neighbourhood search produces the best results of the four, with solutions that are within 2% of the best known solution, with a running time of 3 minutes, and within 2–6% for running times of 10 seconds.

It seems clear that the strongly restricted nature of this problem should be taken into consideration when choosing a suitable solution approach. The results obtained in this paper indicate that the tested traditional metaheuristic local search solution approaches might be insufficient for producing good solutions to the problem – apparently they are unable to cover the irregular solution space well enough. Instead the LNS-algorithm, which allows free movement through the solution space, proves successful. This observation is also supported by the preliminary results of [9], which uses intermediate partial solutions and larger neighbourhoods than the ones presented in the three initial heuristics of this paper.

Additionally tests have been performed on smaller instances with known optimal solutions, and on larger instances to further examine the behaviour of the implemented algorithms, and the findings are consistent with the initial results.

5.1. Future work

Future work on the DTSPMS could focus on either improving the solutions/solution methods or generalising the problem.

One way of attempting to improve the solutions would be by improving the approaches already described here. In particular one could attempt refining the tabu search, e.g. by using variable tabu list length, or by introducing some diversification mechanism, such as performing mutations to the solution when a certain number of iterations have been performed without leading to any improving solutions. Furthermore using a randomised operator choice for tabu search might lead to improvement, since the shorter period lengths are consistently the best. However, although some improvement of the behaviour of the three initial metaheuristics could be obtained, it is questionable whether the improvement would match the results of the LNS.

The three initial metaheuristics are all based on using a combination of different neighbourhoods, and to this end it would also be possible to consider different switching strategies, other than fixed ratios and random selection at each iteration. This could be obtained by using only one operator at a time until a certain number of iterations has been unable to produce improving solutions, or by using some adaptive scheme where the selection probability depends on previous successes.

The current ILS implementation was not intended as an ILS as such, but simply an attempt to test the usefulness of using a very simple steepest descent approach repeatedly. Thus the implementation could be refined using the usual principles from ILS, in particular improving the restart procedure to start at a perturbation of a previous solution, rather than a completely random solution at each iteration.

Additionally it could be attempted to solve the DTSPMS to optimality, and it could be interesting to examine the effect of the number of loading rows on the solution.

The most obvious extension of the problem is to either generalise the TSP aspects of the problem to more general vehicle routing (VRP) and include multiple vehicles/containers and/or multiple depots to form a double VRP with multiple stacks (DVRPMS), or to generalise the problem in the direction of the regular pickup and delivery problem, to form a pickup and delivery problem with multiple stacks (PDPMS).

In the first case, if the choice of depot becomes a decision, it is likely that the cost of the long-haul between the depots will no longer be negligible, and this may be very hard to estimate at the time of planning in real-life applications.

Modifying the problem to include non-uniform objects would complicate the packing considerably, since some objects might still be positioned next to each other, so that their internal delivery order becomes irrelevant. This would heavily influence the use of the LIFO-principle so far, and would approach the problem to the more generalised PDP with loading constraints.

References

[1] D. Applegate, R. Bixby, V. Chvátal, W. Cook, Implementing the Dantzig–Fulkerson–Johnson algorithm for large traveling salesman problems, Mathematical Programming 97 (1–2) (2003) 91–153.

[2] F. Carrabs, R. Cerulli, J.-F. Cordeau, An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with LIFO or FIFO loading. INFOR, December 2006 (Publication CIRRELT-2007-12).

[3] F. Carrabs, J.-F. Cordeau, G. Laporte, Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading, INFORMS Journal on Computing 19 (2007) 618–632.

[4] G. Clarke, J. Wright, Scheduling of vehicles from a central depot to a number of delivery points, Operations Research 12 (1964) 568–581.

[5] J.-F. Cordeau, M. Iori, G. Laporte, J. Salazar-González, A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. Networks (to appear).

[6] J.-F. Cordeau, G. Laporte, J.-Y. Potvin, M.W. Savelsbergh, Transportation on demand, in: C. Barnhart, G. Laporte (Eds.), Transportation, Handbooks in Operations Research and Management Science, vol. 14, Elsevier, Amsterdam, 2006, pp. 429–466 (Chapter 7).

[7] G. Desaulniers, J. Desrosiers, A. Erdmann, M.M. Solomon, F. Soumis, VRP with pickup and delivery, in: P. Toth, D. Vigo, The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications, vol. 9, 2002, pp. 225–242 (Chapter 9).

[8] K.F. Doerner, G. Fuellerer, R.F. Hartl, M. Gronalt, M. Iori, Metaheuristics for the vehicle routing problem with loading constraints, Networks 49 (4) (2007) 294–307.

[9] A. Felipe, M. Ortuño, G. Tirado, Neighborhood structures to solve the double TSP with multiple stacks using local search, in: Proceedings of FLINS 2008.

[10] M. Gendreau, M. Iori, G. Laporte, S. Martello, A tabu search algorithm for a routing and container loading problem, Transportation Science 40 (3) (2006) 342–350.

[11] M. Gendreau, M. Iori, G. Laporte, S. Martello, A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints, Networks 51 (1) (2008) 4–18.

[12] M. Gendreau, G. Laporte, J.-Y. Potvin, Metaheuristics for the capacitated VRP, in: P. Toth, D. Vigo, The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications, vol. 9, 2002 (Chapter 6).

[13] M. Iori, J. Salazar-González, D. Vigo, An exact approach for the vehicle routing problem with two-dimensional loading constraints, Transportation Science 41 (2) (2007) 253–264.

[14] M. Jünger, G. Reinelt, G. Grinaldi, The traveling salesman problem, in: Network Models, Handbooks in Operations Research and Management Science, vol. 7, Elsevier Science, 1995 (Chapter 4).

[15] B. Kalantari, A. Hill, S. Arora, An algorithm for the traveling salesman problem with pickup and delivery customers, European Journal of Operational Research 22 (3) (1985) 377–386.

[16] H.R. Lourenço, O.C. Martin, T. Stützle, Iterated local search, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, Kluwer Acedemic Publishers, 2003, pp. 321–353 (Chapter 11).

[17] N. Mladenović, P. Hansen, Variable neighborhood search, Computers and Operations Research 24 (11) (1997) 1097–1100.

[18] S. Parragh, K. Doerner, R. Hartl, A survey on pickup and delivery models, Parts I & II, Faculty of Business, Economics and Statistics, Department of Business Administration, University of Vienna, 2006.

[19] H.L. Petersen, Heuristic solution approaches to the double TSP with multiple stacks. Technical Report 2006-2, Centre for Traffic and Transport, Technical University of Denmark, 2006.

[20] S. Røpke, D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, Transportation Science 40 (4) (2006) 455–472.

[21] M. Savelsbergh, M. Sol, The general pickup and delivery problem, Transportation Science 29 (1) (1995) 17–29.

[22] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, in: M. Maher, J.-F. Puget (Eds.), Principle and Practice of Constraint Programming, vol. CP98, Springer-Verlag, 1998.

[23] P. Toth, D. Vigo, VRP with backhauls, in: P. Toth, D. Vigo, The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications, vol. 9, 2002, pp. 195–224 (Chapter 8).