Journal of the Operational Research Society

ISSN: 0160-5682 (Print) 1476-9360 (Online) Journal homepage: https://orsociety.tandfonline.com/loi/tjor20

A tabu search heuristic for a routing problem arising in servicing of offshore oil and gas platforms

I Gribkovskaia, G Laporte & A Shlopak

Journal of the Operational Research Society (2008) 59, 1449–1459 © 2008 Operational Research Society Ltd. All rights reserved. 0160-5682/08

www.palgrave-journals.com/jors

A tabu search heuristic for a routing problem arising in servicing of offshore oil and gas platforms

I Gribkovskaia[1]*, G Laporte[2] and A Shlopak[1]

[1]Molde University College, Molde, Norway; and [2]HEC Montréal, Montréal, Canada

This paper introduces a pickup and delivery problem encountered in servicing of offshore oil and gas platforms in the Norwegian Sea. A single vessel must perform pickups and deliveries at several offshore platforms. All delivery demands originate at a supply base and all pickup demands are also destined to the base. The vessel capacity may never be exceeded along its route. In addition, the amount of space available for loading and unloading operations is limited at each platform. The problem, called the *Single Vehicle Pickup and Delivery Problem with Capacitated Customers* consists of designing a least cost vehicle (vessel) route starting and ending at the depot (base), visiting each customer (platform), and such that there is always sufficient capacity in the vehicle and at the customer location to perform the pickup and delivery operations. This paper describes several construction heuristics as well as a tabu search algorithm. Computational results are presented.

Journal of the Operational Research Society (2008) **59,** 1449–1459. doi:10.1057/palgrave.jors.2602469
Published online 8 August 2007

Keywords: pickup and delivery; vehicle routing; capacitated customers; offshore oil and gas platforms

1. Introduction

This paper investigates a routing problem arising in the logistics of offshore oil and gas platforms in the Norwegian Sea. The problem consists of making deliveries and pickups at several platforms at Haltebanken which is currently Norway's second largest oil and gas production area. These operations are performed by vessels based at Kristiansund located on the west coast of Norway. Large amounts of money are at stake in offshore logistics, and it is therefore critical to plan operations as efficiently as possible. To illustrate the magnitude of the costs involved, renting and operating a supply vessel costs about 18 000 euros per day, while delaying a drilling operation costs approximately 5000 euros per hour. The supply base located at Kristiansund serves nine platforms at Haltebanken: Draugen (DRU), Njord A (NJA), Njord B (NJB), Aasgard A (ASA), Aasgard B (ASB), Heidrun (HEI), West Alpha (WAL), Scarabeo 5 (SCA), and Transocean Searcher (TRS) (see Figure 1). The distance between Kristiansund and HEI is about 135 nautical miles, corresponding to 11 h of travel time.

The platforms must be replenished regularly from the base. The commodities they require come in various types, shapes, sizes, weights and volumes. Similarly, platforms return various objects to the base, namely waste, empty containers and rented equipment. The total yearly amount transported from and to Kristiansund is about 100 000 tons. In this paper we consider only one type of commodity which can be measured in numbers of average size containers.

Delivery and pickup operations are currently performed by three supply vessels, each capable of serving from three to six platforms on a single route, taking into account the vessel capacity and speed limitations. Because planning is currently done for one vessel at a time, this paper will focus on planning the route of a single vessel. In addition to vessel capacities, it is important to also consider platform available capacities. A given platform can only accommodate a limited number of containers because of physical limitations, and also because an accumulation of material onboard a platform can hinder the drilling operations. Capacities vary from one platform to another. Part of the platform deck can be occupied by containers while some part is free space. When a platform places an order it provides the size of its available storage capacity. Whenever a vessel arrives at a platform, it can easily perform its delivery operation provided there is sufficient capacity on the platform. Otherwise, some of the return containers must first be removed in order to create free space. This is only possible if there is also some free space on the vessel. Further details on this problem are provided in Aas *et al* (2007).

Our problem belongs to the class of single vehicle pickup and delivery routing problems with combined demands. It is frequently described as a one-to-many-to-one problem, because all deliveries originate at a common location called the depot, and all pickups are sent back to the depot. The relevant literature on this problem can be found in Gendreau *et al* (1999), Nagy and Salhi (2005), Hoff and Løkketangen (2006), and Gribkovskaia *et al* (2007). The latter paper contains a classification and comparison of several solution types

**Correspondence: I Gribkovskaia, Molde University College, Postbox 2110, N-6402, Molde, Norway.*
E-mail: irina.gribkovskaia@himolde.no

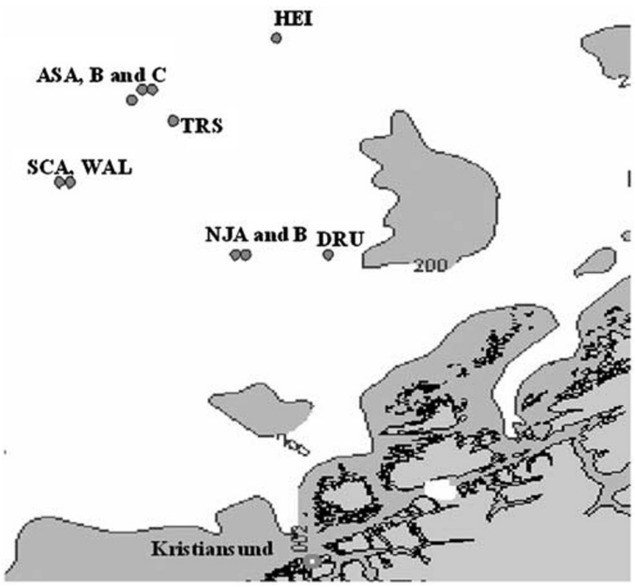

Figure 1 Installations and the supply base.

for the single vehicle one-to-many-to-one pickup and delivery problem. It shows that imposing an *a priori* shape on the solution can be suboptimal. It proposes heuristics capable of generating *general* solutions, that is solutions in which any customer can be visited once or twice. When a customer is visited once, then a pickup and a delivery are performed simultaneously; when a customer is visited twice, the delivery and pickup operations are performed separately. A number of particular solution shapes are of interest. A *lasso* solution consists of a *spoke* rooted at the depot and of a *loop* incident to the end of the spoke. Customers on the spoke are first visited for the deliveries as the vehicle travels from the depot to the loop. Then all customers on the loop are visited once for the simultaneous pickup and delivery services. Finally, the customers of the spoke are visited a second time for pickups as the vehicle returns back to the depot. When the spoke is empty, that is every customer is visited once, the solution is *Hamiltonian*. When a lasso contains only the spoke, the solution is called a *double-path*. In a double-path solution only the last customer on the spoke is visited once for a simultaneous pickup and delivery. With respect to this class of problems, the routing problem considered in this paper is further constrained by the limited available capacity at customer locations. In this context it is particularly important not to impose a solution shape *a priori* because it would unduly restrict the solution space and increase the risk of not identifying a feasible solution or of reaching a suboptimal solution.

Our problem is called the *single vehicle pickup and delivery problem with capacitated customers* (SVPDPCC). It is defined on a graph $G = (V, A)$, where $V = \{0, \dots, n\}$ is the vertex set and $A = \{(i, j) : i, j \in V, i \neq j\}$ is the arc set. Arc (i, j) has a non-negative cost c_{ij} representing the travel time from i to j. Vertices from 1 to n correspond to customers, and vertex 0 is the depot (supply base) at which a vehicle (vessel) of capacity Q starts and ends its trip. Each vertex $i \in V\backslash\{0\}$ has a non-negative pickup demand p_i and a non-negative delivery demand d_i, satisfying $d_i + p_i > 0$. This means that some vertices may have only delivery demands, some may have only pickup demands, and some may have both. Each vertex has a non-negative available capacity C_i at the start of operations. We assume that $C_i \geqslant d_i - p_i$ for every vertex i, for otherwise the problem is infeasible. For the same reason, we assume that $\sum_{i=1}^{n} d_i \leqslant Q$ and $\sum_{i=1}^{n} p_i \leqslant Q$. We will not consider the extreme case $\sum_{i=1}^{n} d_i = \sum_{i=1}^{n} p_i = Q$ and $C_i = 0$ for all vertices, because it will then be practically impossible to perform services when arriving at any vertex.

Vertices can be classified into three categories: category 0 vertices with $d_i = p_i$ and $C_i = 0$; category 1 vertices with $d_i > C_i$; category 2 vertices with $d_i \leqslant C_i$. Vertices of category 0 and 1 can only be visited once for a simultaneous pickup and delivery, and those of category 0 can only be visited when the vehicle is not fully laden. This is obvious because these vertices do not have sufficient available capacity to accept their delivery demand without their pickup demand being collected. Vertices of category 2 can be visited twice on a route. We further assume that neither the delivery demand nor the pickup demand of any vertex can be split between two visits. We denote by n_0, n_1 and n_2 the number of vertices of category 0, 1 and 2, respectively.

The SVPDCC consists of designing a least cost vehicle route starting and ending at the depot, making all pickups and deliveries, such that the vehicle load never exceeds the vehicle capacity along the route, the available capacity at a vertex is always sufficient to perform delivery or simultaneous services, and a fully laden vehicle never arrives at a vertex having zero capacity and equal pickup and delivery demands.

The aim of this paper is to develop a tabu search (TS) heuristic for the SVPDCC. In order to better focus the problem we first present a mathematical model. Construction heuristics and a TS algorithm are described in the next two sections. This is followed by computational experiments and conclusions.

2. Mathematical model

To model the SVPDPCC, we associate with each platform i two vertices i and $i + n$ (a copy). We set $p_{i+n} = p_i$. Two visiting options are allowed for each platform i. The pickup and delivery operations may be performed simultaneously, in which case vertex i is visited and $i + n$ is not visited. Otherwise, platform i is visited twice: delivery is made at vertex i and pickup at vertex $i + n$. To indicate which visiting option is selected for platform i, a binary variable y_i is used, taking value 1 if pickup and delivery are performed simultaneously at platform i in the optimal solution, and value 0 otherwise. To describe in which sequence platforms (or their copies) should be visited on a route, we use binary flow variables x_{ij} equal to 1 if and only if the vessel travels directly from vertex i to vertex j in the optimal solution. Flow variables are not defined for arcs between vertices and their copies reflecting the fact that the second visit at a platform is separated from the first one. We also define a continuous variable u_i equal to an upper bound on the total pickup load in the vehicle upon leaving vertex i and a continuous variable v_i equal to an upper bound on the total delivery load in the vehicle upon leaving vertex i. The model works with an extended cost matrix $\overline{C} = (\bar{c}_{ij})_{(2n+1)\times(2n+1)}$, where

$$\bar{c}_{ij} = \begin{cases} c_{ij} & \text{if } i \leqslant n,\ j \leqslant n \\ c_{i-n,j} & \text{if } i > n,\ j \leqslant n \\ c_{i,j-n} & \text{if } i \leqslant n,\ j > n \\ c_{i-n,j-n} & \text{if } i > n,\ j > n. \end{cases}$$

The model is then as follows:

$$\text{minimize} \sum_{i=0}^{2n} \sum_{j=0}^{2n} \bar{c}_{ij} x_{ij} \tag{1}$$

subject to

$$\sum_{j=0}^{2n} x_{ij} = 1 \qquad (i = 0, \ldots, n) \tag{2}$$

$$\sum_{i=0}^{2n} x_{ij} = 1 \qquad (j = 0, \ldots, n) \tag{3}$$

$$\sum_{j=0}^{2n} x_{ij} = 1 - y_{i-n} \qquad (i = n+1, \ldots, 2n) \tag{4}$$

$$\sum_{i=0}^{2n} x_{ij} = 1 - y_{j-n} \qquad (j = n+1, \ldots, 2n) \tag{5}$$

$$u_0 = 0 \tag{6}$$

$$v_0 = \sum_{i=1}^{n} d_i \tag{7}$$

$$0 \leqslant u_i + v_i \leqslant Q \quad (i = 1, \ldots, 2n) \tag{8}$$

$$u_j \geqslant u_i + p_j y_j - (1 - x_{ij})Q \quad (i = 0, \ldots, 2n;\ j = 1, \ldots, n) \tag{9}$$

$$u_j \geqslant u_i + p_j(1 - y_{j-n}) - (1 - x_{ij})Q \quad (i = 0, \ldots, 2n;\ j = n+1, \ldots, 2n) \tag{10}$$

$$v_j \geqslant v_i - d_j - (1 - x_{ij})Q \quad (i = 0, \ldots, 2n;\ j = 1, \ldots, n) \tag{11}$$

$$C_i \geqslant d_i - p_i y_i \qquad (i = 1, \ldots, n) \tag{12}$$

$$(Q - u_i - v_i) + C_i \geqslant 1 \qquad (i = 1, \ldots, n) \tag{13}$$

$$x_{ij} \in \{0, 1\} \qquad (i, j = 0, \ldots, 2n, i \neq j;\ j \neq i + n \text{ if } 1 \leqslant i \leqslant n;\ j \neq i - n \text{ if } i > n) \tag{14}$$

$$y_i \in \{0, 1\} \qquad (i = 1, \ldots, n) \tag{15}$$

Table 1 Cost matrix

	FBK	*NJA*	*ASB*	*ASC*	*WAL*
FBK	0	360	620	620	590
NJA	360	0	255	260	240
ASB	620	255	0	10	65
ASC	620	260	10	0	75
WAL	590	240	65	75	0

Table 2 Pickup and delivery demands, and available capacities

i	p_i	d_i	C_i
NJA	10	10	25
ASB	39	39	59
ASC	40	40	0
WAL	10	10	80

In this formulation constraints (2)–(5) are degree constraints. Constraints (6) and (7) initialize the pickup and delivery loads. Constraints (8) guarantee that the vessel load never exceeds its capacity. Constraints (9)–(11) control the pickup and delivery load in the vessel after each customer location visit. As in Desrochers and Laporte (1991), they also eliminate subtours. Constraints (12) ensure that the platform capacities are never exceeded. Constraints (13) prevent infeasible situations in which the vessel would arrive fully laden at a location with no free storage space, and the amounts to be picked up and deliver would be the same. More specifically, these constraints state that the amount of free space on the vehicle, $Q - u_i - v_i$, and at the customer location, C_i, cannot both be zero. Constraints (14) and (15) force the x_{ij} and y_i variables to be binary.

The following example based on the real instance of Figure 1 illustrates how customer capacities can effect the solution shape. The data, which are extracted from Aas *et al* (2007), were provided by the Norwegian oil company Statoil. There are four platforms with registered demands (identical for pickup and delivery), and the vessel with the capacity of 99 containers. Tables 1 and 2 provide the cost matrix, the demands and the available capacity at each platform. As can be seen, the vessel is fully laden upon departure from the depot located at Kristiansund (FBK).

If we disregard platform capacities and solve the problem as a pure Single Vehicle Routing Problem with Pickups and Deliveries (Gribkovskaia *et al*, 2007), that is, with constraints (12) and (13) relaxed, we obtain the Hamiltonian solution (FBK, NJA, ASC, ASB, WAL, FBK) with travel time of 1285 min. However, this solution is infeasible because on arrival at platform ASC the vessel will be fully laden and there will be no free space on the platform deck, making it impossible to perform the services. Including constraints (12) and (13) yields the non-Hamiltonian optimal solution (FBK, NJA, ASB, ASC, ASB, WAL, FBK) with travel time of 1290 min (Figure 2). On this route the platform ASB is visited twice. At the first visit to this platform only the delivery is performed,

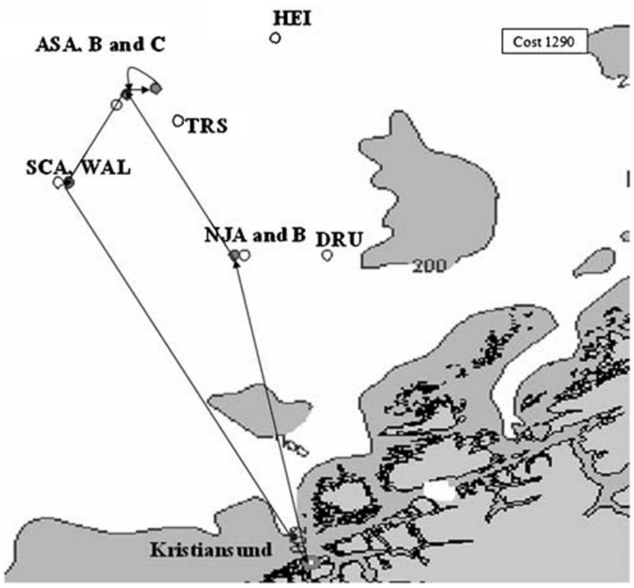

Figure 2 Optimal solution for the real instance depicted in Figure 1, showing two visits at platform ASB. The depot is located in Kristiansund.

thus creating free space for 39 containers on the vessel. When ASC is visited next on the route, both services can now be performed, leaving the same capacity for 39 containers on the vessel. On the second visit to ASB the available space on the vessel is used to collect the pickup demand at that location.

3. Construction heuristics

We have developed several construction heuristics for the SVPDPCC. In what follows, a solution is *load-feasible* if the vehicle capacity is never exceeded. It is *storage-feasible* if none of the vertices belonging to categories 0 or 1 is visited twice. It is *operational-feasible* if a fully laden vehicle never serves a vertex with no available capacity (category 0 vertex).

3.1. Description of the construction heuristics

We now describe the construction heuristics we have developed for the SVPDPCC. Their aim is to provide a load-feasible and storage-feasible initial solution for the TS heuristic. All construction heuristics apply the same three basic steps.

Step 1: *Construction of a spoke*
Only consider the vertices that can be visited twice (category 2). Start from the depot and connect these using a nearest neighbour (NN) rule (Rosenkrantz *et al*, 1977) or a cheapest insertion (CI) rule (Mole and Jameson, 1976). Follow the path in the reverse direction to obtain a spoke.

Step 2: *Construction of a loop*
Now consider the remaining single-visit vertices (category 0 and 1) and construct a loop which will be attached to the end vertex of the spoke generated in Step 1. Two variants of this procedure were tested:

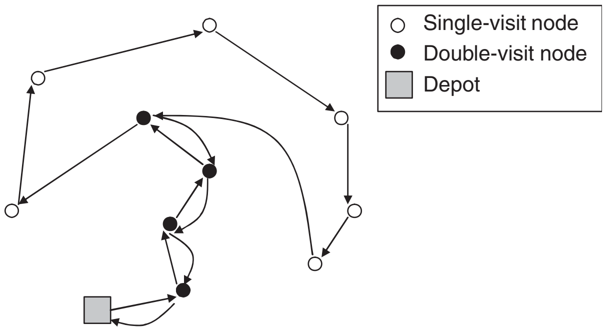

Figure 3 Shape of the initial solution ($n_0 + n_1 > 1$ and $n_2 > 0$). Four vertices are visited twice and six vertices are visited once.

(1) Start from the end vertex of the spoke. First, insert all vertices with positive net delivery demand $d_i - p_i$; then insert vertices of category 0; finally insert vertices with negative net delivery demand. This procedure can be based on the NN or on the CI rule.
(2) Use Mosheiov's (1994) PDαT algorithm: construct, without considering the vehicle capacity, a cycle including all single-visit vertices (categories 0 and 1) and the end vertex of the spoke, which is then treated like an artificial depot, and reinsert it in this cycle to make the route load-feasible.
The resulting solution will be a lasso. If $n_0 + n_1 \leqslant 1$, it will be a double-path, and if $n_2 = 0$ it will be a Hamiltonian circuit. This solution is always storage-feasible, but in a very extreme case it may be operational-infeasible (see below). Figure 3 illustrates a solution obtained after executing Steps 1 and 2 on an artificial instance.

Step 3: *Backward merging*
Backward merging of vertices on the spoke is applied in order to decrease the cost of the initial solution while maintaining feasibility. Starting from the end vertex of the spoke, we try to delete the second visit to a current vertex and check whether this is load-feasible and operational-feasible. If the deletion of the second visit is load- and operational-feasible, we perform it, and proceed towards the depot to the next vertex on the spoke. An illustration of a route after the backward merging is given in Figure 4.
Different versions of the construction heuristics can be obtained: the spoke can be constructed with the use of CI; the loop can be constructed by any of the two variants described above, and in each of them we can use NN or CI. We have programmed and tested the three following versions: **Con1**: NN + 1NN + B; **Con2**: NN + 2CI + B; **Con3**: CI + 2CI + B. The two first letters stand for the ways of constructing a spoke. In the next field, 1 corresponds to the first procedure of loop construction, and 2 to the second procedure; the two letters that follow indicate which

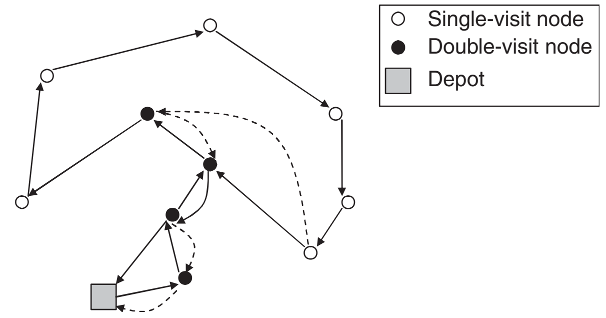

Figure 4 Effects of backward merging using the graph of Figure 3. Now only two vertices are visited twice. The dotted lines correspond to the old route while the full lines correspond to the new route.

heuristic rule is used for loop construction. The final letter B stands for backward merging.

We emphasize the following properties of our construction heuristics. If we use the first procedure for the construction of a loop, we will always obtain an operational-feasible solution assuming one exists. This is so because category 0 vertices are visited when the vehicle is least loaded. Therefore there should be at least one unit of free capacity on the vehicle, for otherwise the problem is infeasible. If we use the second procedure for loop construction, an operational-infeasible solution may be constructed. This can occur if $n_2 = 0$ and $n_0 > 0$. Another extreme infeasible case arises when $n_2 > 0$ but $\sum_{i \in V_2} d_i = 0$ or $\sum_{i \in V_2} p_i = 0$ (where $V2$ is the set containing all category 2 vertices), and $n_0 > 0$. These two cases are very extreme, and we did not consider such instances in our computational experiments. We have therefore always generated operational-feasible solutions.

4. TS heuristic

Once a solution has been constructed, we improve it using TS, a method originally proposed by Glover (1986). TS is a local search metaheuristic that explores the solution space S by moving at each iteration from the current solution s of cost $c(s)$ to the best solution in a subset $M(s)$ of its neighbourhood $N(s)$, defined as all solutions that can be reached by applying a modification to solution s. Since the objective function may deteriorate during the search, anti-cycling rules must be implemented, that is forbidding moves that revoke the effect of recent moves by declaring them tabu. The number of iterations for which a move is declared tabu is called the tabu tenure. It can be fixed or modified dynamically during the search process. An aspiration criterion is used to allow tabu solutions that improve over the best known solution s^*. The most commonly used stopping criteria are: (1) a fixed number of iterations (or a fixed amount of CPU time); (2) a fixed number of iterations without improvement in the objective function value; (3) when the objective reaches a prespecified threshold value. Additional features have to be included in the search strategy to make it fully effective.

Our TS implementation modifies the algorithm developed by Gribkovskaia *et al* (2007), which is based on the Unified Tabu Search Algorithm (UTSA) of Cordeau *et al* (2001). This algorithm has proved to be one of the most successful TS algorithms for the *Vehicle Routing Problem* and some of its variants. One important feature of UTSA is the consideration of infeasible solutions during the search process. Solutions do not have to satisfy capacity or route length restrictions, but penalty terms for their violations are present in the objective function which has the form: $f(s) = c(s) + \alpha q(s) + \beta d(s)$, where $c(s)$ is the routing cost of solution s, $q(s)$ and $d(s)$ are total load and duration violations in this solution, and α and β are positive parameters that dynamically self-adjust depending on the feasibility of previous solutions. Other important features of UTSA are a continuous diversification mechanism, and a route reoptimization procedure. These ideas are present in our algorithm with some modifications. We now discuss the main features of our TS algorithm.

Solution s: Each solution s represents a route in which vertices are visited once or twice. It is important to temporarily allow solutions to be load- or operational-infeasible.

Load feasibility violations: Load feasibility is checked whenever a vertex is visited. The total load infeasibility of a route is equal to the sum of load infeasibilities of all its vertices.

Operational feasibility violations: Operational feasibility means that vertices with zero capacity must not be visited by a fully laden vehicle. The total operational feasibility violation of a route is the number of such vertices.

Storage feasibility violations: We do not allow vertex capacity violations in our algorithm. Before solving an instance, we identify vertices that can be visited only once (these are not dependent on routing).

Penalized objective function: For a solution $s \in S$, let $c(s)$ denote the total routing cost, let $q(s)$ denote the total load feasibility violation of the route, and let $z(s)$ denote the total operational feasibility violation of the route. Solutions $s \in S$ are evaluated with the help of the penalized cost function $f(s) = c(s) + \alpha q(s) + \pi z(s)$, where α and π are positive parameters. The value of the parameter α is dynamically adjusted based on the recent history of the search. At each iteration, the value of a α is modified by a factor $(1+\delta) > 1$, where δ is a positive parameter. If the current solution is load-feasible, the value of α is divided by $(1+\delta)$; otherwise, it is multiplied by $(1+\delta)$. We use a similar rule for parameter π.

Attributes: Another useful ingredient of our TS algorithm is the presence of attributes (i, v). Let v denote the number of visits at vertex i. With each solution s is associated an attribute set $B(s) = \{(i, v) : i = 1, \ldots, n, v = 1, 2\}$ which indicates how many times each vertex is visited. Attributes are used to perform moves, control tabu status, and implement a diversification technique.

Neighbourhood $N(s)$ and definition of a move: The neighbourhood $N(s)$ of solution s is defined by all solutions that can be reached from s by changing the number of visits at one category 2 vertex. A transition from the current solution s to the solution $s' \in N(s)$ is called a move. The move can be expressed by the removal of attribute (i, v) from the set $B(s)$ and the addition of attribute (i, v') to $B(s')$, $(v \neq v')$.

In our problem, some vertices can only be visited once *a priori* (those that belong to categories 0 and 1). Therefore moves can only be applied to some of the category 2 vertices. We use the letter ϕ_i to define the *status* of vertex i, that is the number of times it is visited in the current solution. There are two types of moves because ϕ_i can change from 1 to 2 or from 2 to 1. Moves are performed as follows.

1. *Insertion of the second visit at vertex i, $\phi_i = 1$*: Suppose that in the solution s vertex i is visited once, that is $(i, 1) \in B(s)$. The second visit to vertex i is inserted in the route minimizing the increase in the penalized function $f(s') = c(s') + \alpha q(s') + \pi z(s')$. Hence, $(i, 2) \in B(s')$. The purpose of visiting vertex i twice is to obtain a solution s' with lower total load infeasibility than that of solution s. However, visiting the same vertex twice entails a higher routing cost. That is why the insertion of a second visit to vertex i is based on the penalized function.
2. *Deletion of the second visit at vertex i, $\phi_i = 2$*: Suppose that in the solution s vertex i is visited twice, that is $(i, 2) \in B(s)$. A neighbour solution s' is obtained by deleting the second visit to vertex i from the route maximizing the decrease in $f(s')$, and reconnecting its predecessor and successor. The deletion of the second visit to i implies that $(i, 1) \in B(s')$. In solution s' the vertex i is visited in the same order as its first visit in solution s: If costs satisfy the triangle inequality, deleting the second visit at a vertex will never yield a routing cost increase. However, it may lead to an increase in load infeasibility or to an operational-infeasible solution.

Tabu status of an attribute: We use the attributes to control tabu statuses instead of maintaining an actual tabu list. If the number of visits v at vertex i is changed to v', reverting to v visits at vertex i is forbidden for the next θ iterations. This is done by assigning a tabu status to the attribute (i, v) for the next θ iterations. The tabu tenure θ is an integer randomly selected at each iteration within the interval $[1; \bar{\theta}]$, where $\bar{\theta}$ is a user-controlled parameter.

Aspiration criterion: The assignment of a tabu status to attributes may sometimes be too restrictive, that is, good moves can turn out to be forbidden. To rectify this, we introduce a rule that a tabu status of an attribute (i, v) can be revoked if this would lead to a feasible solution of a smaller cost than that of the best known solution having that attribute. In other words, this move has to result in a solution $s' \in N(s)$ with $q(s') = 0$, $z(s') = 0$ and $c(s') < \sigma_{iv}$, where σ_{iv} is an aspiration level of attribute (i, v). The initial set of σ_{iv} is equal to $c(s)$ if (i, v) belongs to the attribute set of the feasible initial solution and to ∞ otherwise. Every time a feasible solution s is identified, the aspiration level of each of its attributes (i, v) is updated to $\min\{\sigma_{iv}, c(s)\}$.

Admissible subset $M(s)$ of neighbour solutions: At each iteration, the subset $M(s) \subseteq N(s)$ is formed from solutions $s' \in N(s)$ for which the corresponding attribute $\{(i, v') : \phi_i = 2\} \in B(s') \backslash B(s)$ is not tabu, or from solutions s' that are feasible and for which the value of $c(s')$ is less than the aspiration level of attribute (i, v').

Diversification: Without a diversification strategy, the evaluation of the best solution $s' \in M(s)$ would be based on the penalized function f. However, we wish to diversify the search, that is, give a higher chance of being selected to solutions $s' \in M(s)$ having an attribute (i, v') that has not been frequently present in past solutions. The mechanism operates as follows: any solution $s' \in M(s)$ such that $f(s') \geqslant f(s)$ is penalized with a term $p(s')$ proportional to the addition frequency ρ_{iv} of the modified attribute, value of $c(s)$, and parameter ζ. Let ρ_{iv} denote number of times attribute (i, v) has been added to the solution and let t denote tabu iteration counter, where the parameter ζ is used to control the intensity of the diversification. Then the penalty term is $p(s') = \zeta c(s) \rho_{iv} / t$. If $f(s') < f(s)$, we assume that $p(s') = 0$. Finally, the selection of the best solution $s' \in M(s)$ is based on a generalized function $g(s') = f(s') + p(s')$.

Intra-route reoptimization: One more valuable mechanism incorporated in our search algorithm is an *intra-route reoptimization* procedure which attempts to generate a better solution by changing the sequence of vertices in the current route without changing the statuses of vertices. The implementation of an intra-route reoptimization in our algorithm is made through the following reinsertion procedure which is applied to every vertex with the status one in the current route:

Step 1: Delete the current vertex if this does not change the number of visits to other vertices. The predecessor and the successor of the deleted vertex are connected together.

Step 2: Reinsert this vertex in the route minimizing the penalized function f, that is without requiring feasibility of the resulting route. The vertex can be reinserted in the same position from which it was deleted.

Step 3: Consider another vertex on which this procedure has not been applied yet, go to Step 1 and repeat until all vertices are checked.

The intra-route reoptimization procedure is applied every time a new best solution is found, or every φth iteration.

Construction of an initial solution: The constructive heuristics **Con1**, **Con2** and **Con3** described above were used for the generation of the initial solutions for our TS algorithm. We stress that these solutions are always load-feasible and storage-feasible. In addition, **Con1** guarantees the operational feasibility of the initial solutions. If the construction heuristic yields an operational-infeasible solution, it is possible that operational feasibility will be restored during the tabu iterations.

We now provide a detailed description of our TS algorithm.

Notation used in the description of the tabu search algorithm

(i, v)	Attribute: number of visits v at vertex i
ϕ_i	Status of vertex i
$B(s)$	Attribute set of solution s
$c(s)$	Routing cost of solution s
$f(s)$	Routing cost plus penalties for constraint violations in solution s
$g(s)$	Function f plus diversification penalty term in solution s
$q(s)$	Total load violation of solution s
$z(s)$	Operational feasibility violation of solution s
$N(s)$	Neighbourhood of solution s
$M(s)$	An “admissible” subset of $N(s)$
$s, s', \tilde{s}$	Solutions
s_0	Initial solution
s^*	Best solution identified
α	Penalty factor for overload
δ	Parameter used to update α
π	Penalty factor for operational infeasibility
π_0	Initial value of π
ζ	Factor used to adjust the intensity of diversification
η	Total number of iterations to be performed
θ	Tabu tenure
$\bar{\theta}$	Upper bound value for the tabu tenure
t	Iteration counter
ρ_{iv}	Number of times attribute (i, v) has been added to the solution
σ_{iv}	Aspiration level of attribute (i, v)
τ_{iv}	Last iteration for which attribute (i, v) is declared tabu
φ	Parameter used to adjust the intensity of intra-route reoptimization

Our TS algorithm starts from an initial solution s_0 obtained with the use of one of our construction heuristics. The search process is defined by six parameters $\delta, \zeta, \bar{\theta}, \pi, \eta, \varphi$, and returns after execution the best feasible solution found s^*, if any.

```
Step 1: Set $s := s_0$, $\alpha = 1$. If $s$ is operational-feasible ($z(s) = 0$), set $s^* := s$.
Step 2: For every $(i, v)$, do
    • Set $\rho_{iv} := 0$, $\tau_{iv} := 0$.
    • If $(i, v) \in B(s)$ and $s$ is feasible, set $\sigma_{iv} := c(s)$; else, set $\sigma_{iv} := \infty$.
Step 3: For $t = 1, \ldots, \eta$, do
    a. Update $\delta, \zeta, \theta$:
        • $\delta :=$ random value from the interval $(0, 1)$.
        • $\zeta :=$ random value from the interval $(0, 1)$.
        • $\theta :=$ random integer value from the interval $[1, \bar{\theta}]$.
    b. Set $N(s) := \emptyset$.
    c. For each attribute $(i, v') \notin B(s)$ such that $\phi_i = 2$ do
        • Create a solution $s'$ applying a move definition: replace the corresponding attribute $(i, v) \in B(s)$ by attribute $(i, v')$, ie $B(s) := B(s) \setminus \{(i, v)\}$ and $B(s') := B(s) \cup \{(i, v')\}$.
        • Set $N(s) := N(s) \cup \{s'\}$.
    d. Set $M(s) := \emptyset$.
    e. For each $s' \in N(s)$ do
        • For $(i, v) \in B(s') \setminus B(s)$ such that $\tau_{iv} < t$ or such that $c(s) + (f(s') - f(s)) < \sigma_{iv}$, set $M(s) := M(s) \cup \{s'\}$.
    f. For each $s' \in M(s)$, do
        • If $f(s') \geqslant f(s)$, set $g(s') := f(s') + \zeta c(s) \rho_{iv} / t$; else, set $g(s') := f(s')$.
    g. Identify a solution $s' \in M(s)$ minimizing $g(s')$.
    h. For attribute $(i, v) \in B(s') \setminus B(s)$ do
        • Set $\rho_{ik} := \rho_{ik} + 1$ and $\tau_{iv} := t + \theta$.
    i. If $s'$ is feasible ($q(s') = 0$ and $z(s') = 0$), do
        • If $c(s') < c(s^*)$, set $s^* := s'$.
        • For each $(i, v) \in B(s')$, set $\sigma_{iv} := \min\{\sigma_{iv}, c(s')\}$.
        • Set $\alpha := \alpha/(1 + \delta)$; else, set $\alpha := \alpha(1 + \delta)$.
    j. Set $s := s'$.
    k. If $q(s) = 0$ and $z(s) = 0$ and $c(s) = c(s^*)$, or $t = k\varphi, k = 1, 2, \ldots$, do intra-route reoptimization:
        • For each $(i, 1) \in B(s)$ do
            (i) Remove vertex $i$ from its route in solution $s$ and reinsert vertex $i$ in a route in solution $\tilde{s}$ minimizing $f(\tilde{s}) = c(\tilde{s}) + \alpha q(\tilde{s}) + \pi z(\tilde{s})$ such that $B(s) = B(\tilde{s})$.
            (ii) Set $s := \tilde{s}$.
        • If both $q(s) = 0$ and $z(s) = 0$ and $c(s) < c(s^*)$, set $s^* := s$.
    l. Set $t := t + 1$.
```

5. Computational experiments

We have programmed our heuristics in Pelles C for Windows, and run them on a PC with 1.6 GHz Intel Centrino processor and 248 Mb RAM, running under Windows XP.

Table 3 Characteristics of A-instances

Set	μ	*Average* ω
A1	2.2	86.4
A2	1.0	61.1
A3	0.7	40.4
A4	0.6	22.4

Table 4 Characteristics of B-instances

	μ			
Set	*Region* 1	*Region* 2	*Region* 3	*Average* ω
B1	2.2	2.2	2.2	86.3
B2	0.6	1.0	2.2	60.9

5.1. *Test instances*

We have generated 119 test instances: four sets of A-instances, two sets of B-instances, and one set of C-instances. Each set contains 17 instances of different sizes. All instances are derived from the *Capacitated Vehicle Routing Problem* (CVRP) instances of VRPLIB: http://www.or.deis.unibo.it/research_pages/ORinstances/VRPLIB/VRPLIB.html. We have selected 17 CVRP instances containing between 16 and 101 vertices: E-016-03, E-021-04, E-022-04, E-023-03, E-026-08, E-030-03, E-031-09, E-033-04, E-036-11, E-041-14, E-045-04, E-048-04, E-051-05, E-072-04, E-076-07, E-101-08, E-101-10. The instances are generated in the plane with Euclidian distances. Delivery and pickup demands and the amount of available capacity at vertices were generated in different ways, yielding the following three major groups of instances.

5.1.1. A-instances. We have generated four sets of A-instances, 17 in each, with different average percentage of category 2 vertices. Delivery and pickup demands were generated as follows. Let q_i be the demand of vertex i in the original CVRP instance. We then set $d_i = q_i$ and $p_i = [(1-\beta)q_i]$ if i is odd or $p_i = [(1+\beta)q_i]$ if i is even, where $0 \leqslant \beta < 1$. We have selected $\beta = 0.20$. The amount of available capacity at the vertices was generated as follows:

$$C_i = \lambda[\mu(d_i + p_i) - \max\{0, d_i - p_i\}] + \max\{0, d_i - p_i\}$$

where $0 \leqslant \lambda < 1$ is a random number, and $\mu > 0$ is a user-specified parameter. This generation process yields C_i values within the interval $[\max\{0, d_i - p_i\}, \mu(d_i + p_i)]$. By using different values for the coefficient μ, four sets of instances were obtained (Table 3). Let ω be the percentage of category 2 vertices in an instance. We have generated A-instances to investigate the difference in solution shapes between more and less constrained instances.

5.1.2. B-instances. The B-instances have the same delivery demands as the A-instances: $d_i = q_i$ for $i = 1, \ldots, n$. To generate the pickup demands, the vertices were first ranked in increasing distance from the depot. For the first third of vertices (region 1) we set $p_i = 1.2q_i$. For the second third (region 2) p_i was generated as for the A-instances. For the last third (region 3) $p_i = 0.8q_i$. We have generated the amount of available capacity at the vertices by using the same formula as for the A-instances, but we have used two different ways of choosing values for parameter μ to obtain two sets of B-instances (Table 4). When generating B-instances, our idea was that in set B1 there would be a higher likelihood of obtaining more non-Hamiltonian solutions than, for example, in set A1. In particular, we wanted to understand the impact of having less available capacity at vertices. This is why we have generated the set B2.

5.1.3. C-instances. We have generated one set of 17 highly constrained C-instances. These have the same delivery demands (one may be different, and this will be explained below) as the A-instances: $d_i = q_i$ for $i = 1, \ldots, n$. Pickup demands are again generated with respect to vertex location. We use the same partition of the vertices into three regions, as for the B-instances. In region 1 we set $p_i = d_i$ and $C_i = 0$ for all vertices i. In regions 2 and 3 delivery and pickup demands were generated as in the A-instances, and available capacities at vertices were generated with the coefficient $\mu = 2.2$. It is important to note that in the C-instances we also require that $\sum_{i=1}^{n} p_i = \sum_{i=1}^{n} d_i$. We achieve this by adjusting the pickup or the delivery demand at the vertex most remote from the depot: we compute the sum of delivery demands of all other vertices, and compare it with that of their pickup demands. If the former is larger, we assign the pickup demand at this vertex to this difference, otherwise we set its delivery demand equal to this difference. We have generated C-instances in order to study the nature of operational infeasibility. These instances are very constrained and contain a high proportion of category 0 vertices.

In all A-, B- and C-instances, we set $Q = \max\{\sum_{i=1}^{n} p_i, \sum_{i=1}^{n} d_i\}$. The test instances are available on the following website: http://neumann.hec.ca/chairedistributique/data/svpdpcc.

5.2. *Fine tuning of the TS heuristic*

In all tables that follow, the gap is measured with respect to a lower bound. Let s^* be the best solution found by the algorithm, $\underline{s}$ be the solution of a Traveling Salesman Problem (TSP) defined on G, and $c(s)$ be the value of solution s. Then $c(\underline{s}) \leqslant c(s^*)$. The gap is the percentage ratio between $c(s^*)$ and $c(\underline{s})$: gap $= 100[c(s^*) - c(\underline{s})]/c(\underline{s})$. The TSPs are solved by means of the Concorde solver available at http://www.tsp.gatech.edu/concorde.html.

5.2.1. Initial solution. We have first compared the three construction heuristics used to initialize the TS. The gap values obtained with our construction heuristics are given in

Table 5 Average gap values obtained by three construction procedures

Set	*Con1*	*Con2*	*Con3*
A1	51.53	48.80	49.67
A2	70.98	54.83	61.05
A3	82.77	55.57	61.24
A4	84.18	50.77	59.42
Average A	72.37	52.49	57.85
B1	46.80	49.77	50.37
B2	59.39	48.22	55.49
Average B	53.09	49.00	52.93
C	66.83	53.24	60.26
Average	66.07	51.60	56.79

Table 5. Solution times are negligible. The **Con2** heuristic is the best but it is worth remembering that only **Con1** guarantees operational-feasible solutions. However, even for the most constrained C-instances none of our construction heuristics yields an infeasible solution.

5.2.2. Search parameters. In our implementation of the TS heuristic, we have randomly selected δ and ζ in (0,1). These values seem to be relatively stable irrespective of the application (Cordeau *et al*, 2001). We have decided to concentrate on finding good values for the intra-route reoptimization frequency φ, the maximal tabu tenure $\bar{\theta}$, and the penalty factor π for operational infeasibility.

Intra-route reoptimization frequency: In order to determine the best intra-route reoptimization frequency, the TS algorithm was initiated with **Con2**, as it gives the best initial solutions out of the construction heuristics, the tabu tenure was chosen randomly from the interval $[1, [12 \log n]]$, and the operational-infeasibility penalty factor was fixed at value $\pi = \bar{c}n$, where $\bar{c}$ is the average value of the cost matrix. All the instances were run for 10^4 iterations.

We believe that the intra-route reoptimization frequency has to be dependent on the percentage of category 2 vertices in an instance. The reasoning is the following. Our TS algorithm is able to perform moves (changing a number of visits) only on category 2 vertices. If there are several of them in a given instance, then the solution space is larger. The only way to change the position of single-visit vertices is during the intra-route optimization phase. Therefore, we have decided to inversely relate intra-route reoptimization frequency to the value of ω.

We have compared several variants of intra-route reoptimization patterns. Five variants were defined by using different relationships between φ and ω. These are shown in Table 6.

One can observe from this table that in variant 4 the value of φ is not dependent on ω, but set equal to 10. We found that variant 1 performs the best in terms of the average gap. We also observed that in general, for the more constrained

Table 6 Relationship between ω and ϕ for five tested variants

	φ				
ω	*Variant* 1	*Variant* 2	*Variant* 3	*Variant* 4	*Variant* 5
$\geqslant 95$	5	10	20	10	10
$\geqslant 75$	4	8	16	10	10
$\geqslant 55$	3	6	12	10	8
$\geqslant 35$	2	4	8	10	3
< 35	1	2	4	10	3

instances, the best solutions are more often produced during the intra-route reoptimization phase, and not when performing a tabu move. This is true even if φ is constant (variant 4). This in a way confirms our hypothesis that for more constrained instances we should perform intra-route reoptimization more frequently. However, variant 1 took on average longer than the other variants. This behavior was expected since intra-route reoptimization is performed a larger number of times in variant 1 than in the other variants.

Tabu tenure: Having decided to use **Con2** to initialize the search and variant 1 for the dependence of φ over ω, we have attempted to select the best tabu tenure in the interval $[1, \bar{\theta}]$. We have experimented with several values of $\bar{\theta}$: $[12 \log n]$, $[8 \log n_2]$, $[12 \log n_2]$, $[18 \log n_2]$, $[25 \log n_2]$. The last value was not tested for 10^4 iterations, but for 10^5 iterations. We also attempted a different approach, as proposed in Hoff *et al* (2006), and we selected the tabu tenure from intervals $[n/8, n/4]$ and $[n_2/8, n_2/4]$. The best value of average gap among the variants tested for 10^4 iterations is obtained with $\bar{\theta} = [18 \log n_2]$.

Value of the parameter π: To determine the best value of π we again use **Con2** to generate initial solutions, variant 1 for the dependency of φ over w, and $\bar{\theta} = [18 \log n_2]$. In the previous tests, we were using a static value $\bar{c}n$ for parameter π, which was the best one among the static values. We have also applied to π the same adjustment mechanism as for α. We then tested a dynamic self-adjustment mechanism, initiated from $\pi_0 = \bar{c}n$, which proved to perform better.

5.3. *Test results*

In Table 7 we provide the results of the execution of our TS algorithm with the best identified parameters, starting from different initial solutions. These results indicate that **Con2** is better for A-instances, but **Con3** is better for B- and C-instances. The final tests were carried out with **Con2** which is the overall best initialization procedure. The average computation time over all instances was 46 s (48 s for A-instances, 42 s for B-instances, 47 s for C-instances). Computation time is proportional to the number of iterations.

The average gap values observed are reasonably small considering the fact that TSP optimal solution value underestimates that of the SVPDPCC. In fact, the optimal TSP solution is often infeasible for the SVPDPCC: this occurred 64.7% of

Table 7 Average gap values obtained by TS algorithm with three initialization procedures

Set	*Con1*	*Con2*	*Con3*
A1	4.43	4.40	3.65
A2	7.35	4.22	5.69
A3	7.43	5.25	5.08
A4	8.25	5.31	7.62
Average A	6.86	4.80	5.51
B1	5.27	5.65	5.31
B2	5.49	5.69	5.33
Average B	5.38	5.67	5.32
C	13.29	13.36	12.33
Average	7.36	6.27	6.43

Table 8 Percentage of non-Hamiltonian solutions found for each instance set

Instance set	*A1*	*A2*	*A3*	*A4*	*B1*	*B2*	*C*
Percentage of non-Hamiltonian solutions	11.8	17.6	17.6	17.6	47.1	17.6	35.3

the time for A-instances, 88.2% of the time for B-instances and for all C-instances.

Finally, we have analysed the shapes of solutions obtained with our TS heuristic. Table 8 gives for each instance set the percentage of non-Hamiltonian solutions among the best solutions generated by the heuristic.

Looking at A-instances, one can observe that, contrary to what could have been expected, the proportion of non-Hamiltonian solutions is not dependent on how constrained an instance is. The proportion of non-Hamiltonian solutions in A-instances is, in general, relatively small because it is often possible to perform a simultaneous pickup and delivery at all vertices. In contrast, in B-instances, vertices located close to the depot have large pickup demand compared to their delivery demand which means that these two operations may not be performed during the same visit. In this respect, the set B2 differs drastically from B1. In B2, the instances are more constrained and contain fewer vertices that can be visited twice. But what is more important is where vertices with limited available capacity are located. On purpose we have put less capacity at vertices most likely be visited twice, that is those vertices located close to the depot and having large pickup and small delivery demands. By so doing, we have reduced the likelihood of generating non-Hamiltonian solutions. This is confirmed by our results: in B1 the proportion of non-Hamiltonian solutions is 47.1% while in B2 it is only 17.6%.

Our expectations also came true regarding C-instances. We have identified a high proportion of non-Hamiltonian solutions among the best ones. We have on purpose constructed C-instances with several vertices that cannot be visited with a fully laden vehicle, located close to the depot, and a vehicle leaving with a full load. As a result, the vehicle cannot go to any of these vertices just after leaving the depot. And we have also imposed that the sum of delivery demands be equal to the sum of pickup demands, so that the vehicle would be very likely to come back to the region close to the depot (region 1) with a maximum load. In order to obtain a feasible solution, the vehicle first has to go to a vertex where it can free some space onboard, then service vertices with no available capacity, and then probably come back to the first vertex of its route to make a pickup if only a delivery was performed in the first visit.

6. Conclusions

We have introduced the SVPDPCC, a problem encountered in the servicing of offshore oil and gas platforms. We have developed several construction heuristics as well as a TS algorithm for this problem. Our best heuristic consists of applying TS starting with an initial solution generated with **Con2** which combines NN, CI and backward merging procedures. The solutions produced by our algorithm may be Hamiltonian or not, depending partly on vertex demands and relative locations.

Acknowledgements—This work was partly supported by the Norwegian Research Council and by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant 39682-05. This support is gratefully acknowledged. Thanks are also due to Bjørnar Aas who introduced the problem to the authors and provided valuable information, and to two referees for their valuable comments.

References

Aas B, Gribkovskaia I, Halskau Sr Ø and Shlopak A (2007). Routing of supply vessels to petroleum installations. *Int J Phys Distrib Logist Mngt* **37**: 164–179.

Cordeau J-F, Laporte G and Mercier A (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *J Opl Res Soc* **52**: 928–936.

Desrochers M and Laporte G (1991). Improvements and extensions to the Miller–Tucker–Zemlin subtour elimination constraints. *Opns Res Lett* **10**: 27–36.

Gendreau M, Laporte G and Vigo D (1999). Heuristics for the traveling salesman problem with pickup and delivery. *Comput Opns Res* **26**: 699–714.

Glover F (1986). Future paths for integer programming and links to artificial intelligence. *Comput Opns Res* **13**: 533–549.

Gribkovskaia I, Halskau Sr Ø, Laporte G and Vlček M (2007). General solutions to the single vehicle routing problem with pickups and deliveries. *Eur J Opl Res* **180**: 568–584.

Hoff A and Løkketangen A (2006). Creating lasso-solutions for the traveling salesman problem with pickup and delivery by tabu search. *Central European J Opns Res* **14**: 125–140.

Hoff A, Laporte G, Løkketangen A and Gribkovskaia I (2006). Lasso solution strategies for the vehicle routing problem with pickups and deliveries. *Eur J Opl Res*, submitted for publication.

Mole RH and Jameson SR (1976). A sequential route-building algorithm employing a generalized savings criterion. *Opl Res Quart* **27**: 503–511.

Mosheiov G (1994). The travelling salesman problem with pick-up and delivery. *Eur J Opl Res* **79**: 299–310.

Nagy G and Salhi S (2005). Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. *Eur J Opl Res* **162**: 126–141.

Rosenkrantz D, Stearns R and Lewis P (1977). An analysis of several heuristics for the traveling salesman problem. *SIAM J Comput* **6**: 563–581.

Received October 2006;
accepted April 2007 after two revisions