European Journal of Operational Research 223 (2012) 60–75

Contents lists available at SciVerse ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Multiple pickup and delivery traveling salesman problem with last-in-first-out loading and distance constraints

Brenda Cheang[a], Xiang Gao[b], Andrew Lim[b], Hu Qin[a,*], Wenbin Zhu[c]

[a] School of Management, Huazhong University of Science and Technology, No. 1037, Luoyu Road, Wuhan, China
[b] Department of Management Sciences, City University of Hong Kong, Tat Chee Ave., Kowloon Tong, Kowloon, Hong Kong
[c] Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

A R T I C L E I N F O

Article history:
Received 1 April 2011
Accepted 9 June 2012
Available online 17 June 2012

Keywords:
Traveling salesman
Pickup and delivery
Last-in-first-out loading
Distance constraint
Metaheuristic

A B S T R A C T

We extend the traveling salesman problem with pickup and delivery and LIFO loading (TSPPDL) by considering two additional factors, namely the use of multiple vehicles and a limitation on the total distance that a vehicle can travel; both of these factors occur commonly in practice. We call the resultant problem the *multiple pickup and delivery traveling salesman problem with LIFO loading and distance constraints* (MTSPPD-LD). This paper presents a thorough preliminary investigation of the MTSPPD-LD. We propose six new neighborhood operators for the problem that can be used in search heuristics or meta-heuristics. We also devise a two-stage approach for solving the problem, where the first stage focuses on minimizing the number of vehicles required and the second stage minimizes the total travel distance. We consider two possible approaches for the first stage (*simulated annealing* and *ejection pool*) and two for the second stage (*variable neighborhood search* and *probabilistic tabu search*). Our computational results serve as benchmarks for future researchers on the problem.

1. Introduction

The traveling salesman problem with pickup and delivery (TSPPD) is a well-studied problem (e.g., Kalantari et al., 1985; Healy and Moll, 1995; Renaud et al., 2000, 2002; Dumitrescu et al., 2009). In the TSPPD, we are given a set $R = \{1, \ldots, n\}$ of n requests, where each request requires a load to be transported from pickup vertex i^+ to delivery vertex i^-, $i = 1, \ldots, n$. There is a single vehicle with unlimited capacity that starts from a depot vertex 0^+. Its task is to carry out all requests by visiting each pickup vertex before its corresponding delivery vertex (known as the *precedence constraint*) and finally return to a depot vertex 0^- while minimizing the total distance traveled. The TSPPD is defined on a complete weighted undirected graph $G = (V, E)$, where $V = \{0^+, 1^+, \ldots, n^+, 0^-, 1^-, \ldots, n^-\}$ is the vertex set; $E = \{(x, y)\colon x, y \in V, x \neq y\}$ is the edge set; and the edge weight $d(x, y)$ is the non-negative distance between vertices x and y.

A TSPPD variant that has recently received significant attention requires that loading and unloading operations are performed in a last-in-first-out (LIFO) manner, known as the TSPPD with LIFO loading (TSPPDL) (Iori and Martello, 2010). The TSPPDL naturally arises when routing a vehicle whose storage unit has only a single door located at the rear and works like a *stack*; it is especially applicable when the cost of rearranging loaded items is much higher than that of the extra traveling distance caused by the LIFO constraint, i.e., the rearrangement cost dominates the traveling cost. Examples include the transportation of bulky, fragile or hazardous items.

The TSPPDL was first mentioned by Ladany and Mehrez (1984), who solved a real-life delivery problem in Israel using an enumerative approach. Two classes of solution approaches have been applied to the TSPPDL, namely exact algorithms (Carrabs et al., 2007a; Cordeau et al., 2010) and heuristics (Pacheco, 1997; Levitin and Abezgaouz, 2003; Cassani and Righini, 2004; Carrabs et al., 2007b; Li et al., 2011). The difficulty of the TSPPDL is evidenced by the fact that the largest instances optimally solved in the literature contain only 25 requests (Cordeau et al., 2010), so heuristic approaches are required to handle practical instances with hundreds of requests. Currently, the best and latest heuristic is the variable neighborhood search heuristic proposed by Li et al. (2011) that employs rooted ordered trees to represent the feasible solutions of the TSPPDL.

A direct generalization of the TSPPDL is the single vehicle pickup and delivery problem with multiple stacks (1-PDPMS) that considers a vehicle with multiple independent stacks (Côté et al., 2009).

* Corresponding author. Tel.: +852 64117909/+86 13349921096; fax: +86 27 87556437.
E-mail addresses: brendacheang@yahoo.com (B. Cheang), gaoxiang@cityu.edu.hk (X. Gao), lim.andrew@cityu.edu.hk (A. Lim), tigerqin@mail.hust.edu.cn, tigerqin1980@gmail.com (H. Qin), i@zhuwb.com (W. Zhu).

0377-2217/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ejor.2012.06.019

Moreover, a special case of the 1-PDPMS which specifies all pickups to be performed before all deliveries was recently proposed by Petersen and Madsen (2009); this problem is called the double traveling salesman problem with multiple stacks (DTSPMS), which has also been solved by exact algorithms (Lusby et al., 2010; Petersen et al., 2010; Alba et al., in press; Carrabs et al., in press) and heuristics (Felipe et al., 2009a,b, 2011; Petersen and Madsen, 2009).

This paper introduces another useful and practical generalization of the TSPPDL by considering two additional factors. The first is to allow multiple vehicles, which reflects the fact that practical problems of this nature usually involve a fleet of vehicles rather than a single vehicle. The second is to require that the route length of each vehicle cannot exceed a predetermined limit L, which stems from regulations on working hours for drivers. For example, European Union regulation (EC) No. 561/2006 stipulates that the daily driving time shall not exceed 9 h, while in the United States the Federal Motor Carrier Safety Administration (FMCSA) mandates that commercial motor vehicle drivers may only drive up to 11 cumulative hours in a 14-h period. The distance a driver can travel is also limited if they have to return their vehicle to the depot by the end of the working day. This type of distance constraint has been widely applied to many vehicle routing problem (VRP) variants (see Laporte et al., 1985; Li et al., 1992; Nagy and Salhi, 2005; Erera et al., 2010).

We call the resultant problem the *multiple pickup and delivery traveling salesman problem with LIFO loading and distance constraints* (MTSPPD-LD). A solution to this problem is a set of vehicle routes that satisfy the precedence, LIFO and distance constraints. The primary objective of the MTSPPD-LD is to minimize the number of vehicle routes required to fulfill all requests. Once this is achieved, the secondary objective is to minimize the total distance traveled.

In this paper, we use the tree representation of feasible vehicle routes suggested by Li et al. (2011) to design several new search operators for this problem, some of which were adapted from the traditional move operators for the VRP with time windows (VRPTW), such as *relocate, exchange* and *CROSS-exchange* (Bräysy and Gendreau, 2005). These operators are employed in a two-stage heuristic: the first stage concentrates on reducing the number of trees (i.e., vehicle routes) from an initial solution, and the second stage focuses on minimizing the total distance traveled. This strategy of separating vehicle reduction from distance minimization has been successfully employed on other transportation problems (Bent and Van Hentenryck, 2004, 2006; Homberger and Gehring, 2005; Lim and Zhang, 2007).

For the first stage, we designed two vehicle reduction algorithms; one is a simulated annealing algorithm and the other is based on the concept of an ejection pool (Lim and Zhang, 2007; Nagata and Bräysy, 2009). We also designed two distance minimization algorithms for the second stage, namely a variable neighborhood search (VNS) heuristic (Mladenović and Hansen, 1997) and a probabilistic tabu search heuristic (Erdoğan et al., 2009). All of these approaches have been previously successfully applied to related problems. By implementing and examining the effectiveness of these techniques, our aim is to provide a foundation upon which future researchers of this problem can build. To test our approaches, we generated a large number of test instances based on the TSPPDL instances introduced by Carrabs et al. (2007b). Our experiments on these test instances suggest that the best two-stage heuristic consists of using the ejection pool algorithm for vehicle reduction and VNS for distance reduction.

To the best of our knowledge, the only existing research on the MTSPPD-LD is presented in Gao et al. (2011), which designed a VNS heuristic to minimize the total distance for the MTSPPD-LD with a fixed number of free vehicles (i.e., no fixed charge for the used vehicle). Gao et al. (2011) observed that if two trees can be merged without violating the distance constraint, the total distance must be reduced. Therefore, they incorporated a *tree-merge* operator in their VNS heuristic. We do not adopt this operator in the second stage of our approach because the possibility of successfully merging two trees is negligible after the number of trees is minimized in the first stage. Moreover, Gao et al. (2011) used a dynamic programming algorithm to optimize the individual tree with 13 or fewer requests. However, we found by some preliminary experiments that the VNS heuristic proposed in Li et al. (2011) (henceforth referred to as the VNS-SingleTree heuristic) can achieve optimal solutions for almost all TSPPDL instances with such scale using much less computation times; thus, we only apply VNS-SingleTree heuristic when optimizing individual trees.

The remainder of this paper is organized as follows. In Section 2, we briefly explain the tree representation of a feasible vehicle route that we use in our approach (originally proposed by Li et al. (2011)). We then introduce six new inter-tree operators in Section 3; these operators are employed by the various heuristics that we examine in this study. Section 4 describes two possible algorithms for the vehicle reduction stage of our approach, namely a simulated annealing algorithm and an ejection pool algorithm. Similarly, Section 5 describes two possible algorithms for the distance reduction stage, namely a VNS algorithm and a probabilistic tabu search. The effectiveness of these algorithms is analyzed with computational experiments that are detailed in Section 6, where we also explain how our test instances were generated and how the various parameters of the component algorithms were determined. We conclude our study in Section 7 with some additional observations and possible future research directions.

2. The tree representation of feasible routes

A feasible solution of the MTSPPD-LD is a set of vehicle routes, which can be simply represented by a set of vertex sequences; Fig. 1 shows a feasible solution for an MTSPPD-LD instance comprising two routes. Most heuristics for vehicle routing and scheduling problems are based on edge-exchange operations (Toth and Vigo, 2002), which involve the moving or exchanging of edges in vertex sequences. However, if such operations are directly applied to our problem, the resultant routes usually violate the precedence or LIFO constraint (or both). While it is possible to design sequence-based search operators for the problem, considerable effort has to be made to guarantee the feasibility of the resultant routes; this is clearly demonstrated by the complex additional checks required for the sequence-based VNS approach for the TSPPDL by Carrabs et al. (2007b).

To resolve this issue, we represent a feasible solution of the MTSPPD-LD by a *forest*, which consists of a set of ordered trees. Each tree represents a vehicle route with length less than L, and

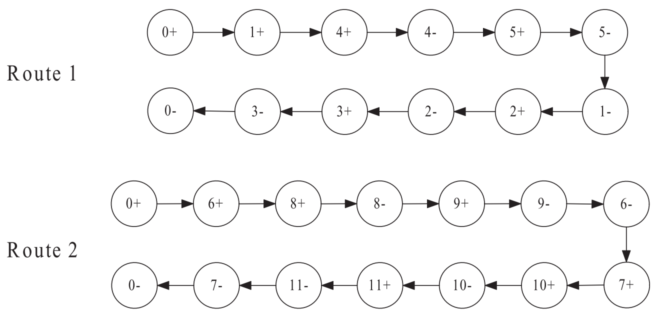

Fig. 1. The sequence representation of a feasible solution.

each node in a tree corresponds to a request in R, except for the root node which represents the depot and is labeled 0. A tree with a single root node 0 denotes a vehicle that does not handle any requests, which we call an *empty tree*. The tree representation of the solution shown in Fig. 1 is given in Fig. 2a. By following the dashed arrows in the ordered trees in Fig. 2b, which is similar to a preorder traversal, we can derive the corresponding vertex sequences. All vertex sequences derived from such trees automatically respect the precedence and LIFO constraints. For the full details on this tree representation, we refer the reader to Li et al. (2011). There is a one-to-one correspondence between the set of all forests of this type and the set of all feasible MTSPPD-LD solutions.

The following notations and terminology will be used for the remainder of the paper. We specify that a *node* refers to a tree node corresponding to a request in R, and a *vertex* refers to the pickup or delivery vertex in V. An MTSPPD-LD solution is denoted by the set of trees $F = \{T_1, \ldots, T_m\}$. For a feasible solution F, all requests $i \in R$ must be in exactly one tree in F; we use $T[i]$ to denote its tree, and the notation τ_i to represent the subtree rooted at node i in $T[i]$. Note that all subtrees τ_i represent a route segment starting at i^+ and ending at i^-.

3. Inter-tree search operators

The various search algorithms examined in this study make use of several search operators that transform (or *move*) a current solution F into another solution F'. We classify our search operators into two categories, namely *intra-tree* and *inter-tree* search operators.

An intra-tree search operator modifies only one tree in F. The number of requests fulfilled by each vehicle is limited by the distance constraint and is usually small. Consequently, we are able to directly employ the VNS algorithm proposed by Li et al. (2011) for the TSPPDL to optimize each tree very quickly. This algorithm integrates several intra-tree search operators, the details of which can be found in the original publication.

In this section, we describe six new inter-tree search operators, namely *node exchange, node relocate, subtree exchange, subtree relocate, node multi-relocate* and *subtree multi-relocate*; these operators modify two or more trees at once. Each operator corresponds to an operation that only considers moves satisfying the distance constraint, i.e., the distance traveled for each resultant route is at most L; this is sufficient to guarantee the feasibility of the resultant solution F' assuming the initial solution F is feasible. A move that reduces the total number of non-empty trees (i.e., uses fewer vehicles) has the highest priority. If no such move exists, then all of these operators perform the move that results in the greatest savings in total distance out of all possibilities. The exceptions are the node multi-relocate and subtree multi-relocate operators, which will be explained in the relevant subsections. If an operator cannot find a move that reduces neither the number of non-empty trees nor the total distance, it will be skipped without modifying the current solution.

3.1. Node exchange

The *node exchange* operation selects a non-root node from two different trees and exchanges their positions. For example, given the initial feasible solution in Fig. 3a, exchanging nodes x and y would create the new solution shown in Fig. 3b. The effect of this operation is to swap a pair of requests between two vehicles. The node exchange operator considers all possible node pairs belonging to two different trees. Since there are n nodes in total, this operator runs in $O(n^2)$ time.

3.2. Node relocate

The *node relocate* operation selects one node $x \in R$ from a tree T_p and relocates it such that it is the child of a node in another tree T_q (T_q can be an empty tree); this moves the request from one vehicle to another. Consider once again the initial solution shown in Fig. 3a. First, node x is removed from T_1 and its children are linked to its parent node such that a new tree $T_1' = T_1 - \{x\}$ is created. Next, node x is inserted into T_2 as a child of node y. If node y has c_y children, then there are $c_y + 1$ possible positions for this insertion. Fig. 4a shows the resultant solution if x is inserted as the leftmost child of y. Note that the total number of positions where node x can be inserted into T_2 is $O(n)$ (each position corresponds

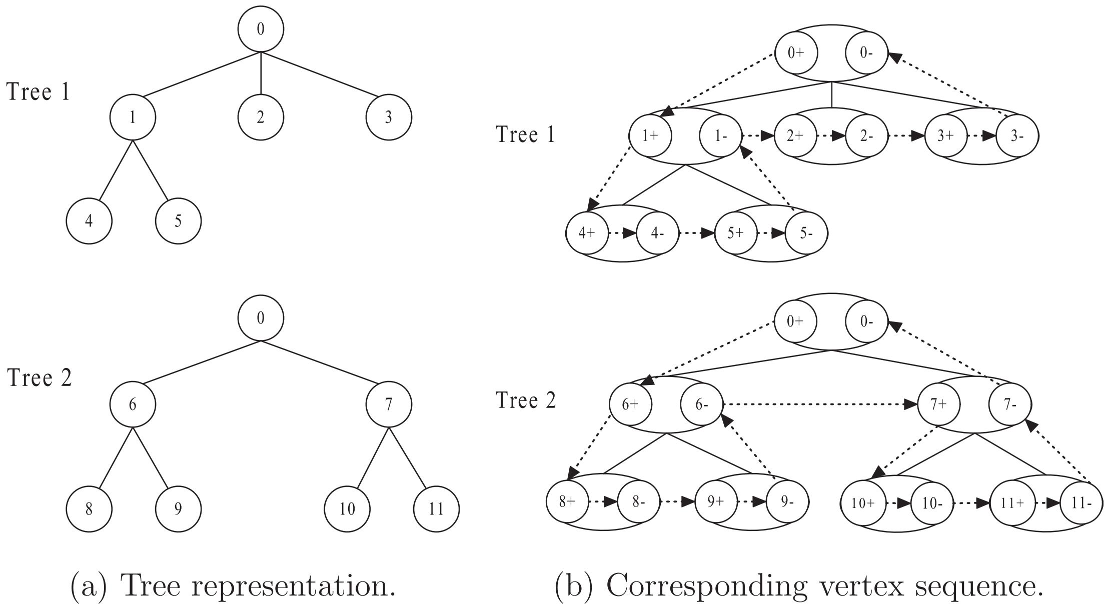

Fig. 2. The tree representation of a feasible solution.

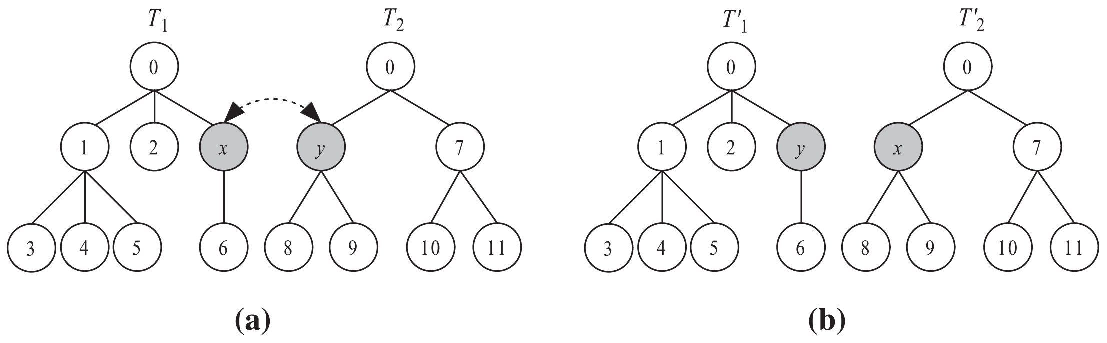

Fig. 3. (a) Initial solution. (b) New solution created by node exchange.

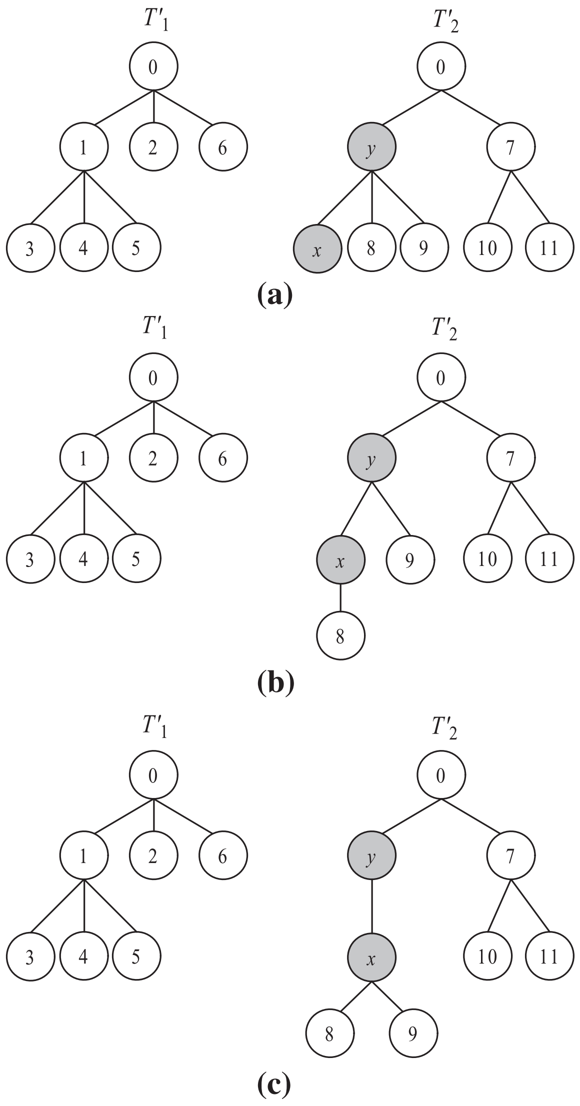

Fig. 4. Three possible resultant solutions created by node relocate.

to x being inserted into a different position in a traversal sequence of T_2).

After node x is inserted into T_2, if it now has α right siblings, we can create further solutions by relocating its $0, 1, \ldots, \alpha$ right siblings as its children. In our example, by repositioning the current right siblings of node x from Fig. 4a, we can create two other solutions as shown in Fig. 4b and c. The node relocate operator considers all possible solutions that can be produced in this manner. Let ω be the maximum number of children for any node in all trees. Since there are $O(n)$ positions where a given node x can be inserted into T_2 and the number of additional solutions that can be created by repositioning its siblings is $O(\omega)$, the time complexity for this operator is $O(n^2\omega)$.

3.3. Subtree exchange

Observe that any set of subtrees whose root nodes are adjacent siblings corresponds to a route segment. The *subtree exchange* operation selects a set of one or more subtrees whose roots are adjacent siblings from each of two different trees and exchanges their positions. For example, consider the initial solution in Fig. 5a. If the two sets selected are $\{\tau_2, \tau_3\}$ and $\{\tau_8\}$, then the subtree exchange operation produces the solution shown in Fig. 5b. The effect of this operation is to exchange the route segments represented by (τ_2, τ_3) and (τ_8) between the two vehicle routes. Note that the relative positions of the subtrees selected from each tree remain unchanged after this operation.

For any node y with c_y children, there are exactly $\frac{c_y}{2}(1 + c_y) = O(\omega^2)$ possible sets of adjacent children. Therefore, all possible such exchanges can be evaluated in $O(n\omega^2 \times n\omega^2) = O(n^2\omega^4)$ time.

3.4. Subtree relocate

The *subtree relocate* operation selects a set of one or more subtrees whose roots are adjacent children of a node x in T_p and relocates them to be the children of another node y in a different tree T_q; this moves the route segment represented by the selected subtrees from one vehicle route to another. For example, given the initial solution shown in Fig. 5a, one of the new solutions created by relocating $\{\tau_2, \tau_3\}$ to T_2 is shown in Fig. 5c. Once again, the relative positions of the relocated subtrees are unchanged.

For a given node x, the total number of adjacent child subtrees of x is exactly $\frac{c_x}{2}(1 + c_x) = O(\omega^2)$. Since there are $O(n)$ positions in T_2 where these subtrees can be relocated, this operator runs in $O(n^2\omega^2)$ time.

3.5. Node multi-relocate

The *node multi-relocate* operation involves more than two trees and can be viewed as a generalization of the node relocate operation. This operation has two forms: *node cyclic-relocate* and *node path-relocate* (Ahuja et al., 2001; Ibaraki et al., 2005). The concept is not new and has been used in designing search operators for several complex combinatorial problems, where it has been termed *multi-exchange* operation. Examples include the VRPTW (Ibaraki et al., 2005; Lim and Zhang, 2007), the capacitated minimum

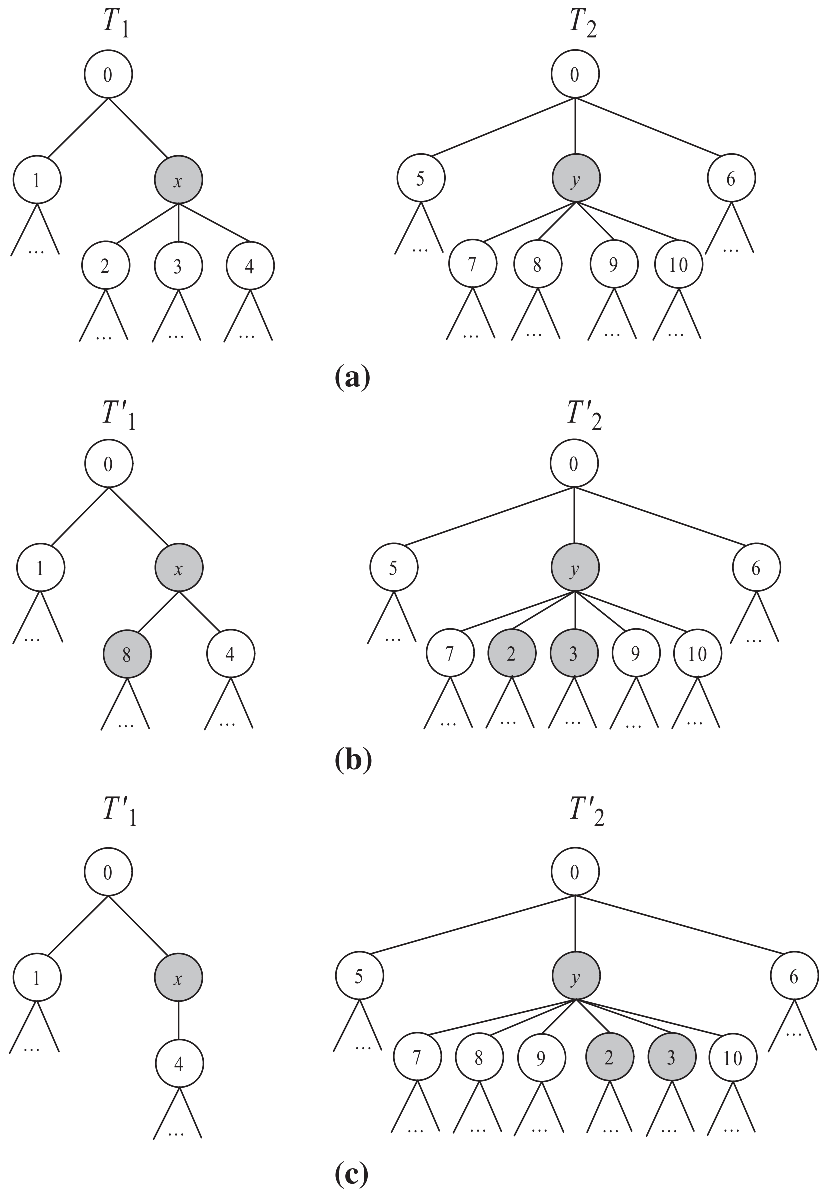

Fig. 5. (a) Initial solution. (b) New solution created by subtree exchange. (c) New solution created by subtree relocate.

spanning tree problem (Ahuja et al., 2001), the capacitated facility location problem (Ahuja et al., 2004) and the parallel machine scheduling problem (Frangioni et al., 2004).

When performing a node multi-relocate operation, we first select h nodes labeled i_1, $i_2,\ldots,i_h$, where $T[i_s] \neq T[i_t]$ for $s \neq t$. Given two selected nodes i_s and i_t, we introduce the following notations: let $i_s \Rightarrow i_t$ represent the node relocation operation (described in Section 3.2) where node i_s is relocated to the best place of tree $T[i_t]$; and let $i_s \to i_t$ represent the node relocation operation where node i_s is relocated to the best place of tree $T[i_t] - \{i_t\}$ (i.e., the node i_t is first removed from $T[i_t]$ and its children are linked to its parent). The node cyclic-relocate operation can be expressed as $i_1 \to i_2 \to \cdots \to i_h \to i_1$. The node path-relocate operation can be expressed as $i_1 \to i_2 \to \cdots \to i_{h-1} \Rightarrow i_h$; it differs from cyclic-relocate in that node i_h is not removed from $T[i_h]$ nor inserted into $T[i_1]$. Fig. 6 illustrates these two operations.

We call the set of solutions that can be obtained from a given feasible solution F by either the node cyclic-relocate or the node path-relocate operations the *node multi-relocate neighborhood of F*. This neighborhood is very large and grows exponentially with the problem size, so identifying the best solution in the neighborhood by enumerating all members is computationally prohibitive. Therefore, our node multi-relocate operator transforms a given feasible solution into the first improving solution it finds in the node multi-relocate neighborhood (i.e., the move leads to a reduction in overall distance).

In order to find an improving solution in the node multi-relocate neighborhood of a feasible solution F, we construct an *improvement graph* (Ahuja et al., 2001; Ibaraki et al., 2005), which is a directed weighted graph denoted by $G_F = (V_F, E_F)$. The node set V_F comprises the set of all requests R, a *pseudonode* p' for each tree T_p, and a node denoted by $0'$.

The edge set E_F is constructed as follows. A directed edge (i, j), i, $j \in R$ and $T[i] \neq T[j]$ is included in E_F if the operation $i \to j$ leads to a feasible resultant tree. If this move is performed, tree $T[j]$ would become tree $T'[i]$. Let $dist(T)$ be the length of the vehicle route

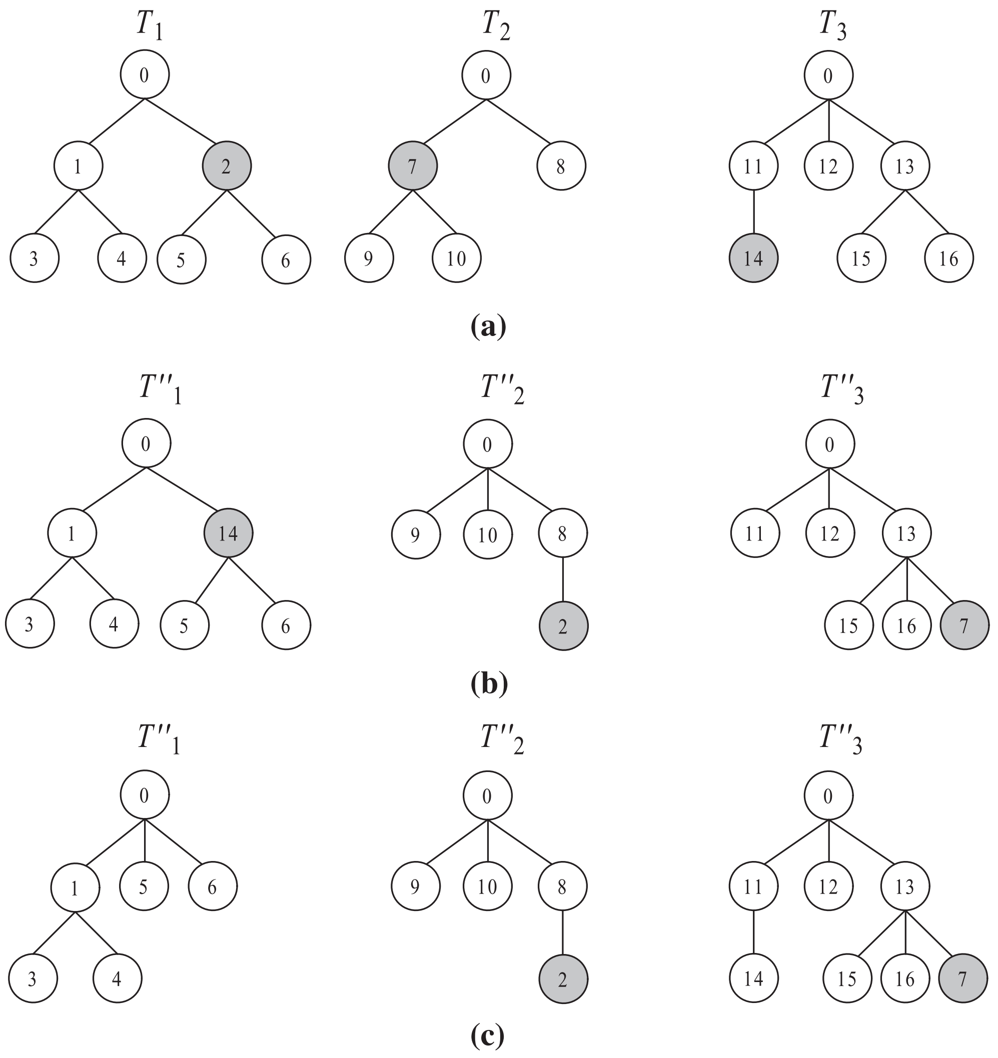

Fig. 6. (a) Initial solution. (b) New solution created by node cyclic-relocate. (c) New solution created by node path-relocate.

associated with tree T; the weight of edge (i, j) is set to be $dist(T'[i]) - dist(T[j])$, which is the increase in travel distance for tree $T[j]$ as the result of removing node j and then inserting node i. We connect node $0'$ to each node $i \in R$ using the directed edge $(0', i)$ and set its weight to be $dist(T[i] - \{i\}) - dist(T[i])$, which is the increase in travel distance for tree $T[i]$ after removing node i. There is a directed edge $(p', 0')$ with zero weight for each tree. We also connect each node $i \in R$ to each pseudonode p' by a directed edge (i, p') if $T[i] \neq T[p']$; this edge indicates that node i is ejected from tree $T[i]$ and then inserted into tree $T[p']$ without ejecting any node from tree $T[p']$. Hence, the cost of edge (i, p') is set to be the distance of the resultant tree after inserting node i into tree $T[p']$ minus $dist(T[i])$; Still, if the operation $i \Rightarrow p'$ leads to an infeasible resultant tree, edge (i, p') will not be constructed.

Having constructed the improvement graph G_F, every node multi-relocate that leads to a distance reduction corresponds to a *negative subset-disjoint cycle* (Ahuja et al., 2001). This operator attempts to identify a negative subset-disjoint cycle in G_F using the variant of the shortest-path label-correcting algorithm proposed by Ahuja et al. (2001), transforms F to F' by performing the moves in the identified cycle, and then reconstructs the improvement graph $G_{F'}$. This is repeated until no negative cycle can be found by the algorithm in the current improvement graph.

Each iteration of this operator consists of constructing the improvement graph and identifying a negative subset-disjoint cycle. It is easy to know that constructing the improvement graph can be done in $O(n^2)$ time. Since identifying a negative subset-disjoint cycle is Np-hard, we apply the heuristic proposed by Ahuja et al. (2001), which runs in $O(n^2 \cdot m \cdot L)$ where m is the number of edges in the graph and L is the number of vehicles. We do not maintain the information of the previous improvement graphs, thus each iteration of this operator requires $O(n^2 \cdot m \cdot L)$ time.

3.6. Subtree multi-relocate

The *subtree multi-relocate* operation is analogous to the node multi-relocate operation except that the moves are applied to subtrees rooted at nodes in R rather than nodes. In a manner similar to the node multi-relocate operation, we construct another improvement graph for the neighborhood of this operation. We then follow the same procedure of identifying a negative subset-disjoint cycle in this improvement graph, performing the moves in the identified cycle and reconstructing the improvement graph until no such cycle can be found. The time complexity of each iteration of this operator is also $O(n^2 \cdot m \cdot L)$.

Table 1
Number of routes obtained by SA and EP algorithms.

Instance	Size	$L = 1.5 \times d_{max}$					$L = 2.0 \times d_{max}$					$L = 2.0 \times d_{max}$				
		CI	SA		EP		CI	SA		EP		CI	SA		EP	
			Min	Ave.	Min	Ave.		Min	Ave.	Min	Ave.		Min	Ave.	Min	Ave.
brd14051	25	**2**	**2**	2.0	**2**	2.0	**2**	**2**	2.0	**2**	2.0	**1**	**1**	1.0	**1**	1.0
	51	**2**	**2**	2.0	**2**	2.0	2	**1**	1.0	**1**	1.0	**1**	**1**	1.0	**1**	1.0
	75	**2**	**2**	2.0	**2**	2.0	2	**1**	1.0	**1**	1.0	**1**	**1**	1.0	**1**	1.0
	101	3	3	3.0	**2**	2.9	**2**	**2**	2.0	**2**	2.0	**2**	**2**	2.0	**2**	2.0
	251	6	**5**	5.0	**5**	5.0	4	4	4.0	**3**	3.8	**3**	**3**	3.0	**3**	3.0
	501	10	9	9.0	**8**	8.0	7	**6**	6.0	**6**	6.0	**5**	**5**	5.0	**5**	5.0
	751	16	14	14.0	**13**	13.0	11	**10**	10.0	**10**	10.6	9	8	8.0	**7**	7.9
d15112	25	3	**2**	2.0	**2**	2.0	**2**	**2**	2.0	**2**	2.0	2	**1**	1.0	**1**	1.0
	51	4	**3**	3.0	**3**	3.0	**2**	**2**	2.0	**2**	2.0	**2**	**2**	2.0	**2**	2.0
	75	5	**4**	4.0	**4**	4.0	4	**3**	3.0	**3**	3.0	3	**2**	2.0	**2**	2.0
	101	6	**5**	5.0	**5**	5.0	5	**4**	4.0	**4**	4.0	4	**3**	3.0	**3**	3.0
	251	13	**11**	11.0	**11**	11.0	9	8	8.0	**7**	7.9	7	**6**	6.0	**6**	6.0
	501	21	19	19.0	**17**	17.0	14	13	13.0	**12**	12.0	11	10	10.0	**9**	9.2
	751	27	25	25.0	**23**	23.0	19	18	18.0	**16**	16.0	14	14	14.0	**12**	12.4
d18512	25	**2**	**2**	2.0	**2**	2.0	**2**	**2**	2.0	**2**	2.0	**1**	**1**	1.0	**1**	1.0
	51	3	**2**	2.0	**2**	2.0	2	**1**	1.0	**1**	1.0	**1**	**1**	1.0	**1**	1.0
	75	**2**	**2**	2.0	**2**	2.0	**2**	**2**	2.0	**2**	2.0	2	**1**	1.0	**1**	1.0
	101	3	3	3.0	**2**	2.0	**2**	**2**	2.0	**2**	2.0	**2**	**2**	2.0	**2**	2.0
	251	6	**5**	5.0	**5**	5.0	**4**	**4**	4.0	**4**	4.0	**3**	**3**	3.0	**3**	3.0
	501	10	9	9.0	**8**	8.0	7	**6**	6.0	**6**	6.0	6	**5**	5.0	**5**	5.0
	751	14	13	13.0	**12**	12.0	10	10	10.0	**9**	9.8	**8**	**8**	8.0	**8**	8.0
fnl4461	25	**1**	**1**	1.0	**1**	1.0	**1**	**1**	1.0	**1**	1.0	**1**	**1**	1.0	**1**	1.0
	51	3	**2**	2.0	**2**	2.0	**2**	**2**	2.0	**2**	2.0	2	**1**	1.0	**1**	1.0
	75	**3**	**3**	3.0	**3**	3.0	3	**2**	2.0	**2**	2.0	**2**	**2**	2.0	**2**	2.0
	101	5	**4**	4.0	**4**	4.0	4	**3**	3.0	**3**	3.0	3	**2**	2.0	**2**	2.0
	251	5	**4**	4.0	**4**	4.0	4	**3**	3.0	**3**	3.0	**3**	**3**	3.0	**3**	3.0
	501	9	**8**	8.0	**8**	8.0	**6**	**6**	6.0	**6**	6.0	5	5	5.0	**4**	4.3
	751	15	14	14.0	**13**	13.0	10	10	10.0	**9**	9.2	8	8	8.0	**7**	7.2
nrw1379	25	**2**	**2**	2.0	**2**	2.0	**1**	**1**	1.0	**1**	1.0	**1**	**1**	1.0	**1**	1.0
	51	**2**	**2**	2.0	**2**	2.0	2	**1**	1.0	**1**	1.0	**1**	**1**	1.0	**1**	1.0
	75	**3**	**3**	3.0	**3**	3.0	**2**	**2**	2.0	**2**	2.0	**2**	**2**	2.0	**2**	2.0
	101	**4**	**4**	4.0	**4**	4.0	**3**	**3**	3.0	**3**	3.0	**2**	**2**	2.0	**2**	2.0
	251	10	**8**	8.0	**8**	8.0	7	**6**	6.0	**6**	6.0	5	5	5.0	**4**	4.0
	501	17	15	15.0	**13**	13.8	11	11	11.0	**10**	10.0	9	**8**	8.0	**8**	8.0
	751	26	24	24.0	**22**	22.4	18	17	17.0	**16**	16.0	14	13	13.0	**12**	12.5
pr1002	25	2	**1**	1.0	**1**	1.0	**1**	**1**	1.0	**1**	1.0	**1**	**1**	1.0	**1**	1.0
	51	4	**3**	3.0	**3**	3.0	**2**	**2**	2.0	**2**	2.0	**2**	**2**	2.0	**2**	2.0
	75	5	**4**	4.0	**4**	4.0	**3**	**3**	3.0	**3**	3.0	3	**2**	2.0	**2**	2.0
	101	6	**5**	5.0	**5**	5.0	5	4	4.0	**3**	3.9	**3**	**3**	3.0	**3**	3.0
	251	10	9	9.0	**8**	8.0	7	**6**	6.0	**6**	6.0	5	5	5.0	**4**	4.0
	501	17	16	16.0	**15**	15.0	12	11	11.0	**10**	10.4	9	9	9.0	**8**	8.0
	751	24	22	22.0	**20**	20.1	16	15	15.0	**14**	14.0	12	12	12.0	**11**	11.0
Sum	–	333	296	296.0	277	279.2	234	213	213.0	201	205.6	181	168	168.0	157	159.5

4. Reducing the number of vehicles

Beginning with an initial feasible solution, the first stage of our approach attempts to reduce the number of non-empty trees, thereby reducing the number of vehicles. In this section, we first describe the algorithm employed to produce the initial feasible solution, and then introduce two algorithms that focus on reducing the number of trees.

4.1. Initial feasible solution

We obtain an initial feasible solution using a *cheapest insertion* (*CI*) *heuristic*. A state in the heuristic is a *feasible partial solution* consisting of the set of unselected requests that have not yet been assigned, and a set of feasible trees representing vehicle routes, one of which is designated as the *current tree*. The heuristic begins with an empty tree as the current tree and R as the set of unselected requests. We repeatedly insert the best unselected request in terms of increased travel distance into the current tree. When no further request can be inserted into the current tree without violating the distance constraint, we continue with another empty tree as the current tree. The above process is repeated until all requests are inserted.

4.2. Simulated annealing algorithm

Having produced a feasible initial solution, we tested two algorithms whose objectives were to reduce the total number of vehicles. The first is a simulated annealing (SA) algorithm, which has been a successful strategy for many VRP variants (e.g., Bent and Van Hentenryck, 2004; Li and Lim, 2003).

The performance of an SA algorithm is largely dependent on its search neighborhood (determined by the set of neighborhood operations used) and its evaluation function. In our SA algorithm, we use the four inter-tree operations, i.e., node exchange, node relocate, subtree exchange and subtree relocate described in Section 3, along with their intra-tree counterparts proposed by Li et al. (2011), for a total of eight possible operations. We do not include the node multi-relocate nor subtree multi-relocate operations because they are designed to decrease the total distance of the solution; preliminary experiments confirmed these two operations are ineffective for reducing the number of vehicles. For a given solution F, our evaluation function $f(F)$ consists of three lexicographically ordered components: (1) the number of trees; (2) the negative sum of the squared number of requests for the individual routes, i.e., $-\sum_{T_p \in F} |T_p|^2$, where $|T_p|$ is the number of requests in tree T_p; and (3) the total distance.

The pseudocode for our SA algorithm is presented in Algorithm 1. In each SA iteration, a neighborhood is randomly chosen, which is similar to the way used in the adaptive large neighborhood search algorithm (Pisinger and Ropke, 2007). Suppose the first component for which two solutions F_1 and F_2 have different values is component k; we define $f(F_1) - f(F_2) = \alpha_k \Delta_k$, where α_k is a controlling parameter and Δ_k is the value difference. We begin with both the current solution F and the best solution found so far F_{best} set to be the initial solution, and initialize a current temperature value t. We then uniformly randomly choose an operation o out of the eight possibilities, and a request $r \in R$. Next, we construct the ordered set $N(o, r, F) = \{F_1, F_2, \ldots, F_\rho\}$, which contains the neighborhood solutions of F that can be obtained using the operation o and the request r (line 9). This set is sorted in a non-decreasing order of evaluation function values.

Algorithm 1. Simulated annealing algorithm

```
1: INPUTS: timeLimit, startTemperature, temperatureLimit,
   SAMaxIterations, $\beta$, $\gamma$, and an initial solution $F_i$;
2: Set $F_{best} \leftarrow F_i$;
3: while time < timeLimit do
4:   $F \leftarrow F_{best}$;
5:   $t \leftarrow$ startTemperature;
6:   while time < timeLimit and $t$ > temperatureLimit do
7:     for $i = 1$ to SAMaxIterations do
8:       Randomly choose an operation $o$ and a request $r$;
9:       Construct ordered set $N(o, r, F) = \{F_1, F_2, \ldots, F_\rho\}$
   containing neighborhood of $F$ using operation $o$ and request
   $r$, where $f(F_i) \leqslant f(F_j)$ if $i < j$;
10:      if $f(F_1) < f(F_{best})$ then
11:        $F_{best} \leftarrow F_1$ and $F \leftarrow F_1$;
12:      else
13:        $\tau \leftarrow \lfloor random(0,1)^\beta \times \rho \rfloor$;
14:        $\Delta \leftarrow \Delta_k \alpha_k = f(F) - f(F_\tau)$;
15:        if $\Delta > 0$ then
16:          $F \leftarrow F_\tau$;
17:        else
18:          Set $F \leftarrow F_\tau$ with a probability of $e^{\Delta/t}$;
19:        end if
20:      end if
21:    end for
22:    $t \leftarrow \gamma \times t$;
23:  end while
24: end while
```

If the best solution in the neighborhood F_1 has a lower evaluation score than F_{best}, then both F_{best} and F are updated to be F_1 (lines 10–12). Otherwise, we randomly choose the τth solution in $N(o, r, F)$ according to $\tau = \lfloor random(0,1)^\beta \times \rho \rfloor$, where $random(0,1)$ uniformly randomly returns a real value between 0 and 1 (line 13). If the cost of solution F_τ is lower than the cost of the current solution F, then we set $F \leftarrow F_\tau$; otherwise we set $F \leftarrow F_\tau$ with probability $e^{\Delta/t}$, where Δ is the difference between the evaluation values of F and F_τ (lines 14–19).

After considering *SAMaxIterations* neighborhoods, we reduce the current temperature by setting $t \leftarrow \gamma \times t$, where γ is a user-defined cooling factor. This process is repeated until t falls below a temperature limit value or the time limit is exceeded.

4.3. Ejection pool algorithm

The second algorithm we examined for reducing the number of vehicles is based on the concept of an *ejection pool*, which has been previously applied to reduce the number of routes for the VRPTW (e.g., Lim and Zhang, 2007; Nagata and Bräysy, 2009). Since the best existing algorithm (devised by Nagata and Bräysy (2009)) for reducing the number of vehicles for the VRPTW makes use of this concept, we feel this is a promising approach for our problem.

In our ejection pool (EP) algorithm, an ejection pool is a stack of unselected requests, and each request in the ejection pool has an associated penalty value $q[r]$ representing the difficulty of the request. Our approach is given in Algorithm 2, which attempts to delete one tree from a feasible solution F; this algorithm is similar to the one described by Nagata and Bräysy (2009). We reduce the number of trees of a given feasible solution one by one repeatedly invoking Algorithm 2 until a given time limit is reached.

Algorithm 2. Ejection pool (EP) algorithm

```
1: INPUT: maxTime and an initial feasible solution $F_i$;
2: Set $F \leftarrow F_i$;
3: Randomly remove a tree from $F$;
4: Initialize ejection pool with the requests from the removed
   tree;
5: Initialize all penalty values $q[r] \leftarrow 0$ for $r \in R$;
6: while ejection pool $\neq \emptyset$ and time < maxTime do
7:   Remove a request $r_{in}$ from ejection pool following LIFO
   rule;
8:   if insert-neighborhood $N_{insert}(r_{in}, F) \neq \emptyset$ then
9:     Randomly select $F'$ from $N_{insert}(r_{in}, F)$ and update $F \leftarrow F'$;
10:  else
11:    $F \leftarrow$ squeeze($r_{in}$, $F$);
12:  end if
13:  if $r_{in}$ is not in $F$ then
14:    Set $q[r_{in}] \leftarrow q[r_{in}] + 1$;
15:    $F \leftarrow$ eject($r_{in}$, $F$);
16:    $F \leftarrow$ perturb_ep($F$);
17:  end if
18: end while
19: if ejection pool $\neq \emptyset$ then
20:   return $F_i$
21: end if
22: return $F$
```

At the beginning of the algorithm, the current solution F is initialized to be the initial feasible solution. We then randomly remove a tree from F such that F becomes a feasible partial solution (line 3). The requests from the removed tree form the initial ejection pool in no particular order (line 4), and the penalty values $q[r]$ for each request $r \in R$ is initialized to zero (line 5). Next, we attempt to insert the requests in the ejection pool into F one by one; the requests are selected from the ejection pool following the last-in-first-out rule. Given a request r_{in}, a partial solution F can be transformed into another partial solution F' by inserting r_{in} into one of the trees of F. We try three possible insertion methods. The first is to perform insertion in the same manner as for the node relocate operation described in Section 3.2. Let $N_{insert}(r_{in}, F)$ be the set of all feasible partial solutions that can result from such an insertion. If $N_{insert}(r_{in}, F)$ is not empty, we randomly select $F' \in N_{insert}(r_{in}, F)$ and update $F \leftarrow F'$ (lines 8–9); otherwise, we try the second method realized by the function *squeeze* (line 11).

Table 2
Performance comparison between the VNS and PTS algorithms for $L = 1.5 \times d_{max}$.

Instance	Size	Initial		VNS				PTS				Gap (%)
		Route	Dist.	Route	Dist.	Time	Improv. (%)	Route	Dist.	Time	Improv. (%)	
brd14051	25	2	5456	2.0	5086.0	1.62	6.78	2.0	5087.2	0.54	6.76	0.02
	51	2	10,922	2.0	9352.0	7.27	14.37	2.0	9438.7	2.96	13.58	0.92
	75	2	10,319	2.0	8543.4	19.05	17.21	2.0	8544.8	6.84	17.19	0.02
	101	2	11,173	2.0	11,169.0	21.31	0.04	2.0	11,169.0	6.42	0.04	0.00
	251	5	29,309	5.0	28,196.1	172.90	3.80	5.0	28,730.2	119.51	1.97	1.86
	501	8	63,084	8.0	59,090.0	965.21	6.33	8.0	60,699.8	907.25	3.78	2.65
	751	13	102,717	13.0	96,421.5	1345.17	6.13	13.0	97,976.6	4559.20	4.62	1.59
d15112	25	2	1,090,841	2.0	108,207.0	1.45	90.08	2.0	108,439.0	0.31	90.06	0.21
	51	3	179,859	3.0	179,086.0	5.29	0.43	3.0	179,859.0	1.13	0.00	0.43
	75	4	252,382	4.0	244,600.7	13.82	3.08	4.0	251,313.9	3.71	0.42	2.67
	101	5	346,055	5.0	339,504.3	14.19	1.89	5.0	346,055.0	5.28	0.00	1.89
	251	11	760,001	*10.8*	720,770.3	98.15	5.16	*10.9*	727,627.5	200.22	4.26	0.94
	501	17	1,220,297	*16.7*	1,165,275.3	252.73	4.51	17.0	1,188,016.3	663.22	2.65	1.91
	751	23	1,705,292	*22.4*	1,622,964.0	1036.52	4.83	*22.0*	1,637,573.1	1765.95	3.97	0.89
d18512	25	2	5456	2.0	5086.0	1.63	6.78	2.0	5087.2	0.54	6.76	0.02
	51	2	9286	2.0	9256.8	6.67	0.31	2.0	9286.0	1.17	0.00	0.31
	75	2	12,134	2.0	10,148.8	16.85	16.36	2.0	10,180.7	8.41	16.10	0.31
	101	2	11,920	2.0	11,765.6	31.13	1.30	2.0	11,909.0	6.24	0.09	1.20
	251	5	30,365	5.0	28,933.5	229.93	4.71	5.0	29,314.2	206.38	3.46	1.30
	501	8	60,660	8.0	57,787.4	733.32	4.74	8.0	58,928.0	1241.01	2.86	1.94
	751	12	99,298	12.0	94,016.2	2346.58	5.32	12.0	95,612.5	3765.94	3.71	1.67
fnl4461	25	1	2511	1.0	2168.0	2.35	13.66	1.0	2168.0	0.40	13.66	0.00
	51	2	4933	2.0	4830.0	4.68	2.09	2.0	4830.0	1.11	2.09	0.00
	75	3	8571	3.0	7432.2	15.83	13.29	3.0	7466.5	8.22	12.89	0.46
	101	4	11,000	4.0	10,765.3	18.29	2.13	4.0	10,897.8	7.32	0.93	1.22
	251	4	31,994	4.0	31,334.1	255.07	2.06	4.0	31,782.4	210.30	0.66	1.41
	501	8	90,027	*7.0*	83,246.3	667.54	7.53	8.0	85,081.8	1596.72	5.49	2.16
	751	13	149,581	*12.0*	138,626.0	2163.77	7.32	12.0	139,106.8	3172.93	7.00	0.35
nrw1379	25	2	5176	2.0	3464.0	2.00	33.08	2.0	3466.1	0.58	33.04	0.06
	51	2	6768	2.0	5423.4	8.71	19.87	2.0	5499.3	2.43	18.75	1.38
	75	3	10,398	3.0	8352.2	20.23	19.67	3.0	8403.4	11.84	19.18	0.61
	101	4	14,228	4.0	12,220.9	33.06	14.11	4.0	12,537.3	18.82	11.88	2.52
	251	8	33,276	*7.5*	31,635.0	96.03	4.93	*7.9*	32,217.6	186.17	3.18	1.81
	501	13	71,240	13.0	68,962.5	293.56	3.20	13.0	70,800.0	775.82	0.62	2.60
	751	22	129,577	22.0	124,711.0	858.91	3.76	22.0	125,859.0	1740.60	2.87	0.91
pr1002	25	1	16,221	1.0	16,221.0	1.75	0.00	1.0	16,221.0	0.21	0.00	0.00
	51	3	47,989	3.0	47,989.0	4.60	0.00	3.0	47,980.6	1.20	0.02	−0.02
	75	4	66,989	4.0	64,889.0	11.94	3.13	4.0	65,197.8	7.49	2.67	0.47
	101	5	88,713	5.0	88,260.8	13.78	0.51	5.0	88,373.9	7.65	0.38	0.13
	251	8	267,307	8.0	263,657.4	60.59	1.37	8.0	264,251.3	157.55	1.14	0.22
	501	15	635,363	*14.9*	611,917.4	396.32	3.69	15.0	620,169.5	776.39	2.39	1.33
	751	20	1,064,222	20.0	1,031,423.3	1045.75	3.08	20.0	1,034,424.5	1723.79	2.80	0.29
Average	–	–	–	–	–	–	8.63	–	–	–	7.71	0.97
Sum	–	277	–	273.3	–	–	–	274.8	–	–	–	–

In the function *squeeze*, we first insert r_{in} into the tree that results in the smallest increase in distance; this resultant partial solution F will be infeasible. Next, we apply the four inter-tree operators, namely, node exchange, node relocate, subtree exchange and subtree relocate, along with their intra-tree counterparts in a standard variable neighborhood descent (VND) procedure to restore the feasibility of F. If the partial solution returned by the VND procedure is feasible, then we are done; otherwise, F is restored to the state before executing this function, the value of $q[r_{in}]$ is incremented by one, and we attempt to insert r_{in} into F using the third method realized by the function *eject* (line 15).

The *eject* function considers the insertion of r_{in} into each tree of F and performs the best such insertion. The insertion is done in the following manner. Given a target tree T_p, we first insert r_{in} at an arbitrary position in T_p. We then invoke the VNS-SingleTree heuristic on T_p. If this procedure has made T_p feasible, i.e., $dist[T_p] \leqslant L$, then we are done; otherwise, we eject the request $r \neq r_{in}$ with the smallest $q[r]$ value from T_p and invoke VNS-SingleTree again. This process is repeated, ejecting one request from T_p each time, until T_p is made feasible. Let $\{r_p^1, \ldots, r_p^k\}$ be the set of ejected requests from tree T_p. We insert r_{in} into the tree T_p with the smallest value of $q[r_p^1] + \cdots + q[r_p^k]$, add the ejected requests $\{r_p^1, \ldots, r_p^k\}$ into the ejection pool, and update F accordingly. The rationale behind this criterion is that ejecting the requests with the smaller sum of penalty values may make future insertions easier.

After inserting r_{in}, we diversify the process by the *perturb_ep* procedure. The *perturb_ep* procedure modifies the partial feasible solution F by randomly selecting and performing one of the same eight neighborhood operations used in our SA algorithm *EPPerturbIterations* times; only the moves that lead to feasible partial solutions are accepted.

The entire process is repeated until the ejection pool is empty, whereupon the number of trees has been successfully reduced by one, or until the time limit is exceeded. At the moment of termination, if the ejection pool is not empty, we restore F to the initial feasible solution F_i (line 20).

5. Minimizing the total distance

After the tree reduction stage, we attempt to minimize the total distance traveled by the vehicles. We propose two approaches for this purpose, namely a variable neighborhood search heuristic

Table 3
Performance comparison between the VNS and PTS algorithms for $L = 2.0 \times d_{max}$.

Instance	Size	Initial		VNS				PTS				Gap (%)
		Route	Dist.	Route	Dist.	Time	Improv. (%)	Route	Dist.	Time	Improv. (%)	
brd14051	25	2	7022	2.0	4863.0	1.93	30.75	2.0	5033.0	0.43	28.33	3.38
	51	1	7749	1.0	7740.0	10.36	0.12	1.0	7740.0	0.99	0.12	0.00
	75	1	7348	1.0	7232.4	39.85	1.57	1.0	7235.6	2.04	1.53	0.04
	101	2	14,154	2.0	11,148.9	43.14	21.23	2.0	11,302.8	12.49	20.14	1.36
	251	3	25,675	3.0	24,651.0	290.21	3.99	3.0	25,023.7	220.85	2.54	1.49
	501	6	60,205	6.0	56,074.6	1571.70	6.86	6.0	57,674.7	2210.90	4.20	2.77
	751	10	106,760	*9.0*	91,949.0	3723.77	13.87	*9.8*	96,218.3	5101.98	9.87	4.44
d15112	25	2	123,242	2.0	104,990.0	2.23	14.81	2.0	105,614.8	0.66	14.30	0.59
	51	2	170,965	2.0	159,159.0	6.81	6.91	2.0	160,303.8	2.08	6.24	0.71
	75	3	247,244	3.0	225,546.2	8.94	8.78	3.0	235,796.0	9.96	4.63	4.35
	101	4	334,778	4.0	309,561.8	23.18	7.53	4.0	312,341.7	19.10	6.70	0.89
	251	7	668,061	7.0	662,487.0	38.93	0.83	7.0	662,636.4	142.76	0.81	0.02
	501	12	1,157,310	12.0	1,091,849.9	571.59	5.66	12.0	1,128,301.9	700.02	2.51	3.23
	751	16	1,570,132	16.0	1,520,234.9	1197.19	3.18	16.0	1,536,716.4	3070.41	2.13	1.07
d18512	25	2	7022	2.0	4863.0	1.94	30.75	2.0	5033.0	0.43	28.33	3.38
	51	1	7561	1.0	7502.0	9.66	0.78	1.0	7502.0	0.76	0.78	0.00
	75	2	13,635	2.0	10,169.7	23.12	25.41	2.0	10,183.8	6.35	25.31	0.14
	101	2	15,798	2.0	11,689.5	63.49	26.01	2.0	11,960.6	21.46	24.29	2.27
	251	4	34,272	4.0	27,381.0	234.47	20.11	4.0	27,484.2	203.04	19.81	0.38
	501	6	59,366	6.0	55,287.8	1496.06	6.87	6.0	56,916.7	1501.90	4.13	2.86
	751	9	101,331	*8.6*	89,522.9	3337.81	11.65	9.0	91,439.3	5233.26	9.76	2.10
fnl4461	25	1	2511	1.0	2168.0	2.38	13.66	1.0	2168.0	0.34	13.66	0.00
	51	2	6270	2.0	4830.0	6.95	22.97	2.0	4847.2	2.57	22.69	0.35
	75	2	6829	2.0	6512.9	25.76	4.63	2.0	6803.4	3.67	0.37	4.27
	101	3	10,575	3.0	9797.1	32.58	7.36	3.0	9903.4	23.98	6.35	1.07
	251	3	31,677	3.0	30,369.8	486.91	4.13	3.0	31,348.2	196.80	1.04	3.12
	501	6	94,759	*5.0*	78,316.9	1913.68	17.35	6.0	81,258.5	1869.35	14.25	3.62
	751	9	137,831	9.0	131,078.1	3015.56	4.90	9.0	134,721.0	3581.66	2.26	2.70
nrw1379	25	1	3501	1.0	3192.0	2.40	8.83	1.0	3192.0	0.35	8.83	0.00
	51	1	5097	1.0	5055.0	9.87	0.82	1.0	5055.0	0.84	0.82	0.00
	75	2	9047	2.0	7451.5	20.77	17.64	2.0	7825.8	5.73	13.50	4.78
	101	3	13,664	3.0	11,117.2	39.67	18.64	3.0	11,465.5	21.08	16.09	3.04
	251	6	31,977	*5.8*	30,270.0	170.42	5.34	*5.0*	30,260.0	109.59	5.37	−0.03
	501	10	68,451	*9.9*	65,908.7	692.18	3.71	10.0	67,196.1	785.44	1.83	1.92
	751	16	122,389	*15.6*	117,459.2	1616.38	4.03	16.0	120,459.4	1855.53	1.58	2.49
pr1002	25	1	19,086	1.0	16,221.0	2.55	15.01	1.0	16,221.0	0.35	15.01	0.00
	51	2	45,316	2.0	39,267.2	8.21	13.35	2.0	39,618.1	3.53	12.57	0.89
	75	3	66,969	3.0	57,195.5	15.12	14.59	3.0	57,742.0	10.09	13.78	0.95
	101	3	75,283	3.0	74,362.0	18.02	1.22	3.0	74,508.0	15.09	1.03	0.20
	251	6	247,410	6.0	235,326.9	165.87	4.88	6.0	245,033.8	122.28	0.96	3.96
	501	10	579,812	10.0	563,081.7	459.23	2.89	10.0	569,556.0	753.56	1.77	1.14
	751	14	988,736	14.0	946,343.7	1942.74	4.29	14.0	967,128.7	3296.91	2.19	2.15
Average	–	–	–	–	–	–	10.43	–	–	–	8.87	1.72
Sum	–	201	–	197.9	–	–	–	199.8	–	–	–	–

and a probabilistic tabu search algorithm. Neither algorithm will increase the number of trees in the solution, although it is possible that the number of trees is further reduced. Therefore, for both algorithms we never perform any neighborhood operation that increases the number of trees, and prioritize the operations that reduce the number of trees.

5.1. Variable neighborhood search heuristic

Variable neighborhood search (VNS) was introduced by Mladenović and Hansen (1997) for complex combinatorial problems. It has been previously applied to vehicle routing problems with precedence and loading constraints (Tricoire et al., 2010; Felipe et al., 2009a; Li et al., 2011). Our VNS heuristic summarized in Algorithm 3 adopts the standard framework, but integrates local search operators and a perturbation function that are tailored for the MTSPPD-LD.

Algorithm 3. The variable neighborhood search heuristic

```
1: INPUTS: VNSMaxIterations and a feasible solution $F$;
2: Initialize $F_{best} \leftarrow F$ and $i \leftarrow 0$;
3: while $i <$ VNSMaxIterations do
4:    $F = localSearch(F)$;
5:    Optimize each tree using VNS-SingleTree heuristic;
6:    if $F$ is better than $F_{best}$ then
7:       $F_{best} \leftarrow F$ and $i \leftarrow 0$;
8:    else
9:       $i \leftarrow i + 1$;
10:   end if
11:   $perturb\_vns(F)$;
12: end while
13: return $F_{best}$
```

Table 4
Performance comparison between the VNS and PTS algorithms for $L = 2.5 \times d_{max}$.

Instance	Size	Initial		VNS				PTS				Gap (%)
		Route	Dist.	Route	Dist.	Time	Improv. (%)	Route	Dist.	Time	Improv. (%)	
brd14051	25	1	5134	1.0	4672.0	2.75	9.00	1.0	4672.0	0.39	9.00	0.00
	51	1	9091	1.0	7740.0	10.62	14.86	1.0	7740.0	0.83	14.86	0.00
	75	1	8990	1.0	7232.4	40.87	19.55	1.0	7235.6	2.15	19.52	0.04
	101	2	15,403	2.0	10,674.0	71.37	30.70	2.0	11,143.4	16.90	27.65	4.21
	251	3	31,338	3.0	24,876.7	512.26	20.62	3.0	25,798.8	348.18	17.68	3.57
	501	5	66,793	*4.9*	54,870.7	2207.33	17.85	5.0	56,326.1	2184.44	15.67	2.58
	751	7	95,065	7.0	89,832.0	3570.15	5.50	7.0	92,880.3	6361.75	2.30	3.28
d15112	25	1	93,981	1.0	93,981.0	2.40	0.00	1.0	93,981.0	0.22	0.00	0.00
	51	2	182,889	2.0	155,400.9	8.04	15.03	2.0	155,787.0	1.13	14.82	0.25
	75	2	218,613	2.0	213,755.2	15.60	2.22	2.0	214,004.0	6.53	2.11	0.12
	101	3	301,170	3.0	294,843.4	26.47	2.10	3.0	296,909.4	9.11	1.41	0.70
	251	6	660,657	6.0	640,857.3	136.24	3.00	6.0	649,633.0	200.78	1.67	1.35
	501	9	1,086,997	9.0	1,045,281.6	695.26	3.84	9.0	1,070,773.1	1156.80	1.49	2.38
	751	12	1,511,530	12.0	1,484,164.4	1404.92	1.81	12.0	1,495,895.5	3666.07	1.03	0.78
d18512	25	1	5134	1.0	4672.0	2.75	9.00	1.0	4672.0	0.39	9.00	0.00
	51	1	8718	1.0	7502.0	9.96	13.95	1.0	7502.0	0.73	13.95	0.00
	75	1	8890	1.0	8629.0	27.98	2.94	1.0	8629.0	1.90	2.94	0.00
	101	2	16,503	2.0	11,281.2	70.20	31.64	2.0	11,582.2	21.89	29.82	2.60
	251	3	31,743	3.0	25,699.5	331.91	19.04	3.0	26,524.1	273.29	16.44	3.11
	501	5	59,300	5.0	53,958.6	2308.02	9.01	5.0	55,448.8	2285.65	6.49	2.69
	751	8	108,713	*7.0*	88,084.0	4571.73	18.98	7.0	90,052.3	5235.76	17.17	2.19
fnl4461	25	1	2511	1.0	2168.0	2.33	13.66	1.0	2168.0	0.35	13.66	0.00
	51	1	4048	1.0	4020.0	10.21	0.69	1.0	4028.4	0.76	0.48	0.21
	75	2	8271	2.0	6498.8	20.62	21.43	2.0	6545.8	8.44	20.86	0.72
	101	2	9617	2.0	9317.3	43.65	3.12	2.0	9449.8	12.50	1.74	1.40
	251	3	36,261	3.0	30,169.4	620.23	16.80	3.0	30,717.9	367.16	15.29	1.79
	501	4	79,758	4.0	74,831.2	3367.81	6.18	4.0	78,069.3	2085.65	2.12	4.15
	751	7	132,516	7.0	126,076.1	3769.14	4.86	7.0	128,758.9	5004.89	2.84	2.08
nrw1379	25	1	3501	1.0	3192.0	2.35	8.83	1.0	3192.0	0.34	8.83	0.00
	51	1	5682	1.0	5055.0	10.69	11.03	1.0	5055.0	0.78	11.03	0.00
	75	2	9706	2.0	7332.8	21.62	24.45	2.0	7582.9	8.11	21.87	3.30
	101	2	12,378	2.0	10,705.0	35.29	13.52	2.0	10,719.7	21.61	13.40	0.14
	251	4	29,881	4.0	29,023.5	175.18	2.87	4.0	29,269.9	177.33	2.05	0.84
	501	8	70,008	8.0	65,203.0	1082.40	6.86	8.0	67,214.7	1766.83	3.99	2.99
	751	12	118,202	12.0	113,634.5	2136.29	3.86	12.0	115,482.8	3554.09	2.30	1.60
pr1002	25	1	19,086	1.0	16,221.0	2.52	15.01	1.0	16,221.0	0.35	15.01	0.00
	51	2	43,985	2.0	37,641.9	9.66	14.42	2.0	38,446.5	2.12	12.59	2.09
	75	2	52,395	2.0	51,090.7	20.81	2.49	2.0	52,268.7	3.79	0.24	2.25
	101	3	85,573	3.0	74,641.6	30.41	12.77	3.0	75,764.2	17.77	11.46	1.48
	251	4	224,709	4.0	218,008.1	238.64	2.98	4.0	217,820.2	394.92	3.07	−0.09
	501	8	564,035	8.0	538,349.1	832.43	4.55	8.0	548,009.1	1045.95	2.84	1.76
	751	11	957,070	11.0	915,391.6	2078.99	4.35	11.0	925,419.6	4386.62	3.31	1.08
Average	–	–	–	–	–	–	10.60	–	–	–	9.38	1.37
Sum	–	157	–	155.9	–	–	–	156.0	–	–	–	–

In this algorithm, we first initialize the current best solution F_{best} to be the feasible solution found after the tree reduction stage (line 2). In each iteration, we first invoke a function *localSearch* (line 4), which performs the six inter-tree operators *node exchange, node relocate, subtree exchange, subtree relocate, node multi-relocate* and *subtree multi-relocate* in order; whenever *localSearch* encounters an improving solution, it restarts from the first operator. Next, we attempt to improve each tree in the current solution individually using the VNS-SingleTree heuristic (line 5). If the current solution is better than F_{best}, then F_{best} is updated (line 6). Finally, a perturbation function called *perturb_vns* is used to diversify the search process (line 11). This function modifies the feasible solution F by randomly performing one of the operations *node exchange, node relocate, subtree exchange* and *subtree relocate*; this is done *VNSPerturbIterations* times. The entire process is repeated until *VNSMaxIterations* consecutive non-improving iterations occur.

5.2. Probabilistic tabu search

Our probabilistic tabu search (PTS) algorithm is motivated by the work done by Erdoğan et al. (2009) on the TSPPD with first-in-first-out loading. This algorithm uses a random sampling phase to realize the tabu effect instead of the traditional tabu list in order to help the search process escape from local optima. One advantage of this approach is that the maximum number of non-improving iterations *PTSMaxIterations* is the only parameter used in the algorithm, which greatly reduces the effort required for parameter tuning.

Let μ be the number of iterations since the last update of the best solution. For the current solution F, we evaluate all possible moves using the operations *node exchange, node relocate, subtree exchange* and *subtree relocate*, and identify a profitable move for each of *node multi-relocation* and *subtree multi-relocation*; these moves constitute a neighborhood $N_{pts}(F)$. If the best move in $N_{pts}(F)$ leads to a solution better than the best-known one, we perform it (and set μ to be zero). If this is not the case, we compute a value λ as follows: if $\mu \leqslant$ *PTSMaxIterations*/2, then $\lambda = \max\{0.05, \mu/\mathit{PTSMaxIterations}\}$; otherwise, $\lambda = \max\{0.05, (\mathit{PTSMaxIterations} - \mu)/\mathit{PTSMaxIterations}\}$. We then randomly select $100 \times \lambda$ percent of moves in $N_{pts}(F)$ and perform the best of these moves. This process repeats until no improving solution has been found in the last *PTSMaxIterations* consecutive iterations. Finally, we further improve

Table 5
Impacts of operators on VNS when $L = 1.5 \times d_{max}$.

Instance	Size	Node exchange		Node relocate		Subtree exchange		Subtree relocate		Node multi-relocate		Subtree multi-relocate	
		Dist.	Time	Dist.	Time	Dist.	Time	Dist.	Time	Dist.	Time	Dist.	Time
brd14051	25	0.00	0.11	0.00	−0.06	0.00	−0.06	0.00	−0.55	0.00	−0.07	0.00	−0.10
	51	0.00	−1.35	0.11	44.80	0.00	−0.99	0.05	9.21	0.11	8.54	0.00	−6.13
	75	0.00	−0.48	0.01	−12.77	0.00	0.77	0.00	2.93	0.00	−9.70	0.00	−1.80
	101	0.00	0.10	0.00	−0.23	0.00	0.00	0.00	−0.16	0.00	−0.14	0.00	−0.23
	251	−0.03	4.30	−0.03	37.03	−0.30	0.40	−0.10	15.89	0.03	19.84	−0.27	27.90
	501	0.26	4.82	0.37	−7.29	−0.07	2.45	0.60	−18.20	0.54	−11.69	0.36	−0.51
	751	−0.17	7.26	0.51	45.68	−0.18	18.51	0.61	19.31	0.45	43.49	0.12	5.02
d15112	25	0.00	0.10	0.00	−0.03	0.00	0.01	0.00	−0.01	1.51	−9.07	0.00	−0.05
	51	0.11	−5.01	0.27	−2.91	0.00	−0.08	0.43	−28.54	0.38	−19.27	0.00	−0.22
	75	0.42	−8.14	0.35	−26.57	−0.12	0.08	1.27	−24.70	−0.04	−1.39	0.23	−2.77
	101	0.05	9.97	−2.26	37.43	0.00	5.72	−0.11	2.34	0.93	−28.14	0.00	−0.60
	251	−0.51	5.48	1.70	−20.72	0.16	1.70	1.00	−2.61	0.21	6.22	0.45	−14.68
	501	0.07	2.24	0.10	26.03	−0.10	−1.38	−0.07	8.29	0.53	22.87	0.17	4.88
	751	0.18	8.17	0.34	7.15	0.10	17.62	0.75	−31.37	0.02	2.43	−0.08	5.34
d18512	25	0.00	0.07	0.00	−0.10	0.00	−0.02	0.00	−0.57	0.00	−0.05	0.00	−0.12
	51	0.02	5.32	−0.04	31.19	0.04	−4.32	0.01	6.49	−0.01	4.24	−0.02	3.43
	75	0.00	−3.79	0.18	30.95	0.00	0.60	0.09	29.04	0.20	15.59	0.00	−4.07
	101	−0.06	11.29	0.14	−21.70	0.00	−0.09	0.03	−6.46	0.00	−0.08	0.00	−0.16
	251	0.43	−16.08	0.40	1.44	0.20	−8.46	0.19	−13.70	0.35	−20.69	0.10	−10.49
	501	−0.03	1.40	−0.56	46.05	−0.15	12.45	−0.24	10.84	−0.25	19.73	0.18	−7.40
	751	0.42	11.74	0.27	−0.25	−0.16	−0.55	0.65	−13.81	0.89	−2.59	0.23	−14.62
fnl4461	25	0.00	0.02	0.00	−0.12	0.00	−0.06	0.00	−0.04	0.00	0.02	0.00	−0.06
	51	0.00	0.37	0.00	4.91	0.00	−0.43	0.00	−3.84	0.00	−0.10	0.00	−0.26
	75	0.16	−11.49	0.51	−29.46	−0.03	1.26	0.17	−18.49	0.11	−5.89	0.11	−3.20
	101	−0.35	24.91	−0.06	12.26	0.29	−8.86	0.69	−12.03	0.90	2.95	0.00	−0.91
	251	−0.04	14.62	−0.17	39.06	−0.09	7.43	−0.01	13.85	0.26	20.62	−0.15	10.95
	501	−0.13	15.59	0.14	−5.68	0.00	−7.77	0.12	−4.27	−0.07	−0.85	0.00	−4.83
	751	−0.25	16.98	−0.34	30.92	−0.14	−1.18	0.21	−7.51	0.25	16.44	−0.32	3.24
nrw1379	25	0.00	0.26	0.00	0.53	0.00	−0.03	0.00	−6.75	0.00	0.05	0.00	0.03
	51	0.07	−0.93	0.60	−3.10	0.00	0.64	−0.34	−10.98	0.20	−5.75	0.00	1.39
	75	0.02	6.56	−0.34	21.05	0.00	−2.75	0.59	−28.49	0.72	0.66	−0.06	−1.08
	101	0.51	−16.05	0.68	11.49	0.11	−8.54	1.82	−27.95	0.04	−16.92	0.41	−5.54
	251	−0.37	5.09	0.49	39.17	0.35	10.51	0.18	10.60	0.44	18.75	0.79	18.06
	501	−0.24	17.26	−0.11	4.11	−0.06	−0.88	−0.20	13.80	0.00	5.75	−0.11	10.48
	751	0.21	18.16	0.02	22.85	−0.43	13.71	0.57	−15.85	0.19	3.85	−0.05	−2.36
pr1002	25	0.00	0.13	0.00	0.12	0.00	0.35	0.00	0.32	0.00	0.25	0.00	0.30
	51	0.00	−0.07	0.00	0.58	0.00	−0.13	0.00	0.10	0.00	−0.33	0.00	−0.23
	75	0.82	−10.86	0.08	−14.35	0.02	−1.22	0.04	−6.16	0.51	−10.70	0.00	−0.50
	101	0.05	1.12	0.13	5.37	−0.05	−2.93	−0.13	−3.96	0.19	0.98	0.00	−0.46
	251	−0.21	8.83	−0.31	12.49	0.00	−0.46	−0.16	1.08	−0.22	6.60	0.00	−1.10
	501	0.17	1.50	−0.30	12.44	0.00	−0.45	0.55	−12.22	0.06	14.44	−0.12	6.03
	751	−0.32	29.35	−0.28	17.28	−0.45	24.80	0.21	−6.33	0.12	−16.50	−0.11	0.62
# improved	–	10	11	12	16	14	23	9	27	5	20	10	28
Average	–	0.03	3.78	0.06	9.45	−0.03	1.60	0.23	−3.84	0.23	1.77	0.04	0.31

each tree in the solution produced by the PTS algorithm using the VNS-SingleTree heuristic.

6. Computational experiments

6.1. Instance generation

To evaluate the algorithms proposed in this paper, we conducted experiments using the data set introduced by Carrabs et al. (2007b). This data set was derived from six TSP instances *fnl4461, brd14051, d15112, d18512, nrw1379* and *pr1002*, taken from TSPLIB (Reinelt, 1991), which is a standard test suite for the TSP. For each of these TSP instances, subsets of vertices were selected with 25, 51, 75, 101, 251, 501 and 751 vertices. One arbitrary vertex was designated as the depot and the rest were paired to form requests using a random matching process, resulting in 42 request sets. This data set is available at http://www.computational-logistics.org/orlib/mtsppdl. Let $d_{max} = \max_{i \in R}\{d(0^+, i^+) + d(i^+, i^-) + d(i^-, 0^-)\}$ be the greatest distance for any route involving a single request; we imposed for each instance a travel distance limit $L = 1.5 \times d_{max}$, $2.0 \times d_{max}$ and $2.5 \times d_{max}$, thereby generating 126 test instances.

6.2. Experimental setup

We conducted a series of preliminary experiments to tune the parameters used in our various algorithms. Each of the algorithms apart from the SA algorithm has only one or two parameters, so we were able to determine good parameter values using trial and error: *maxTime* = 600 s and *EPPerturbIterations* =10 for the EP algorithm, *VNSMaxIterations* = 10 and *VNSPerturbIterations* = 25 for the VNS heuristic, and *PTSMaxIterations* = 1000 for the PTS algorithm.

The values of some parameters in our SA algorithm were selected according to past experience in the literature, namely *startTemperature* = 20,000, *temperatureLimit* = 0.01, *SAMaxIterations* = 250 and $\gamma = 0.98$. We set $\alpha_3 = 1$ and then determined $\beta = 10$, $\alpha_1 = 2000$ and $\alpha_2 = 200$ by conducting a full 2^3 experimental design process based on the instances with 250 vertices (18 instances). When *timeLimit* = 1800 s, these parameter settings resulted in the best solutions on average with the given *startTemperature, temper-*

Table 6
Impacts of operators on VNS when $L = 2.0 \times d_{max}$.

Instance	Size	Node exchange		Node relocate		Subtree exchange		Subtree relocate		Node multi-relocate		Subtree multi-relocate	
		Dist.	Time	Dist.	Time	Dist.	Time	Dist.	Time	Dist.	Time	Dist.	Time
brd14051	25	0.00	−0.03	0.00	0.47	0.00	−0.15	0.00	−7.69	0.00	−0.04	0.00	−1.87
	51	0.00	0.05	0.00	0.19	0.00	0.18	0.00	0.04	0.00	0.17	0.00	0.01
	75	0.00	0.04	0.00	0.10	0.00	0.00	0.00	−0.02	0.00	0.01	0.00	−0.02
	101	−0.13	16.29	−0.31	25.05	−0.28	17.71	−0.35	5.13	−0.04	11.74	−0.28	4.42
	251	0.03	−20.96	0.00	−25.54	0.01	−6.62	0.03	−28.31	0.52	−28.45	0.00	−8.03
	501	0.86	−22.79	−0.69	29.75	0.09	−8.16	0.53	−9.92	1.24	−1.32	0.61	−17.80
	751	0.93	−9.16	1.20	3.30	1.05	−7.99	0.70	−21.62	1.26	13.45	0.49	0.92
d15112	25	0.00	−0.36	0.00	−10.05	0.00	−3.25	0.00	1.10	0.00	8.46	0.00	4.05
	51	−1.20	8.81	0.60	−11.49	0.00	−0.06	0.85	−19.67	0.00	0.43	0.00	−0.13
	75	0.00	−0.02	0.00	17.37	0.00	−4.23	0.00	−10.46	1.87	77.42	0.00	24.75
	101	0.20	−1.69	−0.58	9.43	−0.07	−4.72	−0.05	−1.05	−0.37	3.18	−0.10	−6.28
	251	0.00	1.26	0.00	−2.14	0.00	−0.50	0.00	−1.01	0.00	−0.78	0.00	−1.11
	501	0.12	5.17	0.47	7.12	0.22	3.27	0.15	−1.01	1.87	−25.40	0.02	11.38
	751	0.08	7.53	−0.24	11.48	0.10	−0.85	−0.38	11.34	0.18	2.21	−0.36	−2.93
d18512	25	0.00	0.03	0.00	0.43	0.00	−0.24	0.00	−7.91	0.00	−0.35	0.00	−1.76
	51	0.00	−0.02	0.00	0.08	0.00	0.11	0.00	0.00	0.00	0.03	0.00	0.03
	75	−0.26	8.26	0.06	12.61	−0.24	−1.23	−0.25	13.78	−0.28	21.20	−0.06	7.99
	101	0.47	−8.89	0.97	0.85	0.43	−6.36	1.10	−29.12	0.30	−13.83	−0.05	−7.35
	251	−0.84	8.25	−1.75	44.62	−1.19	13.38	−0.05	21.73	0.20	7.35	0.35	4.32
	501	−0.10	4.97	−0.22	13.25	0.14	4.27	0.20	−8.36	0.36	6.94	−0.07	3.45
	751	0.23	−2.68	0.24	19.91	−0.21	10.82	−0.78	30.04	0.71	20.63	0.42	0.32
fnl4461	25	0.00	0.35	0.00	0.26	0.00	0.05	0.00	0.15	0.00	0.21	0.00	0.24
	51	0.00	−0.31	0.00	17.72	0.00	0.28	0.18	1.78	0.00	9.49	0.00	−0.73
	75	−0.15	16.43	−0.10	11.29	0.00	0.77	0.37	7.19	0.29	0.86	−0.15	4.22
	101	−0.07	1.37	−0.38	12.98	0.08	10.32	0.21	−3.39	−0.06	19.20	0.09	0.20
	251	−0.71	17.13	−0.92	28.75	−0.18	2.90	−1.05	6.65	0.53	11.62	−1.24	11.99
	501	0.60	−1.42	0.75	14.97	−0.24	8.75	0.82	−24.08	0.97	−14.17	0.86	11.12
	751	−0.14	5.93	−0.66	19.39	−0.35	18.23	−0.17	−1.10	0.48	−7.57	−0.39	14.94
nrw1379	25	0.00	0.15	0.00	0.05	0.00	0.07	0.00	0.07	0.00	0.00	0.00	0.00
	51	0.00	0.15	0.00	0.00	0.00	−0.03	0.00	−0.03	0.00	0.04	0.00	−0.05
	75	0.07	−11.47	−0.01	14.82	0.00	0.05	−0.17	−1.36	−0.01	−5.98	−0.04	5.20
	101	−0.30	2.93	0.21	9.20	−0.06	4.16	1.26	2.44	0.30	8.44	1.15	−6.79
	251	0.12	3.40	−0.53	23.10	−0.38	−16.12	−0.61	3.39	−0.22	13.88	−0.26	2.08
	501	−0.07	11.45	−0.30	20.38	0.06	3.10	−0.26	−8.30	−0.13	10.63	0.06	−7.01
	751	0.39	−10.02	−0.06	17.11	0.10	−4.87	0.27	−18.01	0.92	−21.93	0.16	0.06
pr1002	25	0.00	−0.02	0.00	−0.18	0.00	−0.11	0.00	−0.07	0.00	−0.10	0.00	−0.17
	51	0.03	−0.26	0.32	44.86	0.00	−0.25	0.03	11.37	0.34	16.67	0.05	−5.94
	75	−0.06	5.90	−0.19	35.76	−0.15	−0.32	−0.03	3.54	−0.49	26.49	−0.31	4.26
	101	0.00	−0.07	0.00	−0.52	0.00	−0.42	0.00	−0.53	0.00	−0.26	0.00	−0.34
	251	−0.44	−2.88	−0.09	9.75	−0.08	−7.67	0.08	−5.98	−0.24	7.25	−0.18	−1.60
	501	0.49	−17.02	0.89	−40.62	0.00	−0.67	0.38	−25.67	0.37	−12.41	0.01	1.24
	751	0.06	0.20	−0.09	10.08	−0.07	−5.86	0.10	−5.45	0.07	−5.08	0.41	−5.99
# improved	–	13	19	17	7	13	23	12	25	9	15	13	19
Average	–	0.00	0.38	−0.03	9.43	−0.03	0.42	0.07	−2.87	0.26	3.82	0.03	0.98

atureLimit, SAMaxIterations and γ values. We also observed that few improvements could be achieved by the SA algorithm after 600 s, so we finally set *timeLimit* = 600 s.

All algorithms were coded in C++and compiled using the g++compiler. All experiments were conducted on a Dell server with an Intel Xeon E5520 2.26 GHz CPU, 8 GB RAM and Linux operating system. Computation times reported are in CPU seconds on this server.

6.3. Results and analysis

For each instance, we first used the CI heuristic to produce an initial solution and then applied the SA and EP algorithms to minimize the number of routes. Since the SA and EP algorithms are not deterministic, we executed each of these two algorithms 10 times for each instance. Table 1 gives the computational results with respect to the first stage, which include the initial number of routes (columns "CI") generated by the CI heuristic, and the minimum and average number of routes (columns "Min" and "Ave.") generated by the SA and EP algorithms. The smallest number of routes achieved among these approaches for each instance found in the first stage is marked in bold. The results show that the minimum number of routes obtained by the EP algorithm are no larger than those obtained by the SA algorithm for all instances. The EP algorithm generated solutions with fewer routes on average than the SA algorithm for 34 out of the 126 instances. Only for one instance (the 751-node *brd14051* instance when $L = 2.0 \times d_{max}$) is the average number of routes for the EP algorithm greater than for the SA algorithm. This suggests that the EP algorithm is superior to the SA algorithm for reducing the number of vehicle routes for the MTSPPD-LD.

We chose the results with the smallest number of routes generated by the EP algorithm as the initial solutions for the second stage. The VNS and PTS algorithms were also executed 10 times for each instance; the results are summarized in Tables 2–4. In these tables, the number of routes and the total travel distance of the initial solution for each instance are given in columns "Initial Route" and "Initial Dist.", respectively. We present the average number of routes, the average distance and the average computation times in the columns "Route", "Dist." and "Time", respectively.

Table 7
Impacts of operators on VNS when $L = 2.5 \times d_{max}$.

Instance	Size	Node exchange		Node relocate		Subtree exchange		Subtree relocate		Node multi-relocate		Subtree multi-relocate	
		Dist.	Time	Dist.	Time	Dist.	Time	Dist.	Time	Dist.	Time	Dist.	Time
brd14051	25	0.0	0.2	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.2	0.0	0.1
	51	0.0	−0.1	0.0	−0.1	0.0	−0.1	0.0	−0.1	0.0	0.0	0.0	−0.1
	75	0.0	0.1	0.0	0.1	0.0	−0.1	0.0	0.1	0.0	0.0	0.0	0.0
	101	−1.2	5.0	1.2	1.3	−0.6	6.6	−2.9	−2.8	−0.6	15.9	0.7	5.1
	251	−0.9	15.9	0.0	16.5	−0.1	−4.4	0.1	−0.3	−0.4	12.9	−0.2	−3.9
	501	−0.1	−12.7	0.1	46.0	−0.1	9.2	1.2	7.4	−0.1	7.7	0.1	−15.7
	751	−0.7	47.9	−0.3	16.8	−0.3	18.0	0.1	−0.2	−0.1	15.6	−0.1	11.6
d15112	25	0.0	−0.1	0.0	−0.1	0.0	0.0	0.0	−0.1	0.0	−0.1	0.0	0.0
	51	0.0	−2.4	0.1	−6.3	0.0	−0.3	0.0	−10.2	0.0	−2.8	0.1	−8.3
	75	−0.2	14.7	−0.3	5.7	0.0	−0.1	−0.3	4.3	0.0	−1.7	0.0	−0.2
	101	−0.1	−10.6	−0.1	14.6	−0.6	9.9	−0.5	−1.4	0.1	7.6	−0.2	−1.1
	251	−0.6	39.6	−0.5	16.0	−0.1	14.3	−0.7	37.6	0.0	−7.0	0.0	−12.5
	501	0.0	11.9	0.5	−10.9	0.1	1.0	0.3	2.9	0.5	−9.0	0.1	−10.2
	751	−0.4	50.7	0.3	−15.7	−0.1	0.0	0.4	−15.6	0.2	5.3	0.0	0.0
d18512	25	0.0	0.0	0.0	−0.1	0.0	0.0	0.0	−0.1	0.0	0.0	0.0	0.1
	51	0.0	−0.1	0.0	−0.1	0.0	0.0	0.0	−0.2	0.0	−0.3	0.0	0.0
	75	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.1	0.0	0.1
	101	1.3	−9.0	1.6	−9.5	0.9	−3.1	3.0	−41.4	0.4	−8.8	1.0	−4.2
	251	0.0	33.7	0.2	56.3	0.1	−0.1	−0.3	31.8	0.2	8.6	0.1	14.9
	501	−0.3	−4.0	−1.0	34.2	−0.7	0.5	−0.8	−10.4	−0.7	−6.3	−0.6	−0.5
	751	−0.3	15.8	−1.2	27.2	−0.2	4.1	−1.6	−13.4	0.1	11.1	−0.1	13.8
fnl4461	25	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	−0.1
	51	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	−0.1
	75	−0.2	12.7	0.0	16.8	−0.1	17.5	0.1	−7.7	0.4	8.6	0.1	1.2
	101	−0.3	6.7	1.1	−13.3	0.0	−0.1	0.0	−12.3	0.0	−1.8	0.0	−0.2
	251	0.0	7.7	−0.4	24.7	0.1	11.0	−0.4	−8.0	0.1	−4.0	−0.3	−8.1
	501	0.3	−24.1	0.8	−6.6	0.4	−21.7	1.1	−31.2	0.3	−19.9	0.3	−23.7
	751	0.0	15.1	−0.8	73.8	0.1	−8.6	−0.2	−3.5	0.3	0.0	−0.5	20.4
nrw1379	25	0.0	0.5	0.0	0.0	0.0	−0.1	0.0	0.0	0.0	0.0	0.0	0.0
	51	0.0	0.0	0.0	−0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	75	0.0	4.0	−0.5	53.8	0.0	1.1	−0.1	49.8	0.1	−7.4	0.0	−2.5
	101	0.0	0.1	0.0	−8.2	0.0	0.0	0.0	−3.5	0.0	0.3	0.0	−0.2
	251	−0.2	6.4	0.2	−15.8	0.0	−0.2	−0.1	7.4	−0.1	−0.3	0.0	0.0
	501	−0.2	21.4	−0.6	47.8	−0.6	18.0	−1.0	9.3	0.2	15.9	0.3	6.5
	751	−0.2	9.1	0.0	−0.5	0.3	−17.5	0.1	−17.6	0.6	−7.0	0.3	−10.2
pr1002	25	0.0	0.0	0.0	−0.1	0.0	−0.1	0.0	−0.1	0.0	0.0	0.0	−0.1
	51	0.4	−2.5	2.0	4.6	0.0	2.5	3.5	−10.5	0.0	−0.4	−1.2	−3.9
	75	−0.5	17.6	0.2	4.7	−0.2	4.8	−0.5	−9.3	0.1	−9.2	−0.3	7.3
	101	−0.2	−5.0	−0.2	44.5	0.4	−11.7	−0.5	−7.0	−0.1	21.6	−0.3	10.5
	251	0.3	−16.0	0.4	−33.0	0.0	−1.8	0.6	−33.6	0.3	−28.7	0.0	−1.0
	501	0.2	0.1	0.1	8.2	0.3	−16.5	0.1	−2.4	0.5	−21.7	0.1	−6.5
	751	0.0	−7.1	−0.4	2.3	0.0	−3.6	−0.8	2.6	0.2	−9.3	−0.2	2.3
Average	0.00	−0.10	5.80	0.06	9.42	−0.02	0.69	0.00	−2.13	0.07	−0.35	−0.02	−0.47

The columns "Improv. (%)" show the percentage by which each of these two algorithms improved the initial distance; this value is calculated by (*Initial Dist.–VNS Dist.*)/*Initial Dist.* or (*Initial Dist.–PTS Dist.*)/*Initial Dist.* The column "Gap (%)" is used to compare the performance of the VNS and PTS algorithms, which is defined as (*PTS Dist.–VNS Dist.*)/*PTS Dist.*. In the last two lines of each table, the average improvement percentage, the average gap and the total number of routes are reported.

The columns "Improv. (%)" indicate that both the VNS and PTS algorithms can significantly reduce the travel distance of the given initial solutions. An inspection of the columns "Route" in Tables 2–4 reveals that the numbers of vehicle routes for some instances were further reduced in the second stage, which are marked in italics. The average numbers of routes in the solutions generated by the VNS algorithm are no larger than those generated by the PTS algorithm for all instances except instances *d15112-751* with $L = 1.5 \times d_{max}$ and *nrw1379-251* with $L = 2.0 \times d_{max}$. Furthermore, the computation times used by these two algorithms are comparable, and the average gaps in all tables are positive; this clearly shows that on average the VNS algorithm is better suited to handle the second stage of our problem than the PTS algorithm.

Further, we studied the impact of each operator in the VNS heuristic by removing one operator in turn and executing the resulting heuristics 10 times for each instance. The results are reported in Tables 5–7, where the heading of each column refers to the operator removed from the VNS heuristic. For each operator, the columns "Route" and "Dist." give the percentage difference between VNS and the resulting reduced version of VNS in their average solution values and computation times, respectively; these percentage values are calculated by (*reduced version – full version*)/*full version*. A negative value indicates that on average the reduced version produced a better or faster solution than the original VNS for that instance. The row # *improved* in each table shows the number of instances (out of 42) where the reduced version is superior to the original. In terms of the total travel distance, the average percentage values of the six operators over all instances are: −0.02, 0.03, −0.03, 0.10, 0.18, 0.02, which imply that operators *node relocate, subtree relocate, node multi-relocate and subtree multi-relocate* have a positive effect on average solution quality.

The above experimental results suggest that the best two-stage heuristic among those we investigated consists of the EP and VNS algorithms. To facilitate easy comparisons for future researchers on

Table 8
The best solutions for all test instances.

Instance	Size	$L = 1.5 \times d_{max}$		$L = 2.0 \times d_{max}$		$L = 2.5 \times d_{max}$	
		Route	Dist.	Route	Dist.	Route	Dist.
brd14051	25	2	5086	2	4863	1	4672
	51	2	9352	1	7740	1	7740
	75	2	8543	1	7232	1	7232
	101	2	11,169	2	11,099	2	10,102
	251	5	27,195	3	24,575	3	24,161
	501	8	58,028	6	55,152	4	53,843
	751	12	96,068	9	90,671	7	87,538
d15112	25	2	108,207	2	104,990	1	93,981
	51	3	178,863	2	157,242	2	155,358
	75	4	239,511	3	225,539	2	212,760
	101	5	325,761	4	303,285	3	289,621
	251	10	700,366	7	662,487	6	617,293
	501	16	1,145,838	12	1,066,654	9	1,031,519
	751	22	1,596,048	15	1,487,768	12	1,453,373
d18512	25	2	5086	2	4863	1	4672
	51	2	9245	1	7502	1	7502
	75	2	10,147	2	10,113	1	8629
	101	2	11,742	2	11,612	2	10,702
	251	5	27,945	4	26,487	3	25,403
	501	8	56,790	6	53,859	4	52,079
	751	11	91,670	8	87,678	6	84,976
fnl4461	25	1	2168	1	2168	1	2168
	51	2	4830	2	4830	1	4020
	75	3	7399	2	6481	2	6481
	101	4	10,608	3	9689	2	9141
	251	4	30,651	3	29,607	2	29,232
	501	7	81,994	5	76,215	4	73,109
	751	12	135,940	9	127,901	7	123,282
nrw1379	25	2	3464	1	3192	1	3192
	51	2	5398	1	5055	1	5055
	75	3	8207	2	7384	2	7252
	101	4	11,933	3	11,058	2	10,705
	251	7	31,075	5	29,271	4	28,537
	501	13	68,202	9	63,862	7	62,474
	751	21	122,587	15	115,661	12	111,760
pr1002	25	1	16,221	1	16,221	1	16,221
	51	3	47,905	2	39,103	2	37,189
	75	4	64,102	3	56,644	2	50,661
	101	5	87,700	3	74,362	3	73,755
	251	8	257,198	5	224,346	4	214,695
	501	14	597,464	10	550,961	8	529,360
	751	19	1,008,027	14	934,649	10	899,531

the MTSPPD-LD, we report the best solutions for all test instances in Table 8; these best solutions were extracted from the experimental results of the VNS heuristic, the six reduced version of the VNS heuristic and the PTS algorithm.

7. Conclusions

This paper introduces an extension to the TSPPDL that considers multiple vehicles as well as a limit to the distance that a vehicle can travel; both of these additional factors are very common in practice. Using the existing tree representation for feasible TSPPDL solutions, we devised six new inter-tree neighborhood operators. We then examined a two-stage approach for this problem where the first stage aims to reduce the number of vehicles and the second stage focuses on reducing the total distance traveled. For the first stage we investigated an SA algorithm and an ejection pool algorithm, and for the second stage we examined a VNS algorithm and a probabilistic tabu search. Extensive experiments showed that the combination of ejection pool and VNS produces the best results.

The experiments and analysis presented in this study serves as benchmarks for future researchers. Although the four algorithms that we examined are popular choices for solving the VRP and its variants, they are by no means the only choices. Other heuristics and meta-heuristics can be tried, which may employ the six proposed inter-tree operators in their search processes.

We believe the addition of multiple vehicles and a distance constraint makes the MTSPPD-LD a significantly more accurate model of certain practical transportation problems than those proposed in existing literature. Future research can consider further practical constraints, such as time windows, capacitated vehicles or multi-period vehicle routing.

Acknowledgements

The corresponding author was partially supported by the Fundamental Research Funds for the Central Universities, HUST: 2012QN213.

References

Ahuja, R.K., Orlin, J.B., Pallottino, S., Scaparra, M.P., Scutellà, M.G., 2004. A multi-exchange heuristic for the single-source capacitated facility location problem. Management Science 50 (6), 749–760.
Ahuja, R.K., Orlin, J.B., Sharma, D., 2001. Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem. Mathematical Programming 91 (1), 71–97.

Alba, M.A., Cordeau, J.-F., Dell'Amico, M., Iori, M., in press. A branch-and-cut algorithm for the double traveling salesman problem with multiple stacks. INFORMS Journal on Computing.

Bent, R., Van Hentenryck, P., 2004. A two-stage hybrid local search for the vehicle routing problem with time windows. Transportation Science 38 (4), 515–530.

Bent, R., Van Hentenryck, P., 2006. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Computers and Operations Research 33 (4), 875–893.

Bräysy, O., Gendreau, M., 2005. Vehicle routing problem with time windows, part I: route construction and local search algorithms. Transportation Science 39 (1), 104–118.

Carrabs, F., Cerulli, R., Cordeau, J.-F., 2007a. An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with LIFO or FIFO loading. INFOR: Information Systems and Operational Research 45 (4), 223–238.

Carrabs, F., Cerulli, R., Grazia Speranza, M., in press. A branch-and-bound algorithm for the double travelling salesman problem with two stacks. Networks.

Carrabs, F., Cordeau, J.-F., Laporte, G., 2007b. Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. INFORMS Journal on Computing 19 (4), 618–632.

Cassani, L., Righini, G., 2004. Heuristic algorithms for the TSP with rear-loading. In: 35th Annual Conference of the Italian Operational Research Society (AIRO XXXV). Lecce, Italy.

Cordeau, J.-F., Iori, M., Laporte, G., Salazar González, J.J., 2010. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. Networks 55 (1), 46–59.

Côté, J.-F., Gendreau, M., Potvin, J.-Y., 2009. Large Neighborhood Search for the Single Vehicle Pickup and Delivery Problem with Multiple Loading Stacks. Working paper CIRRELT-2009-47.

Dumitrescu, I., Ropke, S., Cordeau, J.-F., Laporte, G., 2009. The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. Mathematical Programming 121 (2), 269–305.

Erdoğan, G., Cordeau, J.-F., Laporte, G., 2009. The pickup and delivery traveling salesman problem with first-in-first-out loading. Computers and Operations Research 36 (6), 1800–1808.

Erera, A.L., Morales, J.C., Savelsbergh, M., 2010. The vehicle routing problem with stochastic demand and duration constraints. Transportation Science 44 (4), 474–492.

Felipe, A., Ortuño, M.T., Tirado, G., 2009a. The double traveling salesman problem with multiple stacks: a variable neighborhood search approach. Computers and Operations Research 36 (11), 2983–2993.

Felipe, A., Ortuño, M.T., Tirado, G., 2009b. New neighborhood structures for the double traveling salesman problem with multiple stacks. TOP 17 (1), 190–213.

Felipe, A., Ortuño, M.T., Tirado, G., 2011. Using intermediate infeasible solutions to approach vehicle routing problems with precedence and loading constraints. European Journal of Operational Research 211 (1), 66–75.

Frangioni, A., Necciari, E., Scutellà, M., 2004. A multi-exchange neighborhood for minimum makespan parallel machine scheduling problems. Journal of Combinatorial Optimization 8 (2), 195–220.

Gao, X., Lim, A., Qin, H., Zhu, W., 2011. Multiple pickup and delivery TSP with LIFO and distance constraints: a VNS approach. Lecture Notes in Computer Science 6704, 193–202.

Healy, P., Moll, R., 1995. A new extension of local search applied to the dial-a-ride problem. European Journal of Operational Research 83 (1), 83–104.

Homberger, J., Gehring, H., 2005. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. European Journal of Operational Research 162 (1), 220–238.

Ibaraki, T., Imahori, S., Kubo, M., Masuda, T., Uno, T., Yagiura, M., 2005. Effective local search algorithms for routing and scheduling problems with general time-window constraints. Transportation Science 39 (2), 206–232.

Iori, M., Martello, S., 2010. Routing problems with loading constraints. TOP 18 (1), 4–27.

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

Ladany, S.P., Mehrez, A., 1984. Optimal routing of a single vehicle with loading and unloading constraints. Transportation Planning and Technology 8 (4), 301–306.

Laporte, G., Nobert, Y., Desrochers, M., 1985. Optimal routing under capacity and distance restrictions. Operations Research 33 (5), 1050–1073.

Levitin, G., Abezgaouz, R., 2003. Optimal routing of multiple-load AGV subject to LIFO loading constraints. Computer and Operations Research 30 (3), 397–410.

Li, C.-L., Simchi-Levi, D., Desrochers, M., 1992. On the distance constrained vehicle routing problem. Operations Research 40 (4), 790–799.

Li, H., Lim, A., 2003. Local search with annealing-like restarts to solve the VRPTW. European Journal of Operational Research 150 (1), 115–127.

Li, Y., Lim, A., Oon, W.-C., Qin, H., Tu, D., 2011. The tree representation for the pickup and delivery traveling salesman problem with LIFO loading. European Journal of Operational Research 212 (3), 482–496.

Lim, A., Zhang, X., 2007. A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. INFORMS Journal on Computing 19 (3), 443–457.

Lusby, R.M., Larsen, J., Ehrgott, M., Ryan, D., 2010. An exact method for the double TSP with multiple stacks. International Transactions in Operational Research 17 (5), 637–652.

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

Nagata, Y., Bräysy, O., 2009. A powerful route minimization heuristic for the vehicle routing problem with time windows. Operations Research Letters 37 (5), 333–338.

Nagy, G., Salhi, S., 2005. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. European Journal of Operational Research 162 (1), 126–141.

Pacheco, J.A., 1997. Metaheuristic based on a simulated annealing process for one vehicle pick-up and delivery problem in LIFO unloading systems. In: Proceedings of the Tenth Meeting of the Europen Chapter of Combinatorial Optimization (ECCO X). Tenerife, Spain.

Petersen, H.L., Archetti, C., Speranza, M., Grazia, S., 2010. Exact solutions to the double travelling salesman problem with multiple stacks. Networks 56 (4), 229–243.

Petersen, H.L., Madsen, O.B.G., 2009. The double travelling salesman problem with multiple stacks – formulation and heuristic solution approaches. European Journal of Operational Research 198 (1), 139–147.

Pisinger, D., Ropke, S., 2007. A general heuristic for vehicle routing problems. Computers and Operations Research 34 (8), 2403–2435.

Reinelt, G., 1991. TSPLIB – traveling salesman problem library. ORSA Journal on Computing 3 (4), 376–384.

Renaud, J., Boctor, F.F., Laporte, G., 2002. Perturbation heuristics for the pickup and delivery traveling salesman problem. Computers and Operations Research 29 (9), 1129–1141.

Renaud, J., Boctor, F.F., Ouenniche, J., 2000. A heuristic for the pickup and delivery traveling salesman problem. Computers and Operations Research 27 (10), 905–916.

Toth, P., Vigo, D. (Eds.), 2002. The vehicle routing problem. SIAM, Philadelphia, PA.

Tricoire, F., Doerner, K.F., Hartl, R.F., Iori, M., 2010. Heuristic and exact algorithms for the multi-pile vehicle routing problem. OR Spectrum 33 (4), 931–959.