ELSEVIER

European Journal of Operational Research 118 (1999) 485–504

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/orms

Theory and Methodology

A route-neighborhood-based metaheuristic for vehicle routing problem with time windows

Fuh-Hwa Franklin Liu *, Sheng-Yuan Shen

Department of Industrial Engineering and Management, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu 30010, Taiwan

Received 1 February 1997; accepted 1 July 1998

Abstract

In this paper, a two-stage metaheuristic based on a new neighborhood structure is proposed to solve the vehicle routing problem with time windows (VRPTW). Our neighborhood construction focuses on the relationship between route(s) and node(s). Unlike the conventional methods for parallel route construction, we construct routes in a nested parallel manner to obtain higher solution quality. Valuable information extracted from the previous parallel construction runs is used to enhance the performance of parallel construction. In addition, when there are only a few unrouted customers left, we design a special procedure for handling them. Computational results for 60 benchmark problems are reported. The results indicate that our approach is highly competitive with all existing heuristics, and in particular very promising for solving problems of large size.

Keywords: Routing; Time windows; Neighborhood; Metaheuristics

1. Introduction

Routing and scheduling problems have been an intensive research area during the past several decades. In order to meet real life needs, a vast number of extensions that generalize the classical vehicle routing problem (VRP) were addressed; see for example surveys in Refs. [3], [11]. One of the important concerns is to provide service within a time window specified by each customer. The purpose of this paper is to develop a method in solving the vehicle routing problem with time windows (VRPTW) such that either the obtainment of a high quality fast-solution or an attempt to find the optimal solution can be achieved.

The VRPTW can be formally stated as follows. Let $G = (V, A)$ be a graph with node set $V = N \cup \{0\}$ and arc set $A = \{(i, j) | i \in V, j \in V, i \neq j\}$, where $N = \{1, 2, \ldots, n\}$ represents the customer set, and node 0 refers to the central depot. Each node $i \in V$ is associated with a customer demand $q_i (q_0 = 0)$, a service time $s_i (s_0 = 0)$, and a service-time window $[e_i, l_i]$. For every arc $(i, j) \in A$, a non-negative distance d_{ij} and a

* Corresponding author. Tel.: 886 3 5731520; fax: 886 3 5729100; e-mail: fliu@cc.nctu.edu.tw

0377-2217/99/$ – see front matter
PII: S0377-2217(98)00315-4

non-negative travel time t_{ij} are known. Moreover, vehicles housed at the central depot are identical. Without loss of generality, each customer demand is assumed to be less than the vehicle capacity Q. In addition, the demand of each customer cannot be split. The VRPTW is to find an optimal set of routes in such a way that

(i) all routes start and end at the depot;

(ii) each customer in N is visited exactly once within its time window;

(iii) the total of customer demands for each route cannot exceed the vehicle capacity Q.

The service-time windows considered in this paper constitute "hard" constraints. That is, a vehicle cannot visit a customer beyond a specified latest arrival time. However, a vehicle can wait if it arrives too early. Our primary objective is to minimize the number of vehicles used, followed by minimizing the total distance traveled. Throughout this paper, route/vehicle will be used interchangeably.

It is obvious that the VRPTW is NP-hard due to the NP-hardness of VRP. The exact algorithms recently developed for solving the VRPTW can be found in Kolen et al. [17], Desrochers et al. [8], Fisher et al. [13], and Kohl and Madsen [16]. However, on the 56 100-customer benchmark problems by Solomon [30], only a total of 11 problems were solved to optimality. Readers interested in the optimization approach can also refer to the surveys by Desrochers et al. [9,10].

Because of the high computational complexity inherent in solving the VRPTW, solution methods in general resort to the heuristic design. Solomon [30] was among the first to generalize VRP heuristics for solving the VRPTW. Since then, several VRPTW heuristics have been developed. A parallel route-building algorithm was contributed by Potvin and Rousseau [24]. Kontoravdis and Bard [18] proposed a GRASP heuristic using a randomized strategy in selecting a candidate to be inserted into the current partial routes. Differing from the traditional two-phase method, a route construction phase followed by a route improvement phase, Russell [27] embedded a local improvement procedure into the route construction phase. These heuristics essentially focused on designing efficient algorithms in constructing routes. Literature focusing on the investigation of improvement procedures based on the node-exchange and edge-exchange can be found in Savelsbergh [28,29], Thompson and Psaraftis [33], and Potvin and Rousseau [25].

In contrast to the classical route construction/improvement heuristics mentioned above, metaheuristics based on the methods such as *tabu search, simulated annealing* and *genetic search* have shown better performance on average. Generally speaking, these metaheuristics try to explore the solution space more thoroughly to find a good solution. Rochat and Taillard [26] discussed the use of diversification and intensification strategies in local search. Thangiah et al. [32] presented hybrid approaches of using genetic, simulated annealing, and tabu search. With a similar framework of Russell [27], Chiang and Russell [6] embedded a simulated annealing improvement procedure into the route construction phase. A tabu search algorithm with the application of an adaptive memory technique was recently proposed by Taillard et al. [31]. To our knowledge, Chiang and Russell using a reactive tabu search method [7] have produced the best record for Solomon's data sets, in terms of the total number of vehicles required. Other related work on VRPTW can refer to papers such as Garcia et al. [14], Potvin and Bengo [22], Potvin et al. [23], Koskosidis et al. [19], and Bramel and Simchi-levi [4,5].

In this paper, we propose a Route-NEighborhood-based Two-Stage methaheuristic (RNETS). In consideration of the geographic characteristics for VRPs, an appropriate solving approach is to spend time on efficient local operations rather than on inefficient global operations. Clearly, this is good for a company to arrange high quality routing schedule many times per day, in particular for large-sized problems. However, we should allow the user to hold the right a pursue much better solutions if seeing a solution is not so urgent. Our RNETS provides the flexibility by ranging from a very small search domain to the entire domain.

The main contribution of this paper is the development of a new neighborhood structure based on a relationship between routes and nodes. Moreover, two new concepts, nested parallel route construction and end-effect handling, are presented in order to enhance solution quality.

This paper is organized as follows. An overview of RNETS is introduced in Section 2. Section 3 describes the method to construct route neighborhoods. Operational procedures for route construction and improvement are stated in Section 4. Section 5 reports computational results for 60 benchmark problems extracted from the literature. Finally, we conclude this paper in Section 6.

2. An overview of RNETS

Past work on the VRPTW has shown that the parallel route construction method is in general superior to the sequential one. To construct routes in a parallel manner, one may start either by estimating a lower bound such as the GRASP [18], or by using a fast-heuristic to get an upper bound as in Potvin and Rousseau [24]. Our RNETS considers a mixed strategy consisting of the following two stages. In the first stage, we construct routes in a nested parallel manner by repeatedly estimating the lower bound for the unrouted customers; several runs corresponding to different parameter settings are computed, and a solution with the fewest number of routes and the minimal distance is recorded. With the solution obtained in Stages I, II is then used to construct routes in parallel by setting the number of vehicles required to one smaller than the minimal number of routes obtained in Stage I. Several runs are also performed in Stage II.

The primary reason for using Stage II is based on the following observation: the lower bound computed in Stage I is in general under-estimated. Thus, the situation of simultaneously constructing the vehicle routes in Stage II might never occur in Stage I. This simply means that we would still have the chance of using fewer vehicles for service. Furthermore, differing from the way of Potvin and Rousseau [24] using Solomon's sequential insertion heuristic to get a fast upper bound solution, we attempt to obtain a high quality solution in Stage I. In other words, unnecessary routes tend to be nongenerated in Stage II, and a loop of Stage II by deducting one vehicle gradually can possibly be avoided. In the following, three main concepts underlying the RNETS are first introduced.

2.1. Main concepts for RNETS

These concepts are proposed not only to achieve high quality solutions within reasonable computation time but also to provide a scheme for thoroughly exploring the search space. We will defer the detailed description of operating these concepts to Sections 3 and 4.

2.1.1. Route neighborhoods

To efficiently deal with large scale VRPs, an intuitive idea is to decompose the original hard problem into several easier subproblems or to divide the original domain into several smaller subdomains. An optimization-based algorithm by Fisher and Jaikumar [12] decomposed the original VRP problem into two subproblems: a generalized assignment problem (GAP) and a traveling salesman problem (TSP). Computational results reported seem to provide support to their method in solving VRP. However, Savelsbergh [28] has shown that finding a feasible solution to the VRPTW is NP-hard when the number of vehicles is fixed. Thus, the method of Fisher and Jaikumar seems not an efficient way for solving VRPTW with hard constraints due to the difficulty in generating a feasible assignment for TSP. Similarly, the heuristics based on the "cluster-first and route-second" approach may appear with the same drawback.

For VRPs, the term "neighborhood" usually refers to a set of nodes. To be clear, we discuss the neighborhood in the following two cases. Case 1: Before the start of route construction, we may divide the customer set into several smaller subsets or neighborhoods, e.g. the way of Garcia et al. [14]. Usually, these neighborhoods are disjoint partitions. Case 2: Having finished route construction, we can apply local search techniques on the current solution, e.g. the λ-interchange mechanism of Osman [21]; the next solution is in

general selected from a restricted neighborhood of the current solution. In both the cases, the construction of neighborhoods primarily focused on the relationship between node(s) and node(s). The relationship between route(s) and node(s) is almost ignored.

Consequently, from a different point of view, we introduce the concept of *route neighborhood*. It is a reasonable conjecture that a customer holds higher probability of being served by closer routes than by farther routes. Under this observation, we attempt to construct a route neighborhood containing a set of "routes" for each node. The primary purpose of our route neighborhoods is to produce a certain kind of search domain when constructing vehicle routes. However, our route neighborhoods also play a role like the neighborhoods in Case 2. There are at least two benefits resulting from the construction of route neighborhoods. The first is that the weakness in handling time window constraints faced by the methods like cluster-first and route-second can be easily improved; this is indeed why we can generate high quality fast-solutions by focusing on a smaller search domain. The second is that we can easily adjust the number of routes to be contained in a route neighborhood from one to a large enough number such that the solution space can be explored more thoroughly.

2.1.2. Nested parallel route construction

Because of the presence of time window constraints, it seems hard to estimate a lower bound tight enough to the number of vehicles actually required. We often encounter that many customers cannot be feasibly served by using only those vehicles estimated. Therefore, we perhaps need to generate newer routes to accommodate the remaining unrouted customers. Kontoravdis and Bard [18] solved VRPTW from a lower bound direction by using a parallel route construction method once, and then they applied a sequential method by adding one vehicle for the rest of unrouted customers repeatedly until all customers were routed. Seeing that the parallel construction method had better performance in average, however, we introduce the concept of *nested parallel route construction* to obtain better solution quality for VRPTW. The nested parallel route construction repeats the following two steps until all customers are routed or a stopping rule is satisfied. At the first step, we estimate a lower bound to the number of vehicles required for the unrouted customers, and construct a corresponding set of partial routes. For the routes just generated, the second step is to use these routes to service the unrouted customers until no feasible insertion locations can be found. We will describe the stopping rules in the next subsection.

2.1.3. End-effect handling

In the presence of time window constraints, to eliminate a scheduled route entirely might not be an easy thing if we only rely on a refinement procedure in the route improvement phase. When there are only "a few" unrouted customers left after a number of parallel construction runs, we do not immediately create a new route or a new set of routes for them. Since our primary objective is to minimize the number of routes, we design a special procedure to handle those remaining unrouted customers. We will say that there are only a few unrouted customers if the total of unrouted demands is smaller than a specified threshold value. Three measurement criteria, called *end-effect thresholds*, are suggested for the threshold value: vehicle capacity, maximal utilization rate $\times$ vehicle capacity, and average utilization rate $\times$ vehicle capacity. Here, the maximal and average utilization rates are computed with respect to the routes constructed in all previous parallel construction runs. These thresholds are the stopping rules mentioned earlier.

2.2. The RNETS metaheuristic

It is well known that a typical approach in solving VRPs consists of the following two phases: route construction phase and route improvement phase. However, results reported by Russell [27] reveal that embedding a local improvement procedure into the route construction phase seems to produce better so-

lution quality in solving the VRPTW. Thus, in addition to an improvement procedure designed for the route improvement phase, a local improvement procedure is also considered during our route construction phase. Russell [27] performed local improvement once after every $0.1n$ customers have been routed. The criteria in determining when to perform the local improvement procedure are called *local improvement rules* in our paper.

As stated at the beginning of this section, there are two stages considered in RNETS. Fig. 1 displays the core framework for each stage of our RNETS. The *initialization block* refers to the construction of an initial set of route neighborhoods (see detail in Section 3). The *parallel route construction block* is responsible for selecting a candidate and inserting it into a specific route. The *parallel route reinitialization block* is to eliminate a number of constructed routes and reconstruct an appropriate set of partial routes. What we called the nested parallel route construction is accomplished through the *parallel route construction block* and the *parallel route reinitialization block* (see detail in Section 4.1). The purpose of *end-effect handling block* is to apply a specially designed procedure on a few unrouted customers who are left with no feasible

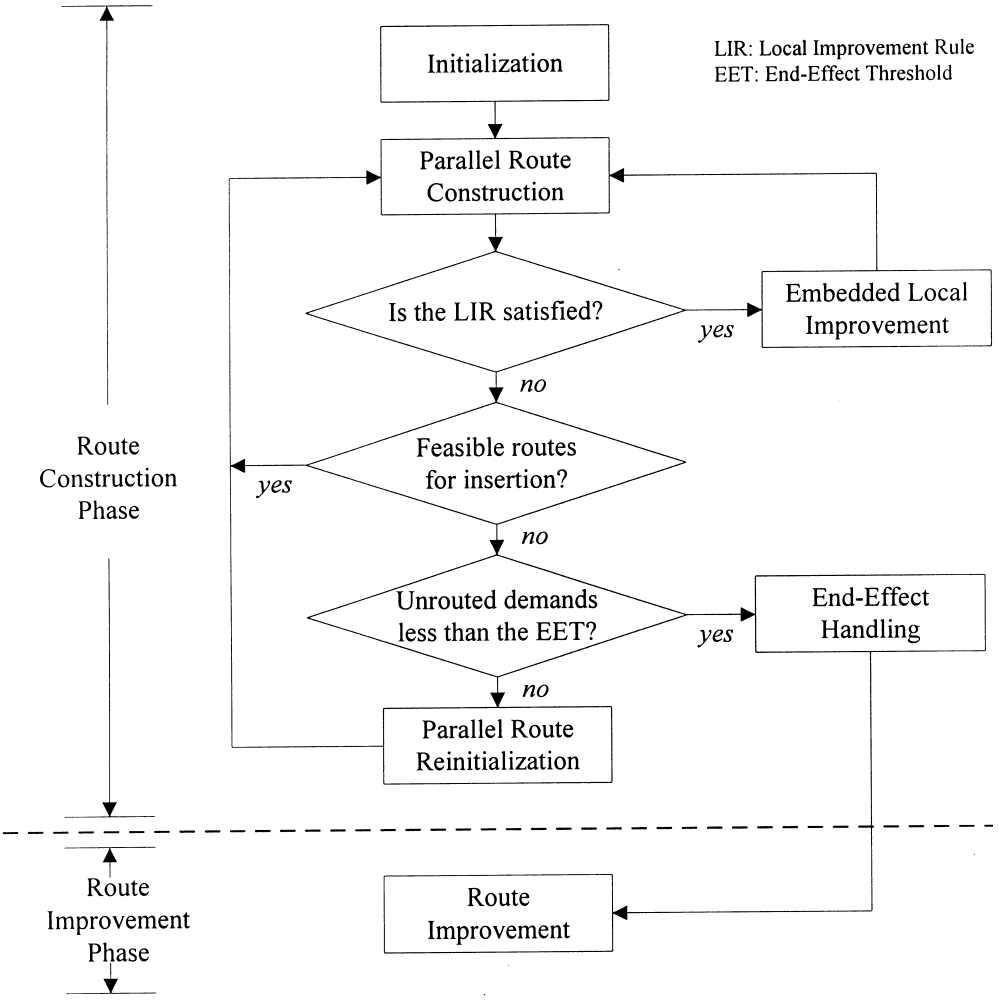

Fig. 1. The core framework for each stage of RNETS.

vehicles with respect to the current set of routes during the route construction phase (see detail in Section 4.2).

For clarity, we summarize the RNETS metaheuristic given below before going into the details.

2.2.1. Notation

ξ: the radius of route neighborhoods;
w: the weight factor for the distance of inserting a customer into a route;
wdelta: the decrement of *w*;
pcont: the perturbation factor for the distance of shifting a customer within a route;
pdelta: the decrement of *pcont*;
maxshift: the maximal number of shifts in a route at a given perturbation constant;
exchiter: the maximal number of iterations for the procedure of swapping routed and unrouted customers to be performed;
exchcont: the constant for restricting the exchange of routed customer *i* and unrouted customer *j* again except that either *i* or *j* has been swapped with other customers *exchcont* times.

2.2.2. The RNETS metaheuristic

Stage I

Step 1. Specify the parameters and rules: (ξ, *w*, *wdelta*, *pcont*, *pdelta*, *maxshift*, *exchiter*, *exchcont*, *end-effect threshold*, *local improvement rule*).

Step 2. (Initialization)*:* Construct route neighborhoods as stated in Section 3.

Step 3. (Parallel construction)*:* Apply parallel route construction method as stated in Section 4.1 until no feasible routes can be found. During the route construction phase, if the specified local improvement rule is satisfied, perform the embedded local improvement procedure as stated in Section 4.3 (Local Improvement).

Step 4. If all customers have been routed, go to Step 7. Otherwise, if the remaining unrouted demands are greater than the end-effect threshold, then go to Step 5, else go to Step 6.

Step 5. (Parallel reinitialization)*:* Eliminate routes, estimate a lower bound and construct an associated set of partial routes as stated in Section 4.1. Go to Step 3.

Step 6. (End-effect handling)*:* Apply the end-effect handling procedure stated in Section 4.2 on each unrouted customer.

Step 7. (Route improvement phase)*:* Apply the perturbation improvement procedure stated in Section 4.3 on the current complete solution. Update the best solution record using the following criteria: (i) number of routes, (ii) total distance, and (iii) total waiting time. If all settings have been executed, go to Step 8, otherwise, go to Step 2.

Stage II

Step 8. Set the number of vehicles required to one smaller than the minimal number of routes obtained in Stage I if route number > 1.

Step 9. Same as Step 1 to Step 7. (Remark: the estimation of a lower bound is not required)

In the next two sections, we describe how to construct route neighborhoods and how to design procedures in constructing vehicle routes and in improving an existing solution based on our neighborhood structure.

3. The construction of route neighborhoods

The key step in constructing route neighborhoods is to assign a *reference route identifier* (RRI) to each node. We now describe a method in generating RRIs and constructing route neighborhoods. Other

methods and more details can be found in Ref. [20]. An example of 25 customers as shown in Fig. 2(a) will be used for illustrating each of the following steps.

Step 1: Generate a giant route (a Hamiltonian cycle) with respect to all customers in N. In generating a giant route, we consider the distance factor (d_{ij}) only. We attempt to find a giant route such that the neighboring nodes on the original Euclidean plane are not dispersed too far on the giant route to be generated. Methods such as the sweep method [15], the spacefilling curve method [2] and the nearest neighbor method can be used to generate the giant route. For the purpose of being used later, let L_N denote the cyclic customer list of a generated giant route. In our example, a giant route is the dashed line shown in Fig. 2(b). Thus, L_N is (1–2–3–$\cdots$–25–1).

Step 2: Estimate a lower bound to the number of vehicles required and select a corresponding set of seed nodes. Generally speaking, these two operations are necessary for all the insertion-type algorithms that construct routes in parallel. Methods in our current implementation for solving the VRPTW are described below.

(a) Lower bound estimation (LBE). First, we regard it as a *bin packing problem*. Therefore, the lower bound, denoted as α, can be estimated by using $\lceil \sum_{i=1}^{n} q_i/Q \rceil$. Second, time window constraints are used in determining customers who cannot be served by the same vehicle. Initially, customers are sorted in an ascending order according to l_i and then e_i. The first element on the sorting list is added into seed set S. Then, searching along the list, we add customer i into S if time window feasibility is violated between i and each j in S. In case of $|S| < \alpha$, we discard all the elements in S, and regenerate α seeds by using the following SEEDGEN procedure. Otherwise, the elements in S are just the seed nodes selected.

(b) *Seed node generation* (SEEDGEN). A basic principle in selecting seed nodes is that each node to be selected is likely to be serviced by a distinct vehicle in an optimal solution. Unfortunately, there is no easy way to find such a good seed set in advance. We roughly estimate the number of customers per route by using $\lfloor n/\alpha \rfloor$. Initially, we select the farthest customer from the depot as the first seed and the farthest customer from the first seed as the second seed. For all the pairs of two adjacent seeds along the L_N, the one that has the largest number of customers within it is chosen for a new seed to be generated from. We further restrict possible candidates for the next seed to the customers excluding the $\lfloor n/\alpha \rfloor/2$ nearest successors and the $\lfloor n/\alpha \rfloor/2$ nearest predecessors with respect to the chosen seed pair. Then a customer with the maximal sum of distances to the chosen seed pair is selected as the next seed node. We operate in this way repeatedly until α seeds are generated.

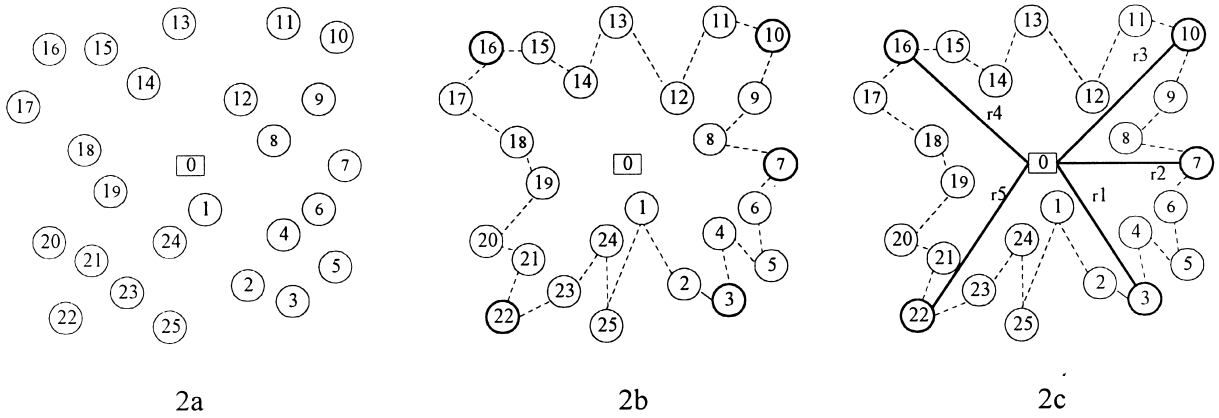

Fig. 2. Illustration of a 25-customer example.

In the example, we simply assume that five vehicles are required and the corresponding five seeds selected are the nodes numbered 3, 7, 10, 16, and 22 as shown in Fig. 2(b).

Step 3: Create partial routes by connecting each seed node to the central depot and label each partial route with a *route identifier*. For the partial routes, we construct a cyclic route list L_R in accordance to the sequence of seed nodes on L_N. In our example, the constructed partial routes are the solid lines labeled r_1, r_2, r_3, r_4, and r_5 in Fig. 2(c). It can be seen that L_R is (r_1–r_2–r_3–r_4–r_5–r_1).

Step 4: Assign a RRI for each node in N. Choosing an arbitrary element from L_R, say r_i, we assigned r_i as the RRI for the seed node of the route labeled r_i. Along the L_N in a forward direction, the succeeding nodes of the seed node are then assigned r_i as their RRIs until the succeeding node encountered is also a seed node. This operation is repeated until each node in N has been assigned an RRI. In the example, the RRI for each node in $\{3,4,5,6\}$ is r_1. Similarly, the RRIs for the node subsets, $\{7,8,9\}$, $\{10,11,12,13,14,15\}$, $\{16,17,18,19,20,21\}$, and $\{22,23,24,25,1,2\}$, are r_2, r_3, r_4, and r_5, respectively.

Step 5: Construct a route neighborhood for each node. Let $R(i,\xi)$ denote the set of *routes* contained in a route neighborhood for node i given ξ. With a specific value of ξ, a route neighborhood for node i can now be defined as follows. The $R(i,\xi)$ consists of the following routes on L_R: the ξ nearest succeeding routes of the RRI for node i, the ξ nearest preceding routes of the RRI for node i, and the RRI itself. Strictly speaking, a route neighborhood contains a set of RRIs. For convenience, we treat the members of a route neighborhood as *routes* when no misleading occurs. It is clear that the number of routes contained in $R(i,\xi)$ equals $2\xi + 1$. In Fig. 2(c), for instance, the RRI for node 13 is r_3, so $R(13,1) = \{r_2, r_3, r_4\}$. Similarly, $R(24,2) = \{r_3, r_4, r_5, r_1, r_2\}$.

Thus far, we have completed the construction of route neighborhoods. Special features on route neighborhoods are exploited through the same example. Since we assume five vehicles required in our example, five classes of RRIs, r_1, r_2, r_3, r_4, and r_5, were assigned to customers. By specifying a value of ξ, each class of RRIs is associated with a class of route neighborhoods. It can be found that we solve the problem in a global domain sense if $2\xi + 1 \geqslant 5$. That is, if $\xi \geqslant 2$, all the constructed neighborhoods contain the same members. In addition, the case of disjoint partitions occurs if $\xi = 0$. Therefore, the following analysis will focus on the case of $\xi = 1$.

When $\xi = 1$, five classes of route neighborhoods, $\{r_5, r_1, r_2\}$, $\{r_1, r_2, r_3\}$, $\{r_2, r_3, r_4\}$, $\{r_3, r_4, r_5\}$, and $\{r_4, r_5, r_1\}$, are constructed for the five classes of RRIs, r_1, r_2, r_3, r_4, and r_5, respectively. It can be seen that the intersection of any two route neighborhood classes is not empty. In other words, these route neighborhood classes are not disjoint partitions. Hence, this result implies that nodes associated with different route neighborhoods are not disjointedly partitioned. As for the nodes belonging to the same route neighborhood class, any route in the corresponding route neighborhood can provide service to them. In our example, for instance, both $R(13,1)$ and $R(14,1)$ are $\{r_2, r_3, r_4\}$; any one of the three routes are potential vehicles for servicing customers 13 and 14. Thus, though these nodes seem to be grouped in a same cluster, the effect of time windows has been diluted in comparison with the traditional approach like cluster-first and route-second.

All our operations designed for route construction and improvement are restricted within the constructed route neighborhoods. For example, the finding of a feasible insertion location for an unrouted customer, say i, is systemically searched as follows. Initially, the route corresponding to the RRI for customer i is considered. Then, following the order as defined in L_R, the routes in $R(i,\xi)$ are examined right and left in turns with respect to the RRI for customer i. Certainly, the value of ξ used in constructing routes can be different from that used in improving routes. In general, the computation time tends to increase as ξ becomes larger. However, there is no guarantee that the solution quality and the value of ξ are always consistently increased. Depending on the computation time allowed, the user may have several choices in setting ξ. In other words, the route neighborhood approach is quite flexible and makes sense, and is suitable for solving VRPs with more complex constraints.

4. Procedures for route construction and improvement

This section describes how to select and insert unrouted customers into the existing partial routes under our neighborhood structure and how to apply the concepts of nested parallel route construction and end-effect handling in constructing vehicle routes. Finally, procedures for improving solution quality are stated.

4.1. Nested parallel route construction

For an unrouted customer, say k, let (i, j) be a link on an arbitrary route in $R(k, \xi)$. The cost for inserting customer k into link (i, j) is measured in a way like Solomon [30] as follows:

- Spatial cost $d_{ikj} = d_{ik} + d_{kj} - d_{ij}$.
- Temporal cost $t_{ikj} = t_j' - t_j$, where t_j and t_j' are the arrival time of j before and after k is inserted.
- Insertion cost $c_{ikj} = w \times d_{ikj} + (1 - w) \times t_{ikj}$, where $0 \leqslant w \leqslant 1$.

To be less myopic in choosing a customer for insertion, we compute the so-called "regret cost" for each unrouted customer. Let $c_k(\rho)$ denote the minimal insertion cost for inserting k into the best link on route ρ in $R(k, \xi)$. If no feasible insertion locations on route ρ can be found for customer k, a large penalty cost is set to $c_k(\rho)$. In our method, the regret cost for customer k is defined as $\sum_{\rho \in R(k,\xi)}\{c_k(\rho) - c_k(\rho_k)\}$, where $\rho_k = \arg\min\{c_k(\rho); \text{for } \rho \text{ in } R(k, \xi)\}$. A customer to be selected for the next insertion is the one with the largest regret cost. This operation is essentially the basic function to be done for the parallel route construction block in Fig. 1. Operations stated below compose the parallel route reinintialization block.

Let Ω_1 denote the set of routes that have been constructed so far. For the remaining unrouted customers after each parallel construction run, a loose lower bound can be computed as $\lceil$the total of unrouted demands / $Q\rceil$. To enhance the performance of nested parallel construction, however, we attempt to find a tight lower bound for the unrouted demands. Information learnt from previous parallel construction runs might be a valuable guideline for the next run. In other words, we try to extract useful information from Ω_1 to compute the lower bound. Let $U_{\max}$ refer to the maximal loading utilization rate for routes in Ω_1, where $0 < U_{\max} \leqslant 1$. A tight lower bound can then be computed as the $\lceil$total of unrouted demands / $(U_{\max} \times Q)\rceil$.

Moreover, during the route construction phase, we also attempt to eliminate a certain number of routes with lower utilization rates. These deleted customers as well as the originally unrouted customers left after the last parallel construction run will be considered simultaneously in the next parallel construction run. Let Ω_2 denote the set of routes constructed during the last parallel construction run, and let U_{ave} refer to the average loading utilization rate for routes in Ω_2. For set Ω_2, routes with utilization rates lower than U_{ave} are eliminated in our current implementation. Clearly, this step results in an increase of computation time, but it can also increase the possibility of using fewer routes or of reducing the distance traveled.

As mentioned above, we estimate the lower bound by using $\lceil$the total of unrouted demands / $(U_{\max} \times Q)\rceil$. This calculation implies that the vehicle utilization rate for the routes providing service to the remaining unrouted customers is assumed to be less than $U_{\max}$. Though this assumption makes sense, we develop a procedure for preventing over-estimation of the lower bound. The procedure attempts to improve $U_{\max}$ before we compute the lower bound. In case of the existence of unrouted customers after each parallel construction run, we may sacrifice some distance cost so that feasible insertion locations for the unrouted customers can possibly be found in Ω_1. Two simple operations that we used are given as follows.

One-move insertion (MV1)*:* For an unrouted customer i, we try to move a customer j on a route ρ in $R(i,\xi)$ to a route ρ' in $R(j,\xi)$. If i and j can be feasibly inserted into routes ρ and ρ', respectively, we carry out the insertions at once without considering the insertion cost.

Two-move insertion (MV2)*:* For an unrouted customer, say i, we try to move a customer j on a route ρ in $R(i, \xi)$ to a route ρ' in $R(j, \xi)$ and move a customer k on the route ρ' to a route ρ'' in $R(k, \xi)$. If i, j, and k can

be feasibly inserted into routes ρ, ρ', and ρ'', respectively, we carry out the insertions regardless of the insertion cost.

Having estimated the number of vehicles required for the unrouted customers, we then select a corresponding set of seed nodes as the way described in Section 3 but ignoring customers who have been routed. Now the next parallel construction run can be triggered after constructing the respected set of partial routes. Note that we need to update the cyclic route list L_R appropriately and reassign an RRI for each remaining unrouted customer.

4.2. End-effect handling

Before entering into the parallel route reinintialization block, we examine whether the total of unrouted demands is less than the specified end-effect threshold. If it is true, we try to reconsider inserting these unrouted customers into the routes in Ω_1 instead of constructing a route (a set of routes) immediately. However, if there are still unrouted customers left after applying the procedure specially designed for those few unrouted customers, a sequential construction procedure is then performed. We repeat the above two procedures until all customers are routed. In the following, several related processing modules are first described.

Intra-route perturbation (IRP)*:* For each customer on an existing route, we attempt to shift it into another feasible location on the route. With this perturbation, feasible insertion locations for the unrouted customers may possibly be found. Considering an arbitrary route like $(0,\ldots,\ i,\ j,\ k,\ldots,\ m,\ n,\ldots,\ 0)$, we reinsert customer j into link (m,n) if $(d_{ik}+d_{mj}+d_{jn}) \leqslant (1+pcont)\times(d_{ij}+d_{jk}+d_{mn})$ and time window feasibility are satisfied, where *pcont* is a non-negative perturbation parameter. The parameter *pcont* will be gradually reduced by a quantity of *pdelta* per iteration until it becomes a negative number. For each route, the number of shifts per iteration is restricted by a specified parameter *maxshift*. Moreover, we do not allow customer j to be reinserted back into link $(i,\ k)$ at the current iteration.

Inter-route improvement (IRIM)*:* To improve solution quality for the current set of routes, we try to shift customers on a particular route to another route and exchange customers between two different routes. For convenience, notation similar to the λ-interchange mechanism of Osman [21] is used below. Three operators $(1,0)$, $(1,1)$ and $(1,2)$ were considered in our current implementation. The λ-interchange mechanism selects all possible combinations of pairs of routes for exchange. In contrast, the possible pairs of routes that we considered are all restricted to the routes within a route neighborhood. For customer i on route r, other routes in $R(i,\xi)$ are searched right and left in turns with respect to r in accordance to the cyclic route list L_R defined in Section 3; we make the shift or exchange when a feasible improvement is first encountered. Accordingly, operator $(1,0)$ represents the shift of customer i from a route under consideration to another route in $R(i,\xi)$. Operator $(1,1)$ refers to the exchange of customer i with a customer on another route in $R(i,\xi)$. For the operator $(1,2)$, we now only consider the exchange of customer i with the two consecutive customers on another route in $R(i,\xi)$. We remark that if a customer is reassigned to another route, its corresponding RRI is updated accordingly.

The exchange of routed and unrouted customers (ERUC)*:* The purpose of this procedure is to generate a different set of unrouted customers having the same number of customers as the original one. For an arbitrary unrouted customer, say i, we examine whether a customer j on a route in $R(i,\ \xi)$ can be feasibly substituted by customer i. If feasibility is maintained, j becomes an unrouted customer and i is inserted into the original location of j. To prevent the swap of i and j too frequently, the exchange of i and j will not be reconsidered except that either i or j has been made the exchange of *exchcont* times with other customers.

With the above descriptions in mind, we now summarize the complete end-effect handling procedure as follows. Recall that *exchiter* refers to the number of iterations for the procedure ERUC to be performed.

```
PROCEDURE End-Effect Handling
While (there are still unrouted customers) Do:
  exchtemp ← exchiter.
  While (exchtemp >0) Do:
    ptemp ← pcont.
    While (ptemp ⩾ 0) Do:
      Perform procedures MV1 and MV2 for each unrouted customer.
      Apply procedure IRP on each existing route.
      Apply procedure IRIM on the current set of routes.
      Unrouted customers are selected and inserted into the existing routes by using
      criteria described in Section 4.1.
      ptemp ← ptemp − pdelta.
    End Do.
    Perform procedure ERUC for each unrouted customer.
    exchtemp ← exchtemp - 1.
  End Do.
  Construct a new route by applying Solomon insertion heuristic I1.
End Do.
```

4.3. The route improvement procedures

Two improvement procedures are designed in searching for a better solution. The basic distinction between these two procedures is that the procedure embedded in the route construction phase purely improves a current partial solution, while the procedure designed in the route improvement phase allows a worse complete solution to be generated.

The embedded local improvement procedure: This procedure considers the searching of possible improvement among routes and within each route. The inter-route improvement procedure IRIM stated earlier is also used here. As to the intra-route improvement, we reinserted each customer into every other link on the same route when a feasible improvement is firstly found.

The perturbation improvement procedure: The purpose of this procedure is in a sense similar to the tabu search method to escape from a local minimum. A loop procedure consisting of two steps is applied on the current complete solution obtained in the route construction phase. At the first step, an improvement procedure same as the embedded one is performed. At the second step, the intra-route perturbation procedure IRP described previously is applied on each route of the solution obtained in the first step; Step 2 attempts to move the solution improved in Step 1 to a worse solution. We repeat the above two steps under a specified setting of *pcont* and *pdelta* until *pcont* that gradually decreases *pdelta* per iteration becomes negative. Once a better solution is found after finishing each first step, the best solution record is updated.

5. Computational studies

To assess the performance of RNETS, we first solved four real world problems obtained from George Kontoravids. Among them, two 249-customer problems, D249 and E249, were generated from a 249-customer data set reported in Baker and Schaffer [1] and two 417-customer problems, D417 and E417, were reported in Russell [27].

In addition, RNETS was tested on 56 benchmark problems by Solomon [30]; each is a 100-customer Euclidean problem. Customer locations were either randomly generated (problem sets R1 and R2), clus-

tered (problem sets C1 and C2) or mixed with randomly generated and clustered customers (problem sets RC1 and RC2). Each set, R1, R2, C1, C2, RC1, and RC2, have 12, 11, 9, 8, 8, and 8 problems, respectively. Problem sets R1, C1, and RC1 have a shorter scheduling horizon while the other sets have a longer horizon.

The RNETS was programmed in Fortran 77 and run on a HP-9000/720 workstation (17 Mflops, 55 Mips), which is approximately 10 or 11 times slower than a Pentium II 233 personal computer according to our experience. The current operating system is HP-UX release 9.01, and the compiler is HP-UX Fortran 77. All computations were real arithmetic without any truncations. Three versions of RNETS were tested; the only difference among them is the method in generating a giant route. The abbreviations SW, NN, and SPF denote the methods for sweep algorithm, nearest neighbor, and spacefilling curve, respectively. The local improvement rules that we used were in a form of min $[\mu_1 \times v, \mu_2 \times (n - v)]$, where$\mu_1$ and μ_2 are given parameters, and v is the number of routes in set Ω_1. For instance, min $[4 \times v, 0.4 \times (n - v)]$ represents that we carry out local improvement procedure once after every 4 customers have been inserted into each route on an average or after $0.4 \times (n - v)$ customers were routed.

We now summarize the parameter settings used in solving the four large-sized problems as follows.

$$w \leftarrow 1.0; wdelta \leftarrow 0.2; pcont \leftarrow 0.5; pdelta \leftarrow 0.1; maxshift \leftarrow 5; exchiter \leftarrow 10;$$
$$exchcont \leftarrow 5; (\mu_1, \mu_2) \leftarrow (4, 0.4); end\text{-}effectthreshold \leftarrow U_{\max} \times Q.$$

In solving Solomon's data sets, more runs were given; *wdelta* was set to 0.1 and three local improvement rules (2, 0.1), (3, 0.25) and (4, 0.4) were considered. In the Appendix, a few best known results are reported.

5.1. Results for the four large-sized problems

Table 1 shows the results for problems D249 and E249 with respect to different values of ξ. Similarly, Table 2 is the results for problems D417 and E417. The three numbers in each entry refer to the number of vehicles required, total distance, and CPU time in minutes, respectively. Since problem D249 required only 3 vehicles, there were no results reported for $\xi > 1$; that is, we have solved problem D249 in a global domain sense. For problems D417 and E417, the value of ξ can be ranged from 1 to 27 according to our definition on route neighborhoods; the optimal number of vehicles required is 54 for both problems. Generally speaking, computation times tend to increase as the values of ξ gets larger; the inconsistency in computation time might be primarily due to the improvement process on a partial or a complete solution. It can also be seen that the solution quality was not always absolutely increased. For problems D417 and E417, solution results corresponding to the smaller settings such as $\xi = 1$ or $\xi = 2$ appeared to be inferior to those of other larger settings. Table 2 also shows that computation times required for the values of ξ such as 25, 26, and 27 were substantially higher than the smaller settings of $\xi = 1, \ldots, 6$. However, the solution quality for $\xi = 4$, 5, or 6 seems competitive with the results for $\xi = 25$, 26, or 27.

Table 1
Results for problems D249 and E249

ξ	D249			E249		
	SW	NN	SPF	SW	NN	SPF
1	3	3	3	5	5	5
	395.60	390.57	394.68	424.62	418.92	440.50
	7.79	7.21	7.84	7.43	6.92	6.92
2	–	–	–	5	5	5
				417.07	415.56	407.03
				8.98	9.45	9.67

Table 2
Results for problems D417 and E417

ξ	D417			E417		
	SW	NN	SPF	SW	NN	SPF
1	55	55	55	55	55	55
	3631.17	3657.87	3628.30	4015.41	4021.02	4047.53
	10.68	9.84	7.00	7.47	6.78	13.27
2	55	55	55	55	55	55
	3570.21	3602.30	3592.42	3957.42	3988.43	4068.09
	23.90	20.15	42.28	24.90	14.94	11.85
3	55	55	55	55	55	55
	3536.96	3582.94	3581.08	3962.71	3980.82	4000.84
	42.86	60.92	31.57	40.80	34.02	33.58
4	54	55	54	54	55	55
	3747.52	3551.30	4075.42	4691.14	4040.57	4017.93
	47.38	46.42	53.27	41.25	41.42	55.06
5	54	54	55	55	55	55
	4216.44	3836.52	3601.32	3957.70	4005.03	4038.65
	66.69	64.35	72.39	50.43	37.05	44.18
6	55	54	54	55	55	55
	3576.36	3927.13	3857.71	3979.76	3978.75	3979.27
	83.62	73.63	55.83	42.53	36.92	43.60
25	54	54	54	55	55	54
	3952.48	4001.62	4104.57	3967.23	3945.11	5038.87
	181.83	194.17	155.12	101.82	87.40	86.31
26	54	54	54	55	55	55
	4146.77	4050.64	4389.35	3884.86	4042.23	3921.45
	201.18	190.07	198.95	98.13	95.87	94.96
27	54	54	54	55	54	55
	4019.49	3978.22	4126.29	3979.14	4685.07	3939.69
	158.96	208.21	184.85	96.73	94.22	98.06

Consequently, a question faced is whether a little improvement in solution quality is worth the extra computational effort. Surely, there is no exact answer and the appropriate solution will depend on your purpose in dealing with the problem. However, the computational results in Table 2 reveal some guidelines for running RNETS as summarized below.

- To solve a problem in a rather limited time, we may set ξ to 1 or 2. In this situation, we can produce a fast solution by sacrificing a certain degree of solution quality. In fact, we will see later the competitiveness of the corresponding results when comparing with other heuristics also taking less computation time.
- In pursuing better solution quality, we suggest that the user should solve the problem several times by using the smaller settings such as $\xi = 3$, 4, and 5, and then choose the best solution among them.
- If finding the best solution is our ultimate goal, we can run RNETS by setting ξ from 1 to a large enough number that ensures the problem to be solved in a global domain sense and the best result among them will then be chosen as our final solution. For instance, the values of ξ can be set as $1, 2, 3, 4, \ldots,$ and 27 for problems D417 and E417. To determine the maximal value for ξ, we may run RNETS at $\xi = 1$ first and then set a reasonable upper limit for ξ according to the solution obtained at $\xi = 1$.

Table 3 compares our results with other heuristics including Kontoravdis and Bard [18] on a SUN SPARC-10 workstation, Chaing and Russell [6] on a 486DX2/66 PC, Thangiah et al. [32] on a NeXT 68080 computer, Rochat and Taillard [26] on a Silicon Graphics workstation, Taillard et al. [31] on a SUN SPARC-10 workstation, and Chiang and Russell [7] on a Pentium-90 PC. The numbers under column RNETS represent the best results in Table 1 and the best results for the row of $\xi = 4$ in Table 2. It can be

Table 3
Comparison of heuristics on the four large-sized problems

Problem	KB *	CR-sa *	TH *	RT *	TA *	CR-tabu *	RNETS
D249	3	4	4	–	–	–	3
	636.3	492.44	457				390.57
	0.45	8.07	25.5				7.21
E249	5	5	5	–	–	–	5
	914.9	546.34	495				407.03
	1.45	6.65	56.25				9.67
D417	55	55	54	54	55	55	54
	4273.4	4232.39	4866	6264.8	3439.8	3455.28	3747.52
	4.57	31.98	32.48	–	–	63.0	47.38
E417	55	55	55	54	55	55	54
	4985.7	4589.61	4149	7211.8	3707.1	3796.61	4691.14
	17.02	22.25	49.47	–	–	60.0	41.25

* KB = Kontoravdis and Bard (1995); CR-sa = Chiang and Russell (1996); TH = Thangiah et al. (1995); RT = Rochat and Taillard (1995); TA = Taillard et al. (1997); CR-tabu = Chiang and Russell (1997).

found that RNETS on all four problems yielded the best solutions in terms of the number of vehicles required. In particular, when comparing with the solutions that require the fewest vehicles, significant improvements on the distance traveled were also achieved; for instance, RNETS produced a distance reduction of 23% and 35% over the previously best known results for D417 and E417, respectively.

Due to the use of different computing machines, programming languages, or computer memory, it is hard to evaluate different heuristics with respect to the computation times. It is known that there is a trade-off between solution quality and computation time. Thus, comparing solution quality among different heuristics seems to be unfair for those heuristics taking less computation time, for instance, the heuristic of Kontoravdis and Bard [18]. However, note that even for the case of $\xi = 1$ our results seem still competitive with other heuristics that yielded solutions in a low computation time. We should point out that the numbers under columns of RT and TA in Table 3 would be time-intensive results by observing their reported data for solving Solomon's problem sets.

5.2. Results for Solomon's problem sets

Table 4 reports the results for our three versions of RNETS using $\xi = 1$ and $\xi = (1, \ldots, 5)$. In fact, only $\xi = 1$ or 2 were used in solving problem sets R2, C2, and RC2 because the number of vehicles required for each problem in these sets is smaller than five. The numbers in each entry are the average results for all problems in the set. The far right column represents the best of the three versions given $\xi = (1, \ldots, 5)$. It can be seen from Table 4 that fast solutions were solved for the case of $\xi = 1$; indeed, we will see later the corresponding solution quality has been competitive in comparison with those published heuristics that also took low computation time. As we expected, much better solution results were obtained for the case of $\xi = (1, \ldots, 5)$ by paying higher computational effort. Among the three different implementations, the SPF version required the fewest number of vehicles in total. However, the NN version seems to be more efficient in solving problems with clustered customers.

Table 5 compares our results with other heuristics. Again, it is hard to justify which heuristic to be the most efficient and effective due to the large variations in computation time used. In fact, no one can dominate all other heuristics. In terms of the total number of vehicles required, RNETS yielded a total of 412 routes, which is one higher than that of the reactive tabu search heuristic by Chiang and Russell [7]. However, RNETS on problem sets C2 and RC2 yielded slightly better solution quality by taking much less

Table 4
Results for Solomon's data sets

Data sets	$\xi = 1$			$\xi = (1, 2, 3, 4, 5)$			
	SW	NN	SPF	SW	NN	SPF	BEST
R1	12.58	12.67	12.5	12.33	12.33	12.25	12.17
	1235.46	1230.02	1258.34	1243.89	1244.13	1253.68	1249.57
	8.16	8.23	6.33	45.92	44.76	41.58	44.28
R2	2.91	2.91	2.82	2.91	2.91	2.82	2.82
	1033.32	1021.67	1022.91	1033.32	1021.67	1022.08	1016.58
	5.78	5.66	5.56	7.05	6.73	6.71	6.64
C1	10	10	10	10	10	10	10
	837.54	833.73	859.07	831.93	830.06	841.33	830.06
	3.78	2.88	4.11	24.39	22.11	24.32	22.01
C2	3	3	3	3	3	3	3
	591.03	591.03	591.03	591.03	591.03	591.03	591.03
	3.64	3.75	3.73	3.66	3.75	3.73	3.59
RC1	12.13	12.25	12.13	12	12.13	12	11.88
	1436.02	1424.62	1430.47	1401.10	1402.17	1416.11	1412.87
	4.99	4.40	5.01	31.06	29.76	31.14	30.46
RC2	3.38	3.38	3.25	3.38	3.38	3.25	3.25
	1205.59	1205.37	1230.31	1196.24	1188.80	1230.31	1204.87
	5.51	5.49	5.33	7.84	7.67	7.17	7.12
Total number of vehicles	421	423	418	417	418	414	412

Table 5
Comparison of heuristics on Solomon's data sets

Data sets	PB *	CR-sa *	TH *	RT *	TA *	CR-tabu *	RNETS
R1	12.6	12.5	12.33	12.58	12.25	12.17	12.17
	1296.8	1308.82	1249.3	1197.42	1216.70	1204.19	1249.57
	11.32	2.86	25.25	45.0	229.56	149.85	44.28
R2	3	2.91	3.09	3.09	3	2.73	2.82
	1117.7	1166.42	1007.3	954.36	995.38	986.32	1016.58
	39.44	3.93	23.15	163.33	337.2	169.88	6.64
C1	10	10	10	10	10	10	10
	838.0	909.8	830.8	828.45	828.45	828.38	830.06
	10.02	2.19	4.25	53.33	243.83	17.9	22.01
C2	3	3	3	3	3	3	3
	589.9	684.1	654.2	590.32	590.30	591.42	591.03
	41.37	2.46	5.25	60.0	272.92	40.02	3.59
RC1	12.1	12.38	12	12.38	11.88	11.88	11.88
	1446.2	1473.9	1438.0	1369.48	1367.51	1397.44	1412.87
	11.22	2.57	16.18	43.33	187.73	78.96	30.46
RC2	3.4	3.38	3.37	3.62	3.38	3.25	3.25
	1360.6	1401.5	1243.7	1139.79	1165.62	1229.54	1204.87
	35.57	3.58	20.20	130.0	193.27	107.39	7.12
Total number of vehicles	422	422	419	427	416	411	412

* PB: Potvin and Bengo (1996); CR-sa = Chiang and Russell (1996); TH = Thangiah et al. (1995); RT = Rochat and Taillard (1995); TA = Taillard et al. (1997); CR-tabu = Chiang and Russell (1997).

computation time when compared with the reactive tabu search heuristic. In addition, we need to emphasize that the computation times displayed in columns TA, CR-tabu, and RNETS were all associated with the best results of several runs or of a number of different implementations. The results from Taillard et al. [31] were the best of five independent runs, and the results from Chiang and Russell [7] were the best of three different heuristics; thus, the computation times actually spent should be about five and three times more, respectively. Surely, the computation times in column RNETS should be about three times more. Under this observation, we can find that the computation times spent in yielding the solutions in columns TA and CR-tabu appeared to be much higher than the effort spent in RNETS.

6. Conclusion

In this paper, we have proposed RNETS for solving VRPTW. In view of the geographic characteristics for VRPs and the conflicts among time window constraints, we introduced a new neighborhood idea, "route neighborhood", to explore solution space more thoroughly and to let users solve their problem in a more controllable way. We also presented concepts such as "nested parallel route construction" and "end-effect handling" to obtain higher solution quality. In addition, useful information extracted from the intermediate solving process is used to compute a tighter lower bound for the next parallel construction run.

Computational testing on 60 benchmark problems extracted from the literature has shown that the overall solution quality of RNETS is highly competitive with all existing heuristics. Moreover, results corresponding to the smaller settings of route neighborhoods indicate that RNETS also behaves well in generating fast solutions. By observing the results for the two 417-customer problems, we think that our approach has potential in solving large-scale problems. A fruitful research direction is to apply the route-neighborhood-based approach to other VRPs with more complex constraints. Since different implementations based on our suggested framework are possible, further testing and development are expected. An interesting issue is to see whether different methods of constructing and operating route neighborhoods have significant impact on solution quality.

Acknowledgements

The authors wish to thank Sam R. Thangiah for providing many reference papers on the VRPTW area, Wen-Chyuan Chiang and Robert A. Russell for providing the paper of the reactive tabu search meta-heuristic, George Kontoravdis for providing the 249 and 417 customer problem sets.

Appendix A. A few best known solutions

Problem RC202: (Number of vehicles = 3; Total distance = 1445.857)

Route #	Tours
1	92–95–85–63–33–31–28–26–27–29–34–62–71–61–44–40–38–41–81–90–8–6–46–5–3–1–43–35–32–89–93–94–80
2	72–37–36–39–42–45–7–47–11–15–16–88–73–79–78–53–99–86–87–9–59–75–97–10–17–13–74–24–25–77– 58–52
3	65–82–12–14–98–69–23–21–19–76–30–50–67–84–51–18–48–49–22–57–20–83–66–56–91–

Problem D249: (Number of vehicles = 3; Total distance = 388.029)

Route #	Tours
1	199–197–196–181–186–190–130–136–140–155–128–126–125–124–112–120–123–121–110–109–108–111– 107–117–115–118–119–116–129–98–97–103–104–102–101–100–99–96–95–24–23–22–21–19–18–3–14–13–12–11–10–9–8–7–6–5–4–16–15–20–27–26–29–28–25–2–17–32–33–30–31–141–139–127–122–105–106–114–113–198–132–133–137–142–145–153–154–150–151–143–138–131–203–205–135–147
2	73–90–91–218–219–227–228–230–229–234–233–241–1–249–248–247–245–244–243–237–223–231–232– 240–242–235–238–239–246–236–226–225–224–222–221–220–215–207–210–214–217–216–213–212–211– 208–206–209–85–86–79–66–59–63–70–56–55–54–52–49–46–42–38
3	34–43–44–48–60–62–61–74–88–87–92–94–93–89–84–83–82–81–78–68–69–72–75–76–80–71–64–51–47–40–39–37–35–36–41–45–57–58–65–67–53–50–152–144–149–148–146–134–162–158–157–156–161–160–164–173–174–171–167–166–169–175–180–182–183–187–188–189–192–193–185–184–177–176–163–159–165– 168–170–172–178–179–191–195–194–200–201–202–204

Problem D417: (Number of vehicles = 54; Total distance = 3701.244)

Route #	Tours	Route #	Tours
1	324–327–328–323–325–319–321–340–338	2	118–63–126–87–79–56–107
3	350–332–348–320–342–361–333	4	165–17–72–114–112–68–89
5	164–151–168–148–36–154–171–106	6	11–58–24–75–155–166–117
7	341–336–344–111–35–172–150	8	122–113–93–34–62–115
9	15–98–84–60–131–53–303	10	285–201–296–278–152–149–167–156
11	334–351–352–326–329–358–322–335	12	76–69–337–339–354–362–356–330
13	178–160–161–169–158–146–41–66	14	159–163–157–162–153–147–133–59
15	282–184–120–170–78–65–140	16	144–67–132–42–315–221–179
17	304–218–176–230–272–177–181–266–188	18	189–240–306–243–318–242–262–301
19	245–241–250–253–256–186–312–222	20	269–247–223–261–232–199–185–284
21	180–174–273–239–258–191–246–316	22	281–227–204–236–307–206–220–217–228
23	263–317–251–244–305–277–224	24	205–173–257–275–286–291–267
25	213–289–190–207–196–194–229–310–175	26	238–290–200–212–210–237–202–209–197–208
27	215–211–279–271–274–198–231	28	195–225–260–299–311–265–287–292
29	123–139–130–9–61–54–125	30	182–248–235–264–255–276–308
31	295–302–309–259–254–216–219–293	32	270–288–313–252–39–375–401–249–371
33	283–297–373–363–369–411–412–381	34	298–387–38–124–121–88–12
35	47–48–378–379–31–23–103–80	36	372–383–137–402–365–368–143
37	374–377–414–410–400–389–380–64	38	10–77–128–110–71–51–280–183
39	203–268–109–46–49–43–45	40	391–366–385–55–2–1–28
41	416–388–394–397–404–359–134–16	42	226–193–192–300–187–233–234–214
43	415–32–50–44–6–5–29–52–57	44	357–408–392–398–405–384–390–21
45	395–409–403–413–396–393	46	129–92–4–141–90–99–136
47	314–37–18–30–127–3–142–294	48	74–346–40–97–96–331–349

49	353–343–145–347–345–360–355–83–73	50	25–101–20–108–119–86–100
51	407–386–399–382–370–376–406–364–367	52	94–102–70–7–95–104–19–135
53	116–27–105–33–417–91–13–26	54	22–82–8–14–81–85–138

Problem RC205: (Number of vehicles = 4; Total distance = 1322.816)

Route #	Tours
1	65–64–19–23–21–18–76–85–84–51–49–22–20–24–74–13–17
2	2–45–42–39–36–72–71–62–94–61–44–40–38–41–81–98–53–90–66–91–56–50– 34–32–26–89–48–25–77–58
3	69–82–11–15–16–47–14–12–88–78–73–79–7–6–8–46–5–3–1–4–100–70
4	92–95–63–33–28–27–29–31–30–67–83–57–99–52–86–87–9–59–75–97–10–60–55–68–54–43–35–37–93–96 –80

Problem E249: (Number of vehicles = 5; Total distance = 394.926)

Route #	Tours
1	147–152–130–124–125–127–123–121–109–107–117–115–118–119–116–33–32–17–2–25–28–29–26–27–20–15–16–14–11–7–5–10–13–4–8–9–6–12–3–18–19–21–24–22–23–156–157–158–172–178
2	42–43–44–38–52–54–66–59–63–61–62–60–56–55–48–49–46–34–36–45–41–37–35–155–141–139–110–98– 97–95–96–99–163–159–166–165–162–160–161–164–168–171–173–174–170–179–187–189–188–191
3	73–87–88–92–94–93–89–84–83–82–81–78–74–70–68–58–57–65–69–72–75–76–77–80–71–64–67–53–50–39–40–47–51–31–30–129–111–108–120–114–113–190–177–176–167–169–175–180–182–183
4	195–196–197–194–192–186–181–184–185–193–199–200–201–202–131–203–138–143–144–150–151–154–153–146–136–134–140–128–126–122–105–104–103–101–100–102–106–112–198–149–148–145–142–133– 132–137–135–205–204
5	207–215–223–226–233–234–231–232–237–235–236–246–238–239–242–240–243–247–1–248–249–244–245–241–230–229–228–227–225–224–222–221–220–210–214–219–217–216–218–213–212–211–208–206–209–90–91–86–85–79

Problem E417: (Number of vehicles = 54; Total distance = 4569.898)

Route #	Tours	Route #	Tours
1	15–93–98–100–116–64–132	2	324–328–323–325–327–352–326–322–89
3	76–336–344–339–354–337–57	4	22–46–49–45–303–42–315
5	10–4–141–333–351–335–330–361–345	6	84–60–130–358–355–349
7	341–356–329–321–319–362–340–338	8	67–54–87–126–62–125
9	179–59–19–133–155–75–24	10	165–102–114–69–151–35–183–310
11	159–163–157–161–117–152–167–65	12	123–7–112–111–148–149–79
13	144–12–135–150–172–171–166	14	156–162–153–36–104–17–168–154–73
15	282–41–106–61–53–129–316	16	164–147–158–169–146–78–140
17	269–181–188–177–296–201–184–66	18	180–266–191–258–277–318–120–11

19	245–240–250–242–262–199–286	20	263–306–244–251–317–243–305
21	280–228–261–223–189–173–205–218	22	295–211–299–198–197–208–210–230–221
23	178–247–246–257–381–386–363–364	24	174–275–312–274–311–271–222–185
25	281–227–204–193–217–220–206–276–288	26	213–259–309–302–256–301–186–216
27	267–234–192–214–187–260–255–229	28	215–279–264–307–225–292–175
29	226–232–284–291–254–237–202–233	30	127–3–182–207–219–289–190–194–293
31	195–248–235–300–231–209–236–200–196	32	407–373–411–412–252–268–28–44–71
33	304–18–124–37–313–387–142–308	34	283–203–38–401–375–39–367–370–297–265
35	285–139–278–160–170–92–58	36	294–128–110–27–109–143–383–371
37	298–400–380–365–368–385–13	38	377–374–382–399–369–406–414–287
39	372–249–253–241–273–239–272–176–224	40	270–80–48–82–1–389–137–410
41	21–33–113–55–346–366–378	42	238–290–212–51–77–118–99–86
43	74–32–392–397–347–343–398–376	44	353–359–384–134–96–14–16
45	122–30–20–29–5–108–52–83	46	416–417–105–405–388–379–23–43
47	9–94–334–348–70–95–68–72	48	395–409–396–393–403–413
49	314–402–415–81–90–136–138	50	357–50–2–31–91–103–119–85
51	391–8–390–408–332–320–342	52	25–101–350–6–88–121
53	47–404–145–394–40–97–331–360	54	26–107–56–115–63–131–34

References

[1] E.K. Baker, J.R. Schaffer, Solution improvement heuristics for the vehicle routing problem with time window constraints, American Journal of Mathematical and Management Sciences 6 (1988) 261–300.

[2] J.J. Barthodi, J.K. Platzman, An O(N log N) planar tavelling salesman heuristic based on spacefilling curve, Operations Research Letters 1 (1982) 121–125.

[3] L. Bodin, B.L. Golden, A. Assad, M. Ball, Routing and scheduling of vehicles and crews: The state of the art, Computers & Operations Research 10 (1983) 62–212.

[4] J. Bramel, D. Simchi-levi, Probabilistic analysis and practical algorithms for the vehicle routing problem with time windows, Operations Research 44 (1996) 501–509.

[5] J. Bramel, D. Simchi-levi, On the effectiveness of set covering formulations for the vehicle routing problem with time windows, Operations Research 45 (1997) 295–301.

[6] W. Chiang, R. Russell, Simulated annealing methaheuristics for the vehicle routing problem with time windows, Annals of Operations Research 63 (1996) 3–27.

[7] W. Chiang, R. Russell, A reactive tabu search methaheuristic for the vehicle routing problem with time windows, Transportation Science (forthcoming).

[8] M. Desrochers, J. Desrosiers, M. Solomon, A new optimization algorithm for the vehicle routing problem with time windows, Operations Research 40 (1992) 342–354.

[9] M. Desrochers, J.K. Lenstra, M.W.P. Savelsbergh, F. Soumis, Vehicle routing with time windows: Optimization and approximation, in: B.L. Golden, A.A. Assad (Eds.), Vehicle Routing: Methods and Studies, North-Holland, Amsterdam, 1988, pp. 65–84.

[10] J. Desrosiers, Y. Dumas, M. Solomon, F. Soumis, Time constrained routing and scheduling, in: M. Ball, T. Magnanti, M. Monma, G. Nemhauser (Eds.), Handbooks in Operations Research and Management Science, vol. 8: Network Routing, North-Holland, Amsterdam, 1995, pp. 35–139.

[11] M. Fisher, Vehicle routing, in: M. Ball, T. Magnanti, M. Monma, G. Nemhauser (Eds.), Handbooks in Operations Research and Management Science, vol. 8: Network Routing, North-Holland, Amsterdam, 1995, pp. 1–33.

[12] M. Fisher, R. Jaikumar, A generalized assignment heuristic for the vehicle routing problem, Networks 11 (1981) 109–124.

[13] M. Fisher, K.O. Jornsten, O.B.G. Madsen, Vehicle routing with time windows: two optimization algorithms, Operations Research 45 (1997) 488–492.
[14] B.L. Garcia, J.Y. Potvin, J.M. Rousseau, A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints, Computers & Operations Research 21 (1994) 1025–1033.
[15] B. Gillett, L. Miller, A heuristic algorithm for the vehicle dispatch problem, Operations Research 22 (1974) 340–349.
[16] N. Kohl, O.B.G. Madsen, An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation, Operations Reseach 45 (1997) 395–406.
[17] A.W.J. Kolen, A.H.G. Rinnooy Kan, H.W.J.M. Trienkens, Vehicle routing with time windows, Operations Research 35 (1987) 266–273.
[18] G. Kontoravdis, J.F. Bard, A GRASP for the vehicle routing problem with time windows, ORSA Journal of Computing 7 (1995) 10–23.
[19] A. Konskosidis, W.B. Powell, M.M. Solomon, An optimization-based heuristic for vehicle routing and scheduling with soft time window constraints, Transportation Science 26 (1992) 69–85.
[20] F.-H.F. Liu, S.Y. Shen, An approach for solving large-scale and complex vehicle routing problems, Working paper, National Chiao Tung University, Hsinchu, Taiwan, 1997.
[21] I.H. Osman, Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem, Annals of Operations Research 41 (1993) 421–451.
[22] J. Potvin, S. Bengo, The vehicle routing problem with time windows: Part II: Genetic search, INFORMS Journal on Computing 8 (1996) 165–172.
[23] J. Potvin, T. Kervahut, B. Garcia, J. Rousseau, The vehicle routing problem with time windows: Part I: Tabu search, INFORMS Journal on Computing 8 (1996) 158–164.
[24] J. Potvin, J. Rousseau, A parallel route building algorithms for the vehicle routing and scheduling problem with time windows, European Journal of Operational Research 66 (1993) 331–340.
[25] J. Potvin, J. Rousseau, An exchange heuristic for routeing problems with time windows, Journal of the Operational Research Society 46 (1995) 1433–1446.
[26] Y. Rochat, E.D. Taillard, Probabilistic diversification and intensification in local search for vehicle routing, Journal of Heuristics 1 (1995) 147–167.
[27] R.A. Russell, Hybrid heuristics for the vehicle routing problem with time windows, Transportation Science 29 (1995) 156–166.
[28] M.W.P. Savelsbergh, Local search in routing problems with time windows, Annals of Operations Research 4 (1985) 285–305.
[29] M.W.P. Savelsbergh, The vehicle routing problems with time windows: minimizing route duration, ORSA Journal of Computing 4 (1992) 146–154.
[30] M. Solomon, Algorithms for the vehicle routing and scheduling with time window constraints, Operations Research 15 (1987) 254–265.
[31] E. Taillard, P. Badeau, M.M. Gendreau, F. Guertin, J. Potvin, A tabu search heuristic for the vehcile routing problem with soft time windows, Transportation Science 31 (1997) 170–186.
[32] S.R. Thangiah, I.H. Osman, and T. Sun, Methaheuristics for vehicle routing problems with time windows, Technical Report, SRU-CpSc-TR-95-32, Artificial Intelligence and Robotics Laboratory, Computer Science Department, Slippery Rock University, Pennsylvania, 1995.
[33] P.M. Thompson, H.N. Psaraftis, Cyclic transfer algorithms for multivehicle routing and scheduling problems, Operations Research 41 (1993) 935–946.