European Journal of Operational Research 243 (2015) 798–814

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A variable neighborhood search for the capacitated vehicle routing problem with two-dimensional loading constraints

Lijun Wei [a], Zhenzhen Zhang [b,*], Defu Zhang [c], Andrew Lim [d,e]

[a] School of Information Technology, Jiangxi University of Finance and Economics, Nanchang 330013, Jiangxi, People's Republic of China
[b] Department of Management Sciences, City University of Hong Kong, Tat Chee Ave, Kowloon Tong, Hong Kong
[c] Department of Computer Science, Xiamen University, Xiamen 361005, China
[d] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore 637371, Singapore
[e] International Center of Management Science and Engineering, School of Management and Engineering, Nanjing University, Nanjing 210093, People's Republic of China

ARTICLE INFO

Article history:
Received 18 September 2013
Accepted 29 December 2014
Available online 6 January 2015

Keywords:
Routing
Packing
Variable neighborhood search
Skyline heuristic
2L-CVRP

ABSTRACT

This paper addresses the capacitated vehicle routing problem with two-dimensional loading constraints (2L-CVRP), which is a generalized capacitated vehicle routing problem in which customer demand is a set of two-dimensional, rectangular, weighted items. The objective is to design the route set of minimum cost for a homogenous fleet of vehicles, starting and terminating at a central depot, to serve all the customers. All the items packed in one vehicle must satisfy the two-dimensional orthogonal packing constraints. A variable neighborhood search is proposed to address the routing aspect, and a skyline heuristic is adapted to examine the loading constraints. To speed up the search process, an efficient data structure (Trie) is utilized to record the loading feasibility information of routes, but also to control the computational effort of the skyline spending on the same route. The effectiveness of our approach is verified through experiments on widely used benchmark instances involving two distinct versions of loading constraints (*unrestricted* and *sequential* versions). Numerical experiments show that the proposed method outperforms all existing methods and improves or matches the majority of best known solutions for both problem versions.

1. Introduction

This paper considers an important extension of the classical vehicle routing problem (VRP), called capacitated vehicle routing problem with two-dimensional loading constraints (2L-CVRP). This problem is a combination of two most important NP-hard optimization problems in distribution logistics, the capacitated vehicle routing problem (CVRP) and two-dimensional bin packing problem (2BPP). In the 2L-CVRP, the demand of customers is formed by several two-dimensional rectangular weighted items, while vehicles are identical with the same weight capacity and loading surface. The purpose is to design routes with minimum cost for given vehicles to serve all the customers and, at the same time, load the corresponding items into the vehicles successfully.

The 2L-CVRP is particularly important in both practical and theoretical aspects. The CVRP is a simplified model for practical operators. However, in the real world, logistics managers usually have to deal with routing and packing problems simultaneously. Moreover, because of the weight, fragility or the large height of freight, numerous real life applications involve the delivery of rectangular-shaped items that cannot be stacked on each other, such as household appliances, delicate pieces of furniture, etc. Therefore, 2L-CVRP has an obvious commercial value. From a theoretical perspective, 2L-CVRP is composed of two NP-hard problems (CVRP and 2BPP); thus, it is also a challenging NP-hard problem of high complexity. The CVRP (Toth & Vigo, 2002) has been extensively studied (Gendreau, Hertz, & Laporte, 1994; Prins, 2004). The loading component of 2L-CVRP is closely related to the two-dimensional bin packing problem (2BPP), which aims at packing a given set of rectangular items into the minimum number of identical rectangular bins. For more related literature, the reader is referred to Lodi, Martello, and Vigo (1999) and Lodi, Martello, and Monaci (2002).

Routing and packing problems have been studied intensively but separately, with the combined problem only being introduced in recent years. The 2L-CVRP was first presented by Iori, Salazar-González, and Vigo (2007), and only small-scale instances were solved via an exact algorithm based on the branch-and-cut technique. For larger scale

* Corresponding author. Tel.: +852 64067667/54323245; fax: +852 34420188.
E-mail addresses: villagerwei@gmail.com (L. Wei), zhenzhenzhang222@gmail.com (Z. Zhang), dfzhang@xmu.edu.cn (D. Zhang), alim.china@gmail.com (A. Lim).

http://dx.doi.org/10.1016/j.ejor.2014.12.048
0377-2217/

problems, Gendreau, Iori, Laporte, and Martello (2008) proposed the first meta-heuristic approach, Tabu Search (TS). Then, Zachariadis, Tarantilis, and Kiranoudis (2009) developed the Guided Tabu Search (GTS), which incorporates the rationale of Tabu Search and Guided Local Search, and relies on a bundle of heuristics to check the feasibility of loading. The Extended Guided Tabu Search (EGTS) and simulation annealing (SA), which introduce a new scoring-based heuristic to improve packing, were proposed by Leung, Zhou, Zhang, and Zheng (2011) and Leung, Zheng, Zhang, and Zhou (2010), respectively. For a population-based algorithm, the effective saving-based Ant Colony Optimization (ACO) was presented by Fuellerer, Doerner, Hartl, and Iori (2009), and its performance was proven to be quite satisfactory. Recently, Duhamel, Lacomme, Quilliot, and Toussaint (2011) proposed the Greedy Randomized Adaptive Search Procedure combined with Evolutionary Local Search (GRASP × ELS) algorithm, whereby the loading component is examined via solving the resource constrained project scheduling problem (RCPSP). This algorithm outperforms all previous methods and obtains new better solutions for several instances; however, only the *Unrestricted* version of 2L-CVRP was solved. Lately, Zachariadis, Tarantilis, and Kiranoudis (2013) proposed an innovative compact meta-heuristic, named as promise routing-memory packing (PRMP), which obtains excellent performance and improves best known solutions for many instances.

The more general problem, capacitated vehicle routing problem with three-dimensional loading constraints (3L-CVRP), has received increasing attention from researchers. Several algorithms proposed for solving 2L-CVRP were extended to 3L-CVRP, e.g., TS (Gendreau, Iori, Laporte, & Martello, 2006), GTS (Tarantilis, Zachariadis, & Kiranoudis, 2009), ACO (Fuellerer, Doerner, Hartl, & Iori, 2010), and GRASP × ELS (Lacomme, Toussaint, & Duhamel, 2013). In addition, Ma, Zhu, and Xu (2011) provided a combined tabu search with local search. Bortfeldt (2012) introduced a hybrid algorithm, which employs tabu search for routing and tree search algorithm for loading. Zhu, Qin, Lim, and Wang (2012) developed a two-stage tabu search for routing, adopted enhanced heuristics for loading and obtained excellent results. Iori and Martello (2010) provided a review in regard to vehicle routing problems with two- and three-dimensional loading constraints. Wei, Zhang, and Lim (2014) introduced a heterogeneous fleet vehicle routing problem with three-dimensional loading constraints (3L-HFVRP).

The VNS (Hansen, Mladenović, Brimberg, & Pérez, 2010) has been proven quite effective to solve the VRP and its variants (Fleszar, Osman, & Hindi, 2009; Hemmelmayr, Doerner, & Hartl, 2009; Imran, Salhi, & Wassan, 2009; Paraskevopoulos, Repoussis, Tarantilis, Ioannou, & Prastacos, 2007). Motivated by these successes, this paper develops a variable neighborhood search (VNS) for 2L-CVRP, which incorporates a skyline heuristic for solving the loading component. To the best of our knowledge, this is the first implementation of VNS for 2L-CVRP. In our method, an insertion-based heuristic is adopted to generate the initial solution, which is used for further improvement. Regarding the routing aspect, VNS can explore the solution space systematically and effectively. Moreover, in order to ensure the loading feasibility of routes, an effective skyline heuristic is employed to solve the packing part of the problem. To accelerate the procedure, a data structure (Trie) is used to record the loading feasibility information, which avoids checking the visited routes repeatedly. In addition, it is also used to control the number of times the skyline heuristic is called for the same route. The proposed algorithm was tested extensively on 2L-CVRP benchmark instances and compared with previously published algorithms. The results demonstrate that our algorithm outperforms all other methods and identifies many new best solutions, especially for large scale instances.

The remainder of this paper is organized as follows. Section 2 describes the problem in detail. Section 3 presents the proposed variable neighborhood search methodology, while Section 4 provides a detailed description of the skyline heuristic for examining the loading subproblem. The extensive computational results on benchmark instances and comparisons with other algorithms are presented in Section 5. Finally, the conclusions are given in Section 6.

2. Problem description

The 2L-CVRP is defined on a complete undirected graph $G = (V, E)$, where $V = \{0, 1, \ldots, N\}$ is the vertex set and $E = \{(i, j) | i, j \in V, i \neq j\}$ is the edge set. Vertex 0 represents the central depot and the set of vertices $\{1, 2, \ldots, N\}$ denotes the location of customers. Each edge $(i, j) \in E$ associates with a travel cost c_{ij} that corresponds to the cost for going from vertex i to j or from vertex j to i. At the central depot, a fleet of K homogeneous vehicles is available. Each vehicle has weight capacity D and a two-dimensional rectangular loading surface of width W and length L. The loading area is denoted as $A = W \times L$. The demand of each customer i $(i = 1, \ldots, N)$ is defined as a set of m_i rectangular items denoted as I_i, and the total weight of I_i is d_i. Each item $I_{ir} \in I_i (r = 1, \ldots, m_i)$ is characterized by a specific width w_{ir} and length l_{ir}. In addition, the total area of items I_i is denoted as $a_i = \sum_{r=1}^{m_i} w_{ir} l_{ir}$. For 2L-CVRP, a feasible solution must satisfy the following constraints:

(a) Every vehicle starts and terminates its route at the central depot.
(b) All the customers must be served by using no more than the given K vehicles.
(c) Each customer is visited exactly once, and all the demanded items must be loaded into the vehicle.
(d) The capacity, length, and width of every vehicle cannot be exceeded by loaded items.
(e) Each item has a fixed orientation that cannot be rotated. In other words, each item is loaded with its width (length) parallel to the corresponding width (length) of the vehicle surface.
(f) All items of customers assigned to the same route must be loaded into the vehicle successfully without any overlap.

The objective of 2L-CVRP is to minimize the total travel cost of routes which serve all the customers and satisfy all the constraints, as well. This paper considers two versions of this problem: *Unrestricted* 2L-CVRP and *Sequential* 2L-CVRP. The *Unrestricted* 2L-CVRP is described in the previous paragraph, which only concerns loading items into the vehicle. *Sequential* 2L-CVRP considers both loading and unloading operators, in which an additional *LIFO (last in first out) constraint* is imposed: when visiting a customer, his or her items can be unloaded by straight movements parallel to the length dimension of the vehicle surface, without the need to rearrange items that belong to other customers in the same route. In other words, no item of customer j in the same route to be served later than customer i can be placed between items of i and the rear door of the vehicle. This arises in practical cases when it is difficult to move the items due to their weight or fragility, especially when the items are unloaded by means of forklift trucks from the rear door of vehicles. Fig. 1 gives an example of the two versions.

3. The VNS algorithm for 2L-CVRP

Variable Neighborhood Search (VNS) is first proposed by Mladenović and Hansen (1997) to solve combinatorial and global optimization problems. VNS is derived from the idea of systematically changing neighborhoods during the search. The underlying theory to obtain a better local optimum is that a local optimum under one neighborhood structure is not necessary so far from another.

In the classical implementation of VNS algorithm, four key components should be specified: (i) method to construct an initial solution;

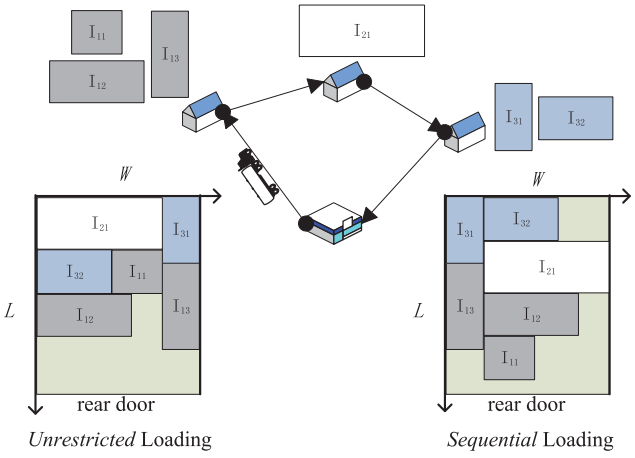


Fig. 1. An example of *Unrestricted* and *Sequential* 2L-CVRP.

(ii) the set of neighborhood structures *NS*, which helps to slightly diversify the starting point of local search; (iii) local search, which improves the solution quickly; and (iv) acceptance criterion that determines the termination of the search. In general, the VNS starts with initializing a solution and the set of neighborhood structures *NS*. Subsequently, the main solution search procedure, which includes a random shaking and a local search component, is repeated until the given stopping criterion is met. In each iteration of the main solution search, given the current solution S, a random neighboring S' of S is generated with a neighborhood structure NS_k (k is set to 1 before the first iteration). S' is taken as a temporary solution, and then optimized by the local search to determine the local optimum S''. If the obtained new solution S'' is better than the incumbent best solution, the algorithm updates the best solution and sets S to S''. Then the search continues with the new starting point S'' and k is reset to 1. In other words, a new random neighboring solution is generated from S'' according to neighborhood structure NS_1 in the next iteration. Otherwise, S remains the starting point and a random neighbor is generated according to neighborhood NS_{k+1} in the next iteration. This process repeats until all neighborhoods are attempted and no new better solution is identified.

Our variable neighborhood search for 2L-CVRP is given in Algorithm 1. Here, our algorithm is slightly different with classical VNS, i.e., instead of generating only one random point, up to K random points are produced from each neighborhood structure (Line 8). For each random point, we try to find the local optimum by the local search. When any improvement is found, we resume the search with the new solution and revert the neighborhood structure back to NS_1 immediately. Otherwise, we jump to the next neighborhood structure. This helps to better explore the current neighborhood structure.

In addition, in order to better explore and exploit the search space, after trying all neighborhood structures, a diversification procedure is introduced to produce a new initial solution based on the incumbent best solution (Line 16). Then the main solution search procedure resumes with the new initial solution. The diversification procedure differs from the neighborhood structures, in that it will drive the search to other regions which neighborhood structures cannot easily reach. This process is repeated until the given time is elapsed.

We describe our method to construct the initial solution in Section 3.1 and the set of neighborhood structures in Section 3.2. Our local search LocalSearch used to find the local optimum is given in Section 3.3 and the diversification procedure Diversify is presented in Section 3.4. Finally, we present some acceleration strategies in Section 3.5.

Algorithm 1 The proposed variable neighborhood search for 2L-CVRP.

```
VNS()
1  Construct the initial solution S
2  Define a set of neighborhood structures NS_i (i = 1, ··· , k)
3  S* = S, nonImp = 0
4  while time limit is not exceeded
5      nonImp = nonImp + 1
6      k = 1
7      while k ≤ |NS|
8          repeat K times
9              Generate a random neighboring S' of S using NS_k
10             S'' ← LocalSearch(S')
11             if S'' is better than S*
12                 S* = S'', S = S''
13                 k = 0, nonImp = 0
14                 break the loop
15         k = k + 1
16     S = Diversify(S*)
17 return S*
```

3.1. Initial solution

The initial solution is constructed via an insertion-based mechanism. At first, K empty routes are generated for all given vehicles. The customers are sorted by the decreasing value of total area of its items a_i ($i = 1, \ldots, N$), as the loading constraints play the key role in determining the feasibility of the solution. Then, the customers are inserted into the routes one by one according to the sorted sequence. Assume the sorted sequence is T. For each customer $T_i (i = 1, \ldots, N)$, we first try to assign it to the route which has the least unoccupied loading surface, but can accommodate T_i feasibly. This helps to increase the utilization of loading surface. If such a route is found, the customer is inserted into this route in the position, which leads to the minimum incremental cost. In particular for *Sequential* 2L-CVRP, the loading feasibility of all possible insertion positions needs to be examined, then the feasible and optimal position is picked. If it is determined that T_i cannot be inserted into any of K routes, the customer sequence is updated by exchanging the positions of customers T_i and T_j (j is randomly selected from $1, \ldots, i-1$), and the construction algorithm restarts with the new customer sequence. The above process repeats until a feasible solution is obtained. The average numbers of iterations to obtain a feasible initial solution are 2 for both versions. In the worst case, the numbers may reach up to 38 and 28 for *Unrestricted* and *Sequential* versions, respectively.

3.2. Neighborhood structures

The neighborhood structures, which are utilized to slightly diversify the starting point of local search, is at the heart of VNS. Each neighborhood should attain a proper balance between perturbing the incumbent solution and retaining the good part of the current solution (Hemmelmayr et al., 2009). We employ six neighborhood structures derived from the following two move classes.

Segment exchange (**SE**): Two sequences of consecutive customers from different routes exchange their positions. The procedure starts by taking a random segment of customers from a randomly chosen route, and then attempts to exchange the positions systemically with another random segment from a different randomly chosen route. If the resulting solution is infeasible, the exchange is cancelled. The above procedure is repeated until the resulting solution is feasible. On average, the numbers of iterations to obtain a feasible solution are 10.8 and 19.5 for *Unrestricted* and *Sequential* versions, whereas

the numbers may reach up to 70 and 108 for the versions in the worst case. In order to obtain a feasible solution easily, the exchange is only performed between segments of the same length. Moreover, the longer the segment length, the lower the probability to obtain a successful exchange. In this study, the segment length is limited to 2 and 3, and the corresponding neighborhoods are denoted as **SE2** and **SE3**, respectively.

Move combination (**MC**): It consists of two randomly chosen simple moves from the six move types presented in Section 3.3. We first select two types randomly from the six types. For each type, we repeatedly try to perform a random move until the resulting solution of this move is feasible. On average, the numbers of iterations to obtain a feasible solution are 8.3 and 12.2 for *Unrestricted* and *Sequential* versions, whereas the numbers may reach up to 27 and 49 for the versions in the worst case. This neighborhood is marked as **MC2**.

Our final neighborhood structures perform each individual move class several times which is controlled by the variable *strength*. Considering the purpose of neighborhood structures, the used *strength* is 1 and 2. Therefore, the derived six neighborhood structures are **SE21** (perform **SE2** once), **SE22** (perform **SE2** two times), **SE31**, **SE32**, **MC21**, and **MC22**. The random order of neighborhoods is adopted by shuffling the sequence of neighborhood structures randomly when finding a new best solution.

3.3. Randomized local search

The local search procedure LocalSearch is utilized to obtain a local optimal solution of the temporal solution generated from neighborhood structures. Our local search is made up of six basic move types (Zachariadis et al., 2009), which can improve the solution quickly and efficiently. The detailed operators are described as follows.

intra-swap: swap the positions of a pair of customers within the same route.
inter-swap: swap the positions of a pair of customers from different routes.
intra-shift: a customer is shifted from its position to elsewhere within the same route.
inter-shift: a customer of one route is sent to another route.
intra-2opt: for a given route, two non-adjacent arcs are deleted, the middle segment of route is reversed, and then two new arcs are added to link the route again. This often helps removing arc crossings.
inter-2opt: two arcs from different routes are eliminated so that each route is cut into two parts, then each first part of one route is concatenated with the last part of the other route to generate two new routes. If one route is empty, the other route is split into two routes. Thus, route split is a special case of this operator. This move brings more dramatic change to the solution.

The local search method adopts the first improvement strategy to systemically explore all possible moves of six types in order to obtain better solutions. In other words, the method tries all possible moves of the current type until the first improving move is found, then this move is performed immediately. Otherwise, the next move type is applied until all types are tried and no improvement is found. Note that only a loading-feasible solution is accepted during the procedure.

In our search, six move types are employed in random order. At the beginning, the order of six move types is randomly generated. The procedure tries each move type in the generated order. If an improving solution is found for the current type, a new order of move types is generated through randomly shuffling the old one, and then the procedure restarts with the new solution and new order of move types. Otherwise, the procedure proceeds to the next type.

Considering an instance with N customers and K vehicles, the number of pairs of customers is $N(N-1)/2$, so the complexity of examining all moves of swap (intra- and inter-) is $O(N^2)$. Similarly, there are $N+K-1$ possible positions for the insertion of each customer; therefore, the complexity of shift types is $O(N(N+K))$. The cardinality of all cutting arcs is $N+K$; hence, trying all intra-2opt and inter-2opt requires $O((N+K)^2) = O(N^2)$ $(K \ll N)$ computational effort.

3.4. Diversification mechanisms

The diversification procedure Diversify is incorporated to escape from the incumbent local optimum, since the local search is usually stuck at a local optimal solution. We use the incumbent best solution as the input of the procedure Diversify to obtain a new initial solution.

A ruin-reconstruct mechanism is utilized to diversify the search process. The basic idea of the ruin-reconstruct approach is to generate a new solution by destructing the input feasible solution first and then rebuilding a complete solution from the resulting partial solution, which is similar to the ruin-recreate heuristic proposed by Ropke and Pisinger (2006) and Schrimpf, Schneider, Stamm-Wilbrandt, and Dueck (2000). First, several randomly chosen customers are removed from the solution and put into a pool temporarily. Then, the rebuilding process starts. The customers in the pool are first sorted by decreasing area of their demanded items and reinserted into the partial solution one by one. The route and position for a customer is decided by the same rule used in the constructing heuristic (Section 3.1). A slight difference occurs when a customer cannot be inserted into any route feasibly. If such a case occurs, a route is chosen randomly, and customers in this route are randomly erased, one at each time, until the given customer can be inserted into this route successfully. This is similar to the concept of the ejection chains method (Lim & Zhang, 2007), which has been proved to be effective to solve tightly constrained problems. The erased customers are put into the pool, and the rebuilding process restarts with the new pool. The rebuilding process is repeated until a complete solution is found, i.e., the pool is empty.

In our implementation, the number of removed customers is set to $\min(0.5 \times N, 0.1 \times N + nonImp)$, where N is the total number of customers, and *nonImp* is the number of iterations without improvement after calling the VNS. The larger the number *nonImp*, the harder it is to escape from the local optimum. By setting the number of removed customers in such a manner, the solution generated retains most good characteristics of the input solution and brings greater difference while the number *nonImp* is larger.

3.5. Acceleration strategies

The packing method is time-consuming and is called quite frequently, which dramatically contributes to the overall computational effort. To accelerate the proposed algorithm, unnecessary calls to the packing method should be avoided, such as duplicated examination and checking for unpromising solutions. Therefore, three strategies used in our methodology are explained as follows.

Based on the characteristics of the problem, a common strategy is employed. During the local search procedure, for a feasible solution, moves within a route (Intra- moves) are always feasible for *Unrestricted* 2L-CVRP. Thus, they are executed without any examination. For Inter-Shift, routes obtained by removing a customer must be loading-feasible for both *Unrestricted* and *Sequential* versions; these examinations can be avoided too.

Furthermore, it is beneficial not to spend time on unpromising solutions by using the "evaluating first-packing second" strategy, which was first explicitly explained by Bortfeldt (2012). The cost of a move is first evaluated, and the packing heuristic is applied only when the

Algorithm 2 TabuPack procedure.

TabuPack(RT, $iter$)

```
1  for each sorting rule
       // Generate initial sequence
2      R = sort sequence of all rectangles belonging to the customers in RT
3      let n be the number of rectangles in R
4      if HeuristicPack(R) places all rectangles
5          return SUCCESS
6      for k = 1 to iter
7          Generate n non-tabu sequences {R_1, ..., R_n} from R by swapping two rectangles
8          Let R* be the sequence with the highest area utilization using HeuristicPack(R)
9          if HeuristicPack(R*) places all rectangles
10             return SUCCESS
11         Let (b_i, b_j) be the rectangles swapped to produce R* from R
12         Add (b_i, b_j) to the tabu list for the next 3n iterations
13         R = R*
14 return FAILURE
```

move leads to a better solution. This operator prevents loads of feasibility examination of unpromising solutions; therefore, much computational time is saved.

In addition, in order to speed up our procedure, we use a special data structure (Trie) to keep track of the loading feasibility of routes already examined. A similar strategy is also employed by other authors, e.g., tree structure (Zachariadis et al., 2009), pool (Fuellerer et al., 2009), hash table (Zachariadis et al., 2013), and have been demonstrated to be quite useful to save running time. However, our Trie is different from the Trie used in previous literature (Leung, Zhang, Zhang, Hua, & Lim, 2013). We will present the details of our Trie in Section 4.

4. The skyline heuristic for the loading subproblem

We use the procedure TabuPack to check the packing feasibility of a route. The TabuPack procedure is given in Algorithm 2, which is adapted from the 2DRPSolver proposed by Wei, Oon, Zhu, and Lim (2011) for the two-dimensional rectangle packing problem. It takes a route tour RT containing the ordered customers visited by a vehicle, and a parameter $iter$ used to control the number of iterations as input. If the procedure finds a feasible packing for this vehicle, it reports a const SUCCESS; otherwise, it returns a const FAILURE.

The TabuPack is actually a sequence-based tabu search procedure. For each sorted sequence of rectangles, we use a heuristic packing subroutine HeuristicPack (see Section 4.1 for details) to try to pack the rectangles into the vehicle surface as much as possible. We first use the following three sorting rules to generate an initial sequence R of the rectangles belonging to the customers in RT (line 2):

1. sort by area of the rectangle in decreasing order;
2. sort by width of the rectangle in decreasing order;
3. sort by length of the rectangle in decreasing order.

Especially for the *Sequential* 2L-CVRP, we will first sort all the rectangles by the reverse visiting order of its customers, and then use the above sorting rules to sort the rectangles of each customer.

If the HeuristicPack finds a packing pattern that places all rectangles in R, SUCCESS will be returned. Otherwise, we attempt to optimize the solution in terms of area utilization through a tabu search mechanism. The number of iterations is determined by the input parameter $iter$ (Lines 6–13). The neighborhood operator simply swaps two randomly selected rectangles in the current sequence, and the same subroutine HeuristicPack will be called on the modified sequence. We generate n (n is the number of rectangles in the input customers) such sequences that are not forbidden by the tabu list, select the one that produces the solution with the highest area utilization, and insert the corresponding swap into the tabu list; and this swap is forbidden for the next $3n$ iterations. At any point in the process, if a packing pattern is identified that places all n rectangles, then the procedure halts with SUCCESS. If this does not occur, then TabuPack reports a FAILURE.

In order to speed up our procedure, we use a special data structure (Trie) to keep track of the loading feasibility of routes already examined. Our Trie stores not only the feasibility information of routes, but also the iteration number $iter$ in the previous call of the TabuPack for the same route. More specifically, given a sequence of customers RT representing a route, in order to determine whether the route is feasible, we first retrieve the information from the Trie. Let the stored feasibility information be *packed*, the stored iteration number be $iter_0$ and the number of rectangles of customers in the route be n. We make different decisions based on the following four cases:

1. the route does not exist in the Trie: call $TabuPack(RT, 1)$, store the results including the $iter$ value 1, sequence of this route and packing results into the Trie, then return the results;
2. $packed ==$ SUCCESS: return SUCCESS;
3. $packed ==$ FAILURE, $iter_0 < n$: call TabuPack (seq, $\min(2 \times iter_0, n)$), update the feasibility of this route and the new iteration value $\min(2 \times iter_0, n)$ in the Trie and return the results;
4. $packed ==$ FAILURE, $iter_0 == n$: return FAILURE.

The Trie could save time in two aspects. For the route whose feasible packing has been found by TabuPack, the stored information in the Trie could avoid the unnecessary call of TabuPack on the same routing later (case 2). For the route whose feasible packing is not found currently, we will iteratively double the effort spent on this route (cases 1 and 3). Thus, more effort will be spent on the difficult instance. However, in order to not spend too much time on the same route, we limit the maximum call to the number of rectangles of the customers in the route. Once this limitation is reached, we regard this route as infeasible and will not call the TabuPack on this route anymore (case 4). The experiments show that this

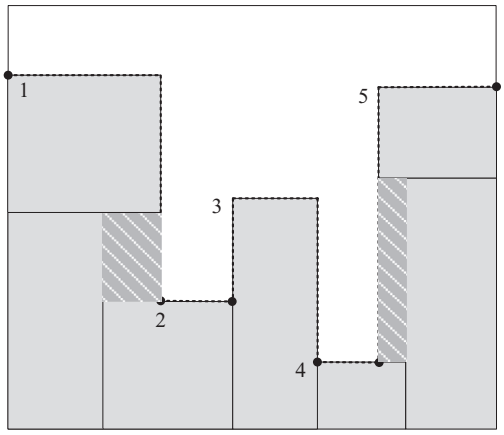


Fig. 2. Example of skyline.

strategy is very useful for the difficult instance; the details are given in Section 5.4.2.

4.1. Heuristic packing subroutine

Our HeuristicPack procedure takes as input a sequence of rectangles R. The purpose of HeuristicPack is to pack the sequence of rectangles into the vehicle surface as much as possible. It is adapted from the 2DRP_Heuristic heuristic proposed by Wei et al. (2011) for the two-dimensional rectangle packing problem. The HeuristicPack heuristic is a skyline-based best-fit algorithm. It uses a skyline to represent the packing pattern, and the candidate positions to place a rectangle are given by the "concave" points on the skyline (see Section 4.1.1 for more details).

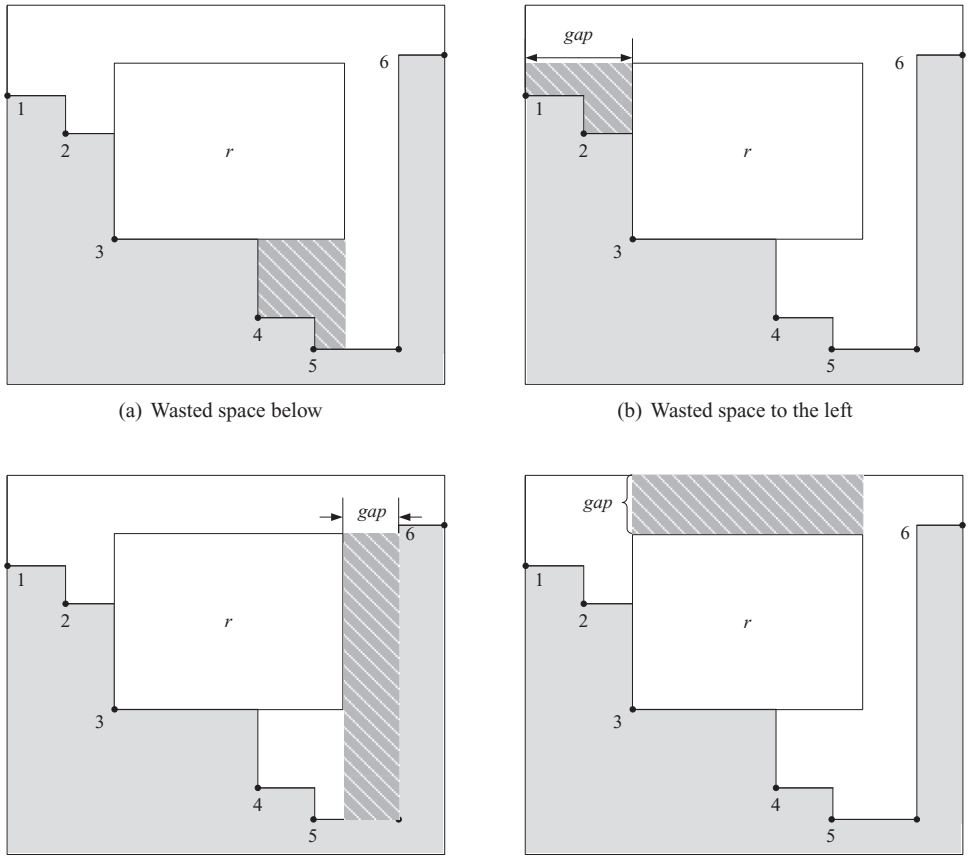
The procedure places the rectangles one by one. In each step, it selects a placement (i.e., a rectangle along with a candidate position) according to a fitness measure and places the selected rectangle at that position. Such step is repeated until no feasible placements remain (either because there are no more rectangles, or no more positions can accommodate any rectangles). For the *Sequential* 2L-CVRP, we only consider the placement whose resulting packing does not violate the LIFO constraint.

We use a function $f(r, p)$ to measure the fitness of packing a rectangle r at a candidate position p. The fitness function $f(r, p)$ produces a vector which consists of six components. The (r, p) is said to be fitter than (r', p') if $f(r, p)$ is lexicographically smaller than $f(r', p')$ (i.e., two vectors are compared component by component, and the order of the first different component determines the order of the two vectors). We select the lexicographically smallest placement from all feasible placements. The six components of $f(r, p)$ are:

$$f(r, p) = (onlyFit(r, p), waste(r, p), -fn(r, p), ord(r), y(p), x(p)) \quad (1)$$

where

1. $onlyFit(r, p)$: if rectangle r is the only rectangle remaining that can fit onto p, this value is 0; otherwise, this value is 1;
2. $waste(r, p)$: the total amount of adjacent wasted space that is created if r is placed at p (see Section 4.1.2 for details);
3. $fn(r, p)$: the number of sides of the rectangle that exactly matches the segment that it is touching in the skyline, called its *fitness number* (see Section 4.1.3 for details);



(a) Wasted space below (b) Wasted space to the left

(c) Wasted space to the right (d) Wasted space above

Fig. 3. Wasted local space.

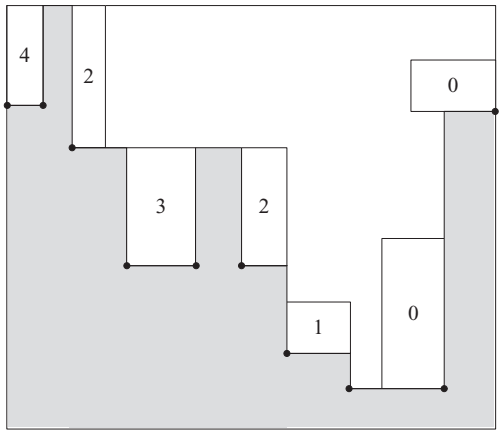


Fig. 4. Fitness numbers.

4. $ord(r)$: the order of r in the ordered set R;
5. $y(p)$: the y-coordinate of position p;
6. $x(p)$: the x-coordinate of position p.

4.1.1. Skyline representation of a packing pattern

A packing pattern is represented by a rectilinear *skyline*, which is the contour of a set of consecutive vertical bars. It is expressed as a sequence of k horizontal line segments $(s_1, s_2, \ldots, s_k)$ satisfying the following properties: (1) the y-coordinate of s_j is different from the y-coordinate of $s_{j+1}, j = 1, \ldots, k-1$; and (2) the x-coordinate of the right endpoint of s_j is the same as the x-coordinate of the left endpoint of $s_{j+1}, j = 1, \ldots, k-1$. Fig. 2 gives an example of a skyline, in which each line segment s_j is labeled as j at its left endpoint. The candidate positions are given by the "concave" points in the skyline, which are depicted by dots in Fig. 2.

4.1.2. Local waste

Our second component of fitness function is the amount of *adjacent waste space*, which is calculated as the total volume of wasted space of the four types given in Fig. 3(a)–(d). Let *w_min* and *h_min* be the minimum width and height, respectively, of the remaining rectangles, excluding r. The shaded area in case (a) is always considered as wasted space. The shaded areas in cases (b) and (c) are considered wasted if the length of the *gap* is less than *w_min*. Finally, the shaded area in case (d) is wasted if the *gap* is less than *h_min*.

4.1.3. Fitness numbers

Our third component of the fitness function is the *fitness number*, which is defined as the number of sides of the rectangle that exactly matches the segment that it is touching in the skyline. If the bottom side of a rectangle is equal to the length of s_j, it is regarded as an exact match. For the left (resp. right) side of a rectangle, an exact match occurs if its height is equal to the y-coordinate of s_{j-1} (resp. s_{j+1}) minus the y-coordinate of s_j. Any side of a rectangle that touches the left, right or bottom side of the sheet is not considered an exact match unless it fills all the remaining space on that side. However, when the top edge of a placed rectangle touches the top of the sheet, this is considered an exact match. The fitness number can be either 0, 1, 2, 3 or 4, as shown in Fig. 4, in which the number in each rectangle depicted is its fitness number. Note that the rectangles in this Fig. 4 are only given for illustrative purpose. They could be larger or smaller, which would lead to other fitness values.

5. Computational results

This section presents the computational results based on the widely used benchmark instances generated by Gendreau et al. (2008). The algorithm was coded in C++, and all experiments were executed on an Intel Xeon E5430 with a 2.66 gigahertz (Quad Core) CPU and 8 gigabytes RAM running the CentOS 5 Linux operating system. The runtime limit for each instance is set according to its scale: 900 seconds for $N \leq 50$, 1800 seconds for $50 < N \leq 100$ and, 3600 seconds for $N > 100$. The detailed computational results presented in this section and the test data are available online at www.computational-logistics.org/orlib/2l-cvrp.

5.1. Benchmark instances

Gendreau et al. (2008) generated the 180 2L-CVRP instances by extending the 36 well-known classical CVRP instances introduced by Toth and Vigo (2002). In particular, each customer is associated with a set of two-dimensional items. In addition, the loading surface (L, W) is fixed as (40, 20) for all instances, and the available vehicle number is specified. According to the characteristics of demanded items, as shown in Table 1, five 2L-CVRP instances are created for each of the 36 CVRP problems. In Class 1, each customer is assigned with one item of unit length and width so that the packing is always feasible. Therefore, Class 1 can be regarded as a pure CVRP problem which is used to evaluate the performance of proposed algorithms in terms of routing aspect. For Classes 2–5, the number of items demanded by customer i, m_i, is a random value in a given interval (see Table 1, column 2). Moreover, each item is classified into one of the three shape categories with equal probability. The *Unrestricted* and *Sequential* versions share the same test data, but *Sequential* 2L-CVRP should consider additional unloading constraint when examining the feasibility of routes.

5.2. Computational results

We executed our approach 10 times for each instance by setting the random seed from 1 to 10. We compare our variable neighborhood search (VNS) with the existing approaches for 2L-CVRP in the literature:

ACO: an ant colony optimization method by Fuellerer et al. (2009);
SA: a simulated annealing method by Leung et al. (2010);
EGTS + LBFH: an extended guided tabu search by Leung et al. (2011);
GRASP × **ELS**: a hybrid procedure combining a greedy randomized adaptive search and evolutionary local search by Duhamel et al. (2011);

Table 1
The characteristics of demanded items of Class 2–5 instances.

Class	m_i	Vertical		Homogeneous		Horizontal	
		Length	Width	Length	Width	Length	Width
2	[1, 2]	[0.4L, 0.9L]	[0.1W, 0.2W]	[0.2L, 0.5L]	[0.2W, 0.5W]	[0.1L, 0.2L]	[0.4W, 0.9W]
3	[1, 3]	[0.3L, 0.8L]	[0.1W, 0.2W]	[0.2L, 0.4L]	[0.2W, 0.4W]	[0.1L, 0.2L]	[0.3W, 0.8W]
4	[1, 4]	[0.2L, 0.7L]	[0.1W, 0.2W]	[0.1L, 0.4L]	[0.1W, 0.4W]	[0.1L, 0.2L]	[0.2W, 0.7W]
5	[1, 5]	[0.1L, 0.6L]	[0.1W, 0.2W]	[0.1L, 0.3L]	[0.1W, 0.3W]	[0.1L, 0.2L]	[0.1W, 0.6W]

Table 2
Computational environments.

Algorithm	CPU	RAM (gigabytes)	Time limit (seconds)
ACO	Pentium IV 3.2 gigahertz	–	10800
SA	Intel 2.4 gigahertz Core Duo	2	–
EGTS + LBFH	Intel Core 2 Duo 2.0 gigahertz	2	–
GRASP × ELS	Opteron 2.1 gigahertz	–	5400
PRMP	Intel Core 2 Duo E6600 2.4 gigahertz	–	–
VNS	Intel Xeon E5430 with a 2.66 gigahertz (Quad Core) CPU	8	3600

Table 3
Comparative results on pure CVRP instances of Class 1.

Inst.	BKS	GRASP × ELS			PRMP			VNS		
		Cost	*t* (seconds)	*Imp*	Cost	*t* (seconds)	*Imp*	Cost	*t* (seconds)	*Imp*
1	278.73	278.73	0.0	0.00	278.73	0.0	0.00	278.73	0.0	0.00
2	334.96	334.96	0.0	0.00	334.96	0.0	0.00	334.96	0.0	0.00
3	358.40	358.40	0.0	0.00	358.40	0.0	0.00	358.40	0.1	0.00
4	430.88	430.88	0.0	0.00	430.88	0.0	0.00	430.89	0.0	0.00
5	375.28	375.28	0.0	0.00	375.28	0.0	0.00	375.28	0.0	0.00
6	495.85	495.85	0.0	0.00	495.85	0.0	0.00	495.85	0.1	0.00
7	568.56	568.56	0.0	0.00	568.56	0.0	0.00	568.56	0.0	0.00
8	568.56	568.56	0.0	0.00	568.56	0.0	0.00	568.56	0.0	0.00
9	607.65	607.65	0.0	0.00	607.65	0.0	0.00	607.65	0.1	0.00
10	**535.74**	535.80	0.0	−0.01	535.80	0.1	−0.01	535.80	0.1	−0.01
11	505.01	505.01	0.0	0.00	505.01	0.0	0.00	505.01	0.0	0.00
12	610.00	610.00	0.2	0.00	610.00	0.2	0.00	610.00	0.9	0.00
13	2006.34	2006.34	0.0	0.00	2006.34	0.3	0.00	2006.34	0.1	0.00
14	837.67	837.67	0.2	0.00	837.67	0.1	0.00	837.67	0.1	0.00
15	837.67	837.67	0.0	0.00	837.67	0.4	0.00	837.67	0.1	0.00
16	698.61	698.61	0.0	0.00	698.61	0.3	0.00	698.61	1.1	0.00
17	861.79	861.79	0.0	0.00	861.79	1.6	0.00	861.79	4.0	0.00
18	723.54	723.54	8.3	0.00	723.54	3.6	0.00	723.54	1.4	0.00
19	524.61	524.61	0.3	0.00	524.61	2.1	0.00	524.61	2.0	0.00
20	241.97	241.97	4.5	0.00	241.97	7.2	0.00	241.97	3.5	0.00
21	687.60	687.60	1.4	0.00	687.60	3.8	0.00	687.60	74.9	0.00
22	740.66	740.66	2.1	0.00	740.66	2.8	0.00	740.66	21.2	0.00
23	835.26	835.26	3391.3	0.00	835.26	48.7	0.00	835.26	159.7	0.00
24	1024.69	1026.60	53.3	−0.19	1024.69	38.1	0.00	1024.69	175.9	0.00
25	826.14	827.39	2.4	−0.15	826.14	8.6	0.00	826.14	332.2	0.00
26	819.56	819.56	0.4	0.00	819.56	11.2	0.00	819.56	1.7	0.00
27	1082.65	1082.65	486.5	0.00	1082.65	172.3	0.00	1082.65	445.5	0.00
28	**1040.70**	1042.12	129.8	−0.14	1042.12	71.2	−0.14	1042.12	1021.5	−0.14
29	1162.96	1162.96	549.6	0.00	1162.96	121.9	0.00	1162.96	172.9	0.00
30	1028.42	1033.42	2165.9	−0.49	1028.42	267.5	0.00	1028.42	1570.0	0.00
31	**1299.56**	1306.07	5096.1	−0.50	**1299.56**	353.8	0.00	1302.48	1813.8	−0.22
32	**1296.91**	1303.52	4492.4	−0.51	**1296.91**	312.0	0.00	1300.22	1976.1	−0.26
33	1299.55	1301.06	4842.1	−0.12	1299.55	434.1	0.00	**1298.02**	2204.1	0.12
34	709.82	713.51	3007.4	−0.52	709.82	328.2	0.00	**708.39**	2125.2	0.20
35	866.06	870.63	2616.5	−0.53	866.06	396.3	0.00	**865.39**	2050.4	0.08
36	**585.46**	592.87	5264.7	−1.27	**585.46**	228.9	0.00	586.49	2420.2	−0.18
Avg.	**769.66**	770.77	892.09	−0.12	769.70	78.20	0.00	769.80	460.53	−0.02

BKS: Best known solution (among SA, EGTS + LBFH, ACO, GRASP × ELS, and PRMP).
Bold entries correspond to higher quality solutions.
t (seconds): Average CPU time to find the best solutions.
Imp: Percentage improvement between the Cost and BKS ($Imp = 100 \times (BKS - Cost)/BKS$).

PRMP: a meta-heuristic named as promise routing-memory packing by Zachariadis et al. (2013).

The computational environments for these approaches are summarized in Table 2. All these approaches were executed 10 times for each instance by setting different random seeds. In the following tables, the cost listed is the best cost achieved over 10 runs. Due to space limitations, we only list the details of the best known solution (BKS) among all previous approaches and two effective methods for each version: **GRASP** × **ELS**, **PRMP** for *Unrestricted* 2L-CVRP and **ACO**, **PRMP** for *Sequential* 2L-CVRP.

5.2.1. Results on pure CVRP benchmark instances

We first test our approach on the pure CVRP benchmark instances of Class 1. Table 3 provides the best results obtained by our method over 10 runs on each instance of Class 1. Our approach managed to produce better value for three instances and robustly matched the best value for 28 instances compared to the best ones obtained by previous approaches. Thus, the routing component of our approach is proved to be rather effective.

5.2.2. Results on Unrestricted 2L-CVRP

Table 4 provides the average and best results obtained by our method on each of the instances of Classes 2–5 for *Unrestricted* 2L-CVRP. We observe that the proposed method is rather stable, as the percentage gaps between the average and best solution achieved over the 10 runs were limited to 0.24 percent, 0.23 percent, 0.23 percent and 0.18 percent, for the problems of Classes 2, 3, 4 and 5, respectively.

Table 4
Results for the *Unrestricted* 2L-CVRP.

Inst.	Class 2				Class 3				Class 4				Class 5			
	C_{avg}	C_{bst}	*gap*	*t* (seconds)	C_{avg}	C_{bst}	*gap*	*t* (seconds)	C_{avg}	C_{bst}	*gap*	*t*(seconds)	C_{avg}	C_{bst}	*gap*	*t*(seconds)
1	278.73	278.73	0.00	0.7	284.52	284.52	0.00	2.5	282.95	282.95	0.00	0.1	278.73	278.73	0.00	1.6
2	334.96	334.96	0.00	0.1	352.16	352.16	0.00	0.2	334.96	334.96	0.00	0.1	334.96	334.96	0.00	0.0
3	387.70	387.70	0.00	0.3	394.72	394.72	0.00	0.8	364.45	364.45	0.00	1.9	358.40	358.40	0.00	0.6
4	430.89	430.89	0.00	0.2	430.89	430.89	0.00	0.1	447.37	447.37	0.00	0.9	430.89	430.89	0.00	0.2
5	375.28	375.28	0.00	0.6	381.69	381.69	0.00	3.4	383.88	383.88	0.00	1.8	375.28	375.28	0.00	0.2
6	495.85	495.85	0.00	0.6	498.16	498.16	0.00	0.9	498.32	498.32	0.00	2.1	495.85	495.85	0.00	0.4
7	725.46	725.46	0.00	1.0	678.75	678.75	0.00	1.8	700.72	700.72	0.00	6.1	657.77	657.77	0.00	1.2
8	674.55	674.55	0.00	3.3	740.12	738.43	0.23	6.7	692.47	692.47	0.00	3.3	609.90	609.90	0.00	1.8
9	607.65	607.65	0.00	0.4	607.65	607.65	0.00	0.5	625.10	625.10	0.00	4.3	607.65	607.65	0.00	0.2
10	689.82	689.82	0.00	20	620.33	620.33	0.00	3.4	710.87	710.87	0.00	3.7	680.51	678.66	0.27	73
11	693.45	693.45	0.00	142	709.49	706.73	0.39	6.8	792.01	784.88	0.90	88	624.82	624.82	0.00	3.5
12	610.57	610.57	0.00	1.7	610.00	610.00	0.00	0.8	614.24	614.24	0.00	6.5	610.03	610.00	0.00	4.2
13	2594.54	2585.72	0.34	66	2468.06	2468.06	0.00	7	2567.84	2548.06	0.77	15	2334.78	2334.78	0.00	13
14	1044.14	1040.74	0.33	502	1003.52	1003.52	0.00	245	981.45	981.00	0.05	6.6	876.22	875.00	0.14	428
15	1018.25	1017.95	0.03	261	1164.35	1155.22	0.78	646	1181.30	1181.30	0.00	49	1160.16	1159.94	0.02	29
16	698.61	698.61	0.00	1.7	698.61	698.61	0.00	1.3	703.35	703.35	0.00	1.9	698.61	698.61	0.00	1.5
17	870.86	870.86	0.00	3.1	861.79	861.79	0.00	2.0	861.79	861.79	0.00	6.5	861.79	861.79	0.00	4.3
18	1004.99	1004.99	0.00	36	1075.56	1070.43	0.48	20	1118.64	1118.57	0.01	119	918.09	917.94	0.02	145
19	760.26	757.44	0.37	255	777.30	773.05	0.55	518	778.18	775.87	0.30	206	648.37	644.59	0.58	23
20	532.17	531.22	0.18	708	522.76	521.31	0.28	702	540.45	539.52	0.17	1045	472.68	471.64	0.22	722
21	999.11	997.63	0.15	507	1120.41	1118.11	0.21	758	974.55	973.03	0.16	849	880.89	877.75	0.36	889
22	1039.12	1035.66	0.33	630	1056.11	1053.76	0.22	1109	1055.00	1054.33	0.06	773	933.79	932.38	0.15	1028
23	1038.00	1035.18	0.27	841	1080.58	1079.78	0.07	811	1072.91	1071.30	0.15	791	936.46	935.33	0.12	969
24	1178.07	1178.07	0.00	638	1081.12	1080.88	0.02	373	1109.68	1108.34	0.12	650	1044.06	1042.83	0.12	627
25	1413.10	1409.24	0.27	1100	1373.13	1369.26	0.28	840	1409.91	1405.85	0.29	1138	1152.57	1150.69	0.16	918
26	1279.27	1272.87	0.50	962	1347.18	1345.45	0.13	1256	1395.76	1394.19	0.11	1183	1217.17	1213.88	0.27	801
27	1319.55	1313.12	0.49	987	1372.89	1370.40	0.18	835	1317.70	1316.19	0.11	870	1247.75	1245.38	0.19	807
28	2581.53	2555.29	1.02	2697	2629.01	2610.56	0.70	2461	2632.80	2602.93	1.13	2096	2312.13	2308.61	0.15	1782
29	2206.84	2201.34	0.25	2630	2090.51	2087.15	0.16	1910	2254.98	2247.23	0.34	2596	2129.55	2128.84	0.03	1795
30	1814.82	1808.77	0.33	2488	1837.79	1830.50	0.40	2416	1836.03	1826.10	0.54	2708	1526.48	1521.91	0.30	2370
31	2274.51	2265.18	0.41	2972	2293.08	2276.14	0.74	2666	2394.29	2382.77	0.48	3169	2001.02	1987.08	0.70	3005
32	2274.18	2255.82	0.81	2447	2250.98	2240.85	0.45	2386	2274.08	2259.83	0.63	2805	1961.34	1954.15	0.37	2957
33	2272.67	2260.49	0.54	2799	2362.61	2348.25	0.61	2738	2384.67	2373.63	0.46	3141	1988.16	1977.25	0.55	3093
34	1184.33	1176.03	0.70	2572	1208.47	1204.58	0.32	2485	1199.41	1193.18	0.52	1860	1022.89	1016.88	0.59	2922
35	1385.19	1373.43	0.85	2681	1454.61	1450.26	0.30	2797	1500.89	1492.98	0.53	2214	1244.70	1237.51	0.58	2791
36	1720.04	1710.26	0.57	2668	1787.29	1776.35	0.61	3147	1665.98	1658.72	0.44	2985	1486.71	1478.68	0.54	3248
Avg.	1133.58	1129.47	0.24	878.4	1145.17	1141.62	0.23	865.4	1157.19	1153.06	0.23	872.1	1031.14	1028.79	0.18	873.8

C_{avg}: Average cost achieved over the 10 runs by **VNS**.
C_{bst}: Best cost achieved over the 10 runs by **VNS**.
gap: Percent gap between average cost and best cost ($gap = 100(C_{avg} - C_{bst})/C_{avg}$).
t (seconds): Average CPU time required for obtaining the best solutions over 10 runs by **VNS**.

Table 5
Comparison for the *Unrestricted* 2L-CVRP (averaged over Classes 2–5).

Inst.	BKS	GRASP × ELS			PRMP			VNS		
		Cost	*t* (seconds)	*Imp*	Cost	*t* (seconds)	*Imp*	Cost	*t* (seconds)	*Imp*
1	281.23	282.65	0.9	−0.50	281.23	0.4	0.00	281.23	1.2	0.00
2	339.26	339.26	0.1	0.00	339.26	0.3	0.00	339.26	0.1	0.00
3	376.32	376.32	0.5	0.00	376.32	0.4	0.00	376.32	0.9	0.00
4	435.00	435.01	0.2	0.00	435.01	0.3	0.00	435.01	0.3	0.00
5	379.03	379.03	0.1	0.00	379.03	1.1	0.00	379.03	1.5	0.00
6	**496.77**	497.04	0.4	−0.05	497.04	0.3	−0.05	497.04	1.0	−0.05
7	690.67	691.11	1.4	−0.06	690.67	1.6	0.00	690.67	2.5	0.00
8	678.84	678.84	0.8	0.00	678.84	2.6	0.00	678.84	3.8	0.00
9	**611.05**	612.01	0.6	−0.16	612.01	1.6	−0.16	612.01	1.3	−0.16
10	675.59	675.79	15.1	−0.03	676.75	26.9	−0.17	**674.92**	25.1	0.10
11	702.96	705.95	11.3	−0.42	703.22	27.2	−0.04	**702.47**	59.9	0.07
12	611.26	611.26	16.9	0.00	611.26	1.4	0.00	**611.20**	3.3	0.01
13	2486.17	2490.62	78.0	−0.18	2491.18	52.7	−0.20	**2484.16**	25.2	0.08
14	**974.04**	984.42	79.9	−1.07	975.88	164.3	−0.19	975.06	295.5	−0.11
15	1128.86	1144.69	257.7	−1.40	1132.91	20.1	−0.36	**1128.60**	246.3	0.02
16	699.79	699.79	6.0	0.00	699.79	4.1	0.00	699.79	1.6	0.00
17	**862.26**	864.05	21.6	−0.21	864.05	2.4	−0.21	864.05	4.0	−0.21
18	1028.61	1029.71	413.5	−0.11	1031.95	33.0	−0.32	**1027.98**	79.8	0.06
19	739.15	739.19	268.5	−0.01	741.78	24.3	−0.36	**737.73**	250.5	0.19
20	**515.44**	522.68	1658.3	−1.41	**515.44**	552.2	0.00	515.92	794.2	−0.09
21	**991.50**	994.58	1450.8	−0.31	992.78	241.5	−0.13	991.63	751.1	−0.01
22	**1017.33**	1021.45	965.0	−0.41	1023.01	166.6	−0.56	1019.03	885.2	−0.17
23	1032.36	1038.16	1373.6	−0.56	1032.36	336.8	0.00	**1030.40**	853.1	0.19
24	**1099.57**	1107.93	480.3	−0.76	1104.64	319.6	−0.46	1102.53	572.1	−0.27
25	1340.18	1345.08	2967.7	−0.37	1341.26	921.7	−0.08	**1333.76**	998.7	0.48
26	1311.79	1317.41	2299.2	−0.43	1311.79	403.5	0.00	**1306.60**	1050.6	0.40
27	1318.04	1323.54	2716.6	−0.42	1318.04	438.2	0.00	**1311.27**	874.5	0.51
28	2530.46	2560.06	5065.5	−1.17	2530.46	3701.9	0.00	**2519.35**	2259.2	0.44
29	2173.02	2191.46	4128.6	−0.85	2173.02	1835.7	0.00	**2166.14**	2232.8	0.32
30	1760.16	1775.44	4753.7	−0.87	1760.59	2151.8	−0.02	**1746.82**	2495.4	0.76
31	2244.13	2282.28	4988.2	−1.70	2244.13	2927.4	0.00	**2227.79**	2952.9	0.73
32	2196.85	2233.27	4900.6	−1.66	2196.85	3713.8	0.00	**2177.66**	2648.7	0.87
33	2261.68	2284.82	4988.9	−1.02	2261.68	1964.8	0.00	**2239.91**	2942.7	0.96
34	1157.22	1191.13	5244.5	−2.93	1157.22	3551.7	0.00	**1147.67**	2459.6	0.83
35	1401.17	1435.22	5015.5	−2.43	1401.17	2756.5	0.00	**1388.55**	2620.6	0.90
36	1669.44	1729.79	4874.0	−3.61	1669.44	4245.6	0.00	**1656.00**	3012.0	0.81
Avg.	1117.14	1127.53	1640.1	−0.70	1118.11	849.8	−0.09	**1113.23**	872.4	0.21

BKS: Best known solution (among SA, EGTS + LBFH, ACO, GRASP × ELS, and PRMP).
Bold entries correspond to higher quality solutions.
t (seconds): Average CPU time to find the best solutions.
Imp: Percentage improvement between the Cost and BKS ($Imp = 100 \times (BKS - Cost)/BKS$).

Table 5 compares the best cost averaged over the four instances (Classes 2–5) of each test problem (problems 1–36) against previous approaches for the *Unrestricted* 2L-CVRP. Our VNS could find better value for 20 problems, compared to the best value obtained by previous approaches, and match the best value for 8 problems.

Table 6 gives the comparisons between the best cost found by VNS and the best cost obtained by previously published approaches on 144 instances of the *Unrestricted* 2L-CVRP. Our VNS could find better value for 65 instances and match the best value for 54 instances, compared to the best solutions obtained by previous approaches.

5.2.3. Results on Sequential 2L-CVRP

Table 7 provides the average and best results obtained by our method on each of the instances of Classes 2–5 for *Sequential* 2L-CVRP. We observe that the percentage gaps between the average and best solution achieved over the 10 runs were limited to 0.48 percent, 0.34 percent, 0.29 percent and 0.35 percent, for the problems of Classes 2, 3, 4 and 5, respectively. Thus, our approach is also rather robust for the *Sequential* 2L-CVRP.

Table 8 compares the best cost averaged over the four instances (Classes 2–5) of each test problem (problems 1–36) against previous approaches for the *Sequential* 2L-CVRP. Our approach could find better value for 25 instances and match the best value for 10 instances, compared to the best solutions obtained by previous approaches.

Table 9 presents the comparison between the best cost found by VNS and the best cost obtained by previously published approaches on 144 instances of the *Sequential* 2L-CVRP. Our VNS could find better value for 88 instances and matches the best value for 54 instances, compared to the best solutions obtained by previous approaches.

In summary, some interesting phenomena can be observed. According to Tables 6 and 9, our VNS algorithm performs better for *Sequential* 2L-CVRP. The VNS method improves the previous best known solutions (BKS) by 0.05–0.53 percent on *Unrestricted* Classes 2, 4 and 5, while the improvement on the *Sequential* classes is at least 0.58 percent. In addition, it is clear that more improvement is obtained on Classes 4 and 5 compared to Classes 2 and 3 for both two problem versions. Furthermore, the proposed VNS performs quite well on large-scale instances, in which almost all BKS are improved for Classes 2–5 of *Unrestricted* problems 25–36 and *Sequential* 18–36. Therefore, our proposed VNS is an effective algorithm for 2L-CVRP.

In terms of computational time, it is difficult to compare the running times of different algorithms fairly because different algorithms were coded in different programming languages and tested on

Table 6
Comparative results on the 144 instances of the *Unrestricted* 2L-CVRP.

Inst.	Class 2			Class 3			Class 4			Class 5		
	BKS	VNS	*Imp*	BKS	VNS	*Imp*	BKS	VNS	*Imp*	BKS	VNS	*Imp*
1	278.73	278.73	0.00	284.52	284.52	0.00	282.95	282.95	0.00	278.73	278.73	0.00
2	334.96	334.96	0.00	352.16	352.16	0.00	334.96	334.96	0.00	334.96	334.96	0.00
3	387.70	387.70	0.00	394.72	394.72	0.00	364.45	364.45	0.00	358.40	358.40	0.00
4	430.88	430.89	0.00	430.88	430.89	0.00	447.37	447.37	0.00	430.88	430.89	0.00
5	375.28	375.28	0.00	381.69	381.69	0.00	383.87	383.88	0.00	375.28	375.28	0.00
6	495.85	495.85	0.00	**497.17**	498.16	−0.20	498.32	498.32	0.00	**495.75**	495.85	−0.02
7	725.46	725.46	0.00	678.75	678.75	0.00	700.72	700.72	0.00	657.77	657.77	0.00
8	674.55	674.55	0.00	738.43	738.43	0.00	692.47	692.47	0.00	609.90	609.90	0.00
9	607.65	607.65	0.00	607.65	607.65	0.00	**621.23**	625.10	−0.62	607.65	607.65	0.00
10	**689.68**	689.82	−0.02	**615.68**	620.33	−0.75	710.87	710.87	0.00	686.12	**678.66**	1.09
11	693.45	693.45	0.00	706.73	706.73	0.00	786.85	**784.88**	0.25	624.82	624.82	0.00
12	610.57	610.57	0.00	610.00	610.00	0.00	614.23	614.24	0.00	610.23	**610.00**	0.04
13	2585.72	2585.72	0.00	**2436.56**	2468.06	−1.29	2587.63	**2548.06**	1.53	2334.78	2334.78	0.00
14	**1038.09**	1040.74	−0.26	**996.25**	1003.52	−0.73	981.00	981.00	0.00	880.80	**875.00**	0.66
15	**1013.29**	1017.95	−0.46	**1154.66**	1155.22	−0.05	1187.30	**1181.30**	0.51	1160.20	**1159.94**	0.02
16	698.61	698.61	0.00	698.61	698.61	0.00	703.35	703.35	0.00	698.61	698.61	0.00
17	**863.66**	870.86	−0.83	861.79	861.79	0.00	861.79	861.79	0.00	861.79	861.79	0.00
18	1004.99	1004.99	0.00	**1069.45**	1070.43	−0.09	1118.71	**1118.57**	0.01	921.29	**917.94**	0.36
19	**754.53**	757.44	−0.39	**771.74**	773.05	−0.17	778.35	**775.87**	0.32	651.97	**644.59**	1.13
20	**525.75**	531.22	−1.04	522.01	**521.31**	0.13	540.58	**539.52**	0.20	473.41	**471.64**	0.37
21	**992.83**	997.63	−0.48	1119.88	**1118.11**	0.16	974.08	**973.03**	0.11	879.21	**877.75**	0.17
22	1035.81	**1035.66**	0.01	**1052.98**	1053.76	−0.07	**1045.91**	1054.33	−0.81	934.60	**932.38**	0.24
23	1035.18	1035.18	0.00	**1079.03**	1079.78	−0.07	1075.36	**1071.30**	0.38	939.88	**935.33**	0.48
24	1178.07	1178.07	0.00	1080.88	1080.88	0.00	1111.27	**1108.34**	0.26	**1028.04**	1042.83	−1.44
25	1412.72	**1409.24**	0.25	1370.27	**1369.26**	0.07	**1405.65**	1405.85	−0.01	1172.08	**1150.69**	1.82
26	1281.51	**1272.87**	0.67	**1344.10**	1345.45	−0.10	1400.38	**1394.19**	0.44	1221.16	**1213.88**	0.60
27	1322.34	**1313.12**	0.70	1377.99	**1370.40**	0.55	1319.87	**1316.19**	0.28	1251.95	**1245.38**	0.52
28	2565.73	**2555.29**	0.41	**2608.27**	2610.56	−0.09	2627.03	**2602.93**	0.92	2320.81	**2308.61**	0.53
29	2211.01	**2201.34**	0.44	2094.60	**2087.15**	0.36	2253.91	**2247.23**	0.30	2132.55	**2128.84**	0.17
30	1816.05	**1808.77**	0.40	1835.79	**1830.50**	0.29	1837.31	**1826.10**	0.61	1551.50	**1521.91**	1.91
31	2279.39	**2265.18**	0.62	2287.11	**2276.14**	0.48	2394.00	**2382.77**	0.47	2016.03	**1987.08**	1.44
32	2272.93	**2255.82**	0.75	2246.74	**2240.85**	0.26	2282.76	**2259.83**	1.00	1984.95	**1954.15**	1.55
33	**2259.29**	2260.49	−0.05	2371.86	**2348.25**	1.00	2408.76	**2373.63**	1.46	2006.82	**1977.25**	1.47
34	1179.76	**1176.03**	0.32	1204.87	**1204.58**	0.02	1209.13	**1193.18**	1.32	1035.13	**1016.88**	1.76
35	1383.93	**1373.43**	0.76	**1446.20**	1450.26	−0.28	1511.99	**1492.98**	1.26	1262.56	**1237.51**	1.98
36	1711.12	**1710.26**	0.05	1781.75	**1776.35**	0.30	1672.44	**1658.72**	0.82	1512.46	**1478.68**	2.23
Avg.	1131.31	**1129.47**	0.05	1141.99	**1141.62**	−0.01	1159.08	**1153.06**	0.31	1036.20	**1028.79**	0.53

BKS: Best known solution (among SA, EGTS + LBFH, ACO, GRASP × ELS, and PRMP).
Bold entries correspond to higher quality solutions.
Cost of VNS is the score of best solution found over 10 runs.
Imp: Percentage improvement between our cost and BKS ($Imp = 100 \times (\text{BKS} - cost_{VNS})/\text{BKS}$).

different machines. However, the time limit for each instance shows that the proposed VNS is rather efficient, as well.

5.3. Convergence analysis

To investigate the convergence behavior of our VNS, we perform an additional experiment on the large-scale instances 31–36 of Class 5. For each instance, we ran VNS one time (the random seed is set to 1) for 3600 CPU seconds. The results are shown in Fig. 5. We plotted the minimum cost found by VNS for each instance as the computational time increases; note that the *x*-axis values increase exponentially. We can see from this figure that our VNS approach exhibits rapid convergence for most of these instances. For the *Unrestricted* 2L-CVRP, our VNS converged after 1024 seconds on all instances. For the *Sequential* 2L-CVRP, our VNS converged after 2048 seconds on all instances, except the instance 36.

5.4. Additional analysis

To investigate the behavior of our VNS, we perform additional experiments on a subset of selected instances. For each class, one out of three instances are selected, i.e., instance 3, 6, ..., 33 and 36. Therefore, 60 instances are selected in total. The following experiments were run on these instances. For each instance, we executed VNS once (the random seed is set to 1).

5.4.1. Improvement achieved by the VNS

Note that our initial solution is constructed via an insertion based heuristic. In order to investigate the improvement achieved by the VNS over this initial solution, we record the cost of the initial solution and the running time of the construction process. The results are shown in Tables 10 and 11. The column *Imp* is the improvement obtained by the VNS, which is calculated as $100(C_{init} - C_{final})/C_{init}$. The results show that the average improvement is larger than 40 percent for all classes of both *Unrestricted* and *Sequential* 2L-CVRP. The average ratio (column *ratio*) of the running time of the construction over the total useful time is about 10 percent for most classes of 2L-CVRP.

5.4.2. Analysis of the skyline heuristic

In order to analyze the effect of the skyline heuristic, we replace the skyline heuristic packing procedure TabuPack with the packing procedure LH_{2SL} used by Gendreau et al. (2008) and LBFH by Leung et al. (2011). All the other parts remain the same. The comparative results are given in Table 12, in which the cost is averaged over the instance with the same id. We can see that the results of VNS are much better than VNS + LH_{2SL} and VNS+LBFH. Combined with the results in

Table 7
Results for the *Sequential* 2L-CVRP.

Inst.	Class 2				Class 3				Class 4				Class 5			
	C_{avg}	C_{bst}	*gap*	*t* (seconds)	C_{avg}	C_{bst}	*gap*	*t* (seconds)	C_{avg}	C_{bst}	*gap*	*t* (seconds)	C_{avg}	C_{bst}	*gap*	*t* (seconds)
1	290.84	290.84	0.00	3.3	286.26	284.52	0.61	11.1	294.25	294.25	0.00	0.5	278.73	278.73	0.00	3.2
2	347.73	347.73	0.00	0.4	352.16	352.16	0.00	1.6	342.00	342.00	0.00	0.2	334.96	334.96	0.00	0.0
3	403.93	403.93	0.00	0.9	394.82	394.72	0.03	1.5	368.56	368.56	0.00	2.0	358.40	358.40	0.00	1.4
4	440.94	440.94	0.00	0.4	441.87	440.68	0.27	0.5	447.37	447.37	0.00	2.6	430.89	430.89	0.00	0.3
5	388.72	388.72	0.00	2.4	381.69	381.69	0.00	9	383.88	383.88	0.00	4.4	375.28	375.28	0.00	0.8
6	499.08	499.08	0.00	0.5	504.68	504.68	0.00	2.5	498.32	498.32	0.00	3.0	495.85	495.85	0.00	0.9
7	734.65	734.65	0.00	1.8	710.17	709.72	0.06	5.2	703.52	703.49	0.00	14	660.37	658.64	0.26	28
8	725.91	725.91	0.00	16	741.12	741.12	0.00	1.9	699.88	697.92	0.28	31	623.39	621.85	0.25	35.9
9	611.49	611.49	0.00	1.2	615.57	613.90	0.27	1.5	625.10	625.10	0.00	11.6	607.65	607.65	0.00	0.6
10	700.20	700.20	0.00	12	642.77	637.38	0.84	123	717.21	715.82	0.19	189	691.01	690.96	0.01	139
11	721.54	721.54	0.00	10	717.44	717.37	0.01	136	821.49	815.68	0.71	41	639.97	636.77	0.50	31
12	619.63	619.63	0.00	5.7	610.00	610.00	0.00	1.6	618.23	618.23	0.00	17	610.23	610.23	0.00	5.8
13	2674.59	2669.39	0.19	50	2497.06	2486.44	0.43	22	2609.69	2609.36	0.01	41	2433.68	2421.88	0.48	101
14	1128.04	1111.94	1.43	488	1087.79	1085.42	0.22	533	987.75	983.20	0.46	183	928.91	924.27	0.50	462
15	1082.84	1041.75	3.80	466	1181.75	1181.68	0.01	69	1248.57	1246.49	0.17	449	1232.36	1230.40	0.16	210
16	698.61	698.61	0.00	2.3	698.61	698.61	0.00	1.6	708.20	708.20	0.00	8.0	698.61	698.61	0.00	1.7
17	870.86	870.86	0.00	3.4	861.79	861.79	0.00	2.3	861.79	861.79	0.00	7.6	861.79	861.79	0.00	4.3
18	1059.14	1053.09	0.57	394	1106.04	1103.45	0.23	251	1135.90	1134.11	0.16	576	927.98	926.53	0.16	365
19	792.55	792.42	0.02	319	801.99	801.13	0.11	243	806.14	801.21	0.61	398	653.81	652.58	0.19	230
20	549.68	547.82	0.34	1102	545.64	541.58	0.74	961	553.46	552.91	0.10	707	480.44	478.73	0.36	917
21	1070.94	1060.72	0.95	941	1154.79	1150.85	0.34	1036	1010.32	1006.21	0.41	973	900.96	893.18	0.86	1062
22	1088.76	1081.44	0.67	1066	1103.42	1094.66	0.79	1264	1095.88	1089.27	0.60	1059	955.40	948.60	0.71	1042
23	1106.33	1093.27	1.18	858	1120.00	1117.54	0.22	1130	1094.41	1093.01	0.13	678	955.20	950.25	0.52	1146
24	1226.69	1222.43	0.35	889	1118.90	1118.44	0.04	897	1143.78	1141.97	0.16	991	1050.82	1048.69	0.20	588
25	1467.56	1458.83	0.59	1183	1441.21	1436.57	0.32	1364	1439.82	1435.18	0.32	1235	1196.05	1183.63	1.04	1442
26	1332.05	1327.47	0.34	1380	1399.85	1396.52	0.24	1005	1452.69	1447.03	0.39	1174	1256.17	1252.65	0.28	1402
27	1377.33	1367.85	0.69	1345	1430.42	1423.74	0.47	1233	1361.97	1357.75	0.31	1116	1280.29	1270.34	0.78	1275
28	2716.25	2699.21	0.63	2690	2802.54	2787.24	0.55	2112	2715.13	2700.66	0.53	1967	2416.43	2399.25	0.71	2924
29	2304.08	2289.84	0.62	2220	2194.22	2172.69	0.98	3057	2318.30	2312.37	0.26	2879	2196.42	2191.69	0.22	2535
30	1890.06	1875.38	0.78	2366	1928.04	1915.42	0.65	2581	1922.51	1910.54	0.62	2659	1583.76	1575.64	0.51	2402
31	2382.25	2369.07	0.55	2390	2380.55	2360.63	0.84	2490	2487.45	2469.40	0.73	2860	2083.96	2072.19	0.56	3303
32	2392.10	2384.29	0.33	2514	2339.61	2325.74	0.59	2873	2372.10	2357.57	0.61	2758	2042.65	2031.92	0.53	2511
33	2393.59	2376.58	0.71	2813	2482.95	2469.85	0.53	2672	2481.91	2470.76	0.45	2570	2067.03	2054.29	0.62	2403
34	1233.27	1226.98	0.51	2619	1260.89	1253.88	0.56	2701	1247.69	1242.26	0.44	3130	1066.56	1062.18	0.41	2853
35	1458.97	1447.30	0.80	3111	1538.44	1529.77	0.56	2701	1572.71	1558.69	0.89	3091	1292.97	1281.90	0.86	3309
36	1805.17	1784.57	1.14	3027	1883.02	1869.38	0.72	3451	1757.04	1740.64	0.93	3444	1563.41	1549.51	0.89	3208
Avg.	1182.95	1175.99	0.48	952.5	1187.72	1182.53	0.34	970.7	1191.81	1187.26	0.29	979.7	1062.01	1057.25	0.35	998.4

C_{avg}: Average cost achieved over the 10 runs by **VNS**.
C_{bst}: Best cost achieved over the 10 runs by **VNS**.
gap: Percent gap between average cost and best cost ($gap = 100(C_{avg} - C_{bst})/C_{avg}$).
t (seconds): Average CPU time required for obtaining the best solutions over 10 runs by **VNS**.

Table 8
Comparison for the *Sequential* 2L-CVRP (averaged over Classes 2–5).

Inst.	BKS	ACO			PRMP			VNS		
		Cost	*t* (seconds)	*Imp*	Cost	*t* (seconds)	*Imp*	Cost	*t* (seconds)	*Imp*
1	287.08	294.48	6.9	−2.58	287.08	1.4	0.00	287.08	4.5	0.00
2	344.21	345.23	0.3	−0.30	344.21	1.0	0.00	344.21	0.5	0.00
3	381.40	381.40	2.9	0.00	381.40	1.3	0.00	381.40	1.4	0.00
4	439.97	441.11	2.9	−0.26	439.97	1.6	0.00	439.97	0.9	0.00
5	382.39	382.39	12.1	0.00	382.39	2.6	0.00	382.39	4.2	0.00
6	499.48	499.48	4.1	0.00	499.48	5.6	0.00	499.48	1.7	0.00
7	702.27	702.27	11.4	0.00	702.27	5.3	0.00	**701.63**	12.2	0.09
8	699.54	711.65	11.9	−1.73	699.54	7.0	0.00	**696.70**	21.3	0.41
9	614.53	614.54	4.8	0.00	615.93	6.2	−0.23	614.53	3.7	0.00
10	**684.00**	697.20	59.2	−1.93	688.48	55.0	−0.66	686.09	115.6	−0.31
11	725.24	728.61	68.8	−0.46	725.83	75.3	−0.08	**722.84**	54.3	0.33
12	614.52	615.77	7.7	−0.20	614.52	7.1	0.00	614.52	7.5	0.00
13	2553.09	2591.76	73.2	−1.51	2554.93	119.6	−0.07	**2546.77**	53.4	0.25
14	1027.38	1042.34	192.6	−1.46	1027.38	637.0	0.00	**1026.21**	416.7	0.11
15	1189.97	1212.94	166.7	−1.93	1189.97	68.1	0.00	**1175.08**	298.5	1.25
16	701.00	701.28	7.8	−0.04	701.00	14.2	0.00	701.00	3.4	0.00
17	864.05	864.06	4.3	0.00	864.05	40.9	0.00	864.05	4.4	0.00
18	1058.00	1068.14	354.1	−0.96	1058.00	95.2	0.00	**1054.29**	396.2	0.35
19	765.64	774.09	146.3	−1.10	766.05	188.3	−0.05	**761.83**	297.4	0.50
20	534.87	541.66	1307.1	−1.27	534.87	1660.9	0.00	**530.26**	921.7	0.86
21	1040.06	1049.25	1277.5	−0.88	1041.77	420.2	−0.16	**1027.74**	1003.0	1.18
22	1066.56	1076.58	894.0	−0.94	1066.56	524.2	0.00	**1053.49**	1107.5	1.23
23	1071.26	1088.05	1482.8	−1.57	1076.19	519.5	−0.46	**1063.52**	953.1	0.72
24	1138.63	1150.80	378.4	−1.07	1139.12	1064.3	−0.04	**1132.88**	841.4	0.50
25	1401.00	1413.29	2988.1	−0.88	1401.00	2319.5	0.00	**1378.55**	1306.1	1.60
26	1370.17	1407.01	2919.7	−2.69	1370.78	1491.2	−0.04	**1355.92**	1240.3	1.04
27	1372.49	1389.46	1873.1	−1.24	1372.49	4163.8	0.00	**1354.92**	1242.5	1.28
28	2668.19	2693.61	10370.6	−0.95	2669.07	8640.1	−0.03	**2646.59**	2423.0	0.81
29	2263.77	2306.16	10016.1	−1.87	2263.76	5484.3	0.00	**2241.65**	2672.8	0.98
30	1849.02	1886.05	10254.4	−2.00	1853.02	4676.9	−0.22	**1819.25**	2502.1	1.61
31	2356.52	2392.08	10508.8	−1.51	2358.26	5845.4	−0.07	**2317.82**	2760.6	1.64
32	2322.71	2364.92	10728.2	−1.82	2322.71	9433.2	0.00	**2274.88**	2664.0	2.06
33	2391.21	2435.22	10668.7	−1.84	2394.32	5662.5	−0.13	**2342.87**	2614.5	2.02
34	1225.53	1251.62	10649.9	−2.13	1225.54	13141.8	0.00	**1196.33**	2825.7	2.38
35	1494.32	1582.32	10657.0	−5.89	1494.32	8989.6	0.00	**1454.42**	3052.9	2.67
36	1762.17	1828.37	10798.8	−3.76	1762.17	10059.6	0.00	**1736.03**	3282.6	1.48
Avg.	1162.84	1181.26	3025.3	−1.30	1163.57	2373.0	−0.06	**1150.76**	975.3	0.75

BKS: Best algorithmic solution (among SA, EGTS + LBFH, ACO, and PRMP).
Bold entries correspond to higher quality solutions.
t (seconds): Average CPU time to find the best solutions.
Imp: Percentage improvement between the Cost and BKS ($Imp = 100 \times (BKS - Cost)/BKS$).

Table 9
Comparative results on the 144 instances of the *Sequential* 2L-CVRP.

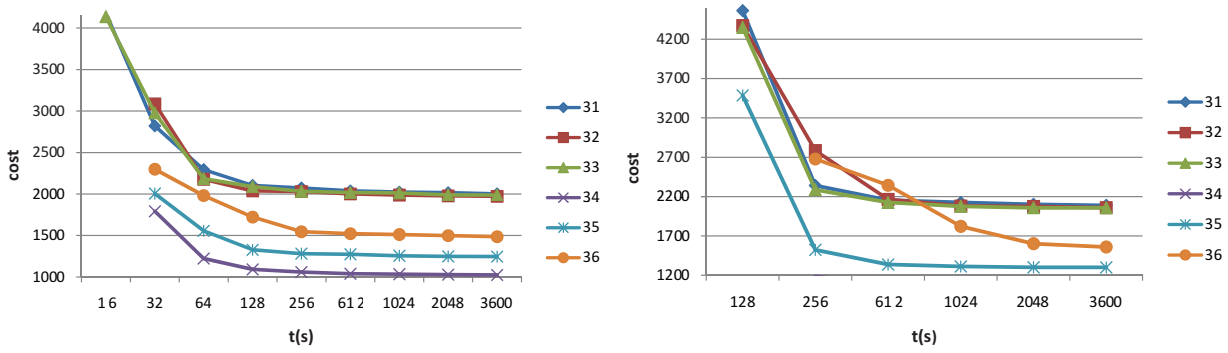
Inst.	Class 2			Class 3			Class 4			Class 5		
	BKS	VNS	*Imp*	BKS	VNS	*Imp*	BKS	VNS	*Imp*	BKS	VNS	*Imp*
1	290.84	290.84	0.00	284.52	284.52	0.00	294.25	294.25	0.00	278.73	278.73	0.00
2	347.73	347.73	0.00	352.16	352.16	0.00	342.00	342.00	0.00	334.96	334.96	0.00
3	403.93	403.93	0.00	394.72	394.72	0.00	368.56	368.56	0.00	358.40	358.40	0.00
4	440.94	440.94	0.00	440.68	440.68	0.00	447.37	447.37	0.00	430.88	430.89	0.00
5	388.72	388.72	0.00	381.69	381.69	0.00	383.87	383.88	0.00	375.28	375.28	0.00
6	499.08	499.08	0.00	504.68	504.68	0.00	498.32	498.32	0.00	495.85	495.85	0.00
7	734.65	734.65	0.00	709.72	709.72	0.00	703.49	703.49	0.00	661.22	**658.64**	0.39
8	725.91	725.91	0.00	741.12	741.12	0.00	697.92	697.92	0.00	633.23	**621.85**	1.80
9	611.49	611.49	0.00	613.90	613.90	0.00	625.10	625.10	0.00	607.65	607.65	0.00
10	700.20	700.20	0.00	**628.94**	637.38	−1.34	715.82	715.82	0.00	691.04	**690.96**	0.01
11	721.54	721.54	0.00	718.09	**717.37**	0.10	815.68	815.68	0.00	645.65	**636.77**	1.37
12	619.63	619.63	0.00	610.00	610.00	0.00	618.23	618.23	0.00	610.23	610.23	0.00
13	2669.39	2669.39	0.00	2497.42	**2486.44**	0.44	2610.57	**2609.36**	0.05	2434.99	**2421.88**	0.54
14	**1101.61**	1111.94	−0.94	1093.08	**1085.42**	0.70	989.60	**983.20**	0.65	925.21	**924.27**	0.10
15	1099.91	**1041.75**	5.29	1181.68	1181.68	0.00	1247.69	**1246.49**	0.10	1230.60	**1230.40**	0.02
16	698.61	698.61	0.00	698.61	698.61	0.00	708.20	708.20	0.00	698.61	698.61	0.00
17	870.86	870.86	0.00	861.79	861.79	0.00	861.79	861.79	0.00	861.79	861.79	0.00
18	1059.44	**1053.09**	0.60	1106.33	**1103.45**	0.26	1137.34	**1134.11**	0.28	928.88	**926.53**	0.25
19	794.15	**792.42**	0.22	801.13	801.13	0.00	807.46	**801.21**	0.77	659.84	**652.58**	1.10
20	549.57	**547.82**	0.32	546.63	**541.58**	0.92	555.59	**552.91**	0.48	487.68	**478.73**	1.84

Continued on next page

Table 9
(*Continued*)

Inst.	Class 2			Class 3			Class 4			Class 5		
	BKS	VNS	*Imp*	BKS	VNS	*Imp*	BKS	VNS	*Imp*	BKS	VNS	*Imp*
21	1070.66	**1060.72**	0.93	1162.07	**1150.85**	0.97	1014.17	**1006.21**	0.78	913.32	**893.18**	2.21
22	1086.25	**1081.44**	0.44	1117.20	**1094.66**	2.02	1101.93	**1089.27**	1.15	960.87	**948.60**	1.28
23	1093.76	**1093.27**	0.04	1121.51	**1117.54**	0.35	1105.73	**1093.01**	1.15	964.02	**950.25**	1.43
24	1222.43	1222.43	0.00	1126.33	**1118.44**	0.70	1150.24	**1141.97**	0.72	1055.51	**1048.69**	0.65
25	1476.34	**1458.83**	1.19	1452.95	**1436.57**	1.13	1467.34	**1435.18**	2.19	1207.36	**1183.63**	1.97
26	1330.94	**1327.47**	0.26	1409.10	**1396.52**	0.89	1468.43	**1447.03**	1.46	1272.21	**1252.65**	1.54
27	1367.87	1367.85	0.00	1449.27	**1423.74**	1.76	1371.88	**1357.75**	1.03	1300.94	**1270.34**	2.35
28	2717.14	**2699.21**	0.66	2796.83	**2787.24**	0.34	2731.04	**2700.66**	1.11	2427.73	**2399.25**	1.17
29	2309.35	**2289.84**	0.84	2201.64	**2172.69**	1.31	2340.96	**2312.37**	1.22	2203.11	**2191.69**	0.52
30	1899.53	**1875.38**	1.27	1947.51	**1915.42**	1.65	1946.53	**1910.54**	1.85	1602.51	**1575.64**	1.68
31	2382.29	**2369.07**	0.55	2413.28	**2360.63**	2.18	2523.98	**2469.40**	2.16	2106.52	**2072.19**	1.63
32	2413.19	**2384.29**	1.20	2386.89	**2325.74**	2.56	2410.15	**2357.57**	2.18	2080.61	**2031.92**	2.34
33	2415.80	**2376.58**	1.62	2522.99	**2469.85**	2.11	2532.13	**2470.76**	2.42	2093.93	**2054.29**	1.89
34	1253.52	**1226.98**	2.12	1278.94	**1253.88**	1.96	1279.67	**1242.26**	2.92	1090.00	**1062.18**	2.55
35	1491.44	**1447.30**	2.96	1566.63	**1529.77**	2.35	1599.02	**1558.69**	2.52	1320.18	**1281.90**	2.90
36	1810.07	**1784.57**	1.41	1889.21	**1869.38**	1.05	1768.88	**1740.64**	1.60	1580.52	**1549.51**	1.96
Avg.	1185.24	**1175.99**	0.58	1194.70	**1182.53**	0.68	1201.14	**1187.26**	0.80	1070.28	**1057.25**	0.99

BKS: Best known solution (among SA, EGTS + LBFH, ACO, and PRMP).
Bold entries correspond to higher quality solutions.
Cost of VNS is the score of best solution found over 10 runs.
Imp: Percent gap between our cost and BKS ($Imp = 100 \times (\text{BKS} - cost_{VNS})/\text{BKS}$).



(a) Unrestricted 2L-CVRP (b) Sequential 2L-CVRP

Fig. 5. The convergence of VNS on instances 31–36 of Class 5.

Table 3, we can conclude that the skyline heuristic makes a significant contribution to the final result in our approach.

Note that when checking the feasibility of a route by TabuPack, we iteratively double the effort spent on the same route until the maximum effort allowed is reached. In our implementation, a parameter *iter* is used to control the effort spent by TabuPack, and *iter* is doubled from 1 until the maximum allowed value n (the number of rectangle) is reached. Another alternative implementation is to set the effort to the maximum effect allowed directly, which means that *iter* is set to n instead of 1 in the first call of TabuPack. Thus, TabuPack is called only once for each route. We modify the VNS to use this alternative implementation and give the results of the modified VNS as VNS1 in Table 12. The results show that VNS is slightly better than VNS1, especially for the large instances (instance 30, 33 and 36) of the *Sequential* version. These instances are the most difficult instances in our test set. Thus, the method of iteratively doubling the effort spent on the same route is very useful for the difficult instances.

5.4.3. Analysis of the number of random points from each neighborhood

One of the differences between our VNS and classical VNS is that up to K (K is the number of vehicles) random points, instead of one, are produced by each neighborhood structure. In order to test the effect of this strategy, we test another three versions by setting the number of random points to 1, $K/2$ and $2K$, respectively. The results are summarized in Table 13. We can see that the results are improved by generating more than one random point, especially for the large instances. However, there is not much difference among the results of generating $K/2$, K, and $2K$ random points. Thus, in our implementation, we set this parameter to K.

5.4.4. Analysis of the diversify procedure

Our diversification procedure DIVERSIFY is used to escape from the incumbent local optimum. To investigate the effect of the diversify procedure, we test another version by removing the procedure DIVERSIFY (Line 16 in Algorithm 1). Moreover, in our implementation of VNS, the number of erased customers is set to $\min(0.5 \times N, 0.1 \times N + nonImp)$. To analyze the sensitivity of this parameter on the final results, we test another two versions by setting this parameter to $N/10$ and $N/2$, respectively. The comparative results are shown in Table 14.

From this table, we can see that the results using diversification are similar to those of without diversification for the *Unrestricted* version. However, for the *Sequential* version, the results are improved after using DIVERSIFY. The results obtained by setting the number of erased customers to the tested three values are similar. Thus, to achieve good performance on different instances, we set the number of erased customers to $\min(0.5 \times N, 0.1 \times N + nonImp)$.

Table 10
The improvement achieved by VNS compared to the initial solution on the *Unrestricted* instances.

Inst.	Class 2						Class 3						Class 4						Class 5					
	C_{init}	C_{final}	*Imp*	t_{init} (seconds)	t (seconds)	*ratio*	C_{init}	C_{final}	*Imp*	t_{init} (seconds)	t (seconds)	*ratio*	C_{init}	C_{final}	*Imp*	t_{init} (seconds)	t (seconds)	*ratio*	C_{init}	C_{final}	*Imp*	t_{init} (seconds)	t (seconds)	*ratio*
3	692.0	403.9	41.6	0.1	0.3	40.9	613.6	394.7	35.7	0.2	0.5	47.3	672.3	368.6	45.2	0.6	2.4	25.4	608.3	358.4	41.1	0.4	2.5	14.1
6	757.6	499.1	34.1	0.1	0.4	15.2	735.1	504.7	31.3	0.5	1.2	37.4	779.8	498.3	36.1	2.3	3.2	71.1	706.2	495.8	29.8	0.0	0.0	50.0
9	876.6	611.5	30.2	0.5	1.2	42.2	892.8	619.5	30.6	0.5	2.1	22.8	931.1	625.1	32.9	0.2	4.4	5.2	932.4	607.7	34.8	0.2	0.3	65.9
12	917.5	619.6	32.5	0.7	4.3	17.2	897.0	610.0	32.0	0.0	3.4	0.6	953.3	618.2	35.1	4.7	28.7	16.5	952.9	610.2	36.0	2.2	7.1	30.7
15	1626.6	1100.5	32.3	0.6	502	0.1	1576.8	1181.7	25.1	0.9	51.0	1.8	1830.7	1249.3	31.8	1.4	590	0.2	1749.5	1234.7	29.4	1.8	473	0.4
18	2116.9	1059.7	49.9	1.4	752	0.2	2365.0	1106.3	53.2	1.7	89.4	1.9	2146.2	1134.1	47.2	2.7	839	0.3	2138.9	928.9	56.6	5.0	193	2.6
21	2318.5	1068.2	53.9	3.0	623	0.5	2330.0	1153.3	50.5	5.3	340	1.6	2137.7	1011.9	52.7	10.2	1224	0.8	2080.4	903.0	56.6	13.2	1385	1.0
24	2445.3	1222.4	50.0	2.0	1359	0.1	2466.4	1118.4	54.7	3.3	710	0.5	2296.1	1143.3	50.2	2.6	573	0.4	2401.6	1051.0	56.2	6.1	1345	0.5
27	3153.9	1381.9	56.2	38.3	753	5.1	3194.8	1423.7	55.4	7.3	1737	0.4	3306.8	1359.9	58.9	12.1	1702	0.7	3177.0	1273.8	59.9	19.6	1706	1.2
30	4651.8	1883.6	59.5	11.8	3378	0.3	4800.5	1923.5	59.9	21.7	2487	0.9	4777.9	1927.3	59.7	35.0	1835	1.9	4107.1	1581.1	61.5	67.2	3566	1.9
33	5657.0	2415.4	57.3	20.1	3365	0.6	6002.2	2481.6	58.7	34.4	3271	1.1	6412.0	2481.3	61.3	44.1	2791	1.6	5522.0	2089.8	62.2	102.4	907	11.3
36	3036.4	1824.1	39.9	40.6	3553	1.1	3319.3	1889.0	43.1	60.0	3570	1.7	3325.4	1772.8	46.7	95.4	3384	2.8	2970.4	1566.4	47.3	192.6	3276	5.9
Avg.	2354.2	1174.2	44.8	9.9	1191	10.3	2432.8	1200.5	44.2	11.3	1022	9.8	2464.1	1182.5	46.5	17.6	1082	10.6	2278.9	1058.4	47.6	34.2	1072	15.4

C_{init}: The cost of the initial solution by the constructive heuristic.
C_{final}: The cost of the final solution found by VNS.
Imp: Improvement rate achieved by VNS ($100(C_{init} - C_{final})/C_{init}$).
t_{init} (seconds): The running time of the constructive heuristic.
t (seconds): Time to find the final solutions.
ratio: The ratio of the running time construction process over the time to find the final solution ($100 \times t_{init}/t$).

Table 11
The improvement achieved by VNS compared to the initial solution on the *Sequential* instances.

Inst.	Class 2						Class 3						Class 4						Class 5					
	C_{init}	C_{final}	*Imp*	t_{init} (seconds)	t (seconds)	*ratio*	C_{init}	C_{final}	*Imp*	t_{init} (seconds)	t (seconds)	*ratio*	C_{init}	C_{final}	*Imp*	t_{init} (seconds)	t (seconds)	*ratio*	C_{init}	C_{final}	*Imp*	t_{init} (seconds)	t (seconds)	*ratio*
3	663.1	395.8	40.3	0.1	0.3	27.5	611.3	394.7	35.4	0.2	1.1	17.0	652.5	366.5	43.8	0.4	1.7	21.6	585.7	358.4	38.8	0.2	1.6	13.0
6	779.1	497.5	36.1	0.1	0.3	19.1	699.7	501.4	28.3	0.3	1.9	13.3	750.6	498.3	33.6	1.3	1.9	69.5	706.2	495.8	29.8	0.0	0.2	8.8
9	845.4	609.6	27.9	0.3	0.8	35.5	852.3	613.6	28.0	0.3	1.1	29.4	952.4	625.1	34.4	0.6	3.1	20.5	941.9	607.7	35.5	0.1	0.1	60.8
12	926.8	615.1	33.6	0.5	3.0	15.5	897.0	610.0	32.0	0.0	2.6	0.8	901.3	616.2	31.6	3.2	22.0	14.5	929.3	610.1	34.3	1.2	5.1	22.5
15	1414.7	1059.2	25.1	0.4	510	0.1	1523.6	1179.1	22.6	0.5	36.8	1.5	1720.7	1215.3	29.4	0.9	322	0.3	1615.0	1197.4	25.9	1.0	252	0.4
18	1938.8	1032.4	46.8	0.8	384	0.2	2175.1	1088.4	50.0	1.0	49.7	2.1	2090.8	1126.4	46.1	1.6	539	0.3	1888.8	923.4	51.1	2.8	104	2.7
21	2161.2	1033.0	52.2	2.0	537	0.4	2223.2	1136.8	48.9	3.2	378	0.8	2017.9	995.0	50.7	5.8	941	0.6	1907.4	892.9	53.2	7.2	1066	0.7
24	2407.5	1200.3	50.1	1.3	891	0.1	2324.6	1099.7	52.7	2.0	414	0.5	2263.9	1126.1	50.3	1.6	533	0.3	2273.7	1046.9	54.0	3.6	1505	0.2
27	2955.8	1353.1	54.2	20.3	735	2.8	3072.7	1403.4	54.3	4.6	1122	0.4	2979.9	1338.4	55.1	7.2	1436	0.5	2912.8	1261.5	56.7	10.6	1206	0.9
30	4339.3	1850.3	57.4	7.3	2857	0.3	4491.8	1883.6	58.1	12.8	1549	0.8	4265.9	1880.3	55.9	19.6	2634	0.7	3893.8	1556.2	60.0	35.5	3543	1.0
33	5319.9	2340.7	56.0	12.3	2967	0.4	5576.0	2426.6	56.5	20.5	2670	0.8	5791.3	2431.1	58.0	25.0	3009	0.8	5200.4	2042.5	60.7	53.9	2136	2.5
36	2905.2	1775.9	38.9	23.9	3543.4	0.7	3123.1	1832.7	41.3	35.1	2793.3	1.3	2991.0	1723.3	42.4	54.0	3017.0	1.8	2753.3	1529.2	44.5	100.9	3421.6	2.9
Avg.	2221.4	1146.9	43.2	5.8	1036	8.6	2297.5	1180.8	42.3	6.7	752	5.7	2281.5	1161.8	44.3	10.1	1038	11.0	2134.0	1043.5	45.4	18.1	1103	9.7

C_{init}: The cost of the initial solution by the constructive heuristic.
C_{final}: The cost of the final solution found by VNS.
Imp: Improvement rate achieved by VNS ($100(C_{init} - C_{final})/C_{init}$).
t_{init} (seconds): The running time of the constructive heuristic.
t (seconds): Time to find the final solutions.
ratio: The ratio of the running time construction process over the time to find the final solution ($100 \times t_{init}/t$).

Table 12
The effect of the skyline heuristic.

Inst.	*Unrestricted*				*Sequential*			
	VNS	VNS1	VNS $+LH_{2SL}$	VNS + LBFH	VNS	VNS1	VNS $+LH_{2SL}$	VNS + LBFH
3	372.74	372.74	372.74	373.57	376.80	376.80	383.43	377.60
6	496.80	496.80	497.27	497.27	498.75	498.75	502.10	499.31
9	611.14	611.14	611.14	611.14	614.27	613.16	618.51	612.96
12	610.96	610.96	613.15	613.15	613.62	613.62	616.81	615.66
15	1074.72	1074.72	1092.74	1108.80	1120.76	1122.14	1138.77	1116.36
18	967.12	969.11	969.88	985.60	990.52	991.88	1011.40	992.51
21	933.32	933.87	945.10	956.50	964.79	968.77	987.01	970.55
24	1087.06	1088.18	1096.10	1104.44	1111.98	1115.10	1131.59	1115.45
27	1271.21	1268.63	1284.26	1298.64	1304.40	1308.64	1337.34	1315.28
30	1611.34	1612.78	1632.69	1653.22	1669.53	1680.58	1717.11	1681.80
33	2063.20	2071.88	2106.94	2130.80	2153.74	2156.26	2211.87	2169.14
36	1452.37	1451.66	1495.30	1505.84	1528.59	1534.78	1568.43	1550.92
Avg.	1046.00	1046.87	1059.77	1069.91	1078.98	1081.71	1102.03	1084.80

VNS: The approach proposed in this paper.
VNS1: Same as VNS, except that the iteration number *iter* in the first call of the TabuPack is set n (the number of rectangles).
VNS $+ LH_{2SL}$: Replace TabuPack by LH_{2SL} used by Gendreau et al. (2008).
VNS + LBFH: Replace TabuPack packing check procedure used by Leung et al. (2011).

Table 13
Analysis of the number of random points from each neighborhood.

Inst.	*Unrestricted*				*Sequential*			
	1	$K/2$	K	$2K$	1	$K/2$	K	$2K$
3	372.74	372.74	372.74	372.74	376.80	376.80	376.80	376.80
6	496.80	496.80	496.80	496.80	498.75	498.75	498.75	498.75
9	611.14	611.14	611.14	611.14	614.27	613.16	614.27	613.16
12	610.96	610.96	610.96	610.96	613.62	613.62	613.62	613.62
15	1073.68	1072.52	1074.72	1073.37	1121.58	1121.00	1120.76	1120.77
18	967.12	967.12	967.12	967.09	991.58	991.01	990.52	991.02
21	936.53	932.74	933.32	933.29	968.19	965.03	964.79	963.46
24	1091.35	1088.01	1087.06	1087.49	1119.63	1116.64	1111.98	1116.29
27	1275.77	1271.80	1271.21	1267.82	1324.37	1310.56	1304.40	1302.00
30	1632.88	1605.79	1611.34	1607.04	1690.31	1675.17	1669.53	1672.11
33	2093.06	2063.25	2063.20	2060.30	2175.35	2142.78	2153.74	2151.91
36	1471.64	1447.82	1452.37	1446.06	1534.48	1526.19	1528.59	1521.82
Avg.	1052.81	1045.06	1046.00	1044.51	1085.75	1079.23	1078.98	1078.48

1: VNS with random point set to 1.
$K/2$: VNS with random point set to $K/2$.
K: VNS with random point set to K.
$2K$: VNS with random point set to $2K$.

Table 14
Analysis of the diversification procedure.

Inst.	*Unrestricted*				*Sequential*			
	Removed	$N/10$	Dynamic	$N/2$	Removed	$N/10$	Dynamic	$N/2$
3	372.74	372.74	372.74	372.74	376.80	376.80	376.80	376.80
6	496.80	496.80	496.80	496.80	498.75	498.75	498.75	498.75
9	611.14	611.14	611.14	611.14	613.16	614.27	614.27	614.27
12	610.96	610.96	610.96	610.96	613.62	613.62	613.62	613.62
15	1072.98	1073.78	1074.72	1074.72	1117.85	1122.11	1120.76	1120.34
18	967.12	967.12	967.12	967.12	992.28	990.52	990.52	990.42
21	936.87	931.83	933.32	933.70	965.25	965.05	964.79	967.04
24	1088.28	1087.06	1087.06	1087.49	1117.15	1112.76	1111.98	1115.48
27	1273.61	1270.00	1271.21	1270.26	1313.59	1304.49	1304.40	1310.44
30	1613.15	1606.77	1611.34	1612.91	1679.38	1668.43	1669.53	1680.23
33	2064.12	2065.52	2063.20	2073.50	2157.20	2149.46	2153.74	2147.03
36	1447.86	1451.54	1452.37	1453.07	1528.90	1526.90	1528.59	1529.42
Avg.	1046.30	1045.44	1046.00	1047.03	1081.16	1078.60	1078.98	1080.32

Removed: VNS removing diversification procedure DIVERSIFY.
$N/10$: VNS with diversification; the number of erased customers is set to $N/10$.
Dynamic: VNS with diversification; the number of erased customers is dynamically set to $\min(N/2, N/10 + nonImp)$.
$N/2$: VNS with diversification; the number of erased customers is set to $N/2$.

6. Conclusions

In this paper, we propose a variable neighborhood search algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints. The variable neighborhood search, which incorporates a diversification procedure to help escape from the local optima, is developed to mainly address the routing aspect. An adapted skyline heuristic is invoked to handle the loading constraints. In addition, some efficient strategies are used to accelerate the procedure. Moreover, an innovative manner of using data structure Trie is introduced not only to record the loading feasibility information of routes already examined, but also to control the effort of examining the loading constraints. Numerical experiments show that the proposed VNS method is superior to all previously published algorithms, and it could find better value for 65 instances of *Unrestricted* 2L-CVRP and 88 instances for *Sequential* 2L-CVRP out of 144 instances. Furthermore, the computational time is also competitive. Thus, the proposed VNS is quite robust, effective and efficient for this problem.

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China (Grant no. 71401065 and 61272003).

References

Bortfeldt, A. (2012). A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints. *Computers & Operations Research*, *39*(9), 2248–2257. 10.1016/j.cor.2011.11.008

Duhamel, C., Lacomme, P., Quilliot, A., & Toussaint, H. (2011). A multi-start evolutionary local search for the two-dimensional loading capacitated vehicle routing problem. *Computers & Operations Research*, *38*(3), 617–640. 10.1016/j.cor.2010.08.017

Fleszar, K., Osman, I. H., & Hindi, K. S. (2009). A variable neighbourhood search algorithm for the open vehicle routing problem. *European Journal of Operational Research*, *195*(3), 803–809. 10.1016/j.ejor.2007.06.064

Fuellerer, G., Doerner, K. F., Hartl, R. F., & Iori, M. (2009). Ant colony optimization for the two-dimensional loading vehicle routing problem. *Computers & Operations Research*, *36*(3), 655–673. 10.1016/j.cor.2007.10.021

Fuellerer, G., Doerner, K. F., Hartl, R. F., & Iori, M. (2010). Metaheuristics for vehicle routing problems with three-dimensional loading constraints. *European Journal of Operational Research*, *201*(3), 751–759. 10.1016/j.ejor.2009.03.046

Gendreau, M., Hertz, A., & Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, *40*(10), 1276–1290.

Gendreau, M., Iori, M., Laporte, G., & Martello, S. (2006). A tabu search algorithm for a routing and container loading problem. *Transportation Science*, *40*(3), 342–350. 10.1287/trsc.1050.0145

Gendreau, M., Iori, M., Laporte, G., & Martello, S. (2008). A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks*, *51*(1), 4–18. 10.1002/net.20192

Hansen, P., Mladenović, N., Brimberg, J., & Pérez, J. A. M. (2010). Variable neighborhood search. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of metaheuristics* (Vol. *146*, pp. 61–86). Springer US. 10.1007/978-1-4419-1665-5_3

Hemmelmayr, V. C., Doerner, K. F., & Hartl, R. F. (2009). A variable neighborhood search heuristic for periodic routing problems. *European Journal of Operational Research*, *195*(3), 791–802. 10.1016/j.ejor.2007.08.048

Imran, A., Salhi, S., & Wassan, N. A. (2009). A variable neighborhood-based heuristic for the heterogeneous fleet vehicle routing problem. *European Journal of Operational Research*, *197*(2), 509–518. 10.1016/j.ejor.2008.07.022

Iori, M., & Martello, S. (2010). Routing problems with loading constraints. *TOP*, *18*(1), 4–27. 10.1007/s11750-010-0144-x

Iori, M., Salazar-González, J.-J., & Vigo, D. (2007). An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation Science*, *41*(2), 253–264. 10.1287/trsc.1060.0165

Lacomme, P., Toussaint, H., & Duhamel, C. (2013). A GRASP × ELS for the vehicle routing problem with basic three-dimensional loading constraints. *Engineering Applications of Artificial Intelligence*, *26*(8), 1795–1810.

Leung, S., Zheng, J., Zhang, D., & Zhou, X. (2010). Simulated annealing for the vehicle routing problem with two-dimensional loading constraints. *Flexible Services and Manufacturing Journal*, *22*(1), 61–82. 10.1007/s10696-010-9061-4

Leung, S. C. H., Zhang, Z., Zhang, D., Hua, X., & Lim, M. K. (2013). A meta-heuristic algorithm for heterogeneous fleet vehicle routing problems with two-dimensional loading constraints. *European Journal of Operational Research*, *225*(2), 199–210. 10.1016/j.ejor.2012.09.023

Leung, S. C. H., Zhou, X., Zhang, D., & Zheng, J. (2011). Extended guided tabu search and a new packing algorithm for the two-dimensional loading vehicle routing problem. *Computers & Operations Research*, *38*(1), 205–215. 10.1016/j.cor.2010.04.013

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.

Lodi, A., Martello, S., & Monaci, M. (2002). Two-dimensional packing problems: A survey. *European Journal of Operational Research*, *141*(2), 241–252. 10.1016/S0377-2217(02)00123-6

Lodi, A., Martello, S., & Vigo, D. (1999). Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, *11*(4), 345–357. 10.1287/ijoc.11.4.345

Ma, H.-w., Zhu, W., & Xu, S. (2011). Research on the algorithm for 3L-CVRP with considering the utilization rate of vehicles. In R. Chen (Ed.), *Intelligent computing and information science* (Vol. *134*, pp. 621–629). Springer Berlin Heidelberg. 10.1007/978-3-642-18129-0_94

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097–1100. 10.1016/s0305-0548(97)00031-2

Paraskevopoulos, D. C., Repoussis, P. P., Tarantilis, C. D., Ioannou, G., & Prastacos, G. P. (2007). A reactive variable neighborhood tabu search for the heterogeneous fleet vehicle routing problem with time windows. *Journal of Heuristics*, *14*(5), 425–455. 10.1007/s10732-007-9045-z

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, *31*(12), 1985–2002. 10.1016/s0305-0548(03)00158-8

Ropke, S., & Pisinger, D. (2006). A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, *171*(3), 750–775. Feature Cluster: Heuristic and Stochastic Methods in Optimization Feature Cluster: New Opportunities for Operations Research. http://dx.doi.org/10.1016/j.ejor.2004.09.004

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., & Dueck, G. (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, *159*(2), 139–171. http://dx.doi.org/10.1006/jcph.1999.6413

Tarantilis, C. D., Zachariadis, E. E., & Kiranoudis, C. T. (2009). A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem. *IEEE Transactions on Intelligent Transportation Systems*, *10*(2), 255–271.

Toth, P., & Vigo, D. (2002). The vehicle routing problem. SIAM monographs on discrete mathematics and applications. Philadelphia: Society for Industrial and Applied Mathematics.

Wei, L., Oon, W.-C., Zhu, W., & Lim, A. (2011). A skyline heuristic for the 2d rectangular packing and strip packing problems. *European Journal of Operational Research*, *215*, 337–346. 10.1016/j.ejor.2011.06.022

Wei, L., Zhang, Z., & Lim, A. (2014). An adaptive variable neighborhood search for a heterogeneous fleet vehicle routing problem with three-dimensional loading constraints. *IEEE Computational Intelligence Magazine*, *9*(4), 18–30.

Zachariadis, E. E., Tarantilis, C. D., & Kiranoudis, C. T. (2009). A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, *195*(3), 729–743. 10.1016/j.ejor.2007.05.058

Zachariadis, E. E., Tarantilis, C. D., & Kiranoudis, C. T. (2013). Integrated distribution and loading planning via a compact metaheuristic algorithm. *European Journal of Operational Research*, *228*(1), 56–71. 10.1016/j.ejor.2013.01.040

Zhu, W., Qin, H., Lim, A., & Wang, L. (2012). A two-stage tabu search algorithm with enhanced packing heuristics for the 3l-cvrp and m3l-cvrp. *Computers & Operations Research*, *39*(9), 2178–2195. 10.1016/j.cor.2011.11.001