European Journal of Operational Research 265 (2018) 843–859

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints

Lijun Wei[a], Zhenzhen Zhang[b,e,*], Defu Zhang[c], Stephen C. H. Leung[d]

[a] School of Information Technology, Jiangxi University of Finance and Economics, Nanchang 330013, Jiangxi, China
[b] Department of Industrial Systems Engineering and Management, National University of Singapore, 1 Engineering Drive 2, Singapore 117576, Singapore
[c] Department of Computer Science, Xiamen University, Xiamen 361005, China
[d] Faculty of Engineering, The University of Hong Kong, Pokfulam, Hong Kong
[e] City University of Hong Kong Shenzhen Research Institute (CityUSRI), Shenzhen, China

ARTICLE INFO

Article history:
Received 7 June 2016
Accepted 20 August 2017
Available online 25 August 2017

Keywords:
Routing
Packing
Simulated annealing
2L-CVRP
Rotation

ABSTRACT

This paper studies the well-known capacitated vehicle routing problem with two-dimensional loading constraints (2L-CVRP). It requires designing a set of min-cost routes, starting and terminating at the central depot, to satisfy customer demands which involve a set of two-dimensional, rectangular, weighted items. A simulated annealing algorithm with a mechanism of repeatedly cooling and rising the temperature is proposed to solve the four versions of this problem, with or without the *LIFO* constraint, and allowing rotation of goods or not. An open space based heuristic is employed to identify the feasible loading patterns. In addition, the data structure Trie is used to accelerate the procedure by keeping track of the packing feasibility information of routes examined, and also by controlling the effort spent on different routes. The proposed algorithm is tested on the widely used instances of 2L-CVRP. The results show that our approach outperforms all existing algorithms on the four problem versions, and reaches or improves the best-known solutions for most instances. Furthermore, we compared the impact of different loading constraints, and observed some interesting results.

1. Introduction

The vehicle routing problem (VRP) is an NP-hard classic combinatorial optimization problem (Toth & Vigo, 2014). It has become a key component of distribution and logistics management. Since Dantzig and Ramser (1959), extensive research related to VRP has been published. Several important variants of this basic problem, with a large number of constraints, have been extensively studied. The constraints make the resultant configurations of problems align better with real-life applications. For example, the capacitated vehicle routing problem with two-dimensional loading constraints (2L-CVRP) considers the routing and loading aspects simultaneously, in order to provide a more practical plan for transportation. In many situations such as the distribution of household appliances, the cargoes have a large shape, then the loading of goods into vehicles becomes a tighter constraint compared to the weight capacity.

In the 2L-CVRP, the customer demands are formed by a set of two-dimensional, rectangular, weighted items. The objective is to design a set of min-cost routes, starting and terminating at the central depot, to satisfy the entire demands. This problem is very important in theoretical as well as practical contexts. It is a highly complex NP-hard problem composed of two NP-hard problems, namely, the capacitated vehicle routing problem (CVRP) and two-dimensional bin packing problem (2BPP). The goods required by the customers, in practice, can often be regarded as rectangular shaped items that cannot be stacked on each other due to their weight, fragility or height (e.g., household appliances, delicate pieces of furniture, etc.). Thus, numerous real-world distribution activities fit into the 2L-CVRP framework. Clearly, the 2L-CVRP can provide a more practical solution and help reduce a significant share of the overall operational expenses. Therefore, an efficient method to solve this problem can reduce costs significantly for enterprises.

However, it is not efficient to simply combine the conventional algorithms for CVRP and 2BPP to solve this problem because the procedure needs to invoke the packing algorithm frequently in order to obtain packing-feasible solution. Precisely, the total number of packing sub-problems to be examined is equal to the cardinality of routes in all candidate solutions. If a sophisticated packing

* Corresponding author at: Department of Industrial Systems Engineering and Management, National University of Singapore, 1 Engineering Drive 2, 117576 Singapore, Singapore.
E-mail address: zhenzhenzhang222@gmail.com (Z. Zhang).

http://dx.doi.org/10.1016/j.ejor.2017.08.035
0377-2217/© 2017 Elsevier B.V. All rights reserved.

algorithm is used, much time is spent on the packing component and then the procedure can explore only a limited solution space. If a simple packing method is chosen, a feasible solution might be classified as infeasible and discarded. In both cases, it would not lead to a good final solution. Thus, it is worth designing algorithms and strategies for this problem exclusively.

In this paper, a simulated annealing method with repeatedly cooling and rising of temperature is proposed, in which a good strategy is developed to control the effort spent on different routes. A two-phase efficient heuristic method is employed to construct a feasible initial solution of high quality. For the search process, four neighborhood structures are defined and employed with probability. Moreover, an open space based heuristic is employed to check the loading feasibility. In this study, our method searches only in the feasible solution space. It means that a newly generated solution can be accepted only if it is examined as feasible in terms of capacity and packing. Moreover, the data structure Trie is used to record the feasibility information of routes already examined, which dramatically speeds the total procedure. However, the random packing procedure may be invoked for several times on the same route, if the route is regarded as infeasible temporally. Here, our Trie can also help control the effort spent on the same route. In order to investigate the performance of the proposed methodology, it was tested on the widely used benchmark instances of the 2L-CVRP and was found to either improve or match the best-known solutions for almost all instances. In addition, although allowing the rotation of goods is frequently encountered, there are only two papers considering this case. Our study tries to close this gap.

The remainder of this paper is organized as follows. After this introductory section, the relevant literature is reviewed in Section 2. Section 3 gives a detailed description of the addressed problem. Section 4 describes the open space based packing heuristic used to check the packing feasibility of routes. Section 5 describes Trie structure used to accelerate the packing procedure. Section 6 presents the proposed simulated annealing meta-heuristic for the 2L-CVRP. Computational results and comparisons on the 2L-CVRP instances are described in Section 7. The paper is concluded in Section 8.

2. Literature review

The 2L-CVRP, first studied by Iori, Salazar-González, and Vigo (2007), is a variant of VRP that combines the capacitated vehicle routing problem (CVRP) with two-dimensional loading constraints. Iori et al. (2007) employed an exact algorithm which uses the branch-and-cut technique to handle the routing aspect of the problem. To check the feasibility of the loadings, a nested branch-and-bound procedure is applied iteratively. The algorithm can solve instances involving up to 30 clients and 90 items. Later, Gendreau, Iori, Laporte, and Martello (2008) proposed a tabu search to solve large scale problems. The guided tabu search, which combines tabu search with guided local search, was developed by Zachariadis, Tarantilis, and Kiranoudis (2009). Another effective tabu search integrated with an extended guided local search and a new packing heuristic was developed by Leung, Zhou, Zhang, and Zheng (2011). A nature inspired meta-heuristic algorithm based on ant colony optimization has been proposed by Fuellerer, Doerner, Hartl, and Iori (2009). Leung, Zheng, Zhang, and Zhou (2010) proposed a simulated annealing approach. Duhamel, Lacomme, Quilliot, and Toussaint (2011) proposed a GRASPxELS algorithm for the 2L-CVRP, however, only the pure CVRP and *unrestricted* 2L-CVRP were solved. Recently, Zachariadis, Tarantilis, and Kiranoudis (2013) provided a promise-based tabu search integrated with an enhanced packing heuristic and obtained excellent results that improve loads of the best-known solutions. Wei, Zhang, Zhang, and Lim (2015) proposed a variable neighborhood search which outperformed all existing approaches. Dominguez, Juan, and Faulin (2014) developed a biased-randomized algorithm, and considered the results of allowing the rotation of items. Moreover, Leung, Zhang, Zhang, Hua, and Lim (2013) proposed a simulated annealing algorithm combined with local search to solve a more generalized problem with heterogeneous vehicles (2L-HFVRP), whose results were improved later by Dominguez, Juan, Barrios, Faulin, and Agustin (2016).

A closely related problem is the capacitated vehicle routing problem with three-dimensional loading constraints (3L-CVRP) which considers the distribution of three-dimensional items. Most authors extended their algorithmic framework proposed for the 2L-CVRP to the 3L-CVRP by introducing the three-dimensional packing methods, for example tabu search (Gendreau, Iori, Laporte, & Martello, 2006), guided tabu search (Tarantilis, Zachariadis, & Kiranoudis, 2009), ant colony algorithm (Fuellerer, Doerner, Hartl, & Iori, 2010) and GRASPxELS (Lacomme, Toussaint, & Duhamel, 2013). Moreover, Bortfeldt (2012), Zhu, Qin, Lim, and Wang (2012) and Tao and Wang (2015) proposed the tabu search with different packing procedures. Ruan, Zhang, Miao, and Shen (2013) developed a honey bee mating optimization for the 3L-CVRP. Later, Wei, Zhang, and Lim (2014) studied the heterogeneous fleet vehicle routing problem with three-dimensional loading constraints. Their algorithm solves both 3L-HFVRP and 3L-CVRP, and outperforms previous algorithms on 3L-CVRP instances. Recently, Zhang, Wei, and Lim (2015) proposed an evolutionary local search to minimize fuel consumption under three-dimensional loading constraints, and improves the best-known solutions for lots of 3L-CVRP instances. In addition, surveys on routing problems with loading constraints were conducted by Iori and Martello (2010), Oliveira (2010), and Pollaris, Braekers, Caris, Janssens, and Limbourg (2015).

The loading component of the 2L-CVRP is closely related to the two-dimensional bin packing problem (2BPP), one of the combinatorial optimization problems having many industrial applications in different fields of operations research. For example, in shipping and transportation industries, packages of identical height have to be packed into bins and the objective is to minimize the number of bins. In addition, many practical problems can either be reduced to the BPP or contain the BPP as a sub-problem. For more reviews, readers are referred to Martello and Vigo (1998), Wei, Oon, Zhu, and Lim (2013) and Zhang, Wei, Leung, and Chen (2013).

3. Problem description

Let n be the number of customers, and $N_0 = N \cup \{0\} = \{0, 1, \ldots, n\}$, where N is the set of customers and 0 denotes the depot. A completely undirected graph $G = (N_0, E)$ is given, where E is the set of edges. For each edge, the associated travel cost c_{ij} is defined, which corresponds to the distance from i to j. In the central depot, there are a fixed number of identical vehicles, each having a weight capacity D and a rectangular loading surface of length L and width W. Each customer i $(i = 1, 2, \ldots, n)$ demands a set of m_i rectangular items, denoted as the set IT_i. Each item $I_{ir} \in IT_i$ $(r = 1, 2, \ldots, m_i)$ has a specific length l_{ir} and width w_{ir}. In addition, we denote $a_i = \sum_{i=1}^{m_i} w_{ir} l_{ir}$ as the total surface area of all items required by the customer i. The total weight of the items in set IT_i is equal to d_i. In the 2L-CVRP, the following constraints are imposed:

(a) The items must be placed on the loading surface orthogonally, namely, their l-edges and w-edges must be parallel to the edges of the vehicle surfaces.
(b) All items in IT_i, demanded by a given customer i, must be loaded on the same vehicle. In other words, every customer can be served only once.
(c) The total weight of items loaded in a vehicle must not exceed the capacity D of the vehicle.

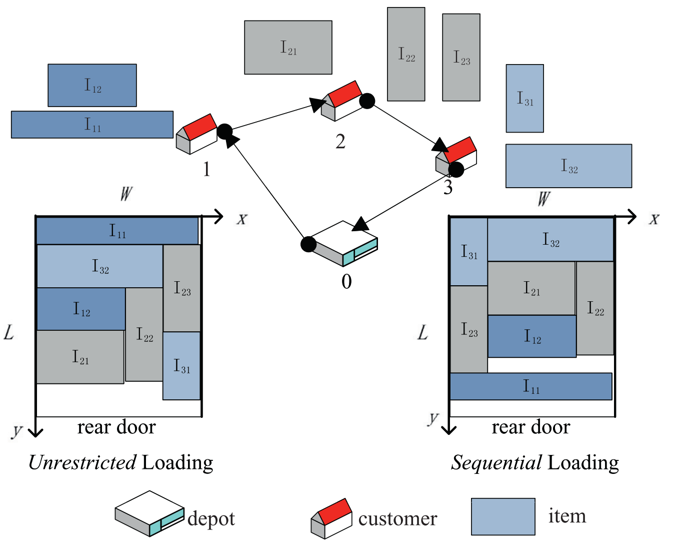


Fig. 1. An example of 2|UO|L and 2|SO|L 2L-CVRP.

(d) All items in a vehicle must be loaded on the surface properly. Overlapping between items in each vehicle is not permitted.
(e) Every vehicle starts from and returns to the central depot.

The objective of the 2L-CVRP is to minimize the total travel cost of vehicles. According to the operational requirements, some extra constraints may be introduced. The rotation (90 °) of items might be allowed, or not. The latter case is also called *oriented* loading, in which the $l-$ and $w-$edges of each item should be parallel to the $L-$ and $W-$edges of the loading surface, respectively. Considering the convenience of unloading, the *sequential* (also referred as *LIFO*) constraint is imposed, which is a more practical constraint: when a customer i is visited, all items of customer i must be unloaded by employing straight movements, parallel to the length of the vehicle, without moving items of other customers. This constraint is very useful when the unloading is performed by forklift trucks. The version without *LIFO* constraint is called *unrestricted* loading. According to the classification scheme suggested by Fuellerer et al. (2009), four versions can be distinguished as follows.

2|UO|L: *unrestricted*, *orientated* loading;
2|UR|L: *unrestricted*, *rotation* allowed loading;
2|SO|L: *sequential*, *orientated* loading;
2|SR|L: *sequential*, *rotation* allowed loading.

Fig. 1 gives an example of 2|UO|L and 2|SO|L, where the loading patterns are quite different due to the *LIFO* constraint.

4. The open space based local search heuristic for the loading subproblem

Before introducing the heuristic for the loading subproblem, we first describe our method used to represent the packing pattern.

4.1. Open space representation of a packing pattern

During the packing procedure, the packing pattern can be represented by the packed region (the dark region in Fig. 2) and the unpacked region (the non-dark region in Fig. 2); the packed region is the area occupied by the packed rectangles. A *free space* is a rectangular space that does not contain any packed area (M_1, M_2 and M_3). Thus, the unpacked region can be represented by the union of a set of free spaces. An *open space* is a special type of free

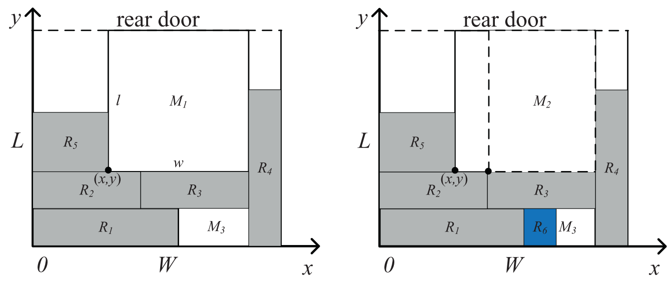


Fig. 2. An example of open space, M_1 and M_2 are open spaces while M_3 is not.

space that has one side coinciding with the rear door of the vehicle. Thus in Fig. 2, M_1 and M_2 are open spaces while M_3 is not. An open space M is described by the following four attributes:

- $M.x$, $M.y$: the coordinates of the bottom-left corner of M; and
- $M.w$, $M.l$: the width and length of M

Given two open spaces M_1 and M_2, for the *unrestricted* version, M_2 is called dominated by M_1 if M_2 is contained in M_1, i.e.,
$(M_1.x \le M_2.x < M_2.x + M_2.w \le M_1.x + M_1.w) \wedge (M_1.y \le M_2.y < M_2.y + M_2.l \le M_1.y + M_1.l)$.

For the *sequential* version, M_2 is dominated by M_1 if M_2 and M_1 share the same x position and M_2 is contained in M_1, i.e.,
$(M_1.x = M_2.x < M_2.x + M_2.w \le M_1.x + M_1.w) \wedge (M_1.y \le M_2.y < M_2.y + M_2.l \le M_1.y + M_1.l)$.

Fig. 2 gives an example of open space. M_2 is dominated by M_1 for the *unrestricted* case, but it is not dominated by M_1 for the *sequential* case. This is because in the *sequential* version, a rectangle r may violate the *sequential* constraint if placed at M_1 (a rectangle is placed at the bottom-left corner of an open space) but it may be feasible if r is placed at M_2.

An *open space* not dominated by other open spaces is called *Maximal Open Space* (MOS). The MOS is similar to *Residual Space* (RS) used by Lai and Chan (1997) and Wang and Chen (2015). The *residual space* is the largest rectangular area that can be obtained in an unpacked region. However, there are two main differences between the MOS and RS. First, the former must have one side coinciding with the rear door of the vehicle while there is no such constraint for the latter. Thus, the hole M_3 in Fig. 2 is not regarded as a MOS in our algorithm while it is a RS. M_3 is regarded as a waste area in our algorithm. Second, the domination rule is different for the *sequential* case. M_2 in Fig. 2 is not regarded as RS while it is MOS in the *sequential* case.

Note that in Fig. 2, M_3 is regarded as a waste area although there maybe some unpacked rectangle that can be placed at M_3. However, this rule will not lose solution quality as we can get the solution in which this waste area is used by swapping the packed order of the rectangles. In Fig. 2, assume the next to be packed rectangle is R_6 which can be placed at M_3, but M_3 is regarded as a waste area so it will not be considered as a candidate position. However, if we change the packed order of the rectangles to R_1, R_6, R_2, R_3, R_4 and R_5, then we can get the packing pattern with R_6 placed at M_3.

During the packing process, a list of MOS is maintained. The initial empty vehicle's surface is represented by a MOS located at the point (0, 0) with size $W \times L$. At each step after a rectangle r is placed at a MOS M, the MOS list is updated in two steps. In the first step, M is removed from the list and two new MOS are introduced. These two new MOS are generated by selecting three edges (that must include the rear door) from M and then selecting one edge from r to form a rectangle. Fig. 3 shows how the new MOS are generated. M_1 is generated by selecting the left, top (rear door) and right edge of M and then selecting the top edge of r. M_2

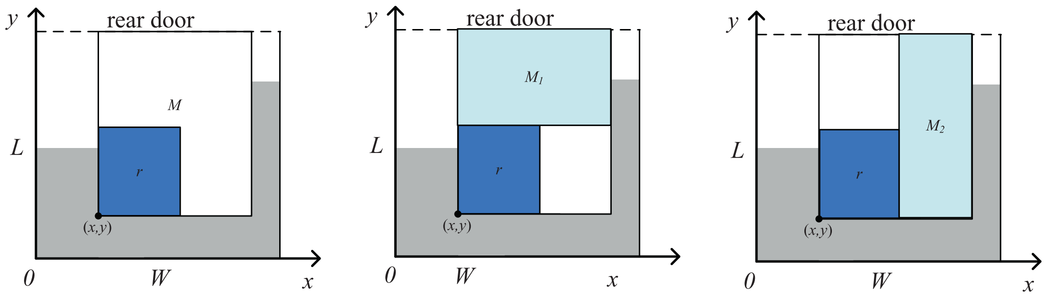


Fig. 3. Generate new open spaces in step 1.

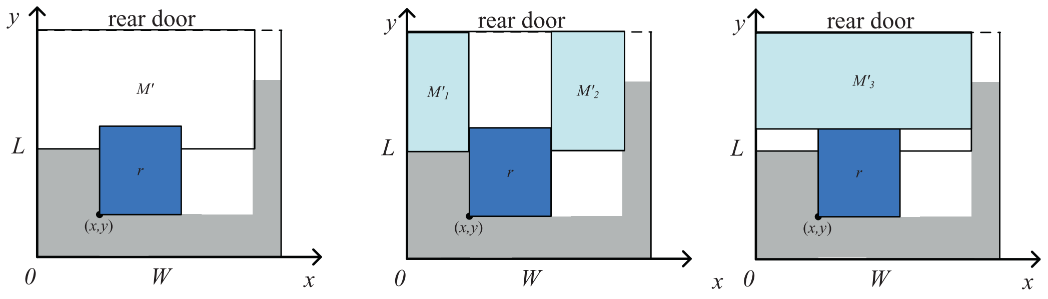


Fig. 4. Generate new open spaces in step 2.

is generated by selecting the top, right and bottom edge of M and then selecting the right edge of r.

In the second step, for any space M' overlapping with r, M' is first removed from the list and then up to three new MOS are generated. All these new MOS can be generated by selecting three edges (must include the rear door) from M', and then selecting one edge from r to form a rectangle that does not overlap with r. As shown in Fig. 4, space M' overlaps with r and three new MOS, M'_1, M'_2 and M'_3 are generated.

Before a new MOS is added into the list, it is compared with all MOS already in the list one by one. The newly generated MOS is added into the list if and only if it is not dominated by any other MOS. In Figs. 3 and 4, if it is the *unrestricted* case, M_1 and M'_2 will not be inserted into the list as M_1 is dominated by M'_3 and M'_2 is dominated by M_2. So the number of MOS after placing r is 3, which is M_2, M'_1 and M'_3. However, if it is the *sequential* case, M_1 will be inserted into the list as M_1 is not dominated by M'_3 in this case. So the number of MOS after placing r is 4, which is M_1, M_2, M'_1 and M'_3.

Algorithm 1. RandomLS procedure.

```
RandomLS(rt)
1  R = All rectangles demanded by the customers in rt
2  iter = number of rectangles in R
3  totalArea = the area of all rectangles in R
4  for each sorting rule
5      Sort R using the sorting rule
6      packedArea = HeuristicPack(R)
7      if packedArea == totalArea
8          return SUCCESS
9      for noImproved = 1 to iter
10         Generate sequence R′ from R by randomly swapping two rectangles
11         packedArea′ = HeuristicPack(R′)
12         if packedArea′ > packedArea
13             noImproved = 1
14             packedArea = packedArea′
15             if packedArea == totalArea
16                 return SUCCESS
17         if packedArea′ ≥ packedArea
18             R = R′
19 return FAILURE
```

4.2. Random local search heuristic for the loading subproblem

To examine the packing feasibility of newly generated routes, we develop a random local search RandomLS (Algorithm 1) which takes a route rt as input. A route consists of the ordered customers to be visited by a vehicle. If a feasible packing for this vehicle is found, a constant value SUCCESS is returned; otherwise, a constant value FAILURE is reported.

RandomLS is a sequence based random local search procedure. Given a sequence of ordered rectangles R, a packing heuristic subroutine HeuristicPack (Section 4.3) is used to pack as many rectangles into the vehicle as possible. The result returned by HeuristicPack is the total area of the packed rectangles. We first use three sorting rules to generate an initial sequence (line 5). The HeuristicPack is called on each sorted sequence. If the result returned by HeuristicPack is equal to the total area, a feasible packing pattern that places all the rectangles in R is found and then the constant value SUCCESS is returned directly. Otherwise, a random local search is used to optimize the solution in terms of area utilization. The number of iterations is controlled by the parameter *iter*, which is set to the number of rectangles in R. At each iteration, a new sequence R' is generated by swapping two randomly selected rectangles in R and HeuristicPack is called on R'. Note that even for the *sequential* version, these two rectangles are also randomly selected, so these two rectangles may belong to different customers. Only if the solution of R' is not worse than R, R is replaced by R'; otherwise, R' is abandoned. The above iteration is repeated until the solution is not improved for *iter* consecutive iterations. At any step in the process, if the area returned by HeuristicPack is equal to the total area, then the procedure halts with SUCCESS. If this does not occur, RandomLS reports a FAILURE.

Three sorting rules are used to generate the initial sequence. For the *unrestricted* version, the first sorting rule is in decreasing order of the area of the rectangle, while the second and third rules are in decreasing order of the width and the length of the rectangle, respectively. The reason is that the rectangle with larger area (width or length) is usually difficult to pack so it should be packed

Algorithm 2. Heuristic procedure.

```
HeuristicPack($R$)
1  $M.x = 0, M.y = 0, M.w = W, M.l = L$
2  $spaceList = \{M\}$
3  if it is unrestricted version
4      Place the first rectangle $r$ at $M$
5      Update the open space list $spaceList$ and remove $r$ from $R$
6      while $spaceList \neq \emptyset$ and $R \neq \emptyset$
7          $M$ = space with the most bottom-left corner
8          Remove $M$ from $spaceList$
9          Find $r$ that is best-fit into $M$
10         if such $r$ is found
11             Place $r$ at $M$
12             Update the open space list $spaceList$ and remove $r$ from $R$
13 else
14     while $spaceList \neq \emptyset$ and $R \neq \emptyset$
15         $r$ = the first unpacked rectangle in $R$
16         Find the first space $M$ that can accommodate $r$
17         if such $M$ is found
18             Place $r$ at $M$ and remove $M$ from $spaceList$
19             Update the open space list $spaceList$ and remove $r$ from $R$
20         else
21             Break the loop
22 return the area of packed rectangles
```

first. If the *rotation* is allowed, the rectangles are rotated if needed in the second rule to make sure the width is not smaller than the length. The rotation is also applied in the third rule to make sure the length is not smaller than the width. For the *sequential* version, the rectangles are first sorted by the reverse visiting order of customers, and then it uses the same rules as *unrestricted* version to sort the rectangles demanded by the same customer.

4.3. Open space based heuristic packing subroutine

The HeuristicPack is given as Algorithm 2, which takes as input a sequence of ordered rectangles R and tries to pack as many rectangles into the vehicle as possible. It uses the MOS to represent the packing pattern and the rectangles are placed at the bottom-left corner of a MOS. The initial empty vehicle surface is represented by a MOS located at the point (0, 0) with size $W \times L$. The procedure involves many steps. In each step, all MOS are first sorted by the non-decreasing y-coordinate, breaking ties by non-decreasing x-coordinate. Then, it selects a placement (a MOS and a rectangle) according to a fitness measure function and places the selected rectangle at that MOS. After that, the MOS list is updated. The above steps are repeated until all the rectangles are placed or there is no MOS that can accommodate any rectangle. Different fitness measures are used for different versions of 2L-CVRP.

For the *unrestricted* version, the space-first rule is used. It first selects the MOS M with the most bottom-left corner, then finds the best-fit rectangle r that can be placed at M. For each rectangle r that can be placed at M, the fitness of the placement is defined as the number of MOS after placing r at M. The best-fit rectangle is the one leading to the smallest number of MOS and the first rectangle is selected if there is a tie. The motivation is that the fewer the open spaces are, the smoother (less inflection points) the unused space is, which can be better used in later stages. If no rectangle can be placed at M, then M is removed from the list, and the procedure proceeds to the next step. Fig. 5 gives an example of the fitness measure. The fitness ranges from 2 to 4 in this example and the resulting packing pattern is smoother (less inflection points) when fitness is equal to 2 compared to other values. As the first placed rectangle has a significant effect to the final solution, in order to bring more varieties, it selects the first rectangle in R as the first placed rectangle other than using the best-fit rectangle. For the *rotation* allowed version, two directions are considered while computing the fitness number of a rectangle and the direction with a smaller fitness number is preferred.

For the *sequential* version, the rectangle-first rule is used. It selects the first unpacked rectangle r in R, then finds the first MOS M that can accommodate r from the list. If no feasible MOS is found for r, the process is stopped and the area of packed rectangles is returned. Since the *sequential* constraint has a major effect in this version, the rectangles are placed one by one strictly accordingly to the input order. For the *rotation* allowed version, two directions are considered while trying to place a rectangle.

5. Acceleration strategy

In order to speed up this procedure, the data structure Trie is used to keep track of the loading feasibility of routes already examined. A similar strategy has been used by Leung et al. (2010), Wei et al. (2014), Zachariadis et al. (2009) and Wei et al. (2015). However, the Trie stores not only the feasibility information of routes, but also the number of calls of RandomLS on this route. To check the feasibility of a route rt that is represented by a sequence of customers, we first retrieve the information from the Trie. The stored information is a pair ($packed$, num), where $packed$ is the feasibility information and num is number of calls of RandomLS on this route. Four cases are classified based on the value of this pair ($maxNum$ is a parameter used to control the maximum allowed calls):

1. $packed ==$ NULL (the route does not exist in the Trie): $res \leftarrow RandomLS(rt)$; store ($res$, 1) of rt into Trie; return res.
2. $packed ==$ SUCCESS: return SUCCESS.
3. $packed ==$ FAILURE, $num < maxNum$: $res \leftarrow RandomLS(rt)$; store ($res$, $num + 1$) of rt into Trie; return res.
4. $packed ==$ FAILURE, $num == maxNum$: return FAILURE.

The Trie can save time in three ways. First, if a feasible packing for this route has been found by RandomLS previously, the stored information in the Trie can avoid unnecessary calls of RandomLS on the same route later (Case 2). Secondly, if no feasible packing for this route has been found, the effort spent on this route (Case 1, 3) is increased. Thus, more effort is spent on the promising routes encountered frequently. Thirdly, if the maximum number $maxNum$ allowed on the same route is reached, this route is regarded as infeasible and no more calls of RandomLS on this route are allowed (Case 4). By setting this limitation, not too much effort is spent on the same route, which can improve the efficiency of our approach. After some preliminary experiments , we set the maximum calling time $maxNum$ to $\max(nr, 100 \times (1 - A_I/(L \cdot W)))$, where nr is the number of items, A_I is the area of all items and $L \cdot W$ is the area of the vehicle surface. By setting $maxNum$ in this manner, the larger the ratio between item area and vehicle area is, the less calling number of RandomLS is allowed.

6. The simulated annealing algorithmic framework

Simulated annealing (SA) is an algorithmic approach to solve combinatorial optimization problems (Kirkpatrick, Gelatt, & Vecchi, 1983; Černý, 1985). It randomizes the local search procedure and accepts changes that worsen the solution with some probability. This constitutes an attempt to reduce the probability of getting trapped in a suboptimal solution. Van Breedam (1995); 2001) demonstrated the effectiveness of SA algorithm compared with its decent alternatives as well as other meta-heuristic implementations in solving the VRP and its variants. In this work, SA is employed to optimize the objective of the 2L-CVRP. To obtain good

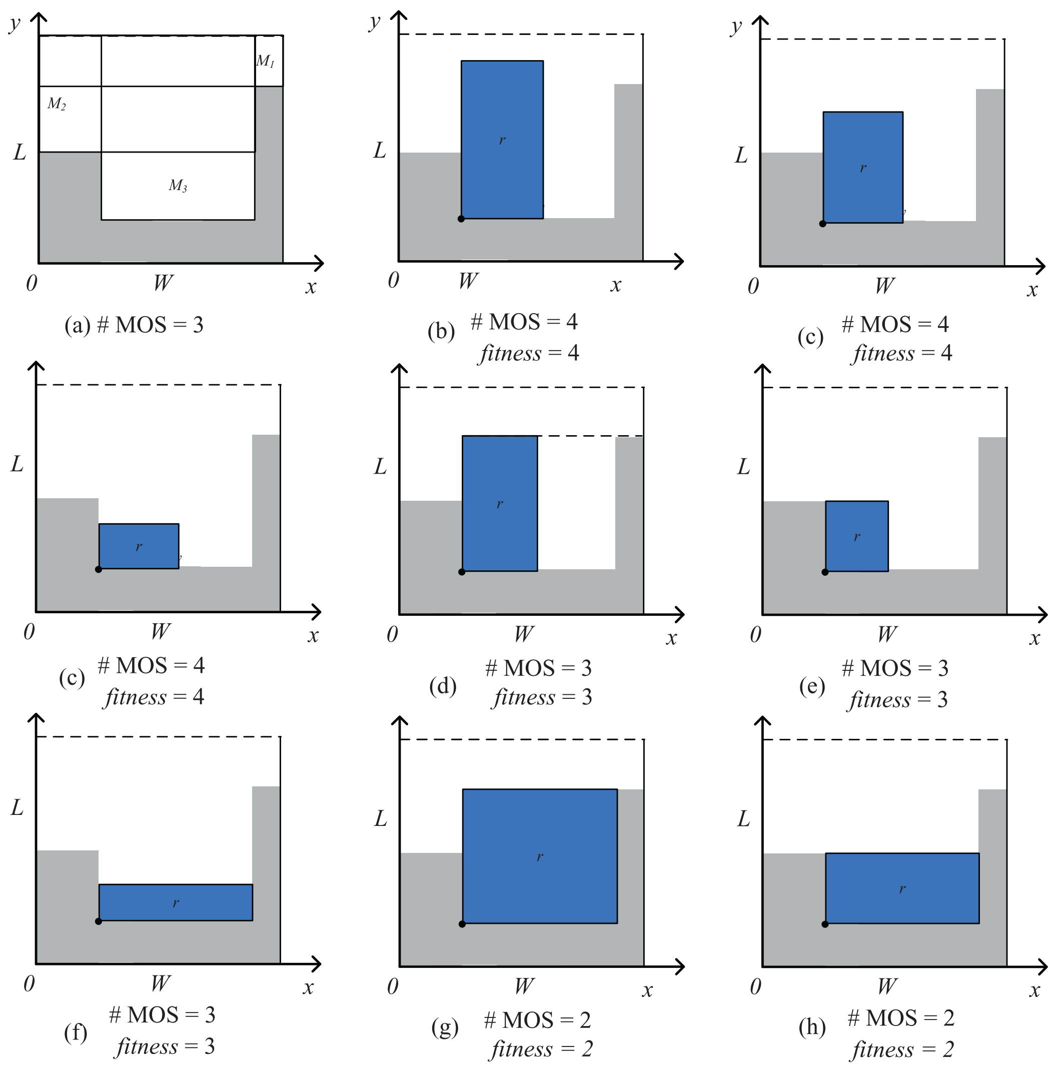

y
L
0
W
x
M_1
M_2
M_3
(a) # MOS = 3
(b) # MOS = 4 fitness = 4
(c) # MOS = 4 fitness = 4
(c) # MOS = 4 fitness = 4
(d) # MOS = 3 fitness = 3
(e) # MOS = 3 fitness = 3
(f) # MOS = 3 fitness = 3
(g) # MOS = 2 fitness = 2
(h) # MOS = 2 fitness = 2
r

Fig. 5. The fitness measure of placing a rectangle at a MOS (#MOS is the number of maximal open spaces).

performance, the algorithm is adapted according to the characteristics of this problem. The pseudo-code of our method is shown in Algorithm 3.

Given a feasible solution S of the 2L-CVRP, S is evaluated with the equation $cost(S) = \sum_{i=1}^{k} f(i)$, where k is the number of available vehicles and $f(i)$ is the travel cost of the ith route. During the

Algorithm 3. Simulated annealing algorithm for 2L-CVRP.

```
SA()
1  Construct the initial solution $S$
2  $S^* = S, T = T_0, T_b = T_0$
3  while time limit is not exceeded
4      for $k = 1$ to $Len$
5          Select a neighborhood structure $NS$ randomly
6          Generate a feasible solution $S'$ from $S$ with $NS$
7          if $cost(S') < cost(S)$
8              $S = S'$
9          else
10             Set $S = S'$ with probability $p$, where $p = exp(-\frac{cost(S') - cost(S)}{T})$
11         if $S'$ is better than $S^*$
12             $S^* = S', T_b = T$
13     $T = \alpha * T$
14     if $T < 0.01$
15         $T_b = 2 * T_b, T = \min\{T_b, T_{max}\}$
16 return $S^*$
```

search process, the algorithm attempts to transform the current solution S into one of its neighbors S' selected at random (Lines 5 and 6). Note that only feasible solutions are considered in our algorithm. Thus, for each temporally generated solution, the packing procedure is invoked to check its feasibility. For the chosen neighborhood structure *NS*, several temporal solutions may be generated until one feasible solution is obtained. If the cost of the new solution is less than the current solution S, the new solution is unconditionally accepted. If the cost of the new solution is higher than the current solution, the probability of acceptance of the new solution is $p = \exp(-\frac{cost(S') - cost(S)}{T})$, where T is a control parameter called temperature (Lines 7–10).

At the beginning of the algorithm, T is assigned an initial value T_0. As suggested by Ropke and Pisinger (2006) and Masson, Lehuédé, and Péton (2013), T_0 is set such that a solution w% worse than the initial solution has a 50% chance to be accepted. For each temperature, a series of *Len* attempts are performed to explore the space (Lines 4–12). During the process, the temperature decreases by multiplying the cooling rate α to enforce the convergence of the search (Line 13). Because this problem is a tightly constrained problem, it is very easy to be trapped in local optima. Thus, the temperature is increased to help the search escape from the trap when the current temperature is lower than 0.01 (Lines 14 and 15). More specifically, T_b is used to record the temperature with which

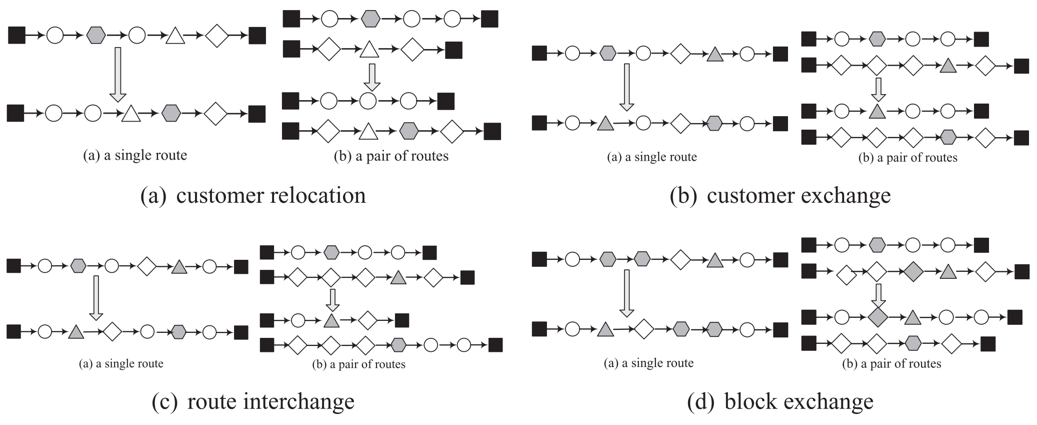


Fig. 6. The neighborhood structures.

the best solution S^* is found. When the temperature is rising, we double the T_b first and then set T as T_b. To avoid the search to restart from a randomly scratched solution, we limit the temperature T to T_{max}.

The detailed ingredients, such as construction of the initial solution and neighborhood structures, are explained in Sections 6.1 and 6.2.

6.1. Constructing initial solution

A two-phase algorithm is proposed to construct an initial solution of high quality. First, the savings algorithm (Clarke & Wright, 1964) is employed. Each customer is assigned to a vehicle, and then two routes are iteratively merged into a new one until no improvement by merger of two routes is possible. In each iteration, the merger with the largest saving is identified and performed. The saving is defined as $c_{i0} + c_{0j} - c_{ij}$, where i is the last customer of the first route and j is the first customer of the second route. Note that all resulting routes must be examined for feasibility in terms of packing.

In the second phase, we aim at reducing the number of routes iteratively such that only the given number of vehicles are used. If the solution obtained in the first phase is already feasible, this phase is skipped. In each iteration, the route with the least space utilization rate is identified and eliminated, and the corresponding customers are put into an unserved pool temporarily. Then, similar insertion strategy proposed by Wei et al. (2015) is applied. Namely, the customers are sorted by decreasing area of required items and reinserted into other routes one by one. More precisely, customers are inserted into the positions and routes with the lowest incremental cost and the obtained route must be packing-feasible. If a customer cannot be feasibly inserted into any route, a route is randomly selected and customers in this route are gradually erased until the given customer can be inserted into this route. The erased customers are put into the pool, and insertion restarts with the new pool. Finally, if all the customers are inserted into the routes, the above procedure restarts with the new solution until only the given number of vehicles are used. In our experiment, this method can find feasible solutions quickly for all instances of the four versions.

6.2. Neighborhood structures

In this paper, four types of neighborhood structures are employed. The first three types are simple operators and the last is efficient for escaping from local optimum but it is time-consuming. Thus, when the SA algorithm attempts to move to a new solution, one of these four types is randomly selected with equal probability. Then the algorithm moves to the new solution which is a feasible solution chosen stochastically among neighboring solutions of this type.

Table 1
The characteristics of items in Classes 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	Time limit(second)
ACO	Pentium IV 3.2 gigahertz	–	10,800
SA1	Intel 2.4 gigahertz Core Duo	2 gigabyte	–
EGTS + LBFH	Intel Core 2 Duo 2.0 gigahertz	2 gigabyte	–
GRASPxELS	Opteron 2.1 gigahertz	–	5400
PRMP	Intel Core 2 Duo E6600 2.4 gigahertz	–	–
MS-BR	Intel Core 2 Duo at 2.4 gigahertz	4 gigabyte	–
VNS	Intel Xeon E5430 with a 2.66 gigahertz (Quad Core) CPU	8 gigabyte	3600
SA	Intel Xeon E5430 with a 2.66 gigahertz (Quad Core) CPU	8 gigabyte	3600

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

Inst.	BKS	PRMP			VNS			SA		
		cost	*t*(second)	*Imp*	cost	*t*(second)	*Imp*	cost	*t*(second)	*Imp*
1	278.73	278.73	0.0	0.00	278.73	0.0	0.00	278.73	0.9	0.00
2	334.96	334.96	0.0	0.00	334.96	0.0	0.00	334.96	0.3	0.00
3	358.40	358.40	0.0	0.00	358.40	0.1	0.00	358.40	1.0	0.00
4	430.88	430.88	0.0	0.00	430.89	0.0	0.00	430.89	0.9	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	0.1	0.00	495.85	2.5	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	0.1	0.00	607.65	1.1	0.00
10	**535.74**	535.80	0.1	−0.01	535.80	0.1	−0.01	535.80	5.8	−0.01
11	505.01	505.01	0.0	0.00	505.01	0.0	0.00	505.01	0.6	0.00
12	610.00	610.00	0.2	0.00	610.00	0.9	0.00	610.00	5.4	0.00
13	2006.34	2006.34	0.3	0.00	2006.34	0.1	0.00	2006.34	0.0	0.00
14	837.67	837.67	0.1	0.00	837.67	0.1	0.00	837.67	0.0	0.00
15	837.67	837.67	0.4	0.00	837.67	0.1	0.00	837.67	0.0	0.00
16	698.61	698.61	0.3	0.00	698.61	1.1	0.00	698.61	4.0	0.00
17	861.79	861.79	1.6	0.00	861.79	4.0	0.00	861.79	22.2	0.00
18	723.54	723.54	3.6	0.00	723.54	1.4	0.00	723.54	6.7	0.00
19	524.61	524.61	2.1	0.00	524.61	2.0	0.00	524.61	9.0	0.00
20	241.97	241.97	7.2	0.00	241.97	3.5	0.00	241.97	14.6	0.00
21	687.60	687.60	3.8	0.00	687.60	74.9	0.00	687.60	343.8	0.00
22	740.66	740.66	2.8	0.00	740.66	21.2	0.00	740.66	101.1	0.00
23	835.26	835.26	48.7	0.00	835.26	159.7	0.00	835.26	838.0	0.00
24	1024.69	1024.69	38.1	0.00	1024.69	175.9	0.00	1024.69	1250.2	0.00
25	826.14	826.14	8.6	0.00	826.14	332.2	0.00	826.14	418.0	0.00
26	819.56	819.56	11.2	0.00	819.56	1.7	0.00	819.56	1.6	0.00
27	1082.65	1082.65	172.3	0.00	1082.65	445.5	0.00	1082.65	1306.0	0.00
28	**1040.70**	1042.12	71.2	−0.14	1042.12	1021.5	−0.14	1042.12	24.6	−0.14
29	1162.96	1162.96	121.9	0.00	1162.96	172.9	0.00	1162.96	35.9	0.00
30	**1028.42**	**1028.42**	267.5	0.00	1028.42	1570.0	0.00	1029.79	1435.8	−0.13
31	**1299.56**	**1299.56**	353.8	0.00	1302.48	1813.8	−0.22	1301.03	1884.0	−0.11
32	**1296.91**	**1296.91**	312.0	0.00	1300.22	1976.1	−0.26	1300.30	2006.9	−0.26
33	1298.02	1299.55	434.1	−0.12	1298.02	2204.1	0.00	**1296.13**	1884.2	0.15
34	**708.39**	709.82	328.2	−0.20	**708.39**	2125.2	0.00	708.66	1658.3	−0.04
35	865.39	866.06	396.3	−0.08	865.39	2050.4	0.00	**862.79**	1611.0	0.30
36	585.46	585.46	228.9	0.00	586.49	2420.2	−0.18	**583.98**	1276.3	0.25
Avg.	**769.56**	769.70	78.20	−0.02	769.80	460.53	−0.02	769.62	448.64	−0.01

BKS: Best-known solution from the literature.
Bold entries correspond to higher quality solutions.
t(second): Average CPU time to find the best solutions (in second).
Imp: Percentage improvement between the cost and BKS (*Imp* = 100*(BKS- *cost*)/BKS).

As shown in Fig. 6a–d, all four types involve one or two routes. The first type is customer relocation in which a customer is relocated to another position in the current solution (Fig. 6a). This type consists of the intra-shift and inter-shift in Wei et al. (2015). The second is the customer exchange that exchanges positions of two customers (Fig. 6b), including the intra-swap and inter-swap (Wei et al., 2015). The third is the route interchange (Fig. 6c). In this type, a pair of customers is selected and, if they are in the same route (intra-2opt in Wei et al. (2015)), positions of customers between them including themselves are reversed. When they are in different routes (inter-2opt in Wei et al. (2015)), the part of the first route before the selected customer is connected with the second part of the other route after the other selected customer, and vice-versa. The fourth type is the block exchange which changes the positions of two segments with length at most 3 (Fig. 6d).

7. Computational results

The algorithm was coded in C++ and tested on an Intel Xeon E5430 with a 2.66 gigahertz (Quad Core) CPU and 8 gigabyte RAM running the CentOS 5 Linux operating system. The classic 2L-CVRP instances that were generated by Gendreau et al. (2008) have been well-studied in several works, thus they are a good benchmark to verify the performance of the developed algorithm. There are 180 instances in total, which can be grouped into five classes according to the characteristics of the items. In Class 1, each customer is assigned one item with unit width and length, which means that the packing of items in one vehicle is always feasible. Thus, this class is a pure CVRP. Table 1 shows the characteristics of items in Classes 2–5 instances. In Classes 2–5, the number of items demanded by customer i, m_i, is generated by a uniform distribution at a certain interval (Column 2). Each item is classified into one of the three shape categories with equal probability. The three categories are vertical (the relative lengths are greater than the relative widths), homogeneous (the relative lengths and widths are generated in the same intervals), and horizontal (the relative lengths are smaller than the relative widths). The dimensions (width and length) of an item are uniformly distributed into the ranges determined by the item's shape category (Columns 3–8). Values $L = 40$ and $W = 20$ were chosen as the dimensions of the loading surface.

The initial temperature has a significant impact on the final solution. Thus, the value w is determined through an extensive calibration procedure on the 2L-CVRP instances. Because the SA converges slowly on this highly constrained problem, a low initial temperature T_0 is preferred. Through preliminary tests, $w = 0.1$ significantly leads to the best results. The method is not sensitive to other parameters. So they are set to the suggested values in the literature. For example, the cooling rate α is usually set as 0.9. The *Len* equals to n^2 which is the approximation for complexity of each

Table 4
Comparison for the 2|UO|L instances (averaged over Classes 2–5).

Inst.	BKS	PRMP			VNS			SA		
		Cost	*t*(second)	*Imp*	Cost	*t*(second)	*Imp*	Cost	*t*(second)	*Imp*
1	281.23	281.23	0.4	0.00	281.23	1.2	0.00	281.23	5.7	0.00
2	339.26	339.26	0.3	0.00	339.26	0.1	0.00	339.26	0.4	0.00
3	376.32	376.32	0.4	0.00	376.32	0.9	0.00	**375.81**	0.7	0.13
4	435.00	435.01	0.3	0.00	435.01	0.3	0.00	435.01	1.0	0.00
5	379.03	379.03	1.1	0.00	379.03	1.5	0.00	379.03	0.6	0.00
6	**496.77**	497.04	0.3	−0.05	497.04	1.0	−0.05	497.04	1.8	−0.05
7	690.67	690.67	1.6	0.00	690.67	2.5	0.00	690.67	1.9	0.00
8	**678.84**	**678.84**	2.6	0.00	**678.84**	3.8	0.00	679.44	5.3	−0.09
9	**611.05**	612.01	1.6	−0.16	612.01	1.3	−0.16	612.01	2.4	−0.16
10	674.92	676.75	26.9	−0.27	674.92	25.1	0.00	**674.88**	16.7	0.01
11	702.47	703.22	27.2	−0.11	702.47	59.9	0.00	**701.07**	23.7	0.20
12	611.20	611.26	1.4	−0.01	611.20	3.3	0.00	611.20	6.9	0.00
13	2484.16	2491.18	52.7	−0.28	2484.16	25.2	0.00	**2480.73**	22.0	0.14
14	974.04	975.88	164.3	−0.19	975.06	295.5	−0.11	**973.23**	53.4	0.08
15	1128.60	1132.91	20.1	−0.38	1128.60	246.3	0.00	**1128.18**	445.3	0.04
16	699.79	699.79	4.1	0.00	699.79	1.6	0.00	699.79	4.0	0.00
17	**862.26**	864.05	2.4	−0.21	864.05	4.0	−0.21	864.05	18.2	−0.21
18	1027.98	1031.95	33.0	−0.39	1027.98	79.8	0.00	**1027.45**	160.5	0.05
19	737.73	741.78	24.3	−0.55	737.73	250.5	0.00	**737.40**	206.6	0.05
20	515.44	515.44	552.2	0.00	515.92	794.2	−0.09	**513.53**	855.9	0.37
21	991.50	992.78	241.5	−0.13	991.63	751.1	−0.01	**988.30**	1658.4	0.32
22	**1017.33**	1023.01	166.6	−0.56	1019.03	885.2	−0.17	1017.56	1740.1	−0.02
23	1030.40	1032.36	336.8	−0.19	1030.40	853.1	0.00	**1029.32**	1353.2	0.10
24	**1099.57**	1104.64	319.6	−0.46	1102.53	572.1	−0.27	1100.64	923.1	−0.10
25	1333.76	1341.26	921.7	−0.56	1333.76	998.7	0.00	**1330.32**	1833.3	0.26
26	1306.60	1311.79	403.5	−0.40	1306.60	1050.6	0.00	**1306.59**	1466.8	0.00
27	1311.27	1318.04	438.2	−0.52	1311.27	874.5	0.00	**1309.92**	1696.0	0.10
28	2519.35	2530.46	3701.9	−0.44	2519.35	2259.2	0.00	**2506.12**	2222.3	0.53
29	2166.14	2173.02	1835.7	−0.32	2166.14	2232.8	0.00	**2163.06**	2169.9	0.14
30	1746.82	1760.59	2151.8	−0.79	1746.82	2495.4	0.00	**1741.87**	1337.1	0.28
31	2227.79	2244.13	2927.4	−0.73	2227.79	2952.9	0.00	**2220.22**	2080.9	0.34
32	2177.66	2196.85	3713.8	−0.88	2177.66	2648.7	0.00	**2167.60**	1954.7	0.46
33	2239.91	2261.68	1964.8	−0.97	2239.91	2942.7	0.00	**2236.73**	1949.9	0.14
34	1147.67	1157.22	3551.7	−0.83	1147.67	2459.6	0.00	**1144.14**	2472.6	0.31
35	1388.55	1401.17	2756.5	−0.91	1388.55	2620.6	0.00	**1380.90**	2447.2	0.55
36	1656.00	1669.44	4245.6	−0.81	1656.00	3012.0	0.00	**1637.04**	2953.5	1.14
Avg.	1112.97	1118.11	849.8	−0.34	1113.23	872.4	−0.03	**1110.59**	891.5	0.14

BKS: Best-known solution from the literature.
Bold entries correspond to higher quality solutions.
t(second): Average CPU time to find the best solutions (in second).
Imp: Percentage improvement between the cost and BKS (*Imp* = 100*(BKS- *cost*)/BKS).

neighborhood structure. $T_{max} = 25$ is large enough to accept a quite deteriorating solution (Leung et al., 2013; Leung et al., 2010).

In addition, to facilitate the comparisons with other algorithms, we set the maximum runtime according to the number of customers in each instance. More specially, the time is limited as follows: 1800 CPU seconds for $n \leq 50$ and 3600 CPU seconds for $n > 50$. For each instance, the algorithm is run for 10 times with random seeds 1–10.

7.1. Computational environments for 2L-CVRP

We executed SA ten times for each instance by setting the random seed from 1 to 10. We compare our SA with some of the most efficient approaches for 2L-CVRP, including ACO (Fuellerer et al., 2009), SA1 (Leung et al., 2010), EGTS + LBFH (Leung et al., 2011), GRASPxELS (Duhamel et al., 2011), PRMP (Zachariadis et al., 2013), MS-BR (Dominguez et al., 2014) and VNS (Wei et al., 2015). The computational environments for these approaches are summarized in Table 2. All these approaches were also executed 10 times for each instance. All four versions described in Section 3 are solved in our study. In the following tables, the cost listed is the best cost achieved over 10 runs. We list the details of only the best-known solution (BKS) among all previous approaches and the two methods with excellent performance: **PRMP**, **VNS** for the 2|UO|L and 2|SO|L versions, and **ACO**, **MS-BR** for the 2|UR|L and 2|OR|L versions (only these two papers studied the *rotation* allowed versions).

7.2. Results for Class 1 instances

Table 3 compares the best cost of each test problem against previous approaches on the pure CVRP (Class 1) instances. Our SA finds better solutions for 3 instances and matches the BKS for 27 instances. Moreover, the average cost of SA is smaller than the other two methods. Thus, the SA is very effective in respect of the routing aspect.

7.3. Results for 2|UO|L instances

Table 4 compares the best average cost of the classes (Classes 2–5) of each test problem (Problems 1–36) against previous approaches for the 2|UO|L instances. Our SA matches the best solutions for 7 instances and finds better solutions for 23 instances, especially the large-size instances. On average, the best-known solutions are improved by 0.14%. The algorithms require similar computational time to obtain the final solutions.

Table 5 gives the detailed comparison of 144 instances of the 2|UO|L version. Our SA finds better solutions for 56 instances and matches the best-known solution for 59 instances. The improvement on a single instance reaches up to 1.66%. In general, the av-

Table 5
Detailed results on the 144 instances of the 2|UO|L version.

Inst.	Class2			Class3			Class4			Class5		
	BKS	SA	*Imp*	BKS	SA	*Imp*	BKS	SA	*Imp*	BKS	SA	*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	**362.41**	0.56	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**	740.85	−0.33	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	689.68	0.00	**615.68**	620.33	−0.75	710.87	710.87	0.00	678.66	678.66	0.00
11	693.45	**684.21**	1.33	706.73	706.73	0.00	**784.88**	788.54	−0.47	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	610.00	610.00	0.00
13	2585.72	2585.72	0.00	**2436.56**	2454.37	−0.73	2548.06	2548.06	0.00	2334.78	2334.78	0.00
14	**1038.09**	1038.20	−0.01	**996.25**	1002.51	−0.63	981.00	981.00	0.00	875.00	**871.22**	0.43
15	**1013.29**	1017.95	−0.46	1154.66	**1149.99**	0.40	**1181.30**	1184.85	−0.30	1159.94	1159.94	0.00
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	1118.57	**1116.45**	0.19	917.94	917.94	0.00
19	**754.53**	757.23	−0.36	771.74	**771.66**	0.01	**775.87**	776.11	−0.03	644.59	644.59	0.00
20	525.75	**524.91**	0.16	521.31	521.31	0.00	539.52	**537.56**	0.36	471.64	**470.33**	0.28
21	**992.83**	993.01	−0.02	1118.11	**1116.58**	0.14	973.03	**970.37**	0.27	877.75	**873.25**	0.51
22	**1035.66**	1037.15	−0.14	**1052.98**	1053.48	−0.05	**1045.91**	1048.78	−0.27	932.38	**930.83**	0.17
23	**1035.18**	1041.58	−0.62	1079.03	**1074.30**	0.44	1071.30	1071.30	0.00	935.33	**930.09**	0.56
24	1178.07	1178.07	0.00	1080.88	1080.88	0.00	1108.34	**1100.76**	0.68	**1028.04**	1042.83	−1.44
25	1409.24	**1407.86**	0.10	1369.26	**1365.37**	0.28	1405.65	**1398.02**	0.54	1150.69	**1150.04**	0.06
26	**1272.87**	1280.13	−0.57	1344.10	**1342.19**	0.14	1394.19	**1390.99**	0.23	1213.88	**1213.04**	0.07
27	**1313.12**	1315.85	−0.21	1370.40	**1369.44**	0.07	1316.19	**1314.05**	0.16	1245.38	**1240.32**	0.41
28	2555.29	**2551.41**	0.15	2608.27	**2592.73**	0.60	2602.93	**2585.92**	0.65	2308.61	**2294.40**	0.62
29	2201.34	**2196.00**	0.24	**2087.15**	2088.44	−0.06	2247.23	**2240.18**	0.31	2128.84	**2127.60**	0.06
30	1808.77	**1803.26**	0.30	1830.50	**1821.83**	0.47	1826.10	**1820.46**	0.31	**1521.91**	1521.93	0.00
31	2265.18	**2254.47**	0.47	2276.14	**2268.64**	0.33	2382.77	**2366.80**	0.67	**1987.08**	1990.95	−0.19
32	2255.82	**2241.02**	0.66	2240.85	**2227.66**	0.59	2259.83	**2252.39**	0.33	1954.15	**1949.34**	0.25
33	2259.29	**2249.68**	0.43	**2348.25**	2348.28	0.00	**2373.63**	2373.83	−0.01	1977.25	**1975.14**	0.11
34	1176.03	**1170.77**	0.45	1204.58	**1196.20**	0.70	**1193.18**	1194.82	−0.14	1016.88	**1014.76**	0.21
35	1373.43	**1364.35**	0.66	1446.20	**1436.52**	0.67	1492.98	**1486.29**	0.45	1237.51	**1236.42**	0.09
36	1710.26	**1681.82**	1.66	1776.35	**1757.43**	1.07	1658.72	**1638.66**	1.21	1478.68	**1470.26**	0.57
Avg.	1128.66	**1126.68**	0.09	1140.05	**1138.25**	0.08	1152.71	**1150.13**	0.14	1028.37	**1027.31**	0.08

BKS: Best-known solution from the literature.
Bold entries correspond to higher quality solutions.
cost of SA is the score of best solution found over 10 runs. *Imp*: Percentage improvement between our cost and BKS ($Imp = 100^{*}(\text{BKS-}\ cost_{SA})/\text{BKS}$).

erage best-known cost of each class is improved by the proposed SA, and the largest improvement comes from the class 4.

7.4. Results for 2|SO|L instances

The 2|SO|L version considers the *LIFO* constraint to facilitate the unloading operation. Table 6 compares the best average cost of the four classes (Classes 2–5) of each test instance (instances 1–36) against previous approaches for the 2|SO|L version. The SA finds better solutions for 20 instances and matches the best solutions for 10 instances. The average improvement for the best-known solutions is 0.24%. Moreover, almost all large-size instances are improved by the proposed SA framework.

Table 7 gives the detailed comparison on each of the 144 instances of the 2|SO|L version. Our SA finds better solutions for 60 instances and matches the best solutions for 62 instances. The

Table 6
Comparison for the 2|SO|L instances (averaged over Classes 2–5).

Inst.	BKS	PRMP			VNS			SA		
		Cost	*t*(second)	*Imp*	Cost	*t*(second)	*Imp*	Cost	*t*(second)	*Imp*
1	287.08	287.08	1.4	0.00	287.08	4.5	0.00	287.08	5.3	0.00
2	344.21	344.21	1.0	0.00	344.21	0.5	0.00	344.21	0.6	0.00
3	381.40	381.40	1.3	0.00	381.40	1.4	0.00	381.40	0.9	0.00
4	439.97	439.97	1.6	0.00	439.97	0.9	0.00	439.97	1.3	0.00
5	382.39	382.39	2.6	0.00	382.39	4.2	0.00	382.39	1.8	0.00
6	499.48	499.48	5.6	0.00	499.48	1.7	0.00	499.48	2.5	0.00
7	701.63	702.27	5.3	−0.09	701.63	12.2	0.00	**699.84**	6.8	0.25
8	696.70	699.54	7.0	−0.41	696.70	21.3	0.00	696.70	45.8	0.00
9	614.53	615.93	6.2	−0.23	614.53	3.7	0.00	614.53	4.0	0.00
10	**684.00**	688.48	55.0	−0.66	686.09	115.6	−0.31	686.52	294.8	−0.37
11	722.84	725.83	75.3	−0.41	722.84	54.3	0.00	**722.22**	173.9	0.09
12	614.52	614.52	7.1	0.00	614.52	7.5	0.00	**614.47**	130.9	0.01
13	2546.77	2554.93	119.6	−0.32	2546.77	53.4	0.00	**2545.31**	274.5	0.06

(*continued on next page*)

Table 6 (*continued*)

Inst.	BKS	PRMP			VNS			SA		
		Cost	*t*(second)	*Imp*	Cost	*t*(second)	*Imp*	Cost	*t*(second)	*Imp*
14	1026.21	1027.38	637.0	−0.11	1026.21	416.7	0.00	**1009.14**	632.8	1.66
15	**1175.08**	1189.97	68.1	−1.27	**1175.08**	298.5	0.00	1175.19	344.2	−0.01
16	701.00	701.00	14.2	0.00	701.00	3.4	0.00	701.00	6.1	0.00
17	864.05	864.05	40.9	0.00	864.05	4.4	0.00	864.05	19.4	0.00
18	**1054.29**	1058.00	95.2	−0.35	**1054.29**	396.2	0.00	1056.07	480.1	−0.17
19	761.83	766.05	188.3	−0.55	761.83	297.4	0.00	**761.64**	435.4	0.03
20	530.26	534.87	1660.9	−0.87	530.26	921.7	0.00	**529.91**	1541.7	0.07
21	1027.74	1041.77	420.2	−1.37	1027.74	1003.0	0.00	**1024.22**	1850.2	0.34
22	**1053.49**	1066.56	524.2	−1.24	**1053.49**	1107.5	0.00	1055.62	1403.6	−0.20
23	**1063.52**	1076.19	519.5	−1.19	**1063.52**	953.1	0.00	1064.15	1499.3	−0.06
24	1132.88	1139.12	1064.3	−0.55	1132.88	841.4	0.00	**1132.24**	1339.6	0.06
25	**1378.55**	1401.00	2319.5	−1.63	**1378.55**	1306.1	0.00	1379.16	1716.9	−0.04
26	1355.92	1370.78	1491.2	−1.10	1355.92	1240.3	0.00	**1354.05**	1550.0	0.14
27	1354.92	1372.49	4163.8	−1.30	1354.92	1242.5	0.00	**1352.11**	1795.2	0.21
28	2646.59	2669.07	8640.1	−0.85	2646.59	2423.0	0.00	**2615.65**	2206.8	1.17
29	2241.65	2263.76	5484.3	−0.99	2241.65	2672.8	0.00	**2228.66**	2281.9	0.58
30	1819.25	1853.02	4676.9	−1.86	1819.25	2502.1	0.00	**1814.92**	1583.8	0.24
31	2317.82	2358.26	5845.4	−1.74	2317.82	2760.6	0.00	**2302.11**	2406.8	0.68
32	2274.88	2322.71	9433.2	−2.10	2274.88	2664.0	0.00	**2261.55**	2704.2	0.59
33	2342.87	2394.32	5662.5	−2.20	2342.87	2614.5	0.00	**2322.03**	2864.5	0.89
34	1196.33	1225.54	13141.8	−2.44	1196.33	2825.7	0.00	**1192.59**	2568.8	0.31
35	1454.42	1494.32	8989.6	−2.74	1454.42	3052.9	0.00	**1443.95**	2809.4	0.72
36	1736.03	1762.17	10059.6	−1.51	1736.03	3282.6	0.00	**1711.02**	3282.2	1.44
Avg.	1150.70	1163.57	2373.0	−0.84	1150.76	975.3	−0.01	**1146.25**	1062.9	0.24

BKS: Best-known solution from the literature.
Bold entries correspond to higher quality solutions.
t(second): Average CPU time to find the best solutions (in second).
Imp: Percentage improvement between the cost and BKS (*Imp* = 100*(BKS- *cost*)/BKS).

largest improvement on the single instance reaches up to 4.27%. On average, the best-known cost of each class is improved by 0.12 to 0.3%, which is larger than the improvement for the 2|UO|L version.

7.5. Results for 2|UR|L instances

Allowing the rotation of goods are common in real-world applications, but the related studies are quite rare that only two papers reported results for the 2|UR|L version. Table 8 provides the com-

Table 7
Detailed results on the 144 instances of the 2|SO|L version.

Inst.	Class2			Class3			Class4			Class5		
	BKS	SA	*Imp*	BKS	SA	*Imp*	BKS	SA	*Imp*	BKS	SA	*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	**702.59**	1.00	703.49	703.49	0.00	658.64	658.64	0.00
8	725.91	725.91	0.00	741.12	741.12	0.00	697.92	697.92	0.00	621.85	621.85	0.00
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**	639.09	−1.61	715.82	715.82	0.00	690.96	690.96	0.00
11	721.54	721.54	0.00	**717.37**	719.91	−0.35	**815.68**	822.61	−0.85	636.77	**624.82**	1.88
12	619.63	619.63	0.00	610.00	610.00	0.00	618.23	618.23	0.00	610.23	**610.00**	0.04
13	2669.39	2669.39	0.00	2486.44	2486.44	0.00	2609.36	2609.36	0.00	2421.88	**2416.04**	0.24
14	1101.61	**1092.51**	0.83	1085.42	**1039.06**	4.27	983.20	**982.25**	0.10	924.27	**922.75**	0.16
15	1041.75	1041.75	0.00	1181.68	1181.68	0.00	**1246.49**	1247.39	−0.07	1230.40	**1229.95**	0.04
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	**1053.09**	1059.44	−0.60	1103.45	**1102.17**	0.12	**1134.11**	1136.31	−0.19	926.53	**926.34**	0.02
19	792.42	**792.07**	0.04	801.13	801.13	0.00	801.21	801.21	0.00	652.58	**652.15**	0.07
20	547.82	**545.68**	0.39	**541.58**	544.11	−0.47	552.91	**551.72**	0.22	478.73	**478.15**	0.12
21	1060.72	1060.72	0.00	1150.85	**1149.90**	0.08	1006.21	**1000.25**	0.59	893.18	**886.00**	0.80
22	1081.44	1081.44	0.00	**1094.66**	1099.59	−0.45	**1089.27**	1092.86	−0.33	948.60	948.60	0.00
23	**1093.27**	1095.41	−0.20	**1117.54**	1117.97	−0.04	**1093.01**	1094.52	−0.14	950.25	**948.68**	0.17
24	1222.43	1222.43	0.00	1118.44	1118.44	0.00	**1141.97**	1142.01	0.00	1048.69	**1046.08**	0.25
25	1458.83	**1453.98**	0.33	1436.57	**1433.92**	0.18	**1435.18**	1444.77	−0.67	**1183.63**	1183.96	−0.03
26	1327.47	**1323.23**	0.32	1396.52	**1392.43**	0.29	**1447.03**	1447.36	−0.02	**1252.65**	1253.18	−0.04
27	**1367.85**	1369.52	−0.12	**1423.74**	1426.69	−0.21	1357.75	**1353.06**	0.35	1270.34	**1259.17**	0.88
28	2699.21	**2632.55**	2.47	2787.24	**2737.42**	1.79	2700.66	**2690.69**	0.37	**2399.25**	2401.93	−0.11
29	2289.84	**2285.84**	0.17	2172.69	**2150.35**	1.03	2312.37	**2299.32**	0.56	2191.69	**2179.12**	0.57

(*continued on next page*)

Table 7 (*continued*)

Inst.	Class2			Class3			Class4			Class5		
	BKS	SA	*Imp*	BKS	SA	*Imp*	BKS	SA	*Imp*	BKS	SA	*Imp*
30	**1875.38**	1877.21	−0.10	1915.42	**1912.09**	0.17	1910.54	**1904.42**	0.32	1575.64	**1565.96**	0.61
31	2369.07	**2341.08**	1.18	2360.63	**2354.21**	0.27	2469.40	**2459.59**	0.40	2072.19	**2053.57**	0.90
32	2384.29	**2365.99**	0.77	2325.74	**2320.35**	0.23	2357.57	**2343.29**	0.61	2031.92	**2016.58**	0.75
33	2376.58	**2349.98**	1.12	2469.85	**2447.20**	0.92	2470.76	**2446.05**	1.00	2054.29	**2044.88**	0.46
34	1226.98	**1217.24**	0.79	1253.88	**1249.07**	0.38	1242.26	**1241.13**	0.09	**1062.18**	1062.93	−0.07
35	1447.30	**1434.99**	0.85	1529.77	**1511.66**	1.18	1558.69	**1550.24**	0.54	1281.90	**1278.90**	0.23
36	1784.57	**1755.33**	1.64	1869.38	**1833.97**	1.89	1740.64	**1713.71**	1.55	1549.51	**1541.07**	0.54
Avg.	1175.71	**1170.05**	0.27	1182.29	**1176.54**	0.30	1187.25	**1184.42**	0.12	1057.25	**1054.01**	0.24

BKS: Best-known solution from the literature.
Bold entries correspond to higher quality solutions.
Cost of SA is the score of best solution found over 10 runs.
Imp: Percent gap between our cost and BKS (*Imp* = 100*(BKS- $cost_{SA}$)/BKS).

Table 8
Comparison for the 2|UR|L instances (averaged over Classes 2–5).

Inst.	BKS	ACO			MS-BR			SA		
		cost	*t*(second)	*Imp*	cost	*t*(second)	*Imp*	cost	*t*(second)	*Imp*
1	**280.84**	281.16	–	−0.11	**280.84**	3.3	0.00	281.13	0.7	−0.10
2	339.26	341.02	–	−0.52	339.26	0.8	0.00	339.26	0.4	0.00
3	**370.62**	372.93	–	−0.62	**370.62**	34.8	0.00	371.62	2.1	−0.27
4	435.00	435.01	–	0.00	435.00	2.0	0.00	435.00	1.1	0.00
5	378.59	378.59	–	0.00	378.59	30.0	0.00	378.59	0.7	0.00
6	497.04	497.04	–	0.00	497.05	3.8	0.00	497.04	1.7	0.00
7	681.00	688.50	–	−1.10	681.00	20.3	0.00	681.00	1.2	0.00
8	675.46	678.75	–	−0.49	675.46	17.0	0.00	**675.45**	5.1	0.00
9	612.01	612.02	–	0.00	612.01	32.3	0.00	612.01	2.8	0.00
10	666.67	671.00	–	−0.65	667.65	182.0	−0.15	**660.43**	5.4	0.94
11	690.56	698.25	–	−1.11	690.56	146.5	0.00	690.56	5.2	0.00
12	611.06	611.12	–	−0.01	611.06	15.3	0.00	**610.06**	6.8	0.16
13	**2437.15**	2468.19	–	−1.27	**2437.15**	114.5	0.00	2437.58	4.9	−0.02
14	968.55	974.80	–	−0.65	968.55	176.5	0.00	**961.11**	10.9	0.77
15	1111.81	1132.49	–	−1.86	1112.00	183.5	−0.02	**1110.54**	46.4	0.11
16	699.79	699.79	–	0.00	699.80	26.5	0.00	699.79	5.1	0.00
17	861.79	862.36	–	−0.07	861.79	12.5	0.00	861.79	15.4	0.00
18	999.22	1012.19	–	−1.30	999.22	182.5	0.00	**997.97**	23.7	0.13
19	722.17	726.96	–	−0.66	722.17	268.8	0.00	**716.24**	168.4	0.82
20	501.90	508.69	–	−1.35	501.90	210.5	0.00	**500.18**	143.9	0.34
21	977.03	989.24	–	−1.25	977.03	348.8	0.00	**971.45**	1346.4	0.57
22	1000.79	1008.52	–	−0.77	1001.75	297.3	−0.10	**994.77**	1402.0	0.60
23	1011.19	1024.25	–	−1.29	1011.19	420.5	0.00	**998.81**	903.8	1.22
24	1092.90	1098.60	–	−0.52	1092.90	213.5	0.00	**1083.69**	986.4	0.84
25	1320.27	1323.84	–	−0.27	1320.27	362.5	0.00	**1298.87**	1389.2	1.62
26	1302.52	1314.34	–	−0.91	1302.52	332.0	0.00	**1284.92**	1145.9	1.35
27	1303.75	1309.76	–	−0.46	1304.14	362.0	−0.03	**1279.22**	1529.9	1.88
28	2516.24	2526.81	–	−0.42	2518.51	401.0	−0.09	**2454.97**	1796.6	2.43
29	2159.75	2175.33	–	−0.72	2161.43	417.8	−0.08	**2121.31**	1520.7	1.78
30	1735.98	1742.15	–	−0.35	1742.01	337.8	−0.35	**1701.36**	1717.4	1.99
31	2203.91	2227.74	–	−1.08	2204.44	472.8	−0.02	**2160.70**	2244.6	1.96
32	2165.44	2180.18	–	−0.68	2167.61	394.0	−0.10	**2119.46**	2036.5	2.12
33	2222.42	2239.04	–	−0.75	2222.42	459.5	0.00	**2174.71**	1925.1	2.15
34	1140.47	1149.87	–	−0.82	1142.25	458.0	−0.16	**1114.40**	2399.1	2.29
35	1384.37	1387.45	–	−0.22	1392.05	471.8	−0.55	**1345.26**	2376.0	2.83
36	1648.75	1670.67	–	−1.33	1653.05	375.3	−0.26	**1598.88**	2646.5	3.02
Avg.	1103.51	1111.63	1818.10	−0.66	1104.31	216.3	−0.05	**1089.45**	772.7	0.88

BKS: Best-known solution from the literature.
Bold entries correspond to higher quality solutions.
t(second): Average CPU time to find the best solutions (in second).
Imp: Percentage improvement between the cost and BKS (*Imp* = 100*(BKS- *cost*)/BKS).

parisons of the average cost of classes 2–5, where the dash line (-) means that no detailed time information was reported for each instance in the original paper. Among the 36 instances, the proposed SA improves the best-known solutions for 24 instances, and matches the best solutions for 9 instances. The improvements for all large-size instances 25–36 are larger than 1%, and reach up to 3.02%.

The detailed results for each 2|UR|L instance are reported in Table 9. The proposed SA improves or matches the best-known solutions for 137 of the total 144 instances. More precisely, the best-known solutions of 81 instances are improved, and of 56 instances are matched. Almost all the best-known costs of large-size instances 18–36 of classes 2–5 are improved with gap up to 4.84%.

Table 9
Detailed results on the 144 instances of the 2|UR|L version.

Inst.	Class2			Class3			Class4			Class5		
	BKS	SA	*Imp*	BKS	SA	*Imp*	BKS	SA	*Imp*	BKS	SA	*Imp*
1	278.73	278.73	0.00	**282.95**	284.11	−0.41	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	380.35	380.35	0.00	385.32	385.32	0.00	**358.40**	362.41	−1.12	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	379.94	379.94	0.00	383.87	383.88	0.00	375.28	375.28	0.00
6	495.85	495.85	0.00	498.16	498.16	0.00	498.32	498.32	0.00	495.85	495.85	0.00
7	715.02	715.02	0.00	664.96	664.96	0.00	686.26	686.26	0.00	657.77	657.77	0.00
8	665.17	665.17	0.00	738.43	738.43	0.00	688.32	688.32	0.00	609.90	609.90	0.00
9	607.65	607.65	0.00	607.65	607.65	0.00	625.10	625.10	0.00	607.65	607.65	0.00
10	**667.42**	667.86	−0.07	615.36	**591.61**	3.86	703.64	703.64	0.00	680.26	**678.62**	0.24
11	**664.48**	666.16	−0.25	699.35	699.35	0.00	773.58	**771.93**	0.21	624.82	624.82	0.00
12	610.00	610.00	0.00	610.00	610.00	0.00	614.23	**610.23**	0.65	610.00	610.00	0.00
13	**2502.65**	2504.53	−0.08	2377.39	2377.39	0.00	2533.79	2533.79	0.00	2334.78	**2334.59**	0.01
14	1029.34	1029.34	0.00	988.79	988.80	0.00	981.00	**955.09**	2.64	875.07	**871.22**	0.44
15	1001.51	1001.51	0.00	1120.75	**1116.07**	0.42	1164.77	**1164.63**	0.01	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	861.79	861.79	0.00	861.79	861.79	0.00	861.79	861.79	0.00	861.79	861.79	0.00
18	988.61	**987.10**	0.15	986.30	986.30	0.00	1100.66	**1100.52**	0.01	921.29	**917.94**	0.36
19	726.51	**723.93**	0.36	752.06	**749.43**	0.35	765.51	**747.03**	2.41	644.59	644.59	0.00
20	489.23	**488.69**	0.11	511.46	511.46	0.00	534.14	**533.77**	0.07	472.77	**466.79**	1.26
21	**964.49**	968.42	−0.41	1089.75	**1086.72**	0.28	967.85	**959.82**	0.83	886.04	**870.82**	1.72
22	**976.70**	985.16	−0.87	1031.79	**1024.11**	0.74	1052.60	**1041.80**	1.03	942.06	**928.02**	1.49
23	985.18	**984.00**	0.12	1056.56	**1041.60**	1.42	1064.76	**1047.32**	1.64	938.25	**922.34**	1.70
24	1152.35	**1140.13**	1.06	1073.01	**1066.15**	0.64	1099.40	**1086.09**	1.21	1046.84	**1042.37**	0.43
25	1356.24	**1345.89**	0.76	1353.90	**1333.64**	1.50	1402.08	**1366.28**	2.55	1168.87	**1149.66**	1.64
26	1262.43	**1257.00**	0.43	1335.80	**1311.11**	1.85	1391.02	**1362.22**	2.07	1220.83	**1209.34**	0.94
27	1283.66	**1271.10**	0.98	1354.76	**1329.33**	1.88	1318.45	**1284.94**	2.54	1258.12	**1231.52**	2.11
28	2517.25	**2491.86**	1.01	2587.25	**2541.02**	1.79	2638.07	**2510.29**	4.84	2322.37	**2276.71**	1.97
29	2151.68	**2129.10**	1.05	2067.69	**2040.83**	1.30	2267.37	**2199.79**	2.98	2152.26	**2115.53**	1.71
30	1755.89	**1740.87**	0.86	1811.22	**1767.72**	2.40	1834.68	**1784.14**	2.75	1542.14	**1512.71**	1.91
31	2171.60	**2162.88**	0.40	2246.54	**2196.26**	2.24	2385.63	**2314.76**	2.97	2011.88	**1968.89**	2.14
32	2191.58	**2165.96**	1.17	2219.26	**2166.18**	2.39	2267.57	**2206.72**	2.68	1983.34	**1938.96**	2.24
33	2175.85	**2157.23**	0.86	2325.36	**2276.31**	2.11	2387.22	**2318.77**	2.87	2001.26	**1946.51**	2.74
34	1140.83	**1121.67**	1.68	1176.71	**1165.57**	0.95	1208.19	**1163.96**	3.66	1036.16	**1006.38**	2.87
35	1340.41	**1310.33**	2.24	1437.30	**1393.90**	3.02	1503.42	**1452.59**	3.38	1256.34	**1224.21**	2.56
36	1679.27	**1625.42**	3.21	1739.36	**1708.05**	1.80	1670.84	**1605.00**	3.94	1505.54	**1457.05**	3.22
Avg.	1100.82	**1093.90**	0.41	1124.13	**1110.58**	0.85	1152.81	**1130.55**	1.30	1036.27	**1022.76**	0.94

BKS: Best-known solution from the literature.
Bold entries correspond to higher quality solutions.
Cost of SA is the score of best solution found over 10 runs.
Imp: Percentage improvement between our cost and BKS ($Imp = 100^{*}(\text{BKS- } cost_{SA})/\text{BKS}$).

7.6. Results for 2|SR|L instances

The 2|SR|L version was only solved by the ACO (Fuellerer et al., 2009) previously. Thus, we compare the average results of classes 2–5 obtained by the SA and ACO in Table 10. All the best-known solutions can be either improved or matched by the SA. In fact, the SA obtains better results for 30 instances with improvement up to 4.88%. The average improvement of these instances is 1.7%. Regarding the computational time, the SA requires much less time to obtain the final solutions.

Table 11 gives the detailed result of each instance. Among the 144 instances, the SA finds better solutions for 108 instances and reaches the best solutions for 32 instances. The improvement on the single instance reaches up to 6.37%. On average, the greatest improvement is obtained on the class 4 with 2.38%, and the least improvement is got for the class 2 with 0.68%.

7.7. Impact of different loading constraints

To study the impact of different loading constraints, we summarize the average results of different classes over different versions in Table 12.

The row “Avg.” reports the average cost over classes 2–5 for each version. As expected, the 2|SO|L version gets the largest cost of 1146.25, because it has the tightest loading constraints, such as the *LIFO* and *oriented* constraints. The 2|UR|L version obtains the least cost of 1089.45. Hence, the percentage gap between the 2|SO|L and 2|UR|L versions reaches to 5.21%.

The column “Avg.” gives the average cost of different versions for each class. According to the results, the class 5 seems to be the easiest to solve, while class 4 has the largest average cost.

The row “Gap-S/U” shows the impacts of the *LIFO* constraint on the problems with and without item rotation. We can see that the impact of *LIFO* constraint depends on whether the item rotation is allowed. Its impact becomes less significant when the rotation is allowed, as the percentage gap decreases from 3.21 to 1.95%.

The row “Gap-O/R” studies the impacts of the rotation on the problems with and without *LIFO* constraints. Similar result can be observed that the impact of rotation becomes less significant if the *LIFO* constraint is not imposed.

These interesting observations can provide some helpful insights to the decision makers. The managers should make a trade-off between the convenience and cost according to their requirements. For example, to reduce the travel cost, the manager may need to adopt some tools to rotate the cargos if necessary.

7.8. Analysis of different sorting rules in RandomLS

Note that we use three different sorting rules to generate the initial sequence in the process RandomLS. To analyze the impact

Table 10
Comparison for the 2|SR|L instances (averaged over Classes 2–5).

Inst.	BKS	ACO			SA		
		Cost	*t*(second)	*Imp*	Cost	*t*(second)	*Imp*
1	281.70	281.70	–	0.00	**281.16**	9.4	0.19
2	341.02	341.02	–	0.00	**339.26**	0.4	0.52
3	376.65	376.65	–	0.00	**373.28**	2.9	0.90
4	435.01	435.01	–	0.00	435.01	1.2	0.00
5	378.59	378.59	–	0.00	378.59	1.0	0.00
6	497.62	497.62	–	0.00	497.62	2.4	0.00
7	696.23	696.23	–	0.00	**682.31**	4.5	2.00
8	690.09	690.09	–	0.00	**678.75**	95.1	1.64
9	612.02	612.02	–	0.00	612.01	2.9	0.00
10	679.67	679.67	–	0.00	**672.88**	193.5	1.00
11	711.67	711.67	–	0.00	**700.33**	299.5	1.59
12	613.89	613.89	–	0.00	**611.06**	13.6	0.46
13	2513.67	2513.67	–	0.00	**2466.95**	128.2	1.86
14	992.53	992.53	–	0.00	**985.75**	301.2	0.68
15	1165.38	1165.38	–	0.00	**1121.33**	637.6	3.78
16	699.79	699.79	–	0.00	699.79	4.6	0.00
17	861.79	861.79	–	0.00	861.79	19.0	0.00
18	1018.95	1018.95	–	0.00	**1014.83**	287.8	0.40
19	737.73	737.73	–	0.00	**732.46**	608.9	0.71
20	517.79	517.79	–	0.00	**506.92**	1785.9	2.10
21	1010.65	1010.65	–	0.00	**987.36**	1660.3	2.30
22	1029.08	1029.08	–	0.00	**1010.64**	1514.7	1.79
23	1042.27	1042.27	–	0.00	**1021.23**	1422.6	2.02
24	1115.85	1115.85	–	0.00	**1098.54**	1159.3	1.55
25	1351.34	1351.34	–	0.00	**1326.23**	1711.7	1.86
26	1341.83	1341.83	–	0.00	**1310.39**	1855.3	2.34
27	1344.19	1344.19	–	0.00	**1308.93**	1564.3	2.62
28	2600.20	2600.20	–	0.00	**2532.21**	2517.0	2.61
29	2217.32	2217.32	–	0.00	**2164.76**	2258.7	2.37
30	1795.53	1795.53	–	0.00	**1744.15**	1831.2	2.86
31	2293.88	2293.88	–	0.00	**2224.42**	2414.6	3.03
32	2249.85	2249.85	–	0.00	**2181.29**	2497.9	3.05
33	2305.09	2305.09	–	0.00	**2237.98**	2566.7	2.91
34	1187.81	1187.81	–	0.00	**1148.73**	2772.1	3.29
35	1444.85	1444.85	–	0.00	**1387.27**	2765.8	3.99
36	1734.97	1734.97	–	0.00	**1650.22**	3317.9	4.88
Avg.	1135.74	1135.74	2247.90	0.00	**1110.73**	1061.9	1.70

BKS: Best-known solution from the literature.
Bold entries correspond to higher quality solutions.
t(second): Average CPU time to find the best solutions (in second).
Imp: Percentage improvement between the cost and BKS (*Imp* = 100*(BKS- *cost*)/BKS).

Table 11
Detailed results on the 144 instances of the 2|SR|L version.

Inst.	Class2			Class3			Class4			Class5		
	BKS	SA	*Imp*	BKS	SA	*Imp*	BKS	SA	*Imp*	BKS	SA	*Imp*
1	278.73	278.73	0.00	284.52	**284.23**	0.10	282.95	282.95	0.00	280.60	**278.73**	0.67
2	334.96	334.96	0.00	352.16	352.16	0.00	342.00	**334.96**	2.06	334.96	334.96	0.00
3	384.93	384.93	0.00	394.72	**385.32**	2.38	368.56	**364.45**	1.11	358.40	358.40	0.00
4	430.89	430.89	0.00	430.89	430.89	0.00	447.37	447.37	0.00	430.89	430.89	0.00
5	375.28	375.28	0.00	379.94	379.94	0.00	383.88	383.88	0.00	375.28	375.28	0.00
6	498.16	498.16	0.00	498.16	498.16	0.00	498.32	498.32	0.00	495.85	495.85	0.00
7	716.82	716.82	0.00	706.99	**668.39**	5.46	702.45	**686.26**	2.30	658.64	**657.77**	0.13
8	674.20	674.20	0.00	741.12	**738.43**	0.36	705.89	**692.47**	1.90	639.18	**609.90**	4.58
9	607.65	607.65	0.00	607.65	607.65	0.00	625.13	**625.10**	0.01	607.65	607.65	0.00
10	685.21	**684.70**	0.07	617.62	**615.68**	0.31	722.70	**710.87**	1.64	693.15	**680.26**	1.86
11	**694.60**	695.02	−0.06	706.73	**704.77**	0.28	800.88	**776.69**	3.02	644.46	**624.82**	3.05
12	615.87	**610.00**	0.95	610.23	**610.00**	0.04	619.21	**614.24**	0.80	610.23	**610.00**	0.04
13	**2526.07**	2534.97	−0.35	2469.98	**2436.41**	1.36	2623.65	**2561.65**	2.36	2434.99	**2334.78**	4.12
14	1041.61	**1032.96**	0.83	1012.46	**1006.69**	0.57	988.25	**981.90**	0.64	927.79	**921.45**	0.68
15	1009.87	**1005.26**	0.46	1170.82	**1146.66**	2.06	1245.94	**1172.43**	5.90	1234.87	**1160.96**	5.99
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	861.79	861.79	0.00	861.79	861.79	0.00	861.79	861.79	0.00	861.79	861.79	0.00
18	989.21	**988.37**	0.09	1031.94	**1031.49**	0.04	1128.25	**1118.18**	0.89	926.40	**921.29**	0.55
19	**732.64**	737.10	−0.61	**757.59**	764.20	-0.87	796.42	**776.59**	2.49	664.28	**651.97**	1.85
20	496.93	**495.01**	0.39	536.58	**519.43**	3.20	549.38	**541.17**	1.49	488.28	**472.09**	3.32
21	998.48	**986.35**	1.21	1126.49	**1104.72**	1.93	1006.61	**977.00**	2.94	911.01	**881.38**	3.25
22	1009.25	**1001.03**	0.81	1060.79	**1044.34**	1.55	1089.59	**1061.43**	2.58	956.71	**935.74**	2.19
23	1019.73	**1001.74**	1.76	1090.74	**1064.72**	2.39	1098.82	**1076.34**	2.05	959.79	**942.10**	1.84
24	1183.02	**1173.04**	0.84	1091.54	**1076.30**	1.40	1129.79	**1102.37**	2.43	1059.06	**1042.43**	1.57

(*continued on next page*)

Table 11 (*continued*)

Inst.	Class2			Class3			Class4			Class5		
	BKS	SA	*Imp*	BKS	SA	*Imp*	BKS	SA	*Imp*	BKS	SA	*Imp*
25	1383.57	**1371.51**	0.87	1374.51	**1364.61**	0.72	1442.71	**1395.23**	3.29	1204.57	**1173.55**	2.58
26	1283.32	**1278.62**	0.37	1372.46	**1341.01**	2.29	1450.79	**1395.93**	3.78	1260.75	**1226.01**	2.76
27	1310.13	**1292.78**	1.32	1399.26	**1367.12**	2.30	1369.55	**1324.92**	3.26	1297.83	**1250.91**	3.62
28	2548.28	**2524.37**	0.94	2710.90	**2588.78**	4.50	2724.74	**2647.03**	2.85	2416.88	**2368.67**	1.99
29	2197.20	**2170.03**	1.24	2138.30	**2089.73**	2.27	2327.21	**2247.12**	3.44	2206.55	**2152.16**	2.46
30	1805.65	**1774.70**	1.71	1874.66	**1821.19**	2.85	1908.00	**1838.87**	3.62	1593.81	**1541.85**	3.26
31	2265.21	**2231.33**	1.50	2326.58	**2260.59**	2.84	2486.10	**2380.00**	4.27	2097.62	**2025.75**	3.43
32	2258.99	**2226.32**	1.45	2305.18	**2234.74**	3.06	2372.77	**2277.96**	4.00	2062.45	**1986.14**	3.70
33	2251.88	**2209.40**	1.89	2407.87	**2350.43**	2.39	2479.42	**2381.54**	3.95	2081.18	**2010.55**	3.39
34	1172.76	**1156.30**	1.40	1238.94	**1195.80**	3.48	1251.31	**1201.39**	3.99	1088.21	**1041.41**	4.30
35	1375.14	**1352.64**	1.64	1483.13	**1430.28**	3.56	1606.54	**1504.19**	6.37	1314.60	**1261.97**	4.00
36	1737.64	**1673.51**	3.69	1842.36	**1759.82**	4.48	1765.72	**1656.29**	6.20	1594.14	**1511.25**	5.20
Avg.	1123.73	**1113.03**	0.68	1158.73	**1134.31**	1.59	1191.83	**1155.62**	2.38	1068.65	**1039.98**	2.12

BKS: Best-known solution from the literature.
Bold entries correspond to higher quality solutions.
Cost of SA is the score of best solution found over 10 runs.
Imp: Percent gap between our cost and BKS ($Imp = 100^*(\text{BKS-}\ cost_{SA})/\text{BKS}$).

Table 12
The average results of different classes over different versions.

	2|UO|L	2|SO|L	2|UR|L	2|SR|L	Avg.
Class 2	1126.68	1170.05	1093.90	1113.03	1125.92
Class 3	1138.25	1176.54	1110.58	1134.31	1139.92
Class 4	1150.13	1184.42	1130.55	1155.62	1155.18
Class 5	1027.31	1054.01	1022.76	1039.98	1036.02
Avg.	1110.59	1146.25	1089.45	1110.73	
Gap-S/U	3.21		1.95		
Gap-O/R			1.94	3.20	

of these three rules, we added another sorting rule called *Random* which generates the initial sequence randomly for the *unrestricted* version. For the *sequential* version, *Random* first sorts the rectangles by the reverse visiting order of customers, and then sorts the rectangles belonging to the same customer randomly. We tested the modified algorithm on the largest instance 36 for the versions 2|UO|L and 2|SO|L. For each sorting rule, we recorded the times that the loading problem is uniquely solved with the sequence generated from the initial order of the rule. The results are shown as Fig. 7. It can be seen that the times of uniquely solved from the sequence by *Width* are much larger than those by other rules for both versions. For the *unrestricted* version, the times of uniquely solved from sequence by *Random* are similar to those by *Length* and smaller than those by *Area* and *Width*. While for the *sequential* version, the times of uniquely solved from the sequence by *Random* are much smaller than those of other rules. This is because for the *unrestricted* version, the rectangle is selected by the *fitness* measure. While for the *sequential* version, the rectangle is packed by the given order. Therefore, the sorting rules have a greater impact on the results of the *sequential* version compared to the *unrestricted* one.

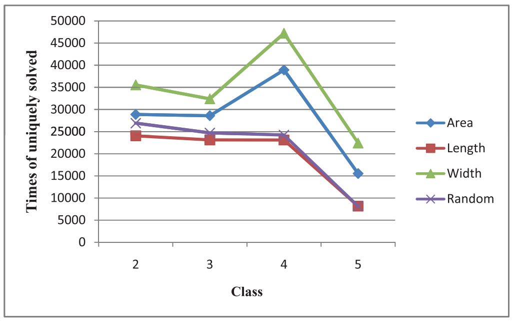
7.9. Analysis of the open space based local search heuristic

In order to analyze the effect of the open space based local search heuristic, we replaced the packing procedure RandomLS with the packing procedure LH_{2SL} used by Gendreau et al. (2008), **LBFH** by Leung et al. (2011) and TabuPack by Wei et al. (2015). All the other parts remain the same.

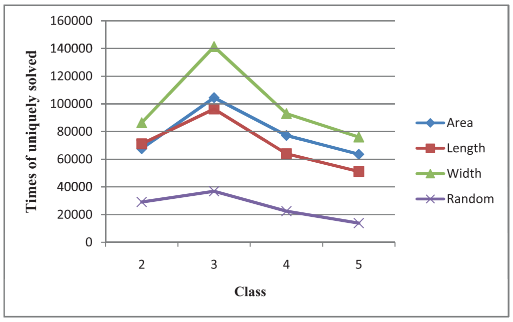
For each class 2–5, one out of three instances are selected, i.e., instance 3, 6,..., 33 and 36. Therefore, 48 instances are selected in total. For each instance, we executed SA once by setting the random seed to 1. The comparative results are given in Table 13, in which the cost is averaged over the instances with the same id. We can see that the results of SA are much better than those of SA+LH_{2SL}, SA+LBFH and SA + TabuPack. Thus, it can be concluded that the open space based local search heuristic makes a significant contribution to the final result of our approach.

8. Conclusions

This paper studies the well-known integrated routing and packing problem, 2L-CVRP. A novel SA framework with repeated cooling and rising of temperature is developed to solve this highly constrained problem. A two-phase method employing savings and reinsertion algorithm is proposed to construct the initial solution.



(a) 2|UO|L



(b) 2|SO|L

Fig. 7. Times of each sorting rule that the packing problem is uniquely solved on instance 36.

Table 13
The effect of the open space based local search heuristic.

Inst.	Unrestricted				Sequential			
	SA	SA+LH_{2SL}	SA+LBFH	SA+TabuPack	SA	SA+LH_{2SL}	SA+LBFH	SA+Tabupack
3	376.32	380.88	376.32	376.32	381.40	385.38	382.76	382.25
6	497.04	501.70	500.98	497.04	499.48	504.68	504.22	499.48
9	612.01	618.46	612.01	612.01	614.53	621.23	621.23	614.54
12	611.20	618.92	616.21	611.20	614.52	621.51	619.82	615.80
15	1128.62	1196.55	1190.95	1129.10	1175.59	1221.25	1214.62	1196.66
18	1027.45	1075.22	1066.26	1028.41	1057.80	1092.65	1086.87	1070.88
21	989.88	1043.66	1031.83	993.12	1028.83	1070.57	1060.19	1048.57
24	1100.89	1145.06	1136.23	1104.11	1133.88	1169.52	1157.29	1152.77
27	1312.10	1386.67	1370.42	1314.05	1357.91	1420.99	1407.70	1379.01
30	1745.93	1863.46	1824.89	1749.66	1818.75	1912.09	1892.37	1842.37
33	2244.31	2394.82	2352.21	2243.08	2345.08	2448.84	2417.45	2372.04
36	1644.41	1760.44	1717.17	1694.01	1721.49	1783.46	1776.11	1907.98
Avg.	1107.51	1165.48	1149.62	1112.68	1145.77	1187.68	1178.38	1173.53

SA: The approach proposed in this paper.
SA+LH_{2SL}: Replace RandomLS by LH_{2SL} used by Gendreau et al. (2008).
SA+LBFH: Replace RandomLS by packing check procedure used by Leung et al. (2011).
SA+TabuPack: Replace RandomLS by TabuPack used by Wei et al. (2015).

In addition, we develop a new open space based heuristic to identify the packing patterns for routes, and an efficient data structure (Trie) that not only records the loading feasibility information of routes already examined but also controls the effort spent on each route. Four versions with different loading constraints are studied in our work. More specially, the versions allowing the rotation of items are studied and the corresponding benefits are investigated. Numerical experiments on the 2L-CVRP instances show that the proposed SA outperforms all existing algorithms, and matches or improves the best-known solutions for most instances. Furthermore, the computational time is also comparable. Thus, the proposed SA is illustrated to be quite effective and efficient for the 2L-CVRP. Finally, some interesting observations are provided.

Acknowledgements

The work described in this paper was partially supported by the National Nature Science Foundation of China (Grant No. 71390335, 61272003, 61672439, 71401065, and 71501075), a grant from City University of Hong Kong (Project No. 7004149), the Natural Science Foundation of Jiangxi Province (Grant no. 20151BAB217006), the Educational Commission of Jiangxi Province (Grant no. GJJ160457) and Guangdong Natural Science Funds for Distinguished Young Scholar (Grant No. 2015A030306007).

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.

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

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science, 6*(1), 80–91.

Dominguez, O., Juan, A., Barrios, B., Faulin, J., & Agustin, A. (2016). Using biased randomization for solving the two-dimensional loading vehicle routing problem with heterogeneous fleet. *Annals of Operations Research, 236*(2), 383–404.

Dominguez, O., Juan, A., & Faulin, J. (2014). A biased-randomized algorithm for the two-dimensional vehicle routing problem with and without item rotations. *International Transactions in Operational Research, 21*(3), 375–398.

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.

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.

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.

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.

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.

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

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.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

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

Lai, K. K., & Chan, J. W. M. (1997). Developing a simulated annealing algorithm for the cutting stock problem. *Computers & Industrial Engineering, 32*(1), 115–127.

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.

Leung, S. C. H., 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.

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.

Martello, S., & Vigo, D. (1998). Exact solution of the two-dimensional finite bin packing problem. *Management Science, 44*(3), 388–399.

Masson, R., Lehuédé, F., & Péton, O. (2013). An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Science, 47*(3), 344–355.

Oliveira, J. (2010). Comments on: Routing problems with loading constraints. *TOP, 18*(1), 31–33.

Pollaris, H., Braekers, K., Caris, A., Janssens, G. K., & Limbourg, S. (2015). Vehicle routing problems with loading constraints: state-of-the-art and future directions. *OR Spectrum, 37*(2), 297–330.

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

Ruan, Q., Zhang, Z., Miao, L., & Shen, H. (2013). A hybrid approach for the vehicle routing problem with three-dimensional loading constraints. *Computers & Operations Research, 40*(6), 1579–1589.

Tao, Y., & Wang, F. (2015). An effective tabu search approach with improved loading algorithms for the 3l-cvrp. *Computers & Operations Research, 55*, 127–140.

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. (2014). Vehicle routing: Problems, methods, and applications, second edition. *MOS-SIAM series on optimization*. Philadelphia: Society for Industrial and Applied Mathematics.

Van Breedam, A. (1995). Improvement heuristics for the vehicle routing problem based on simulated annealing. *European Journal of Operational Research, 86*(3), 480–490.

Van Breedam, A. (2001). Comparing descent heuristics and metaheuristics for the vehicle routing problem. *Computers & Operations Research, 28*(4), 289–315.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications, 45*(1), 41–51.

Wang, Y., & Chen, L. (2015). Two-dimensional residual-space-maximized packing. *Expert Systems with Applications, 42*(7), 3297–3305.

Wei, L., Oon, W.-C., Zhu, W., & Lim, A. (2013). A goal-driven approach to the 2d bin packing and variable-sized bin packing problems. *European Journal of Operational Research, 224*(1), 110–121.

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.

Wei, L., Zhang, Z., Zhang, D., & Lim, A. (2015). A variable neighborhood search for the capacitated vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research, 243*(3), 798–814.

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.

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.

Zhang, D., Wei, L., Leung, S. C. H., & Chen, Q. (2013). A binary search heuristic algorithm based on randomized local search for the rectangular strip-packing problem. *INFORMS Journal on Computing, 25*(2), 332–345.

Zhang, Z., Wei, L., & Lim, A. (2015). An evolutionary local search for the capacitated vehicle routing problem minimizing fuel consumption under three-dimensional loading constraints. *Transportation Research Part B: Methodological, 82*, 20–35.

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