5-2021

A matheuristic algorithm for the vehicle routing problem with cross-docking

Aldy GUNAWAN
Singapore Management University, aldygunawan@smu.edu.sg

Audrey Tedja WIDJAJA
Singapore Management University, audreyw@smu.edu.sg

Pieter VANSTEENWEGEN
Katholieke Universiteit Leuven

Vincent F. YU
National Taiwan University of Science and Technology

Citation

GUNAWAN, Aldy; WIDJAJA, Audrey Tedja; VANSTEENWEGEN, Pieter; and YU, Vincent F.. A matheuristic algorithm for the vehicle routing problem with cross-docking. (2021). *Applied Soft Computing*. 103, 1-13.
Available at: https://ink.library.smu.edu.sg/sis_research/6039

A matheuristic algorithm for the vehicle routing problem with cross-docking

Aldy Gunawan [a,*] Audrey Tedja Widjaja [a], Pieter Vansteenwegen [b], Vincent F. Yu [c]

a School of Computing and Information Systems, Singapore Management University, Singapore 178902, Singapore
b KU Leuven, Leuven Mobility Research Center, CIB, Celestijnenlaan 300, 3001 Leuven, Belgium
c Department of Industrial Management, National Taiwan University of Science and Technology, Taipei 106, Taiwan

Published in Applied Soft Computing, 103, May 2021, 107163. DOI: 10.1016/j.asoc.2021.107163

Abstract: This paper studies the integration of the vehicle routing problem with cross-docking (VRPCD). The aim is to find a set of routes to deliver products from a set of suppliers to a set of customers through a cross-dock facility, such that the operational and transportation costs are minimized, without violating the vehicle capacity and time horizon constraints. A two-phase matheuristic based on column generation is proposed. The first phase focuses on generating a set of feasible candidate routes in both pickup and delivery processes by implementing an adaptive large neighborhood search algorithm. A set of destroy and repair operators are used in order to explore a large neighborhood space. The second phase focuses on solving the set partitioning model to determine the final solution. The proposed matheuristic is tested on the available benchmark VRPCD instances and compared with the state-of-the-art algorithms. Experimental results show the competitiveness of the proposed matheuristic as it is able to improve the best known solutions for 80 instances and to obtain the same results for the remaining 10 instances, with an average improvement of 12.6%. On new and larger instances, our proposed matheuristic maintains its solution quality within acceptable CPU times and outperforms a pure ALNS algorithm. We also explicitly analyze the performance of the matheuristic considering the solution quality and CPU time.

Keywords: Adaptive large neighborhood search, Cross-docking, Matheuristic, Scheduling, Set-partitioning formulation, Vehicle routing problem

Corresponding author: aldygunawan@smu.edu.sg (A. Gunawan).

1. Introduction

Cross-docking is an intermediate activity within a supply chain network for enabling a transshipment process. The two key points of the cross-docking are simultaneous arrival and consolidation [1]. Different shipments for a particular destination are consolidated in a full truckload, such that direct shipment with less than truckload can be avoided, and thus the transportation cost can be minimized [2]. Whenever an incoming (inbound) vehicle arrives at the cross-dock facility, its loads are sorted, moved, and loaded to the outgoing (outbound) vehicle for an immediate delivery elsewhere in the network [3].

The vehicle routing problem with cross-docking (VRPCD), as the integration of the vehicle routing problem (VRP) and cross-docking, is quite common in practice, however, only few papers consider both simultaneously [1]. Obviously, many papers consider the classical VRP, which involves the service of a set of customers with known demands by vehicles from a single distribution center or warehouse. The main objective is to minimize the total distance and the number of vehicles which start and end their tours at the central depot. The VRPCD was first introduced by Lee et al. [4]. It aims to construct a set of routes to deliver a single type of product from a set of suppliers to a set of customers through a cross-dock facility, such that the operational and transportation costs are minimized, with respect to vehicle capacity and time limitations.

In this study, we design a matheuristic based on a branch-and-price/column generation approach, which employs a restricted master heuristic scheme [5]. This approach is commonly used since it only requires a heuristic scheme to generate columns. In the implementation, the column generation is performed by an adaptive large neighborhood search (ALNS), due to its ability to explore a large neighborhood space. The proposed matheuristic is tested on benchmark VRPCD instances, and the results are compared against those of the state-of-the-art algorithms: tabu search (TS) [4], improved tabu search (imp-TS) [1], and simulated annealing (SA) [6]. Experimental results on the available benchmark VRPCD instances show that our proposed matheuristic outperforms the state-of-the-art algorithms. New sets of larger instances which are originally developed for the VRPCDTW [7] are also introduced. On those instances, our matheuristic outperforms an ALNS algorithm. An explicit analysis is included on the added value of solving the set partitioning formulation and implementing the adaptive scheme when selecting operators in the neighborhood search process.

The main contributions of this work are summarized as follows:

- A matheuristic algorithm is developed in order to improve the solutions of the available benchmark VRPCD instances. The optimal solutions of one set of instances which were not available before are also provided.
- New sets of larger instances are introduced in order to evaluate the performance of our matheuristic. These instances and our solutions can also be used for future research.
- A comprehensive analysis of the properties of our algorithm, such as the set partitioning formulation, the adaptive scheme when selecting operators, and the performance of operators, is presented.

The rest of the paper is organized as follows. Section 2 summarizes related work to the VRPCD. Section 3 defines the VRPCD network. The proposed matheuristic is presented in Section 4, followed by the computational results and some analysis in Section 5. Finally, Section 6 concludes this study.

2. Related work

Lee et al. [4] first introduced the concept of VRPCD as the integration of vehicle routing problem (VRP) and cross-docking. A mathematical model for the VRPCD network is presented and a tabu search (TS) approach is proposed to solve the problem, since it is an NP-hard problem. The proposed TS is straightforward. It generates a new solution by selecting two adjacent arcs that have the highest transportation cost. The two selected arcs are then exchanged with other arcs on the same positions from a randomly selected vehicle, if any. Otherwise, only the sequence of the two arcs is changed. If the newly generated solution improves the total cost, then the (arc, vehicle) pairs found are preserved in the tabu list. Benchmark instances are generated in order to compare the results obtained by an enumeration approach and the proposed TS. Experimental results show that TS is able to obtain solutions with an average gap under 5% compared to the enumeration results, in a reasonable CPU time.

Liao et al. [1] studied the same variant of VRPCD as in [4] and solve the instances by introducing another TS. The main differences of their TS with the one proposed by Lee et al. [4] are: (1) the former [4] exchanged routes between two vehicles simultaneously, while the latter [1] arranged it one at a time, (2) the latter [1] allows a reduction of the number of vehicles used during the search process, while this property is not observed in the former approach [4]. This new TS is able to improve the solutions by 10%–36% for different sizes of problems with significantly shorter computational times.

Yu et al. [6] proposed a simulated annealing (SA) algorithm which is primarily designed for solving a variant of the VRPCD, namely the open VRPCD (OVRPCD). In this problem, an open network wherein the flow starts from a pickup node and ends at the cross-dock or begins at the cross-dock and finishes at one of the delivery node without returning to the cross-dock is introduced. The proposed SA uses a string of numbers representing a permutation of the delivery nodes and pickup nodes with dummy zeros to separate the routes. In order to create a neighborhood solution, three common moves: swap, reverse, and N-insertion, are implemented. Other than solving the OVRPCD, benchmark VRPCD instances as in [4] and [1] are also solved. The proposed SA is able to further improve the solutions of 78 out of 90 problem instances.

Santos et al. [8] studied another variant of the VRPCD by considering costs when loads are transferred from one vehicle to another during the delivery and pickup process. Moreover, no time horizon constraint is imposed in the proposed problem. A branch-and-price algorithm is proposed. It dominates a linear programming approach based on branch-and-bound by determining better lower and upper bounds during the search. Wen et al. [7] extended the VRPCD by considering time windows, namely VRPCDTW. The VRPCDTW however, has a different objective function compared to the VRPCD, since it only aims to minimize the total transportation time, while VRPCD aims to minimize the total of transportation and vehicle operational costs. Various algorithms have been proposed to solve the VRPCDTW, such as TS [7,9,10], variable neighborhood search (VNS) [10], genetic algorithm (GA) [11], matheuristic [12], local search [13], and iterated local search (ILS) [14]. Grangier et al. [15] then adapted the matheuristic in [12] to solve the VRPCDTW by considering the limitation of the number of dock doors that can be used simultaneously. Wang et al. [16] allowed split deliveries in customer nodes as well as split pickups in supplier nodes in an effort to maximize the vehicle utilization. A two-layer variant of SA and TS is proposed and it is able to effectively solve large-size problems within a reasonable CPU time.

While both VRPCD and VRPCDTW only consider the vehicle routing for delivering and picking up products, Ting and Chen [17] studied the truck scheduling to determine the sequence of truck arrivals at the cross-dock facility, as well as the routing problem. This truck scheduling problem is important to address when the number of docking doors at the cross-dock facility is insufficient (or less than the number of vehicles used). In order to solve this problem, an ant colony optimization (ACO) algorithm is proposed. It is able to obtain optimal solutions in small size instances. This problem is further extended by considering the docking door assignment and a heterogeneous fleet [18]. A new rigorous mixed-integer linear programming formulation improved by an approximate sweep-based model is introduced. It is able to solve instances with up to 50 nodes with three different fleet sizes up to 10 vehicles within acceptable CPU times. Table 1 summarizes the recent literature on the VRPCD.

The idea of combining metaheuristics with elements of exact mathematical programming algorithms, known as matheuristics, for solving the VRP was first introduced by Foster and Ryan [19]. Matheuristics also have been used to solve other VRP variants, such as truck and trailer routing problems [20], the pollution-routing problem [21], and the technician routing and scheduling problem [22].

One of the matheuristic approaches is based on a branch-and-price/column generation approach, which employs a restricted master heuristic scheme [5]. In this approach, a heuristic algorithm generates a set of columns, and the set partitioning formulation is solved on that subset of columns to find a feasible solution. There are two different approaches to perform the column generation phase:

- Use a heuristic algorithm without considering the dual information given by the solution of the restricted master problem, or
- Consider the dual information, but only a restricted set of columns is generated.

We observe that previous research on VRPCD mostly focuses on developing (meta)heuristics which thus motivates us to design a matheuristic approach in this paper.

3. Problem description

The VRPCD network consists of a set of suppliers $S = \{1, 2, \dots, |S|\}$ delivering products to a set of customers $C = \{1, 2, \dots, |C|\}$ through a cross-dock facility, denoted as node 0. Two major

Table 1
VRPCD literature.

Literature	VRPCD	Time windows	Open network	Limited dock doors	Heterogeneous fleet	Split pickups-deliveries	Approach
Lee et al. [4]	✓	–	–	–	–	–	Tabu search
Liao et al. [1]	✓	–	–	–	–	–	Improved tabu search of [4]
Yu et al. [6]	✓	–	✓	–	–	–	Simulated annealing
Santos et al. [8]	✓[a]	–	–	–	–	–	Branch and price
Wen et al. [7]	✓	✓	–	–	–	–	Tabu search
Tarantilis [9]	✓	✓	–	–	–	–	Adaptive multi-restart tabu search
Sadri Esfahani and Fakhrzad [10]	✓	✓	–	–	–	–	Tabu search and variable neighborhood search
Touihri et al. [11]	✓	✓	–	–	–	–	Genetic algorithm
Grangier et al. [12]	✓	✓	–	–	–	–	Matheuristic based large neighborhood search
Urtasun and Montero [13]	✓	✓	–	–	–	–	Greedy randomized adaptive search procedure
Morais et al. [14]	✓	✓	–	–	–	–	Iterated local search
Grangier et al. [15]	✓	✓	–	✓	–	–	Matheuristic based large neighborhood search
Wang et al. [16]	✓	✓	–	–	–	✓	Two-layer simulated annealing and tabu search
Ting and Chen [17]	✓[b]	–	–	–	–	–	Ant colony optimization
Dondo and Cerdá [18]	✓	✓	–	✓	✓	–	Sweep-heuristic based-models
This research	✓	–	–	–	–	–	Matheuristic based adaptive large neighborhood search and simulated annealing

[a]Also considering load transfer costs between vehicles.
[b]Also considering the truck scheduling at the cross-dock.

processes in a VRPCD network are: the pickup process at the suppliers and the delivery process to the customers. However, the processes inside the cross-dock facility, such as loading, unloading and sorting processes, are assumed to be fast enough. Therefore, they are not taken into consideration. P_i products must be picked up from node i in S, and D_i products must be delivered to node i in C.

Each pair of nodes (i, j) in S is connected by travel time t'_{ij} and transportation cost c'_{ij}. Each pair of nodes (i, j) in C is connected by travel time t''_{ij} and transportation cost c''_{ij}. The two processes together cannot exceed the time horizon, T_{max}. The VRPCD network is illustrated in Fig. 1.

A fleet of homogeneous vehicles $V = \{1, 2, \ldots, |V|\}$ with capacity Q is available at the cross-dock facility to be utilized for shipments. Each vehicle may only be used to perform either a pickup process or a delivery process, or neither. In the pickup process, vehicles depart from the cross-dock, visit one (or more) supplier(s) to pickup their products, and return to the cross-dock for consolidating products. After the products are consolidated according to customers' demand, vehicles depart from the cross-dock, visit one (or more) customer(s) to deliver their demand, and return to the cross-dock. For each vehicle used, a fixed operational cost H will be charged.

The VRPCD aims to determine the number of vehicles used and its corresponding routes, such that the operational and transportation costs are minimized. The constraints in the VRPCD are as follows:

- The total transportation time for the pickup and delivery processes together does not exceed T_{max}. This implies that all products are first collected from the suppliers and then (after consolidation) these products are distributed to the customers, all within T_{max}.
- Each supplier and customer can only be visited exactly once. This implies that split deliveries (pickups) in customer (supplier) nodes are not allowed.
- Each vehicle starts its trip from and ends its trip at the cross-dock. Multiple trips are not allowed.
- The number of vehicles utilized in both the pickup and delivery process together does not exceed $|V|$. VRPCD assumes that each vehicle can only be used for one route, which is either a pickup route or a delivery route.
- The amount of loads on the pickup route and on the delivery route in each vehicle does not exceed Q.

One may think that solving the VRPCD is similar to solving two independent VRPs. However, this is not the case due to the above-mentioned constraints which require the two VRPs to be solved simultaneously. This definitely increases the complexity of the problem. Interested readers are referred to [4] for the VRPCD mathematical model.

4. Proposed algorithm

In this section, we present the two phases of our matheuristic. First, we briefly describe the overview of the matheuristic and the motivation behind it. Sections 4.2 and 4.4 then describe each phase respectively. Section 4.3 provides the list of destroy and repair operators used in ALNS.

4.1. Overview of the matheuristic

The matheuristic pseudocode is presented in Algorithm 1. Sol_0, Sol^*, and Sol' are defined as the current solution, the best found solution so far, and the starting solution at each iteration respectively. An initial solution is constructed based on a greedy approach, where the node with the least additional transportation cost is inserted, such that each vehicle starts (ends) its route from (in) the cross-dock without violating the vehicle capacity and time horizon constraints. At the beginning, the Sol_0, Sol^*, and Sol' are equal to the initial solution (Line 1 in Algorithm 1).

The proposed matheuristic is decomposed into two phases: (i) ALNS and (ii) the set partitioning formulation. In the first phase (Lines 7–38 in Algorithm 1), the aim is to generate as many as feasible candidate routes, represented as columns. A column is defined as a feasible candidate route of a vehicle which visits a set of nodes, starting and ending at the cross-dock. A candidate route is feasible if it visits any node at most once and it does not violate the vehicle capacity constraint. In the second phase (Line 39 in Algorithm 1), a set partitioning formulation is solved to find a combination of routes that satisfies the VRPCD constraints.

The motivation of the proposed matheuristic is explained as follows. In a pure metaheuristic or heuristic search, a set of routes could be completely ignored or rejected if it does not improve the solution quality or it leads to an infeasible solution. In fact, some of those routes could be good routes or part of the optimal solution. Therefore, a matheuristic is introduced to tackle this problem, where all routes constructed during the heuristic

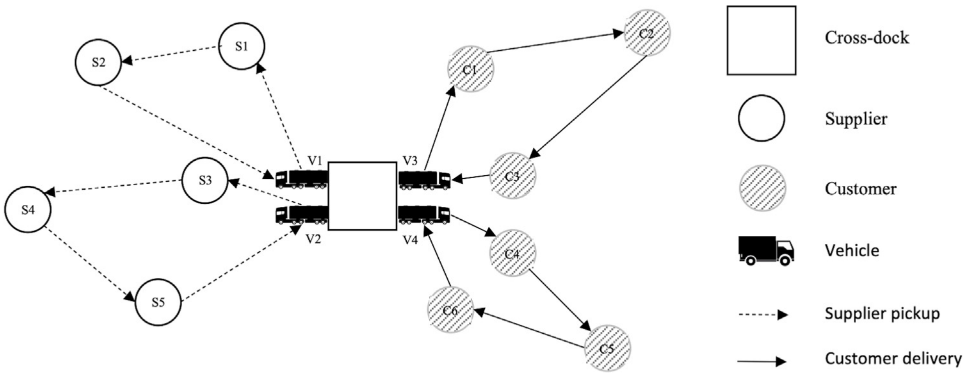

Fig. 1. VRPCD network.

search are stored and treated independently regardless of their performances (the first phase) and a combination of routes is then determined to minimize the objective function as long as it satisfies all constraints (formulated as a set partitioning problem in the second phase). By implementing this approach, some good routes that were ignored in the first phase are still "remembered" and are available to be chosen in the second phase.

Since there are two processes in the VRPCD (pickup and delivery processes), the feasible candidate routes which are obtained in the first phase are accommodated in two different pools. Let Ω_s and Ω_c be a pool to accommodate the list of feasible candidate routes in the pickup process and in the delivery process respectively. Both Ω_s and Ω_c are expanded during the first phase with the routes explored in ALNS after performing destroy and repair operators (see Section 4.2). However, every time we insert any routes to the Ω_s or Ω_c, we avoid route duplication. In the second phase, we solve the set partitioning formulation (see Section 4.4) over the set of routes stored in Ω_s and Ω_c. After generating the initial solution, the constructed routes (pickup routes and delivery routes) are added into Ω_s and Ω_c respectively (Line 2).

The current temperature (*Temp*) is set to be equal to the initial temperature (T_0) (Line 3), Iter and NoImpr are set to zero (Line 4), and FoundBestSol is set to False (Line 5). We introduce Iter as a variable to count the number of iterations that have been passed, NoImpr as a variable to count the number of successive temperature reductions when no new best solution is found, and FoundBestSol is a boolean variable which is True when a new best solution is found and False otherwise.

Algorithm 1: Matheuristic pseudocode

```
1  $Sol_0, Sol^*, Sol' \leftarrow$ Initial Solution
2  UpdatePool(InitialSolution, $\Omega_s$, $\Omega_c$)
3  $Temp \leftarrow T_0$
4  NoImpr, Iter $\leftarrow 0$
5  FoundBestSol $\leftarrow$ False
6  Set $s_j$ and $w_j$ such that $p_j$ is equally likely ($\forall j \in R \cup I$)
7  while NoImpr $< \theta$ do
8      RemovedNodes $\leftarrow 0$
9      while RemovedNodes $< \pi$ do
10         $Sol_0 \leftarrow$ Destroy ($R_i$)
11         UpdatePool($Sol_0$, $\Omega_s$, $\Omega_c$)
12         UpdateRemovedNodes(RemovedNodes, $R_i$)
13     end
14     while RemovedNodes $> 0$ do
15         $Sol_0 \leftarrow$ Repair ($I_i$)
16         UpdatePool($Sol_0$, $\Omega_s$, $\Omega_c$)
17         UpdateRemovedNodes(RemovedNodes, $I_i$)
18     end
19     AcceptanceCriteria($Sol_0$, $Sol^*$, $Sol'$, $Temp$)
20     if $Sol_0 < Sol^*$ then
21         FoundBestSol $\leftarrow$ True
22     end
23     Update $s_j (\forall j \in R \cup I)$
24     if Iter mod$\eta_{ALNS}$ = 0 then
25         Update $w_j$ and $p_j$ ($\forall j \in R \cup I$)
26     end
27     if Iter mod$\eta_{SA}$ = 0 then
28         if FoundBestSol = False then
29             NoImpr $\leftarrow$ NoImpr + 1
30         end
31         else
32             NoImpr $\leftarrow 0$
33         end
34         FoundBestSol $\leftarrow$ False
35         $Temp \leftarrow Temp \times \alpha$
36     end
37     Iter $\leftarrow$ Iter + 1
38 end
39 $Sol^* \leftarrow$ solve the set partitioning formulation
40 Return $Sol^*$
```

4.2. Phase 1: Adaptive large neighborhood search algorithm

Adaptive large neighborhood search (ALNS) removes nodes from the current solution by using destroy operators and reinserts these nodes in a more profitable position by using repair operators iteratively. This process is illustrated in Fig. 2. Let us consider an example with $|S| = 5$, $|C| = 6$, and $|V| = 5$. In Fig. 2(a), an initial solution consists of visiting Customers 3 and 4 in a non-optimal route. Additionally, there are three vehicles needed to perform the pickup process in supplier nodes. After applying a destroy operator to remove two customers and three suppliers from this solution, the updated yet incomplete solution is illustrated in Fig. 2(b). Subsequently, a repair operator is applied to the solution in order to re-insert these five nodes in more profitable positions, resulting in a better solution with a lower transportation cost and lesser number of vehicles used, illustrated in Fig. 2(c).

Instead of only using a single operator during the entire search process, ALNS employs multiple operators. However, the probability of selecting which operator to be used in a certain iteration depends on its performance in the previous iterations. The better the performance is (i.e. obtains a better objective function value), the higher its probability to be selected in the latter iterations.

In this phase, we aim to fill the Ω_s and Ω_c with any routes that are generated by ALNS, as illustrated in Fig. 3. Any routes observed in Fig. 2(a) are added to the pools, as illustrated in

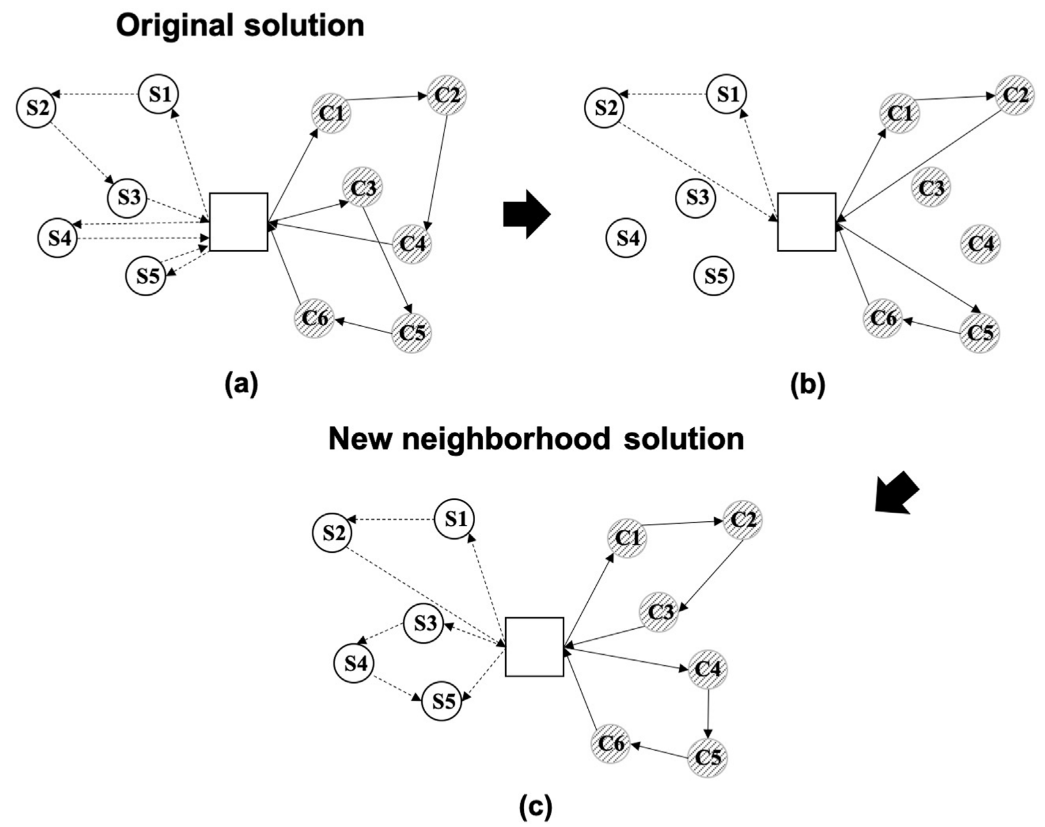

Fig. 2. Example on how ALNS finds neighborhood solution: (a) original solution, (b) incomplete solution after applying destroy operator, (c) new neighborhood solution after applying repair operator.

Fig. 3(a). Subsequently, when a destroy operator changes the solution as in Fig. 2(b), our pools are also updated with the newly observed routes, as shown in Fig. 3(b). We keep these routes since they could be part of a good solution. Finally, when a new neighborhood solution is constructed as in Fig. 2(c), these routes are added to the pools, as illustrated in Fig. 3(c). When doing this process, duplications are avoided (e.g. 0 - S1 - S2 - 0 is only kept once instead of twice, although it is observed twice during the process).

Let $R = \{R_1, R_2, \ldots, R_{|R|}\}$ be a set of destroy operators and $I = \{I_1, I_2, \ldots, I_{|I|}\}$ be a set of repair operators. The score s_j and weight w_j of each operator j ($j \in R \cup I$) is set such that the probability of choosing each operator j, p_j, in R and I is equally likely in the beginning (Line 6).

The first phase of the matheuristic is run until NoImpr reaches the threshold θ (Line 38). At each iteration, a destroy operator R_i ($R_i \in R$) is randomly selected to remove π nodes from Sol_0 (Lines 9–13). Subsequently, a repair operator I_i ($I_i \in I$) is selected to reinsert back the π removed nodes to the Sol_0 (Lines 14–18), resulting in a new neighborhood solution. In our implementation, we use $\pi = 5$. The operators are further explained in Section 4.3. Each of the removed nodes is only considered as a candidate to be inserted in a route of Sol_0 if it satisfies both the vehicle capacity and time horizon (T_{max}) constraints. Therefore, the feasibility of Sol_0 is guaranteed, unless some of the removed nodes cannot be inserted to any positions in Sol_0. If that happens, a high penalty value is added to the objective function value (total cost TC).

Sol_0 is directly accepted if its objective function value is better than Sol^* or Sol'. Otherwise, it will only be accepted with probability $exp(-\frac{Sol_0 - Sol'}{Temp})$ (Line 19). Furthermore, if a new best solution is found (i.e. Sol_0 improves Sol^*), FoundBestSol is set to True (Lines 20–22). Each of the operators' score s_j is then updated (Line 23) by Eq. (1), where $\delta_1 > \delta_2 > \delta_3$. We implemented 0.5, 0.33, and 0.17 for δ_1, δ_2, and δ_3 respectively.

$$s_j = \begin{cases} s_j + \delta_1, & \text{if } j \text{ is selected and the new solution is the best found solution so far} \\ s_j + \delta_2, & \text{if } j \text{ is selected and the new solution improves the current solution} \\ s_j + \delta_3, & \text{if } j \text{ is selected and the new solution does not improve the current solution, but it is accepted} \end{cases} \quad \forall j \in R \cup I \tag{1}$$

After η_{ALNS} iterations, each of the operators' weight w_j and probability p_j are updated (Lines 24–26). Operators' weight w_j is updated by Eq. (2), where γ refers to the reaction factor ($0 < \gamma < 1$) to control the influence of the recent success of an operator on its weight and χ_j is the frequency of using operator j.

$$w_j = \begin{cases} (1-\gamma)w_j + \gamma \frac{s_j}{\chi_j}, & \text{if } \chi_j > 0 \\ (1-\gamma)w_j, & \text{if } \chi_j = 0 \end{cases} \quad \forall j \in R \cup I \tag{2}$$

Operators' probability p_j is then updated by Eq. (3).

$$p_j = \begin{cases} \frac{w_j}{\sum_{k \in R} w_k} & \forall j \in R \\ \frac{w_j}{\sum_{k \in I} w_k} & \forall j \in I \end{cases} \tag{3}$$

If, after η_{SA} iterations, there is no better solution than Sol^*, NoImpr is increased by one (Lines 28–30), otherwise NoImpr is reset to zero (Lines 31–33). FoundBestSol is then set to False (Line 34) to see if in the subsequent η_{SA} iterations the algorithm is able to improve Sol^* and $Temp$ is decreased by α (Line 35). Iter is also increased by one (Line 37) to continue to the next iteration.

This entire phase is repeated until there is no solution improvement after θ successive temperature reductions. The Sol^*

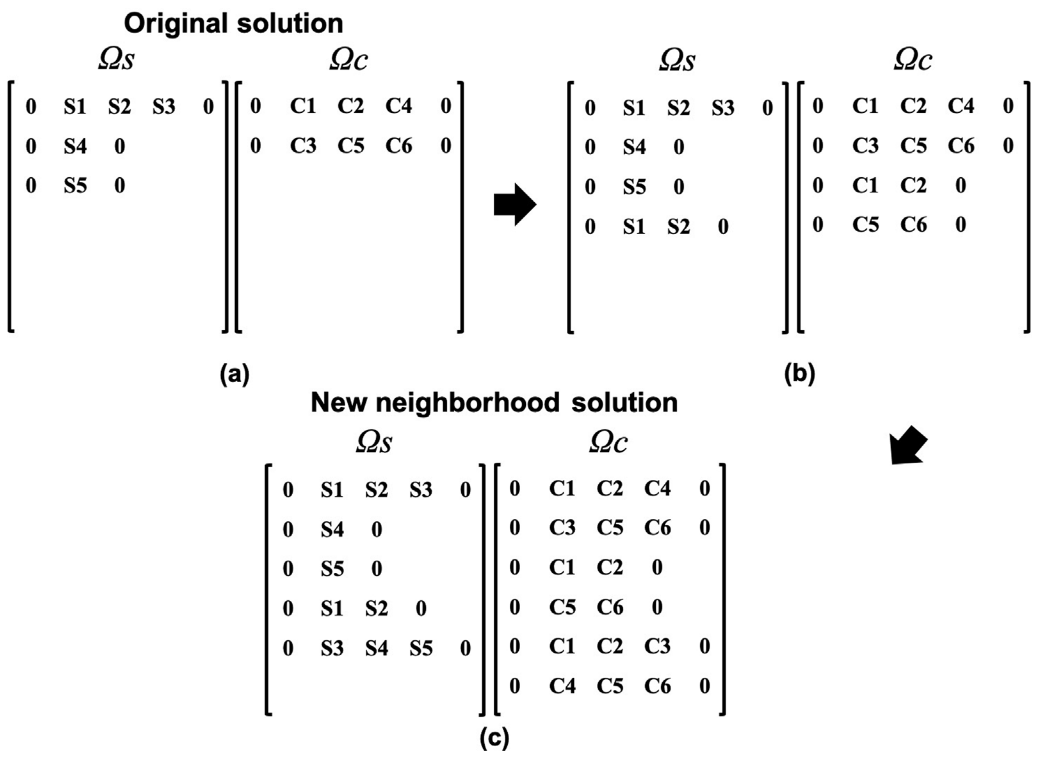

Fig. 3. Example on how observed routes are kept inside the pools during phase 1: (a) observed routes in original solution, (b) new routes from incomplete solution after applying destroy operator are added to pools, (c) new routes from neighborhood solution after applying repair operator are added to pools.

constructed by this phase becomes an upper bound of the VRPCD solution. It means that solving the following set partitioning formulation (Line 39) will yield a lower (or at least the same) objective function value as the Sol^* constructed by this phase.

4.3. Adaptive large neighborhood search operators

Below is the list of destroy and repair operators used in ALNS:
Random removal (R_1): remove a randomly selected node from Sol_0. Once R_1 is applied, RemovedNodes += 1.
Worst removal (R_2): remove a node with the x^{th} highest removal gain [12]. The removal gain is defined as the difference in objective function value between including and excluding a particular node. For each node in Sol_0, this cost is calculated and sorted in a descending order. x is determined by Eq. (4), in which the value is ranged between 1 to ξ, with $x = 1$ having the highest chance to be drawn, and the chance is decreasing for larger xs. It means that R_2 tries to remove a node which contributes a high cost if it is located in the current position. However, instead of always choosing the node with the highest removal gain (e.g. $x = 1$), R_2 also considers other nodes in order to diversify the removal process (in which $2 \le x \le \xi$), with a lower chance. We implement $p = 3$. y_1 is introduced to maintain the randomness, $y_1 \sim U(0, 1)$. ξ is the number of candidate nodes which is formally formulated in Eq. (5) case 1. By referring to Algorithm 1 (Lines 9 – 13), if we consider $|C| = 6$, $|S| = 4$ with RemovedNodes = 0, then, the number of candidate nodes that can be removed in the current iteration is $|C| + |S| - 0 = 10$. Once a node is removed, RemovedNodes += 1, then in the next iteration, the number of candidate nodes that can be removed becomes $|C| + |S| - 1 = 9$. It means that every time a node is removed from the Sol_0, then, the number of candidate nodes that can be removed from the Sol_0 in the next iteration is decreased.

$$x = \lceil y_1^p \times \xi \rceil \tag{4}$$

$$\xi = \begin{cases} |C| + |S| - \textsc{RemovedNodes}, & \text{for } R_2 \\ |C| + |S| - \textsc{RemovedNodes} - 2, & \text{for } R_4, R_5 \\ \textsc{RemovedNodes}, & \text{for } I_9 \end{cases} \tag{5}$$

Route removal (R_3): select a vehicle randomly and remove its z visited nodes. The value of z is determined by Eq. (6), where β is the number of nodes visited by the corresponding vehicle. Therefore, once R_3 is applied, RemovedNodes += z. R_3 is implemented to speed up the removal process compared to the one of R_1. Take note that implementing R_1 requires at least π iterations (see Lines 9 – 13 of Algorithm 1). On the other hand, the implementation of R_3 may only require one iteration if $\beta \ge \pi$ (see Eq. (6) case 1). The worst case is that if all vehicles only visit one node each, then, the implementation of R_3 would also need π iterations (see Eq. (6) case 2).

$$z = \begin{cases} \pi, & \text{if } \beta \ge \pi \\ \beta, & \text{otherwise} \end{cases} \tag{6}$$

Node pair removal (R_4): remove a pair of nodes with the x^{th} highest additional transportation cost. x is determined by Eq. (4) while ξ is determined by Eq. (5) case 2. However, it should be noted that applying R_4 means that two nodes are removed simultaneously from the Sol_0 in one iteration and therefore, RemovedNodes is increased by 2. When applying R_4 is impossible (e.g. when RemovedNodes $= 4$ while $\pi = 5$), then, we apply R_1 instead. R_4 tries to remove two adjacent nodes with a high transportation cost from the Sol_0, such that when repair reinserts them back to Sol_0, they can be located in better, and probably separated, positions.
Worst pair removal (R_5): similar to R_2, but instead of only one node, R_5 chooses a pair of nodes. The underlying difference between R_4 and R_5 is that R_4 only focuses in the transportation cost between two nodes, while R_5 considers the overall costs. Same

as R_4, applying R_5 means that two nodes are removed simultaneously from the Sol_0 in one iteration and therefore, RemovedNodes is increased by 2. When applying R_5 is impossible, we apply R_1 instead. x is determined by Eq. (4) while ξ is determined by Eq. (5) case 2.

Shaw removal (R_6): remove a node that is highly related with other removed nodes in a predefined way. R_6 tries to remove some similar nodes, such that it is easier to replace the positions of one another during the repair process. The last removed node is denoted as node i while the next candidate of the removed node is denoted as node j. The relatedness value (φ_j) of node j to node i is calculated by Eq. (7). R_6 starts by randomly selecting a node to be removed, set this node as i, and then calculating φ_j for the remaining nodes in Sol_0. The next removed node is the node with the lowest φ_j. Since R_6 removes one node at a time, RemovedNodes is increased by 1. The ϕ_1 to ϕ_4 are weights given to each of the relatedness components, in which we set as 0.25 each. $l_{ij} = -1$ if nodes i and j are in the same vehicle; 1 otherwise.

$$\varphi_j = \begin{cases} \phi_1 c'_{ij} + \phi_2 t'_{ij} + \phi_3 l_{ij} + \phi_4 |P_i - P_j|, & \text{if } i \in S \\ \phi_1 c''_{ij} + \phi_2 t''_{ij} + \phi_3 l_{ij} + \phi_4 |D_i - D_j|, & \text{if } i \in C \end{cases} \tag{7}$$

Greedy insertion (I_1): insert a node to a position with the lowest insertion cost. The insertion cost is defined as the difference in objective function values between after and before inserting a node to a particular position. For each of the removed nodes, the insertion cost to all possible positions is calculated and sorted in an ascending order. A node with the lowest insertion cost is then selected and inserted to the corresponding position. Once I_1 inserts one node to the Sol_0, RemovedNodes −= 1.

***k*-regret insertion** (I_2, I_3, I_4): a regret value is defined as the difference in objective function values when node j is inserted in the best position (denoted as $TC_1(j)$) and in the k-best position (denoted as $TC_k(j)$). The idea is to select a node which leads to the largest regret value if it is not inserted in its best position, which is formally formulated in Eq. (8). This node is then inserted in its best position. In other words, this operator tries to insert the node that we will regret the most if it is not inserted now [23]. We implement $k = 2, 3$ and 4. Once I_2, I_3, or I_4 inserts one node to the Sol_0, RemovedNodes −= 1.

$$\underset{j \in \text{RemovedNodes}}{\text{argmax}} \left\{ \sum_{i=2}^{k} (TC_i(j) - TC_1(j)) \right\} \tag{8}$$

Greedy insertion with noise function (I_5): an extension of I_1. A noise function is applied to the objective function value by Eq. (9) when selecting the best position of a node [24], where $\bar{e}$ is the maximum transportation cost between nodes (problem-dependent), μ is a noise parameter that we set to 0.1, and $y_2 \sim U(-1, 1)$.

$$TC_{new} = TC + \bar{e} \times \mu \times y_2 \tag{9}$$

***k*-regret insertion with noise function** (I_6, I_7, I_8): an extension of I_2, I_3, and I_4 by applying a noise function to the objective function value by Eq. (9) when calculating the regret value [24].

GRASP insertion (I_9): similar to I_1, but instead of choosing a node with the lowest insertion cost, I_9 chooses the node with the x^{th} lowest insertion cost. x is determined by Eq. (4) while ξ is determined by Eq. (5) case 3.

4.4. Phase 2: Set partitioning formulation for the VRPCD

Phase 2 focuses on solving the set partitioning model to determine the final solution: optimal routes for pickup and delivery processes. Recall that Ω_s contains a set of candidate routes in pickup process $\Omega_s = \{1, 2, \ldots, |\Omega_s|\}$ and Ω_c contains a set of candidate routes in delivery process $\Omega_c = \{1, 2, \ldots, |\Omega_c|\}$. Each of the candidate routes r in Ω_s is associated to a transportation cost of c'_r and a transportation time of t'_r, while each of the candidate routes r in Ω_c is associated to a transportation cost of c''_r and a transportation time of t''_r. It should be noted that all candidate routes satisfy the vehicle capacity constraint, which has been handled during the route construction process by the repair operators.

Let a'_{ir} be a binary parameter equal to 1 if route r visits node i; 0 otherwise ($r \in \Omega_s, i \in S$) and a''_{ir} be a binary parameter equal to 1 if route r visits node i; 0 otherwise ($r \in \Omega_c, i \in C$). Several decision variables in the set partitioning formulation are as follows:

- x'_r equals to 1 if route r is selected; 0 otherwise ($r \in \Omega_s$).
- x''_r equals to 1 if route r is selected; 0 otherwise ($r \in \Omega_c$).
- Tp_{max} records the maximum transportation time for the pickup process.
- Td_{max} records the maximum transportation time for the delivery process.

$$Min \sum_{r \in \Omega_s} c'_r x'_r + \sum_{r \in \Omega_c} c''_r x''_r + H \left(\sum_{r \in \Omega_s} x'_r + \sum_{r \in \Omega_c} x''_r \right) \tag{10}$$

$$\sum_{r \in \Omega_s} a'_{ir} x'_r = 1 \quad \forall i \in S \tag{11}$$

$$\sum_{r \in \Omega_c} a''_{ir} x''_r = 1 \quad \forall i \in C \tag{12}$$

$$\sum_{r \in \Omega_s} x'_r + \sum_{r \in \Omega_c} x''_r \le |V| \tag{13}$$

$$t'_r x'_r \le Tp_{max} \quad \forall r \in \Omega_s \tag{14}$$

$$t''_r x''_r \le Td_{max} \quad \forall r \in \Omega_c \tag{15}$$

$$Tp_{max} + Td_{max} \le T_{max} \tag{16}$$

The objective is to minimize the total of transportation and operational costs, which is formulated in Eq. (10). All supplier and customer nodes must be visited, as required in Eqs. (11) and (12) respectively. Eq. (13) limits the number of selected routes (i.e. does not exceed the number of available vehicles). Eqs. (14) and (15) record the maximum transportation time in pickup and delivery process respectively. Finally, the two processes must be done within the time horizon, as expressed in Eq. (16).

We acknowledge that the proposed matheuristic is designed for the VRPCD that considers the processes inside the cross-dock facility (e.g. loading, unloading, and sorting processes) to be fast enough (see Section 3), which might not be the case in practice. Without considering those processes, a vehicle might be waiting (ready for the loading process) while the products are still moved/prepared inside the cross-dock facility. Then, further cost reductions might be achieved by eliminating this vehicle waiting time, if the decision on when the vehicle should be ready in the cross-dock (respecting the arrival of the last vehicle picking up products and handling time) is taken into consideration. This can be adapted with a slight modification by specifying a time (t_h) that should be added between the arrival of the last vehicle picking up products from suppliers and the departure of the first vehicle distributing products to customers. Subsequently, Eq. (16) can be replaced by Eq. (17).

$$Tp_{max} + Td_{max} + t_h \le T_{max} \tag{17}$$

Table 2
Matheuristic parameter values.

Parameter	Value
γ	**0.7**, 0.8, 0.9
θ	10, **20**, 30
T_0	300, **500**, 700
α	0.85, **0.9**, 0.95
η_{ALNS}	**200**, 300, 400
η_{SA}	$(\lvert\mathbf{S}\rvert + \lvert\mathbf{C}\rvert) \times \mathbf{2}$, $(\lvert S\rvert + \lvert C\rvert) \times 3$, $(\lvert S\rvert + \lvert C\rvert) \times 4$

Table 3
VRPCD parameter values [4].

Parameter	Set 1	Set 2	Set 3
$\lvert S\rvert$	4	7	12
$\lvert C\rvert$	6	23	38
$\lvert V\rvert$	10	20	30
Q	70	150	150
H	1000	1000	1000
T_{max}	960	960	960
c'_{ij}, c''_{ij}	U~(48,560)	U~(48,480)	U~(48,560)
t'_{ij}, t''_{ij}	U~(20,200)	U~(20,100)	U~(20,200)
P_i, D_i	U~(5,50)	U~(5,20)	U~(5,30)

5. Computational results

In this section, we first present the experimental setup of this study, which includes the details about the environment used, the selected parameter configurations of the proposed matheuristic, the benchmark instances and the metrics used. Section 5.2 presents the experimental results on the given benchmark instances and a thorough analysis. The results of the newly proposed larger instances solved by the matheuristic and a pure ALNS are presented in Section 5.3. Finally, the importance of solving the set partitioning formulation, implementing the adaptive scheme, and analyzing the ALNS operators is discussed in Sections 5.4, 5.5, and 5.6 respectively.

5.1. Experimental setup

The proposed matheuristic is implemented in C++ with CPLEX 12.9.0.0 to solve the set partitioning formulation. All experiments were performed on a computer with Intel Core i7-8700 CPU @ 3.20 GHz processor, 32.0 GB RAM.

For the purpose of tuning parameters, we used the OFAT (one-factor-at-a-time) method and randomly selected instances with various values of parameters, as listed in Table 2. The best values for parameters are highlighted in **bold** which yield good quality of solutions. The following experiments are then conducted based on this setting.

To assess the performance of our proposed matheuristic, we use the available benchmark VRPCD instances which were introduced by Lee et al. [4]. The benchmark VRPCD instances consist of three problem sets, which are differentiated based on the number of nodes: 10-nodes, 30-nodes, and 50-nodes. The parameter values are listed in Table 3. The benchmark VRPCD instances can be downloaded at http://web.ntust.edu.tw/~vincent/ovrpcd/.

In order to further assess our matheuristic performance, we adopted larger instances with up to 300 nodes (150 customer nodes and 150 supplier nodes) from VRPCDTW instances [7]. The instances are grouped into five sets with 40, 60, 100, 200 and 300 nodes each. Due to some differences between VRPCD and VRPCDTW problems, some modifications are required:

- Time windows constraints in VRPCDTW instances are ignored.
- Parameters H and $\lvert V\rvert$ are added.

Table 4
New VRPCD instances parameter values.

Parameter	Set 4	Set 5	Set 6	Set 7	Set 8
$\lvert S\rvert$, $\lvert C\rvert$, $\lvert V\rvert$	20	30	50	100	150
Q			33		
H			\$100		
T_{max}			16 h		
v			60 km/h		
Total nodes	40	60	100	200	300
Number of instances	11	5	5	5	5

Furthermore, instead of directly providing the travel time and transportation cost for connecting two nodes i and j in the network, VRPCDTW instances provide each node's coordinate (x,y) in meters and the vehicle speed (v). Therefore, the travel distance between nodes i and j, denoted as s'_{ij} (for connecting supplier nodes) and s''_{ij} (for connecting customer nodes), is calculated by the euclidean distance and converted to kilometers. Consequently, to derive the travel time and transportation cost, we use Eqs. (18) and (19) respectively. Because of the above-mentioned differences, comparing our results with VRPCDTW results (e.g. [7, 12,15]) makes no sense. Detailed parameters are listed in Table 4. These large instances are available online on https://www.mech.kuleuven.be/en/cib/op/opmainpage#section-47.

$$\begin{cases} t'_{ij} = \lfloor \frac{\lfloor s'_{ij} \rfloor}{v} \rfloor, & \forall i,j \in S \\ t''_{ij} = \lfloor \frac{\lfloor s''_{ij} \rfloor}{v} \rfloor, & \forall i,j \in C \end{cases} \tag{18}$$

$$\begin{cases} c'_{ij} = \lfloor s'_{ij} \times \$1 \rfloor, & \forall i,j \in S \\ c''_{ij} = \lfloor s''_{ij} \times \$1 \rfloor, & \forall i,j \in C \end{cases} \tag{19}$$

5.2. Results for the benchmark VRPCD instances

The results for the benchmark VRPCD instances are summarized in Tables 5–7. Our matheuristic results are compared to those of the state-of-the-art algorithms: Tabu search (TS) [4], improved tabu search (imp-TS) [1] and simulated annealing (SA) [6]. For Set 1, we solved the mathematical model [4] using CPLEX to obtain the optimal solutions. Take note that none of the state-of-the-art algorithms report the optimal solutions. They only report the average results where the best known solutions (BKS) are consolidated from all of them. For this set, since the optimal solution is available for each instance, solutions should be compared to these optimal solutions and the BKS is not very useful anymore.

For each algorithm, the average gaps from the optimal solutions are calculated by Eq. (20). $\overline{TC}_{algorithm}$ represents the average objective function value by a particular algorithm. We observe that our matheuristic performs best and it is able to obtain all optimal solutions with less CPU time. Table 5 summarizes the results of Set 1 instances.

$$Gap\ (\%) = \frac{(\overline{TC}_{algorithm} - Opt)}{Opt} \times 100 \tag{20}$$

For Sets 2 and 3, CPLEX is unable to generate the optimal solutions. Therefore, the gap values are calculated by comparing the average results of the different algorithms with the BKS. Tables 6 and 7 summarize results of Sets 2 and 3 respectively. On average, our matheuristic is able to further improve the best known solutions by 14.0% and 22.5% respectively. The CPU times required are lower than others, except imp-TS [1]. In addition, we are able to improve all best known solutions. The new best known solutions are made available online on https://www.mech.kuleuven.be/en/cib/op/opmainpage#section-47.

Table 5
Total cost comparison of the proposed matheuristic, state-of-the-art algorithms and CPLEX for Set 1.

Instance	TS [4]		imp-TS [1]		SA [6]		BKS	CPLEX		Matheuristic			Gap (%)			
	$\overline{TC}$	CPU (s)	$\overline{TC}$	CPU (s)	$\overline{TC}$	CPU (s)		Opt	CPU (s)	$\overline{TC}$	Best	CPU (s)	[4]	[1]	[6]	Matheuristic
1	7571.4	1.52	6847.6	0.22	6953.0	1.91	6847.6	6823	0.92	6823.0	6823	0.26	11.0	0.4	1.9	0.0
2	7103.7	1.74	6816.8	0.23	6741.0	1.78	6741.0	6741	1.36	6741.0	6741	0.19	5.4	1.1	0.0	0.0
3	9993.5	2.37	9615.6	0.19	9269.0	2.44	9269.0	9269	1.36	9269.0	9269	0.18	7.8	3.7	0.0	0.0
4	8338.0	1.60	7289.7	0.27	7255.0	1.75	7255.0	7229	1.09	7229.0	7229	0.18	15.3	0.8	0.4	0.0
5	8709.9	2.28	6599.0	0.21	6524.0	1.75	6524.0	6475	0.86	6475.0	6475	0.18	34.5	1.9	0.8	0.0
6	9143.5	1.82	9324.6	0.03	7613.0	1.81	7613.0	7434	0.66	7434.0	7434	0.13	23.0	25.4	2.4	0.0
7	12721.2	2.80	12083.0	0.01	11990.0	2.77	11990.0	11713	0.47	11713.0	11713	0.14	8.6	3.2	2.4	0.0
8	9275.7	1.85	8719.6	0.04	8158.0	2.21	8158.0	8158	1.26	8158.0	8158	0.16	13.7	6.9	0.0	0.0
9	8096.5	2.04	7362.2	0.25	7120.0	1.75	7120.0	6989	1.22	6989.0	6989	0.15	15.8	5.3	1.9	0.0
10	7044.8	1.82	6204.5	0.36	6056.0	2.05	6056.0	5960	1.37	5960.0	5960	0.16	18.2	4.1	1.6	0.0
11	8051.8	1.80	7635.3	0.00	7434.0	1.93	7434.0	6916	1.53	6916.0	6916	0.14	16.4	10.4	7.5	0.0
12	8661.0	1.72	7867.2	0.24	7800.0	1.76	7800.0	7656	2.90	7656.0	7656	0.16	13.1	2.8	1.9	0.0
13	7370.2	1.54	7097.9	0.24	6934.0	1.88	6934.0	6783	3.31	6783.0	6783	0.16	8.7	4.6	2.2	0.0
14	7132.3	1.53	5208.0	0.00	4704.0	1.69	4704.0	4417	0.75	4417.0	4417	0.15	61.5	17.9	6.5	0.0
15	7563.4	1.61	7103.2	0.41	7088.0	1.75	7088.0	7072	2.17	7072.0	7072	0.15	6.9	0.4	0.2	0.0
16	9983.6	2.05	8768.7	0.03	8616.0	2.05	8616.0	8440	3.98	8440.0	8440	0.13	18.3	3.9	2.1	0.0
17	9538.1	2.26	9003.0	0.06	9003.0	2.60	9003.0	9003	2.04	9003.0	9003	0.15	5.9	0.0	0.0	0.0
18	8057.4	1.74	6887.5	0.19	6911.0	1.88	6887.5	6760	2.39	6760.0	6760	0.15	19.2	1.9	2.2	0.0
19	9042.6	2.21	7123.0	0.03	7051.0	1.92	7051.0	7051	4.59	7051.0	7051	0.13	28.2	1.0	0.0	0.0
20	10478.0	2.55	10471.0	0.00	10004.0	2.32	10004.0	9786	1.39	9786.0	9786	0.14	7.1	7.0	2.2	0.0
21	8380.5	2.06	5431.4	0.00	4753.0	1.51	4753.0	4644	1.69	4644.2	4644	0.16	80.5	17.0	2.3	0.0
22	9016.9	2.42	6908.0	0.01	6442.0	1.85	6442.0	6442	3.78	6442.0	6442	0.13	40.0	7.2	0.0	0.0
23	9489.2	2.31	9224.1	0.11	9156.0	2.28	9156.0	9156	2.96	9156.0	9156	0.16	3.6	0.7	0.0	0.0
24	12513.6	2.64	11976.0	0.00	11976.0	2.78	11976.0	11976	1.79	11976.0	11976	0.12	4.5	0.0	0.0	0.0
25	7114.3	1.68	6638.0	0.10	6346.0	1.94	6346.0	6346	5.69	6346.0	6346	0.16	12.1	4.6	0.0	0.0
26	8421.3	2.04	7216.9	0.03	6880.0	1.86	6880.0	6817	5.46	6817.0	6817	0.16	23.5	5.9	0.9	0.0
27	10666.8	2.47	9709.8	0.06	9541.0	2.39	9541.0	9541	5.69	9541.0	9541	0.15	11.8	1.8	0.0	0.0
28	10123.3	2.69	7408.0	0.01	7107.0	1.84	7107.0	6782	4.85	6782.0	6782	0.13	49.3	9.2	4.8	0.0
29	7503.2	1.73	6748.5	0.18	6762.0	1.86	6748.5	6591	4.04	6591.0	6591	0.17	13.8	2.4	2.6	0.0
30	7642.6	1.82	7304.4	0.09	6942.0	3.51	6942.0	6919	7.18	6919.0	6919	0.16	10.5	5.6	0.3	0.0
Average		2.02		0.12		2.06			2.62			0.16	19.6	5.2	1.6	0.0

Table 6
Total cost comparison of the proposed matheuristic and state-of-the-art algorithms for Set 2.

Instance	TS [4]		imp-TS [1]		SA [6]		BKS	Matheuristic			Gap (%)			
	$\overline{TC}$	CPU (s)	$\overline{TC}$	CPU (s)	$\overline{TC}$	CPU (s)		$\overline{TC}$	Best	CPU (s)	[4]	[1]	[6]	Matheuristic
1	12366.7	3.00	7692.9	0.37	7550.2	2.80	7550.2	6620.0	6401	1.87	63.8	1.9	0.0	−12.3
2	14173.0	3.55	7787.2	0.16	7832.7	2.90	7787.2	6658.2	6593	1.77	82.0	0.0	0.6	−14.5
3	13836.8	4.32	7893.6	0.43	7747.4	3.00	7747.4	6626.3	6510	1.82	78.6	1.9	0.0	−14.5
4	10995.4	2.09	7792.2	0.23	7677.4	3.00	7677.4	6377.7	6259	1.36	43.2	1.5	0.0	−16.9
5	11757.8	2.26	7224.8	0.39	7579.7	3.50	7224.8	6289.6	6105	2.01	62.7	0.0	4.9	−12.9
6	11027.7	2.10	7245.9	0.22	7053.0	3.10	7053.0	5656.6	5617	1.21	56.4	2.7	0.0	−19.8
7	11899.2	2.61	8206.9	0.11	7720.1	3.00	7720.1	6865.1	6679	2.41	54.1	6.3	0.0	−11.1
8	12825.5	3.00	7880.9	0.16	7709.8	3.30	7709.8	6606.3	6351	2.04	66.4	2.2	0.0	−14.3
9	12718.6	3.10	8157.3	0.16	7882.5	2.80	7882.5	6820.1	6569	1.54	61.4	3.5	0.0	−13.5
10	11794.7	2.38	7924.7	0.20	7734.9	2.60	7734.9	6729.0	6492	1.77	52.5	2.5	0.0	−13.0
11	12094.9	3.03	7452.6	0.29	7721.7	3.00	7452.6	6421.3	6322	1.26	62.3	0.0	3.6	−13.8
12	12132.5	2.64	8320.0	0.16	7899.8	2.90	7899.8	6698.1	6631	1.49	53.6	5.3	0.0	−15.2
13	13223.4	2.92	8222.7	0.15	7863.7	2.90	7863.7	6766.3	6628	1.63	68.2	4.6	0.0	−14.0
14	12413.9	2.76	8211.7	0.18	8141.1	3.60	8141.1	6775.4	6695	1.44	52.5	0.9	0.0	−16.8
15	12521.4	3.06	8144.6	0.38	7941.6	3.10	7941.6	6711.4	6535	1.67	57.7	2.6	0.0	−15.5
16	12044.4	3.15	7451.7	0.28	7901.9	3.10	7451.7	6763.9	6719	1.58	61.6	0.0	6.0	−9.2
17	12699.4	2.14	8086.2	0.34	8055.0	3.00	8055.0	6709.7	6612	1.43	57.7	0.4	0.0	−16.7
18	11001.4	1.70	7576.0	0.28	7798.3	2.90	7576.0	6655.3	6541	1.53	45.2	0.0	2.9	−12.2
19	12724.4	2.77	7871.2	0.36	7964.3	3.10	7871.2	6788.5	6562	1.80	61.7	0.0	1.2	−13.8
20	12357.7	2.72	7883.7	0.24	7522.4	2.50	7522.4	6775.9	6382	2.06	64.3	4.8	0.0	−9.9
21	13177.0	3.39	7914.1	0.26	7886.2	3.40	7886.2	6655.6	6489	1.49	67.1	0.4	0.0	−15.6
22	11545.0	2.43	8005.3	0.35	7841.1	2.90	7841.1	6694.7	6566	1.55	47.2	2.1	0.0	−14.6
23	12308.1	2.93	7883.5	0.38	7791.5	3.20	7791.5	6574.4	6479	1.58	58.0	1.2	0.0	−15.6
24	12722.7	2.87	7731.2	0.48	7957.8	3.00	7731.2	6611.9	6537	1.56	64.6	0.0	2.9	−14.5
25	12844.9	2.67	7884.8	0.20	7839.4	3.00	7839.4	6766.7	6490	1.55	63.9	0.6	0.0	−13.7
26	13297.5	3.31	8001.6	0.16	7846.2	3.30	7846.2	6956.1	6703	2.36	69.5	2.0	0.0	−11.3
27	13415.2	3.25	8899.4	0.17	8128.6	3.30	8128.6	7164.1	6733	1.64	65.0	9.5	0.0	−11.9
28	12613.0	2.60	10131.0	0.00	8367.5	2.80	8367.5	7266.2	7191	2.09	50.7	21.1	0.0	−13.2
29	12840.8	3.28	8276.9	0.38	8003.1	2.90	8003.1	6900.8	6692	1.90	60.4	3.4	0.0	−13.8
30	13796.2	3.78	8251.6	0.28	7760.9	2.70	7760.9	6630.5	6520	1.66	77.8	6.3	0.0	−14.6
Average		2.86		0.26		3.02				1.70	61.0	2.9	0.7	−14.0

Table 7
Total cost comparison of the proposed matheuristic and state-of-the-art algorithms for Set 3.

Instance	TS [4]		imp-TS [1]		SA [6]		BKS	Matheuristic			Gap (%)			
	$\overline{TC}$	CPU (s)	$\overline{TC}$	CPU (s)	$\overline{TC}$	CPU (s)		$\overline{TC}$	Best	CPU (s)	[4]	[1]	[6]	Matheuristic
1	24284.6	5.62	20704.6	0.49	19804.5	7.49	19804.5	15905.4	15731	7.06	22.6	4.5	0.0	−19.7
2	23435.6	5.73	20816.8	0.64	18248.1	6.62	18248.1	14110.1	13687	6.31	28.4	14.1	0.0	−22.7
3	23449.4	5.80	19612.2	0.30	18133.9	7.10	18133.9	13872.0	13554	5.89	29.3	8.2	0.0	−23.5
4	23471.1	5.87	19549.0	0.44	19083.6	6.89	19083.6	15087.1	14624	5.74	23.0	2.4	0.0	−20.9
5	23406.2	7.28	20448.0	0.61	18877.3	7.09	18877.3	14664.3	14091	6.74	24.0	8.3	0.0	−22.3
6	24026.6	5.38	21212.0	0.56	19783.0	7.47	19783.0	15206.3	14240	8.26	21.5	7.2	0.0	−23.1
7	24190.0	7.65	20640.2	0.55	19690.0	6.11	19690.0	15946.7	15689	7.06	22.9	4.8	0.0	−19.0
8	23158.9	9.88	20664.1	0.42	18939.2	6.91	18939.2	14643.2	14108	5.14	22.3	9.1	0.0	−22.7
9	23594.7	5.54	18920.0	0.38	18510.6	7.11	18510.6	14163.4	14009	6.01	27.5	2.2	0.0	−23.5
10	23530.5	5.77	20384.2	0.52	19607.3	7.10	19607.3	15754.5	15248	7.00	20.0	4.0	0.0	−19.6
11	23371.7	5.37	19941.6	0.37	18675.8	7.15	18675.8	14703.5	14368	6.98	25.1	6.8	0.0	−21.3
12	21082.8	4.46	17258.4	0.17	17550.3	6.75	17258.4	13126.3	12710	5.66	22.2	0.0	1.7	−23.9
13	21610.7	4.62	17829.9	0.16	18039.2	6.32	17829.9	13366.3	12559	6.03	21.2	0.0	1.2	−25.0
14	23397.9	5.44	19845.2	0.53	18252.5	7.38	18252.5	13685.6	13190	5.87	28.2	8.7	0.0	−25.0
15	24041.9	6.31	21863.0	0.55	19803.8	7.97	19803.8	15445.9	15015	6.99	21.4	10.4	0.0	−22.0
16	22893.4	5.18	20144.2	0.30	18808.3	6.95	18808.3	14304.4	13614	6.79	21.7	7.1	0.0	−23.9
17	22950.4	6.93	20093.3	0.44	18713.8	6.72	18713.8	14628.0	13853	5.48	22.6	7.4	0.0	−21.8
18	24358.2	6.25	20244.8	0.53	18579.7	7.07	18579.7	14291.9	13683	7.85	31.1	9.0	0.0	−23.1
19	25068.7	5.68	19955.0	0.28	18453.2	7.93	18453.2	14956.0	14554	7.16	35.9	8.1	0.0	−19.0
20	23232.1	4.79	19267.7	0.36	18167.3	7.56	18167.3	13932.5	13518	5.99	27.9	6.1	0.0	−23.3
21	22564.8	5.43	19533.4	0.61	19226.0	6.97	19226.0	14264.4	13838	6.83	17.4	1.6	0.0	−25.8
22	24360.7	6.04	19032.1	0.37	18551.6	7.83	18551.6	14170.5	13740	6.35	31.3	2.6	0.0	−23.6
23	24377.9	5.88	20562.5	0.51	18514.8	7.35	18514.8	14463.9	14175	5.68	31.7	11.1	0.0	−21.9
24	22008.7	5.36	19288.2	0.05	18558.5	6.81	18558.5	13589.9	13185	5.76	18.6	3.9	0.0	−26.8
25	24256.6	5.76	19695.9	0.33	18574.2	7.55	18574.2	14512.4	14212	6.78	30.6	6.0	0.0	−21.9
26	23424.9	5.05	20610.5	0.14	18995.7	7.80	18995.7	14554.5	13871	6.31	23.3	8.5	0.0	−23.4
27	22961.4	5.17	18942.8	0.31	18128.7	7.12	18128.7	14571.2	14300	7.41	26.7	4.5	0.0	−19.6
28	23822.3	5.56	20097.3	0.39	18952.4	7.41	18952.4	14332.3	13875	6.11	25.7	6.0	0.0	−24.4
29	23678.3	64.84	22248.1	0.20	19056.1	6.96	19056.1	14713.1	13948	5.19	24.3	16.8	0.0	−22.8
30	23149.8	5.91	19321.9	0.65	18268.6	8.29	18268.6	14451.7	14022	5.83	26.7	5.8	0.0	−20.9
Average		7.82		0.41		7.19				6.41	25.2	6.5	0.1	−22.5

5.3. Results for the new large VRPCD instances

As mentioned earlier, sets of larger instances are introduced in this paper. In order to have some reference to compare with, we run a pure ALNS algorithm with the same CPU time as the matheuristic uses. The results are presented in Table 8. We also calculate the gap between the matheuristic and the ALNS average objective function values.

It can be concluded that ALNS alone is slightly outperformed by the matheuristic. Table 9 summarizes the average results for each set of large instances. The matheuristic is able to produce better solutions especially for large instances, e.g. 100, 200 and 300 nodes. We remark the importance of solving the set partitioning formulation towards getting better solution quality, which will be analyzed further in Section 5.4. By referring to the CPU time, our proposed matheuristic is able to tackle up to a total of 100 nodes within half a minute of CPU time on average. When the total number of nodes is increased up to 200 and 300, it requires around nine minutes and 40 min of CPU time on average. This is due to a long checking process in the first phase of the matheuristic (the ALNS part) when the operators are applied to modify the solution. Anyway, these computation times seem acceptable for practical applications. The matheuristic results on this new larger instances are made available online on https://www.mech.kuleuven.be/en/cib/op/opmainpage#section-47.

5.4. The importance of solving the set partitioning formulation

We conducted additional experiments in order to assess the added value of the set partitioning compared to a pure metaheuristic approach, ALNS. The improvement made by our proposed matheuristic compared to a pure ALNS is calculated by Eq. (21). The improvement of each instance is then plotted in Fig. 4.

$$\mathit{Improvement}\ (\%) = \frac{(\overline{TC}_{matheuristic} - \overline{TC}_{ALNS})}{\overline{TC}_{ALNS}} \times 100 \qquad (21)$$

From Fig. 4 (top) we observe that ALNS itself generates good solutions for small instances, i.e. Set 1 with 10-nodes. In this set of instances, ALNS may obtain exactly the same results as the matheuristic (ALNS + set partitioning). Therefore, the improvement is 0%. When the number of nodes is increased, e.g. 30-nodes (Set 2), the ALNS result deteriorates and thus solving the set partitioning helps to improve the results up to 0.4% on average, with an individual instance improvement ranged from 0% to 1.9% at most. Similar observations can be found when we further increase the number of nodes to 50 (Set 3). Solving the set partitioning improves the ALNS results up to 0.9% on average, with an individual instance improvement ranged from 0.2% to 1.7% at most.

We then evaluate this improvement on sets with larger instances (Sets 4 to 8), as plotted in Fig. 4 (bottom). It is observed that the improvement made by solving the set partitioning is even larger. It improves the results up to 0.9% on average, with an individual instance improvement ranged from 0.1% to 2.6% at most. However, solving the set partitioning of course takes longer CPU time. For solving all instances, it is increased to 125 s from 113 s on average. Anyway, a trade-off can be made here between solution quality and CPU time. According to us, this increase in CPU time from 113 s to 125 s is certainly acceptable given the increase in solution quality.

5.5. The importance of implementing the adaptive scheme

The first phase of our proposed matheuristic, which is the column generation by ALNS, employs an adaptive scheme for the

Table 8
Total cost comparison of the proposed matheuristic and ALNS for large instances.

Instance	$\overline{TC}$		Gap (%)	CPU (s)
	Matheuristic	ALNS		
40–a	3600.4	3623.0	0.6	3.64
40–b	4601.0	4601.2	0.0	1.20
40–c	3524.7	3539.0	0.4	1.55
40–d	5038.1	5056.0	0.4	1.07
40–e	3983.2	3990.6	0.2	1.43
40–f	4658.4	4675.7	0.4	1.25
40–g	4695.0	4696.5	0.0	1.52
40–h	4802.0	4802.3	0.0	3.14
40–i	4037.0	4037.4	0.0	1.52
40–j	4016.6	4030.3	0.3	1.34
40–k	4015.4	4066.0	1.3	1.06
60–a	5357.6	5362.0	0.1	105.29
60–b	6781.6	6803.7	0.3	4.01
60–c	7081.2	7229.8	2.1	4.68
60–d	5331.9	5336.5	0.1	105.91
60–e	6969.0	7024.9	0.8	4.14
100–a	9205.8	9218.3	0.1	138.18
100–b	10331.1	10594.1	2.5	47.37
100–c	10468.0	10554.4	0.8	36.39
100–d	9958.4	9972.1	0.1	131.78
100–e	10584.5	10698.3	1.1	28.36
200–a	18499.7	18564.6	0.4	557.30
200–b	20510.0	20653.1	0.7	474.30
200–c	19912.9	20202.8	1.5	484.19
200–d	19268.2	19543.6	1.4	601.65
200–e	19983.1	20397.3	2.1	431.08
300–a	28416.5	28512.5	0.3	2474.20
300–b	29635.5	30200.3	1.9	2191.69
300–c	28734.1	29104.1	1.3	2169.73
300–d	29366.6	29461.2	0.3	2408.02
300–e	28436.4	28745.3	1.1	2497.29

Table 9
Summary of new large instances.

Instance	$\overline{TC}$		Gap (%)	Avg. CPU (s)
	Matheuristic	ALNS		
Set 4 – 40 nodes	4270.2	4283.5	0.3	1.70
Set 5 – 60 nodes	6304.3	6351.4	0.7	44.80
Set 6 – 100 nodes	10109.6	10207.4	0.9	76.41
Set 7 – 200 nodes	19634.8	19872.3	1.2	509.70
Set 8 – 300 nodes	28917.8	29204.7	1.0	2348.19

operator selection. Each operator's performance upon generating a new neighborhood solution is evaluated. When it performs well, its score is increased, and so is its probability to be selected in the subsequent iterations. This approach allows the algorithm to learn and select which operators are more suitable to be used.

In this section, we compare this adaptive scheme and the static scheme. The static scheme is defined by giving the same probability to each operator to be selected during the search process despite of its performance. Hence, for this static scheme, p_j is calculated by the following Eq. (22).

$$p_j = \begin{cases} \frac{1}{|R|} & \forall j \in R \\ \frac{1}{|I|} & \forall j \in I \end{cases} \tag{22}$$

For each instance, we calculate the difference (%) made by implementing the adaptive scheme compared to the static scheme, as formulated in Eq. (23). $\overline{TC}_{adaptive}$ and $\overline{TC}_{static}$ refer to average costs by the adaptive and static schemes respectively. The results are summarized in Table 10.

$$Difference\ (\%) = \frac{(\overline{TC}_{adaptive} - \overline{TC}_{static})}{\overline{TC}_{adaptive}} \times 100 \tag{23}$$

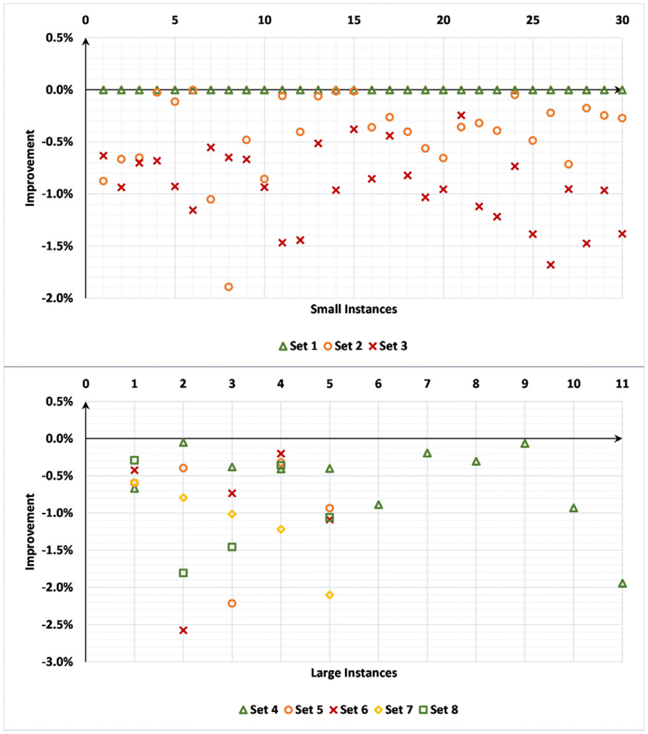

Fig. 4. Improvement by solving the set partitioning formulation.

Table 10
Results of adaptive and static schemes.

Instance	$\overline{TC}$		Difference	Avg. CPU (s)	
	Adaptive	Static	(%)	Adaptive	Static
Set 1	7529.6	7529.7	0.0	0.16	0.20
Set 2	6684.5	6685.3	0.0	1.70	1.99
Set 3	14513.9	15261.0	−4.1	6.41	7.17
Set 4	4270.2	4270.2	0.0	1.70	1.71
Set 5	6304.3	6304.5	0.0	44.80	45.34
Set 6	10109.6	10110.1	0.0	76.41	80.62
Set 7	19634.8	19647.1	−0.1	509.70	603.52
Set 8	28917.8	28920.7	0.0	2348.19	2536.01

We can conclude that only for Set 3 the adaptive scheme seems to make sense. Actually, a simplified algorithm with a static scheme, basic LNS, would have been sufficient to solve these instances and the added value of the adaptive scheme is very limited. Since the results for the adaptive scheme are always at least as good, and sometimes better, than the static scheme, we decided to report the results of the adaptive scheme in the rest of this paper. Moreover, the CPU time for the adaptive scheme is slightly better than for the static scheme. Finally, the adaptive scheme provides us with information about the performance of the different operators, which will be discussed in detail in the next section.

5.6. Analysis on ALNS operators

We introduce a total of 15 operators (Section 4.3) that are employed in our matheuristic, consisting of six destroy operators and nine repair operators. This huge amount of operator choices is actually linked to the adaptive scheme, where the operators will not be equally used since its selection is based on the performance during previous iterations. We initially set p_j to be equally likely for each operator (see Algorithm 1 Line 6). When operator j generates a better solution in a particular iteration,

Table 11
Results of employing all operators and removing some operators.

Instance	$\overline{TC}$			Gap (%)		Avg. CPU (s)		
	(i)	(ii)	(iii)	(ii)	(iii)	(i)	(ii)	(iii)
Set 1	7529.6	7529.7	7529.7	0.0	0.0	0.16	0.12	0.11
Set 2	6684.5	6729.8	6777.5	−0.6	−1.3	1.70	1.90	1.59
Set 3	14513.9	15184.5	15529.5	−4.0	−6.0	6.41	7.42	5.14
Set 4	4270.2	4270.3	4271.2	0.0	0.0	1.70	1.54	1.21
Set 5	6304.3	6305.2	6311.5	−0.1	−0.1	44.80	43.27	26.99
Set 6	10109.6	10110.1	10121.7	−0.1	−0.1	76.41	75.78	74.86
Set 7	19634.8	19637.2	19648.1	−0.1	−0.1	509.70	617.05	453.04
Set 8	28917.8	28941.9	28933.0	0.0	0.0	2348.19	3402.69	2310.53

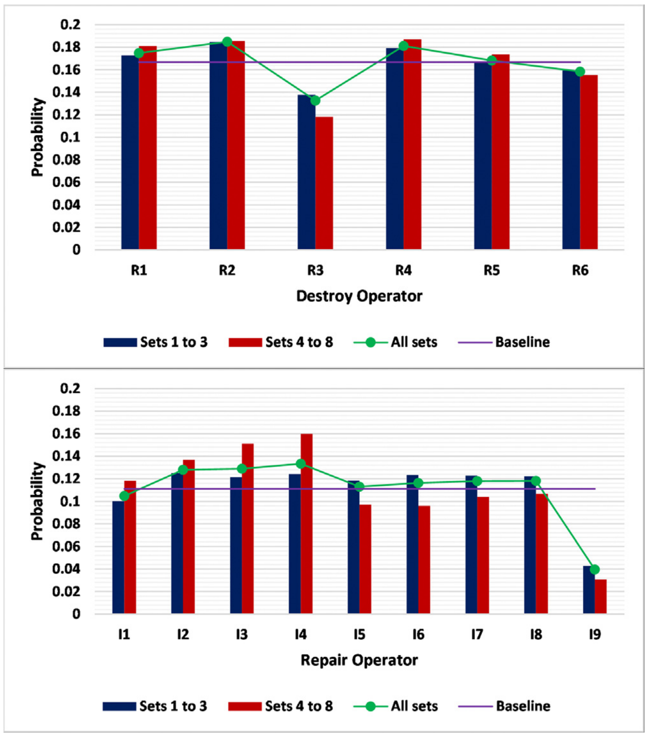

Fig. 5. Comparison of operators' probability.

p_j is then increased. In this manner, operators with a higher p_j value frequently produce good solutions compared to those with a lower p_j value. Hence, in order to analyze the performance of each operator, we record the p_j of each operator upon the matheuristic termination, and calculate the average value of it in small instances (Sets 1 to 3), large instances (Sets 4 to 8), and all sets (Sets 1 to 8). We then compare these values with the initial p_j (the so-called baseline) and plot the results in Fig. 5.

We observe a similar p_j pattern for all sets in the destroy operators, as shown in Fig. 5 (top). It seems that R_1, R_2 and R_4 frequently produce good solutions in contrast to operators R_3 and R_6. R_5 is around the baseline. From Fig. 5 (bottom), we see different patterns on how repair operators perform in small instances (Sets 1 to 3) and large instances (Sets 4 to 8). In our case, I_1 to I_4 play an important role in solving large instances, while I_5 to I_8 play an important role in solving small instances. I_9, on the other hand, seems less beneficial compared to any other repair operators both in small and large instances.

Based on these results, somewhat simplified matheuristics could be designed with (much) less operators. Therefore, we remove several operators to see whether it will influence the matheuristic performance in terms of solution quality and CPU time. We first try to remove a set of operators who are below the baseline, which are R_3 and R_6 for destroy operators, and I_1 and I_9 for repair operators (called as "alternative (ii)"). We also try to remove a larger set of operators, which are R_3, R_5, and R_6 for destroy operators, and I_1, I_5 to I_9 for repair operators (we call this "alternative (iii)"). We then compare both alternatives towards the original results when we use all operators (we call this "alternative (i)") by calculating the gap (%), as formulated in Eq. (24). The results for these experiments are summarized in Table 11.

$$Gap\ (\%) = \frac{(\overline{TC}_{alternative\ (i)} - \overline{TC}_{alternative\ (ii)\ or\ (iii)})}{\overline{TC}_{alternative\ (i)}} \times 100 \qquad (24)$$

Based on Table 11, we conclude that only for Sets 2 and 3 some changes in solution quality can be observed. Clearly, alternative (i) slightly outperforms alternatives (ii) and (iii). The operators with lower probability turn out to be still useful to explore the search space. Possibly surprising, the CPU times are also very similar for different alternatives. So, removing certain operators does not speed up the calculations. With all operators, alternative (i), better solutions can be found in the earlier phase and thus the algorithm terminates faster. With few operators, alternative (iii), it might be more difficult to escape from local optima. Then the algorithm sometimes terminates slightly faster, but with worse solutions.

6. Conclusion

We study the integration of the vehicle routing problem with cross-docking (VRPCD). A set of homogeneous vehicles is used to deliver products from a set of suppliers to a set of customers through a cross-dock facility. The aim is to select a set of vehicles to be used and its corresponding routes, such that the operational and transportation costs are minimized.

A matheuristic approach is proposed. It consists of two phases: adaptive large neighborhood search (ALNS) and set partitioning. ALNS is used to generate a list of candidate feasible routes. A total of 15 operators are employed, consisting of six destroy and nine repair operators. The selection of operators is based on the adaptive scheme where the probability of being selected in subsequent iterations increases if the operators perform well. A set partitioning formulation is then developed to find the best route combination with respect to other VRPCD constraints, e.g. the limited number of vehicles and the total transportation time available.

Computational results show that the proposed matheuristic clearly outperforms the state-of-the-art algorithms in terms of solution quality. It is able to obtain optimal solutions for all Set 1 instances. In summary, it improves the best known solutions for 80 instances and obtains the same results for the remaining 10 instances. The improvement made towards the best known solutions is 12.6% on average. A new set of larger instances, adopted from available benchmark VRPCDTW instances, is also introduced. Experimental results show that our matheuristic can also solve these instances and obtain high quality results in acceptable computation times. Compared to a pure ALNS algorithm, our matheuristic is able to obtain 0.7% better results on average.

We further present comprehensive experiments on different properties of our matheuristic. We conclude that solving the set partitioning formulation has a clear impact on the solution quality, as it may produce improvements up to 2.6% on large instances, although it requires more CPU time. It is observed that the implementation of the adaptive scheme when selecting the operators in the matheuristic only leads to slight improvements in performance. Finally, we observe that all operators during the column generation phase are required in order to obtain the high quality results. However, some of them only slightly contribute and might be removed to simplify the matheuristic. Nevertheless, this would have no impact on the required CPU time.

Possibly, the performance of the algorithm could be further improved by extending the set of routes considered during the set partitioning. Some routes are currently not accepted since they are infeasible in the ALNS solution due to the combined time horizon of supplier routes and customer routes. However, individually, these routes could be considered as feasible during the set partitioning phase. Solving other recent variants of VRPCD, such as VRP with forward and reverse cross-docking, by modifying our proposed matheuristic is also a possible interesting future direction. A number of underlying assumptions within the current model can be modified in future research in order to advance the vehicle routing and cross-docking combinations, such as multiple items or products, split deliveries, a heterogeneous fleet, or multiple cross-docks.

New and larger instances with up to 300 nodes are introduced together with benchmark results. This is another useful contribution since it corresponds to real-sized instances that could be faced by practitioners and the supply chain management community. Therefore, this paper could increase the awareness about the advantages of the cross-docking. From a practical perspective, especially when solving large instances, it may be preferred to implement a pure ALNS instead of the matheuristic when making trade-off between solution quality and CPU time.

CRediT authorship contribution statement

Aldy Gunawan: Conceptualization, Methodology, Formal analysis, Investigation, Writing - review & editing. **Audrey Tedja Widjaja:** Software, Formal analysis, Investigation, Methodology, Writing - original draft. **Pieter Vansteenwegen:** Conceptualization, Writing - review & editing. **Vincent F. Yu:** Conceptualization, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research is supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

References

[1] C.J. Liao, Y. Lin, S.C. Shih, Vehicle routing with cross-docking in the supply chain, Expert Syst. Appl. 37 (10) (2010) 6868–6873.
[2] U.M. Apte, S. Viswanathan, Effective cross docking for improving distribution efficiencies, Int. J. Logist. 3 (3) (2000) 291–302.
[3] N. Boysen, M. Fliedner, Cross dock scheduling: classification, literature review and research agenda, Omega 38 (6) (2010) 413–422.
[4] Y.H. Lee, J.W. Jung, K.M. Lee, Vehicle routing scheduling for cross-docking in the supply chain, Comput. Ind. Eng. 51 (2) (2006) 247–256.
[5] C. Archetti, M.G. Speranza, A survey on matheuristics for routing problems, EURO J. Comput. Optim. 2 (4) (2014) 223–246.
[6] V.F. Yu, P. Jewpanya, A.A.N.P. Redi, Open vehicle routing problem with cross-docking, Comput. Ind. Eng. 94 (2016) 6–17.
[7] M. Wen, J. Larsen, J. Clausen, J.F. Cordeau, G. Laporte, Vehicle routing with cross-docking, J. Oper. Res. Soc. 60 (12) (2009) 1708–1718.
[8] F.A. Santos, G.R. Mateus, A.S. da Cunha, A branch-and-price algorithm for a vehicle routing problem with cross-docking, Electron. Notes Discrete Math. 37 (2011) 249–254.
[9] C.D. Tarantilis, Adaptive multi-restart tabu search algorithm for the vehicle routing problem with cross-docking, Optim. Lett. 7 (7) (2013) 1583–1596.
[10] A. Sadri Esfahani, M. Fakhrzad, Modeling the time windows vehicle routing problem in cross-docking strategy using two meta-heuristic algorithms, Int. J. Eng. 27 (7) (2014) 1113–1126.
[11] A. Touihri, O. Dridi, S. Krichen, A multi operator genetic algorithm for solving the capacitated vehicle routing problem with cross-docking problem, in: 2016 IEEE Symposium Series on Computational Intelligence, SSCI, IEEE, 2016, pp. 1–8.
[12] P. Grangier, M. Gendreau, F. Lehuédé, L.M. Rousseau, A matheuristic based on large neighborhood search for the vehicle routing problem with cross-docking, Comput. Oper. Res. 84 (2017) 116–126.
[13] J.M. Urtasun, E. Montero, An study of operator design under an adaptive approach for solving the cross-docks vehicle routing problem, in: 2019 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2019, pp. 2098–2105.
[14] V.W. Morais, G.R. Mateus, T.F. Noronha, Iterated local search heuristics for the vehicle routing problem with cross-docking, Expert Syst. Appl. 41 (16) (2014) 7495–7506.
[15] P. Grangier, M. Gendreau, F. Lehuédé, L.-M. Rousseau, The vehicle routing problem with cross-docking and resource constraints, J. Heuristics (2019) 1–31.
[16] J. Wang, A.K.R. Jagannathan, X. Zuo, C.C. Murray, Two-layer simulated annealing and tabu search heuristics for a vehicle routing problem with cross docks and split deliveries, Comput. Ind. Eng. 112 (2017) 84–98.
[17] C.-J. Ting, T.-D. Chen, Integrate vehicle routing and truck sequencing in cross-docking operations, in: 13th IMHRC Proceedings, Cincinnati, Ohio. USA – 2014, 2014.
[18] R. Dondo, J. Cerdá, The heterogeneous vehicle routing and truck scheduling problem in a multi-door cross-dock system, Comput. Chem. Eng. 76 (2015) 42–62.
[19] B.A. Foster, D.M. Ryan, An integer programming approach to the vehicle scheduling problem, J. Oper. Res. Soc. 27 (2) (1976) 367–384.
[20] J.G. Villegas, C. Prins, C. Prodhon, A.L. Medaglia, N. Velasco, A matheuristic for the truck and trailer routing problem, European J. Oper. Res. 230 (2) (2013) 231–244.
[21] R. Kramer, A. Subramanian, T. Vidal, F.C. Lucídio dos Anjos, A matheuristic approach for the pollution-routing problem, European J. Oper. Res. 243 (2) (2015) 523–539.
[22] V. Pillac, C. Gueret, A.L. Medaglia, A parallel matheuristic for the technician routing and scheduling problem, Optim. Lett. 7 (7) (2013) 1525–1535.
[23] S. Ropke, D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, Transp. Sci. 40 (4) (2006) 455–472.
[24] E. Demir, T. Bektaş, G. Laporte, An adaptive large neighborhood search heuristic for the pollution-routing problem, European J. Oper. Res. 223 (2) (2012) 346–359.