9-2022

# Two-phase matheuristic for the vehicle routing problem with reverse 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*

# Two-phase Matheuristic for the vehicle routing problem with reverse cross-docking

Aldy Gunawan, School of Computing and Information Systems, Singapore Management University, Singapore, Singapore

Audrey Tedja Widjaja, School of Computing and Information Systems, Singapore Management University, Singapore, Singapore

Pieter Vansteenwegen, KU Leuven Mobility Research Center - CIB, KU Leuven, Belgium

Vincent F. Yu, Department of Industrial Management, National Taiwan University of Science and Technology, Taipei, Taiwan; Center for Cyber-Physical System Innovation, National Taiwan University of Science and Technology, Taipei, Taiwan

Published in Annals of Mathematics and Artificial Intelligence, 2021

https://doi.org/10.1007/s10472-021-09753-3

**Abstract:**

Cross-docking is a useful concept used by many companies to control the product flow. It enables the transshipment process of products from suppliers to customers. This research thus extends the benefit of cross-docking with reverse logistics, since return process management has become an important field in various businesses. The vehicle routing problem in a distribution network is considered to be an integrated model, namely the vehicle routing problem with reverse cross-docking (VRP-RCD). This study develops a mathematical model to minimize the costs of moving products in a four-level supply chain network that involves suppliers, cross-dock, customers, and outlets. A matheuristic based on an adaptive large neighborhood search (ALNS) algorithm and a set partitioning formulation is introduced to solve benchmark instances. We compare the results against those obtained by optimization software, as well as other algorithms such as ALNS, a hybrid algorithm based on large neighborhood search and simulated annealing (LNS-SA), and ALNS-SA. Experimental results show the competitiveness of the matheuristic that is able to obtain all optimal solutions for small instances within shorter computational times. For larger instances, the matheuristic outperforms the other algorithms using the same computational times. Finally, we analyze the importance of the set partitioning formulation and the different operators.

**Keywords:**

Vehicle routing problem, Cross-docking, Reverse logistics, Matheuristic, Adaptive large neighborhood search

Corresponding author: Aldy Gunawan aldygunawan@smu.edu.sg

## 1 Introduction

Companies often aim to improve their distribution systems in order to reduce their operational cost and gain more market share by putting cross-docking into practice [1]. A cross-dock is a transshipment facility that transfers products or items in large quantities from suppliers to customers. Operating a cross-dock could reduce the distribution costs by avoiding long origin-to-destination routes and decreasing the fleet size [2]. One successful application of cross-docking was introduced by Walmart [3].

Lee et al. [4] highlighted the importance of considering vehicle routing and scheduling in both pickup and delivery processes when efficiently operating a cross-dock. Since this problem is happening more and more in the real world, the literature has introduced the integration of cross-docking in vehicle routing problems, namely the vehicle routing problem with cross-docking (VRPCD). The VRPCD targets to design a route sequence for picking up products from supplier nodes to the cross-dock and subsequently delivering these products to customers, all within a given time horizon such that no storage occurs inside the cross-dock facility. The objective is to minimize the overall costs, including transportation costs and fixed vehicle operations costs. Since the problem is considered to be NP-hard, various algorithms have been proposed to solve the benchmark instances, such as tabu search (TS) [4, 5], simulated annealing (SA) [6], adaptive large neighborhood search (ALNS) [7], and a matheuristic [8, 9].

While the VRPCD deals with a forward logistics system, many companies have recently focused on reverse logistics, whereby items may be sent back from customers to suppliers due to various reasons, such as defective or unsold products [10]. Rogers and Tibben-Lembke [11] defined reverse logistics (RL) as the process of planning, implementing, and controlling the efficient and cost effective flow of raw materials, in-process inventory, finished goods, and related information from the point of consumption to the point of origin for the purpose of recapturing value or proper disposal. Shen and Li [12] studied the benefits of reverse logistics by considering the effects of unsold products from an outsourced fashion supply chain.

Motivated by the tendency to optimize RL systems and the success of incorporating cross-docking in a forward logistics system, Widjaja et al. [13] extended the benefit of cross-docking with RL to facilitate return process management, namely the vehicle routing problem with reverse cross-docking (VRP-RCD). Unlike the VRPCD that focuses on product delivery from suppliers to customers (forward flow), the VRP-RCD looks at the product delivery from customers to suppliers (reverse flow). Instead of directly sending the returned products back to suppliers (as in Kaboudani et al. [10]), the VRP-RCD [13] considers sending the returned products to the outlets first for a reselling activity [14].

The three main processes in the VRP-RCD are as follows: 1) the customer pickup process that aims to pick up (or collect) returned products from the customers and bring them to the cross-dock for the next process, 2) the outlet simultaneous delivery and pickup process that aims to satisfy outlets' demands based on the availability of the customers' returned products, for reselling at the outlets, while at the same time collecting the returned products from the outlets, and 3) the supplier delivery process that aims to deliver the returned products from both customers and outlets to the supplier that supplied those products. The VRP-RCD framework is illustrated in Fig. 1, while the network model will be explained in Section 3. Here, we use the number of customers $|C| = 6$, the number of outlets $|O| = 5$, the number of suppliers $|S| = 4$, and the number of vehicles $|V| = 5$.

This reverse process usually happens in a business with seasonal demand patterns, such as fashion, books, or some electronic devices [15]. Once a new product is launched, the

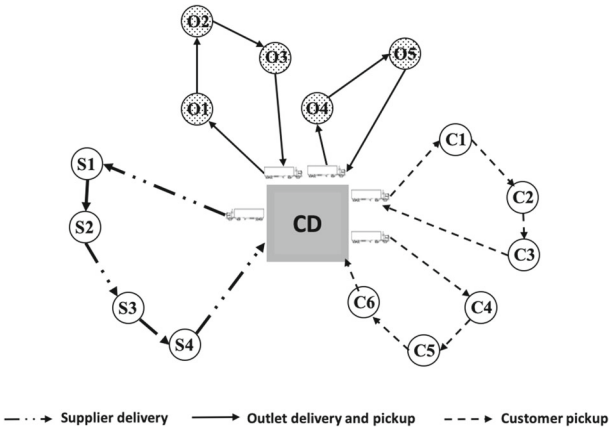


**Fig. 1** VRP-RCD with $|C| = 6$, $|O| = 5$, $|S| = 4$, and $|V| = 5$

primary sellers (we call them customers in the VRP-RCD) no longer sell the old products. Hence, the old products are returned to the suppliers that supplied those products. However, there might be some secondary sellers (i.e., outlets) that are willing to sell these old products at lower prices. Therefore, the returned products from the primary sellers are sent to the secondary sellers. Afterwards, if there are still some unsold products at the secondary sellers, then those products need to be returned to the respective suppliers for further processings (e.g., recycle or remanufacture). The VRP-RCD thus aims to minimize the overall costs occurring in the distribution of the RL system, such as the transportation cost and fixed vehicle operational cost.

This paper extends the matheuristic algorithm that was originally designed for the VRPCD in Gunawan et al. [8, 9], in order to solve the VRP-RCD problem. The matheuristic consists of two phases: routes generation and solving the set partitioning formulation (SPF). The first phase generates as many routes as possible, which is performed by the ALNS of Gunawan et al. [16], while the second phase finds the optimal route combination from any routes found in the first phase that satisfies the VRP-RCD constraints. We modify the SPF of Gunawan et al. [8, 9] to deal with the VRP-RCD and introduce a new set of larger benchmark VRP-RCD instances to show how the matheuristic performs well on these newly generated instances. These results can be used as a baseline for future research. Finally, we present how every component in our matheuristic, such as the set partitioning formulation and the operators, contributes toward generating better solution quality.

After discussing the related state-of-the-art algorithms in Section 2, Section 3 discusses the VRP-RCD in detail and presents a mathematical model. Section 4 explains the matheuristic designed to tackle the VRP-RCD. Section 5 presents a comprehensive experimental study in order to compare the matheuristic performance with the performance of the optimization software CPLEX and three different types of ALNS algorithm: a pure ALNS, LNS-SA, and ALNS-SA [16]. The LNS-SA does not incorporate an adaptive scheme when selecting the operators, but worse solutions found might be accepted with some probability (i.e., employs the SA acceptance criterion), while the ALNS-SA [16] combines the ALNS with the SA acceptance criterion. Section 6 summarizes the contributions of different elements of the matheuristic, the conclusions, and future work directions.

## 2 Literature review

Lee et al. [4] first introduced the VRPCD as an integration of vehicle routing scheduling and cross-docking. They formulated a mathematical model and proposed a tabu search (TS) algorithm to solve their own generated instances. By comparing against enumeration results, the proposed TS is able to obtain on average a 4% gap from the optimal solutions within a reasonable CPU time. Liao et al. [5] then designed a new TS with two main differences with the one of Lee et al. [4]: 1) the former moves one node at a time to another vehicle, while the latter always exchanges two nodes at two different vehicles simultaneously, and 2) the former allows the removal of an empty vehicle from the solution, while the latter does not. Compared to TS [4], this new modified TS [5] is able to improve the quality of solutions by 10.6%, 36%, and 14.9% on average for small, medium, and large instances, respectively, within significantly shorter CPU times.

Yu et al. [6] proposed a new variant in the VRPCD by considering an open network (namely OVRPCD) and developed a simulated annealing (SA) algorithm to solve both benchmark VRPCD instances and newly introduced benchmark OVRPCD instances. Experimental results show the competitiveness of the proposed SA in solving the VRPCD instances as it is able to improve the best known solutions (from Lee et al. [4] and Liao et al. [5]) on average by 3.3%, 2%, and 5.9% for small, medium, and large instances, respectively. Gunawan et al. [7] designed an adaptive large neighborhood search (ALNS) algorithm to solve the available benchmark VRPCD instances and showed that ALNS is able to further improve the best known solutions (from Lee et al. [4], Liao et al. [5], and Yu et al. [6]) by 1.4%, 13.6%, and 21.8% on average for small, medium, and large instances, respectively. Gunawan et al. [9] proposed a two-phase matheuristic and showed the competitiveness of the proposed matheuristic by optimally solving 29 out of 30 problems in small benchmark VRPCD instances. Gunawan et al. [8] extended the proposed algorithm in Gunawan et al. [7, 9] by incorporating more operators and adding larger instances for the VRPCD in order to justify the performance of the proposed algorithm. Another variant of the VRPCD by considering multiple products is studied in Gunawan et al. [17]. Only small and medium instances were introduced and solved by an optimization software, CPLEX.

Apart from the original VRPCD, there are many other variants of the VRPCD that have been studied so far, such as the VRPCD with time windows (VRPCDTW) [18–21], the VRPCDTW with split pickups in supplier nodes and split deliveries in customer nodes [22], the VRPCD by considering the dock door utilization [23, 24], and the profitable heterogeneous VRPCD [25]. However, all of the above-mentioned research only deal with the forward flow of products, from suppliers to customers.

Rezaei and Kheirkhah [2] introduced an integrated forward/reverse cross-docking for a multi-product supply chain network in response to the needs of sustainability in logistics networks and growing environmental and economic concerns (e.g., recycling of used products). They presented a mixed-integer linear programming (MILP) model with the objective of minimizing total costs. The model is solved by the general algebraic modeling system (GAMS) software. They also highlighted the importance of developing metaheuristic methods to solve the model in large-scale problems.

Zuluaga et al. [15] discussed the concept of cross-docking in a reverse logistics context as well as the importance of managing return flows, especially in businesses with seasonal demand patterns, including selling unsold products to secondary channels. A reverse cross-docking mathematical model is introduced with the objective of minimizing the cost of return process management for unsold products. The mathematical model is solved by

commercial software, CPLEX, for dealing with simulated data. However, the method for solving instances with realistic sample sizes, which are close to real life problems, is still missing. One requires (meta)heuristics to solve these realistic instances within reasonable computational times.

An integrated model of the VRP with cross-docking for handling both forward and reverse logistics was first introduced by Kaboudani et al. [10]. A mathematical model is formulated that includes transferring products from suppliers to customers and returning products from customers to suppliers through a cross-dock with minimum transportation cost. Since the problem is NP-hard, a simulated annealing based heuristic algorithm was proposed and compared with commercial software, GAMS. The proposed algorithm showed the efficiency at solving large-scale problems.

Widjaja et al. [13] studied the VRP-RCD by introducing a reselling process at secondary customers (i.e., outlets). Hence, the distribution system in the network consists of: 1) picking up customers' returned products, 2) delivering outlets' demand based on the amount of customers' returned products and simultaneously picking up their returned products, and 3) delivering those returned products from both customers and outlets back to the suppliers that supplied the products. A mathematical model was presented to formulate the problem. Small instances were introduced and solved by optimization software, CPLEX.

Gunawan et al. [16] presented an algorithm that is based on an adaptive large neighborhood search and the simulated annealing acceptance criteria to solve the VRP-RCD [13]. The algorithm employs DESTROY and REPAIR operators for repetitively removing some nodes from a solution and re-inserting them back to a more profitable position, respectively. In terms of the performance, the proposed algorithm performs well on solving two sets of small and medium benchmark VRP-RCD instances. All above-mentioned references are summarized in Table 1 in order to visualize how this research differs from the state of the art concerning, for example supply chain levels, product flow, and other additional considerations.

## 3 Problem description

This paper studies a four-level supply chain network that considers suppliers, customers, outlets, and a cross-dock. We define $C = \{1, 2, \ldots, |C|\}$ as a set of customers from which returned products are sent to a set of outlets $O = \{1, 2, \ldots, |O|\}$ for the reselling process. At each outlet, some unsold products are sent back to a set of suppliers $S = \{1, 2, \ldots, |S|\}$ that originally supplied those products. Therefore, there are three main processes in the VRP-RCD: 1) customer pickup process, 2) outlet simultaneous delivery and pickup process, and 3) supplier delivery process.

A cross-dock facility as an intermediate point is set up to consolidate the products in-between those processes. Let $V = \{1, 2, \ldots, |V|\}$ be a set of homogeneous vehicles with capacity $q$ that is available at the cross-dock. Each vehicle only serves one of the above-mentioned processes. A fixed operational cost $H$ is charged for every vehicle used in the process.

In a union of cross-dock and customer nodes, each arc connecting nodes $i$ and $j$ has a travel distance of $e'_{ij}$ and transportation time of $t'_{ij}$. Let $c$ be the transportation cost per unit distance. Each customer $i \in C$ has $r'_{ik}$ units of returned product $k$, where $p_k(\%)$ of it are considered as defective products. Starting from the cross-dock, a vehicle must travel to visit any customers with returned products. Those returned products are inspected inside the cross-dock facility to be separated between $p_k \times r'_{ik}$ of defective products and $(1 - p_k) \times r'_{ik}$

**Table 1** VRP-RCD literature

| Literature | VRP | Cross-docking | | | Product Flow | | Additional Consideration | Approach |
|---|---|---|---|---|---|---|---|---|
| | | Supplier | Customer | Secondary Channel* | Forward | Reverse | | |
| Lee et al. [4] | ✓ | ✓ | ✓ | – | ✓ | – | – | Tabu search (TS) |
| Liao et al. [5] | ✓ | ✓ | ✓ | – | ✓ | – | – | Improved TS of [4] |
| Yu et al. [6] | ✓ | ✓ | ✓ | – | ✓ | – | Open network | Simulated annealing (SA) |
| Gunawan et al. [17] | ✓ | ✓ | ✓ | – | ✓ | – | Multiple product types | Mixed-integer linear programming (MILP) |
| Gunawan et al. [7] | ✓ | ✓ | ✓ | – | ✓ | – | – | Adaptive large neighborhood search (ALNS) |
| Gunawan et al. [9] | ✓ | ✓ | ✓ | – | ✓ | – | – | Matheuristic |
| Gunawan et al. [8] | ✓ | ✓ | ✓ | – | ✓ | – | – | Improved Matheuristic of [9] |
| Wen et al. [18] | ✓ | ✓ | ✓ | – | ✓ | – | Time windows | TS |
| Tarantilis [19] | ✓ | ✓ | ✓ | – | ✓ | – | Time windows | Adaptive multi-restart TS |
| Grangier et al. [20] | ✓ | ✓ | ✓ | – | ✓ | – | Time windows | Matheuristic |
| Morais et al. [21] | ✓ | ✓ | ✓ | – | ✓ | – | Time windows | Iterated local search |
| Wang et al. [22] | ✓ | ✓ | ✓ | – | ✓ | – | Time windows with split pickups and deliveries | Two-layer SA and TS |
| Dondo and Cerdá [23] | ✓ | ✓ | ✓ | – | ✓ | – | Dock door utilization | Sweep-heuristic based-models |
| Grangier et al. [24] | ✓ | ✓ | ✓ | – | ✓ | – | Dock door utilization | Matheuristic |
| Baniamerian et al. [25] | ✓ | ✓ | ✓ | – | ✓ | – | Profit and heterogeneous fleet | Modified VNS[b] and genetic algorithm |
| Rezaei and Kheirkhah [2] | – | ✓ | ✓ | ✓ | ✓ | ✓ | Multiple product types | MILP |
| Zuluaga et al. [15] | – | – | ✓ | ✓ | – | ✓ | Considers "other channel" instead of "supplier" | MILP |
| Kaboudani et al. [10] | ✓ | ✓ | ✓ | – | ✓ | ✓ | – | SA |
| Widjaja et al. [13] | ✓ | ✓ | ✓ | ✓ | – | ✓ | Multiple product types | MILP |
| Gunawan et al. [16] | ✓ | ✓ | ✓ | ✓ | – | ✓ | Multiple product types | ALNS |
| This research | ✓ | ✓ | ✓ | ✓ | – | ✓ | Multiple product types | Matheuristic |

[a] Secondary channel / secondary customer / outlet

[b] VNS: variable neighborhood search

of non-defective products. The non-defective products are then consolidated according to outlets' demand and delivered to them.

In a union of cross-dock and outlet nodes, each arc connecting nodes $i$ and $j$ has a travel distance of $e''_{ij}$ and transportation time of $t''_{ij}$. Each outlet $i \in O$ has demand for product $k$ as much as $d''_{ik}$. However, not each outlet may be able to receive all of its demand, depending on the availability of the non-defective returned products from customers. Therefore, we have to decide which outlets need to be visited. Moreover, some outlets might also have returned products $r''_{ik}$ that should be sent back to the suppliers. Those products also need to be collected. Starting from the cross-dock, a vehicle visits outlets with either delivery or pickup, or both, and returns to the cross-dock.

In a union of cross-dock and supplier nodes, each arc connecting nodes $i$ and $j$ has a travel distance of $e'''_{ij}$ and transportation time of $t'''_{ij}$. The following products: 1) the returned products from outlets, 2) the defective products from customers (if any), and 3) the returned products from customers that are not sent to any outlets during the second process, are then consolidated inside the cross-dock and are returned to the appropriate supplier. This overall process is illustrated in Fig. 1 and must be done within the time horizon $T_{max}$.

It should be noted that in the VRP-RCD considered herein, the number of products returned ($r'_{ik}$ and $r''_{ik}$) is assumed to be known and fixed when the routing decisions are made [16]. We consider this as a realistic assumption. In a previous work [13], the number of returned products is fixed as a fraction of the number of products delivered to a certain customer or outlet. However, that is only realistic when multiple periods are considered with similar flows or when the return flow depends on the forward flow of (one of) the previous period(s). The list of decision variables used in this problem is summarized in Table 2, and the mathematical model is formulated as follows.

$$Min \; c\left(\sum_{v\in V}\sum_{i\in C\cup 0}\sum_{j\in C\cup 0} x'^{v}_{ij}e'_{ij} + \sum_{v\in V}\sum_{i\in O\cup 0}\sum_{j\in O\cup 0} x'''^{v}_{ij}e''_{ij} + \sum_{v\in V}\sum_{i\in S\cup 0}\sum_{j\in S\cup 0} x''''^{v}_{ij}e'''_{ij}\right) + H\left(\sum_{v\in V}\sum_{j\in C} x'^{v}_{0j} + \sum_{v\in V}\sum_{j\in O} x''^{v}_{0j} + \sum_{v\in V}\sum_{j\in S} x''''^{v}_{0j}\right) \quad (1)$$

$$\sum_{j\in C} x'^{v}_{0j} + \sum_{j\in O} x''^{v}_{0j} + \sum_{j\in S} x''''^{v}_{0j} \le 1 \quad \forall v \in V \quad (2)$$

$$\sum_{i\in C\cup 0}\sum_{j\in C\cup 0, j\neq i} x'^{v}_{ij}t'_{ij} \le Tcp_{max} \quad \forall v \in V \quad (3)$$

$$\sum_{i\in O\cup 0}\sum_{j\in O\cup 0, j\neq i} x'''^{v}_{ij}t''_{ij} \le Todp_{max} \quad \forall v \in V \quad (4)$$

$$\sum_{i\in S\cup 0}\sum_{j\in S\cup 0, j\neq i} x''''^{v}_{ij}t'''_{ij} \le Tsd_{max} \quad \forall v \in V \quad (5)$$

$$Tcp_{max} + Todp_{max} + Tsd_{max} \le T_{max} \quad (6)$$

The VRP-RCD objective function minimizes the total transportation and operational costs, as expressed in (1). Constraint (2) ensures that each vehicle is only used in either one

**Table 2** List of VRP-RCD decision variables

| Decision Variables | |
|---|---|
| $x_{ij}^{'v}$ | 1 if vehicle $v$ travels from node $i$ to $j$ in the customer pickup process; 0 otherwise $(i, j \in C \cup 0, v \in V)$ |
| $x_{ij}^{''v}$ | 1 if vehicle $v$ travels from node $i$ to $j$ in the outlet delivery and pickup process; 0 otherwise $(i, j \in O \cup 0, v \in V)$ |
| $x_{ij}^{'''v}$ | 1 if vehicle $v$ travels from node $i$ to $j$ in the supplier delivery process; 0 otherwise $(i, j \in S \cup 0, v \in V)$ |
| $y_k$ | 1 if the demand for product $k$ from all outlets is less than the amount of non-defective returned product $k$ from all customers; 0 otherwise $(k \in S)$ |
| $A_i^{'v}$ | Amount of products picked up from node $i$ by vehicle $v$ in the customer pickup process $(i \in C, v \in V)$ |
| $A_{ik}^{''v}$ | Amount of product $k$ delivered to node $i$ by vehicle $v$ in the outlet delivery and pickup process $(i \in O, k \in S, v \in V)$ |
| $A_i^{'''v}$ | Amount of products delivered to node $i$ by vehicle $v$ in the supplier delivery process $(i \in S, v \in V)$ |
| $q_0^{'v}$ | Initial load of vehicle $v$ upon leaving the cross-dock in the customer pickup process $(v \in V)$ |
| $q_0^{''v}$ | Initial load of vehicle $v$ upon leaving the cross-dock in the outlet delivery and pickup process $(v \in V)$ |
| $q_0^{'''v}$ | Initial load of vehicle $v$ upon leaving the cross-dock in the supplier delivery process $(v \in V)$ |
| $q_i'$ | Amount of load remaining in the vehicle upon visiting node $i$ in the customer pickup process $(i \in C)$ |
| $q_i''$ | Amount of load remaining in the vehicle upon visiting node $i$ in the outlet delivery and pickup process $(i \in O)$ |
| $q_i'''$ | Amount of load remaining in the vehicle upon visiting node $i$ in the supplier delivery process $(i \in S)$ |
| $Tcp_{max}$ | Maximum traveling duration time in the customer pickup process |
| $Todp_{max}$ | Maximum traveling duration time in the outlet delivery and pickup process |
| $Tsd_{max}$ | Maximum traveling duration time in the supplier delivery process |
| $u_i'$ | Order in which node $i$ is visited on a tour in the customer pickup process $(i \in C)$ |
| $u_i''$ | Order in which node $i$ is visited on a tour in the outlet delivery and pickup process $(i \in O)$ |
| $u_i'''$ | Order in which node $i$ is visited on a tour in the supplier delivery process $(i \in S)$ |

of the processes. Constraints (3) to (5) record the maximum time of each of the three processes, where the service times (e.g., administration, loading, unloading, and consolidating) are included in the transportation times. Constraint (6) ensures all processes must be done within the time horizon $T_{max}$.

### Customer pickup process constraints

$$L \sum_{v \in V} \sum_{i \in C \cup 0, i \neq j} x_{ij}^{'v} \geq \sum_{k \in S} r_{jk}' \quad \forall j \in C \tag{7}$$

$$\sum_{k \in S} r_{jk}' \geq \epsilon - L\left(1 - \sum_{v \in V} \sum_{i \in C \cup 0, i \neq j} x_{ij}^{'v}\right) \quad \forall j \in C \tag{8}$$

$$\sum_{i \in C} \sum_{j \in C, j \neq i} x_{ij}^{'v} \leq L \sum_{j \in C} x_{0j}^{'v} \quad \forall v \in V \tag{9}$$

$$\sum_{i \in C \cup 0, i \neq l} x_{il}^{'v} = \sum_{j \in C \cup 0, j \neq l} x_{lj}^{'v} \quad \forall l \in C, \forall v \in V \tag{10}$$

$$\sum_{v \in V} \sum_{i \in C \cup 0} x_{ij}^{'v} \leq 1 \quad \forall j \in C \tag{11}$$

$$A_j^{'v} = \sum_{k \in S} r'_{jk} \sum_{i \in C \cup 0} x_{ij}^{'v} \quad \forall j \in C, \forall v \in V \tag{12}$$

$$q_0^{'v} = 0 \quad \forall v \in V \tag{13}$$

$$q'_i \geq q_0^{'v} + A_i^{'v} - L(1 - x_{0i}^{'v}) \quad \forall i \in C, \forall v \in V \tag{14}$$

$$q'_i \leq q_0^{'v} + A_i^{'v} + L(1 - x_{0i}^{'v}) \quad \forall i \in C, \forall v \in V \tag{15}$$

$$q'_j \geq q'_i + A_j^{'v} - L(1 - x_{ij}^{'v}) \quad \forall i, j \in C, \forall v \in V \tag{16}$$

$$q'_j \leq q'_i + A_j^{'v} + L(1 - x_{ij}^{'v}) \quad \forall i, j \in C, \forall v \in V \tag{17}$$

$$\sum_{j \in C} A_j^{'v} \leq q \quad \forall v \in V \tag{18}$$

$$u'_j \geq u'_i + 1 - |C|\left(1 - \sum_{v \in V} x_{ij}^{'v}\right) \quad \forall i, j \in C \tag{19}$$

Constraints (7) and (8) ensure that if there is any returned product from a customer, then a vehicle will visit that customer. $L$ refers to a very large integer number, while $\epsilon$ refers to a very small positive number. Constraint (9) ensures that for every vehicle used in this process, it always starts its trip from the cross-dock. Constraint (10) ensures the outflow and inflow of a vehicle in each customer node. Constraints (11) and (12) ensure that each customer is visited at most once by one vehicle, such that no split pickup occurs. Constraints (13) to (17) track the total load inside a vehicle. Constraint (18) ensures that the total amount of picked-up products from any customers assigned to a vehicle does not violate the vehicle capacity. Constraint (19) is the sub-tour elimination constraint.

**Outlet delivery and pickup process constraints**

$$\sum_{i \in O} d''_{ik} - (1 - p_k) \sum_{i \in C} r'_{ik} \geq -L y_k \quad \forall k \in S \tag{20}$$

$$\sum_{i \in O} d''_{ik} - (1 - p_k) \sum_{i \in C} r'_{ik} \leq L(1 - y_k) \quad \forall k \in S \tag{21}$$

$$\sum_{v \in V} \sum_{i \in O} A_{ik}^{''v} \geq (1 - p_k) \sum_{i \in C} r'_{ik} - L y_k \quad \forall k \in S \tag{22}$$

$$\sum_{v\in V}\sum_{i\in O} A''^{v}_{ik} \le (1-p_k)\sum_{i\in C} r'_{ik} + Ly_k \quad \forall k \in S \tag{23}$$

$$\sum_{v\in V}\sum_{i\in O} A''^{v}_{ik} \ge \sum_{i\in O} d''_{ik} - L(1-y_k) \quad \forall k \in S \tag{24}$$

$$\sum_{v\in V}\sum_{i\in O} A''^{v}_{ik} \le \sum_{i\in O} d''_{ik} + L(1-y_k) \quad \forall k \in S \tag{25}$$

$$\sum_{v\in V} A''^{v}_{ik} \le d''_{ik} \quad \forall i \in O, \forall k \in S \tag{26}$$

$$L\sum_{i\in O\cup 0, i\neq j} x''^{v}_{ij} \ge \sum_{k\in S} A''^{v}_{jk} \quad \forall j \in O, \forall v \in V \tag{27}$$

$$L\sum_{v\in V}\sum_{i\in O\cup 0, i\neq j} x''^{v}_{ij} \ge \sum_{v\in V}\sum_{k\in S} A''^{v}_{jk} + \sum_{k\in S} r''_{jk} \quad \forall j \in O \tag{28}$$

$$\sum_{v\in V}\sum_{k\in S} A''^{v}_{jk} + \sum_{k\in S} r''_{jk} \ge \epsilon - L\Big(1 - \sum_{v\in V}\sum_{i\in O\cup 0, i\neq j} x''^{v}_{ij}\Big) \quad \forall j \in O \tag{29}$$

$$\sum_{i\in O}\sum_{j\in O, j\neq i} x''^{v}_{ij} \le L\sum_{j\in O} x''^{v}_{0j} \quad \forall v \in V \tag{30}$$

$$\sum_{i\in O\cup 0, i\neq l} x''^{v}_{il} = \sum_{j\in O\cup 0, j\neq l} x''^{v}_{lj} \quad \forall l \in O, \forall v \in V \tag{31}$$

$$\sum_{v\in V}\sum_{i\in O\cup 0} x''^{v}_{ij} \le 1 \quad \forall j \in O \tag{32}$$

$$q''^{v}_{0} = \sum_{j\in O}\sum_{k\in S} A''^{v}_{jk} \quad \forall v \in V \tag{33}$$

$$q''_i \ge q''^{v}_{0} - \sum_{k\in S} A''^{v}_{ik} + \sum_{k\in S} r''_{ik} - L(1 - x''^{v}_{0i}) \quad \forall i \in O, \forall v \in V \tag{34}$$

$$q''_i \le q''^{v}_{0} - \sum_{k\in S} A''^{v}_{ik} + \sum_{k\in S} r''_{ik} + L(1 - x''^{v}_{0i}) \quad \forall i \in O, \forall v \in V \tag{35}$$

$$q''_j \ge q''_i - \sum_{k\in S} A''^{v}_{jk} + \sum_{k\in S} r''_{jk} - L(1 - x''^{v}_{ij}) \quad \forall i, j \in O, \forall v \in V \tag{36}$$

$$q''_j \le q''_i - \sum_{k\in S} A''^{v}_{jk} + \sum_{k\in S} r''_{jk} + L(1 - x''^{v}_{ij}) \quad \forall i, j \in O, \forall v \in V \tag{37}$$

$$q''^{v}_{0} \le q \quad \forall v \in V \tag{38}$$

$$q''_j \le q \quad \forall j \in O \tag{39}$$

$$u''_j \ge u''_i + 1 - |O|\left(1 - \sum_{v\in V} x''^{v}_{ij}\right) \quad \forall i, j \in O \tag{40}$$

Constraints (20) and (21) serve to determine the value of $y_k$. Constraints (22) to (26) then determine the amount of products delivered to the outlets, while ensuring that this amount does not exceed each outlet's demand. Constraint (27) ensures no split delivery occurs in the process. Constraints (28) and (29) ensure that if there is a delivery process to an outlet or there are returned products from an outlet, then a vehicle will visit that outlet. Constraint (30) ensures that for every vehicle used in this process, it always starts its trip from the cross-dock. Constraint (31) ensures the outflow and inflow of a vehicle in each outlet node. Constraint (32) ensures that each outlet is visited at most once. Constraints (33) to (37) track the total load inside a vehicle. Constraint (38) ensures the vehicle capacity limitation is addressed when a vehicle leaves the cross-dock, while constraint (39) ensures the vehicle capacity limitation is addressed when a vehicle leaves any outlet node. Constraint (40) is the sub-tour elimination constraint.

**Supplier delivery process constraints** Due to similarities with the customer pickup process constraints, the constraints are listed in Appendix A. It is worth noting that the differences lie on: 1) how to decide the visited suppliers (e.g., suppliers with either returned products from customers or outlets, or both), and 2) the vehicle load is checked at the beginning of the tour, and it is decreasing upon delivering its load to the supplier.

Given the flow constraints (14)–(17), (34)–(37), and (84)–(87), the subtour elimination constraints (19), (40), and (89) are not strictly necessary, but we do note that they speed up the optimization process.

## 4 Proposed algorithm

In the context of the VRPCD, we assume that all customers have to be visited [4]. However, in the VRP-RCD, only customers with returned products need to be visited. Thus, a node selection process is first performed, as briefly explained in Section 4.1, followed by implementing ALNS to find possible route sequences based on selected nodes, as explained in Section 4.2. This ALNS is adopted from Gunawan et al. [16]. The matheuristic then determines the best combination of the routes found.

Another main difference between the VRPCD and VRP-RCD is that the former involves an individual pickup (delivery) process in supplier (customer) nodes, while the latter also involves the simultaneous pickup and delivery processes in outlet nodes. In order to handle this additional process, the vehicle capacity must be checked at every edge of the route. Obviously, the solution representation includes outlet nodes together with supplier and customer nodes. This will be described in Section 4.2.

### 4.1 Node selection

Let $m'_i$, $m''_i$, and $m'''_i$ be binary variables, where each equals 1 if customer $i (i \in C)$, outlet $i (i \in O)$, and supplier $i (i \in S)$ must be visited, respectively, and 0 otherwise. In the customer pickup process, customer $i$ is visited ($m'_i = 1$) only if it has any returned products, while in the supplier delivery process, supplier $k$ is visited ($m'''_k = 1$) only if there is any returned product sent to supplier $k$.

In order to determine the value of $m''_i$, we should first calculate the amount of product $k$ delivered to outlet $i$, denoted as $\vartheta''_{ik}$, where $i \in O$ and $k \in S$. If the amount of non-defective returned product $k$ from all customers is more than the cumulative of the outlet demand $k$, then each outlet is able to receive product $k$ as much as its demand. Hence, $\vartheta''_{ik} = d''_{ik} \ \ \forall i \in O, \forall k \in S$; otherwise, when the demand from the outlets is larger than the

amount of non-defective returned products, we apply sorting criteria on the outlets and then iteratively assign $\vartheta''_{ik}$ according to this sorting until the amount of available units is reached. For each instance, one of the following sorting criteria is randomly selected to determine $\vartheta''_{ik}$:

- outlet with the highest demand of product $k$ first
- all outlets with demand of product $k$ are fulfilled by splitting the total amount of non-defective returned products $k$ equally over all those outlets
- outlet with demand of product $k$ that is located nearest to the cross-dock and all other outlets first
- outlet with the highest number of different product types first
- outlet with the highest cumulative demand of all product types first
- outlet with the lowest number of different returned product types first
- outlet with the lowest cumulative returned products of all product types first

Subsequently, outlet $i$ is visited ($m''_i = 1$) if there is any delivered product to and/or returned products from outlet $i$, as formulated in (41).

$$m''_i = \begin{cases} 1, & \text{if } \sum_{k \in S} \vartheta''_{ik} + \sum_{k \in S} r''_{ik} > 0 \\ 0, & \text{if } \sum_{k \in S} \vartheta''_{ik} + \sum_{k \in S} r''_{ik} = 0 \end{cases} \quad \forall i \in O \tag{41}$$

### 4.2 Matheuristic

The pseudocode of the proposed matheuristic is presented in Algorithm 1. The matheuristic starts by building an initial solution that will be explained in detail in Section 4.2.1. Next, the matheuristic continues with two main phases. The first phase generates as many feasible routes as possible by implementing adaptive large neighborhood search (ALNS) (Section 4.2.2). The second phase of the matheuristic determines the best combination (i.e., minimum total cost) of the routes found in the first phase by solving the set partitioning formulation (Section 4.2.3).

#### 4.2.1 Initial solution

We construct an initial solution by assigning nodes to vehicles without violating the vehicle capacity, regardless of the route sequence. The additional decision variables are listed in Table 3. The initial solution is generated by solving the following equations.

$$Min \quad \sum_{v \in V} x'^{v} + x''^{v} + x'''^{v} \tag{42}$$

$$\sum_{v \in V} a_i'^{v} = m'_i \quad \forall i \in C \tag{43}$$

$$\sum_{v \in V} a_i''^{v} = m''_i \quad \forall i \in O \tag{44}$$

**Algorithm 1** Matheuristic pseudocode.

$Sol_0, Sol^*, Sol' \leftarrow$ INITIALSOLUTION
UpdatePool(INITIALSOLUTION, $\Omega_c$, $\Omega_o$, $\Omega_s$)
$Temp \leftarrow T_0$
NOIMPR, ITER $\leftarrow 0$
FOUNDBESTSOL $\leftarrow$ False
Set $s_j$ and $w_j$ such that $p_j$ is equally likely
**while** NOIMPR $< \theta$ ***and*** $|\Omega_c|, |\Omega_o|, |\Omega_s| < \Omega_{max}$ **do**
    REMOVEDNODES $\leftarrow 0$
    **while** REMOVEDNODES $< \pi$ **do**
        $Sol_0 \leftarrow$ Destroy $(R_r)$
        UpdatePool($Sol_0$, $\Omega_c$, $\Omega_o$, $\Omega_s$)
        UpdateRemovedNodes(REMOVEDNODES, $R_r$)
    **end**
    **while** REMOVEDNODES $> 0$ **do**
        $Sol_0 \leftarrow$ Repair $(I_i)$
        UpdatePool($Sol_0$, $\Omega_c$, $\Omega_o$, $\Omega_s$)
        UpdateRemovedNodes(REMOVEDNODES, $I_i$)
    **end**
    AcceptanceCriteria($Sol_0$, $Sol^*$, $Sol'$, $Temp$)
    **if** $Sol_0 < Sol^*$ **then**
        FOUNDBESTSOL $\leftarrow$ True
    **end**
    Update $s_j$
    **if** ITER mod $\eta_{ALNS} = 0$ **then**
        Update $w_j$ and $p_j$
    **end**
    **if** ITER mod $\eta_{SA} = 0$ **then**
        **if** FOUNDBESTSOL = *False* **then**
            NOIMPR $\leftarrow$ NOIMPR + 1
        **end**
        **else**
            NOIMPR $\leftarrow 0$
        **end**
        FOUNDBESTSOL $\leftarrow$ False
        $Temp \leftarrow Temp \times \alpha$
    **end**
    ITER $\leftarrow$ ITER + 1
**end**
$Sol^* \leftarrow$ solve the set partitioning formulation
**Return** $Sol^*$

$$\sum_{v \in V} a_i'''^v = m_i''' \quad \forall i \in S \tag{45}$$

$$\sum_{i \in C} \sum_{k \in S} a_i'^v r'_{ik} \leq q \quad \forall v \in V \tag{46}$$

**Table 3** List of additional decision variables to construct an initial solution

| Additional Decision Variables | |
|---|---|
| $a_i^{'v}$ | 1 if node $i$ is visited by vehicle $v$ in the customer pickup process; 0 otherwise ($i \in C, v \in V$) |
| $a_i^{''v}$ | 1 if node $i$ is visited by vehicle $v$ in the outlet delivery and pickup process; 0 otherwise ($i \in O, v \in V$) |
| $a_i^{'''v}$ | 1 if node $i$ is visited by vehicle $v$ in the supplier delivery process; 0 otherwise ($i \in S, v \in V$) |
| $x^{'v}$ | 1 if vehicle $v$ is used in the customer pickup process; 0 otherwise ($v \in V$) |
| $x^{''v}$ | 1 if vehicle $v$ is used in the outlet delivery and pickup process; 0 otherwise ($v \in V$) |
| $x^{'''v}$ | 1 if vehicle $v$ is used in the supplier delivery process; 0 otherwise ($v \in V$) |

$$\sum_{i \in O} a_i^{''v} \times \max\left(\sum_{k \in S} \vartheta_{ik}^{''}, \sum_{k \in S} r_{ik}^{''}\right) \leq q \quad \forall v \in V \tag{47}$$

$$\sum_{k \in S} a_k^{'''v} \left(\sum_{i \in C} r_{ik}^{'} - \sum_{i \in O} \vartheta_{ik}^{''} + \sum_{i \in O} r_{ik}^{''}\right) \leq q \quad \forall v \in V \tag{48}$$

$$|C|\, x^{'v} \geq \sum_{i \in C} a_i^{'v} \quad \forall v \in V \tag{49}$$

$$|O|\, x^{''v} \geq \sum_{i \in O} a_i^{''v} \quad \forall v \in V \tag{50}$$

$$|S|\, x^{'''v} \geq \sum_{i \in S} a_i^{'''v} \quad \forall v \in V \tag{51}$$

$$x^{'v} + x^{''v} + x^{'''v} \leq 1 \quad \forall v \in V \tag{52}$$

Objective function (42) minimizes the number of vehicles used. All mandatory visited nodes are visited by exactly one vehicle, as addressed in constraints (43)–(45). The vehicle capacity limitation is formulated by constraints (46)–(48). It is noted that our current aim is to assign nodes to vehicles, regardless of their route sequences. Therefore, in the outlet simultaneous pickup and delivery process (constraint (47)), the balance of vehicle capacity checking cannot be performed at the edge of each route, since the route sequence has not been generated yet. In order to overcome this, we use an upper bound product value of each node, which is max(delivery, pickup). Only in the subsequent parts of the proposed algorithm when the route sequences are already observed can the vehicle capacity checking then be implemented by checking the vehicle capacity balance at every edge of the route (e.g., balance - delivery + pickup $\leq q$). Constraints (49)–(51) keep track of the used vehicles in each process, and finally constraint (52) ensures that each vehicle is being used in at most one of the three processes.

After all nodes are assigned to exactly one vehicle, the route sequence of these nodes in each vehicle are then constructed. We implement the nearest neighbor approach. When the route sequences have been constructed for all vehicles, Constraints (3)–(6) are checked. For a case when the time horizon (i.e., Constraint (6)) is violated, a node that is visited by a vehicle with the longest transportation time is allocated to another used vehicle, as long as the vehicle capacity and time horizon constraints are satisfied. Only when the insertion is

| | | | | | | |
|---|---|---|---|---|---|---|
| **Vehicle 1** | **0** | **S1** | **S2** | **S3** | **S4** | **0** |
| **Vehicle 2** | **0** | **O1** | **O2** | **O3** | **0** | |
| **Vehicle 3** | **0** | **O4** | **O5** | **0** | | |
| **Vehicle 4** | **0** | **C1** | **C2** | **C3** | **0** | |
| **Vehicle 5** | **0** | **C4** | **C5** | **C6** | **0** | |

**Fig. 2** Example of solution representation with $|C| = 6$, $|O| = 5$, $|S| = 4$, and $|V| = 5$

not possible in any positions will this node then be allocated to a new vehicle. This process is repeated until the time horizon constraint is satisfied. By the end of this initial solution construction process, a solution as illustrated in Fig. 2 is generated, where each row indicates a route sequence of a particular vehicle.

### 4.2.2 Phase 1: Adaptive large neighborhood search (ALNS)

ALNS iteratively removes a subset of nodes from the current solution by using DESTROY operators and re-inserts them back by using REPAIR operators in a more profitable position, hence resulting in a new neigborhood solution. Each operator is associated with a score that will be increased when it is able to generate a better solution. Subsequently, this score is then used to calculate the weight. The higher the weight of an operator is, the greater is the probability that it will be selected in the following iterations.

$Sol_0$, $Sol^*$, and $Sol'$ represent the current solution, the best found solution so far, and the starting solution in every iteration, respectively. Initially, they all equal the initial solution (Line 1 in Algorithm 1). $\Omega_c$, $\Omega_o$, and $\Omega_s$ are defined as pools to retain any observed feasible route during this first phase of the matheuristic. All customer pickup routes, outlet pickup and delivery routes, and supplier delivery routes observed in the initial solution are kept in $\Omega_c$, $\Omega_o$, and $\Omega_s$, respectively (Line 2 of Algorithm 1). A feasible route is defined as a route that starts and ends its trip at the cross-dock, visits one or more non-duplicate nodes, and does not violate vehicle capacity. An illustration of these pools from the routes observed in Figs. 1 and 2 is shown in Fig. 3.

For the initialization, the current temperature ($Temp$) is set to the initial temperature ($T_0$) (Line 3), NOIMPR and ITER are set to zero (Line 4), and FOUNDBESTSOL is set to False (Line 5). NOIMPR is a variable to count the number of iterations without any solution improvement, ITER is a variable that counts how many iterations have been passed so far, while FOUNDBESTSOL is a boolean variable with the value set to True only when a new

$$\Omega c = \begin{bmatrix} 0 & C1 & C2 & C3 & 0 \\ 0 & C4 & C5 & C6 & 0 \end{bmatrix} \quad \Omega o = \begin{bmatrix} 0 & O1 & O2 & O3 & 0 \\ 0 & O4 & O5 & 0 & \end{bmatrix} \quad \Omega s = \begin{bmatrix} 0 & S1 & S2 & S3 & S4 & 0 \end{bmatrix}$$

**Fig. 3** Example of retaining feasible routes in the pools

best found solution is observed. Each operator is initialized with an equal score, weight, and probability to be selected (Line 6).

Let $R = \{R_r | r = 1, 2, \ldots, |R|\}$ and $I = \{I_i | i = 1, 2, \ldots, |I|\}$ be defined as the sets of DESTROY and REPAIR operators, respectively (all these operators are discussed in detail in Section 4.2.4). In every iteration, a set of $\pi$ nodes (e.g. $\pi = 5$) is removed from $Sol_0$ by using a randomly selected $R_r$, and the newly observed routes during the process are added into $\Omega_c$, $\Omega_o$, and $\Omega_s$ (Lines 9–13). Subsequently, those $\pi$ nodes are re-inserted back into $Sol_0$ using a randomly selected $I_i$ (Lines 14–18), resulting in a new neighborhood solution $Sol_0$. The quality of this new $Sol_0$ is compared against $Sol^*$ and $Sol'$ (Line 19). It is directly accepted if it improves $Sol'$ and accepted with a probability of $e^{\frac{-(Sol_0 - Sol')}{Temp}}$ otherwise. When it improves $Sol^*$, FOUNDBESTSOL is set to True (Lines 20–22). Based on the quality of this new $Sol_0$, the operators' scores ($s_j$) are updated (Line 23) by (53), where $\delta_1 > \delta_2 > \delta_3$ (e.g., $\delta_1 = 0.5, \delta_2 = 0.33, \delta_3 = 0.17$).

$$s_j = \begin{cases} s_j + \delta_1, & \text{if } j \text{ is selected and } Sol_0 < Sol^* \\ s_j + \delta_2, & \text{if } j \text{ is selected and } Sol_0 < Sol' \\ s_j + \delta_3, & \text{if } j \text{ is selected and } Sol_0 > Sol', \\ & \text{but it is accepted} \end{cases} \quad \forall j \in R \cup I \tag{53}$$

After $\eta_{ALNS}$ iterations, each operator's weight ($w_j$) is updated by (54), and subsequently each operator's probability ($p_j$) is adjusted by (55) (Lines 24–26).

$$w_j = \begin{cases} (1-\gamma) w_j + \gamma \frac{s_j}{\chi_j}, & \text{if } \chi_j > 0 \\ (1-\gamma) w_j, & \text{if } \chi_j = 0 \end{cases} \quad \forall j \in R \cup I \tag{54}$$

$$p_j = \begin{cases} \frac{w_j}{\sum_{k \in R} w_k} & \forall j \in R \\ \frac{w_j}{\sum_{k \in I} w_k} & \forall j \in I \end{cases} \tag{55}$$

After $\eta_{SA}$ iterations, FOUNDBESTSOL is evaluated. When no improved solution is found, NOIMPR is increased by one (Lines 28–30); otherwise, it is set to zero (Lines 31–33). FOUNDBESTSOL is then set to False to see whether there is any solution improvement in the following iteration (Line 34). Subsequently, $Temp$ is decreased by $\alpha$ (Line 35), and ITER is increased by one for the next iteration. This whole process is terminated when there is no solution improvement after $\theta$ successive temperature reductions or the amount of routes generated in either one of $\Omega_c$, $\Omega_o$, or $\Omega_s$ has reached $\Omega_{max}$ (e.g., $\Omega_{max} = 2000$ in our case).

### 4.2.3 Phase 2: Solving the set partitioning formulation

Given the routes kept in $\Omega_c$, $\Omega_o$, and $\Omega_s$, phase 2 finds a combination of routes that minimizes the total cost of the entire processes. In order to complete this task, we build the following set partitioning formulation. Additional parameters and decision variables needed in this phase are listed in Table 4.

$$Min \sum_{r \in \Omega_c} c'_r x'_r + \sum_{r \in \Omega_o} c''_r x''_r + \sum_{r \in \Omega_s} c'''_r x'''_r + H \left( \sum_{r \in \Omega_c} x'_r + \sum_{r \in \Omega_o} x''_r + \sum_{r \in \Omega_s} x'''_r \right) \tag{56}$$

$$\sum_{r \in \Omega_c} a'_{ir} x'_r = m'_i \quad \forall i \in C \tag{57}$$

**Table 4** List of additional parameters and decision variables in phase 2

| | |
|---|---|
| Additional Parameters | |
| $c'_r$ | transportation cost of route $r$ ($r \in \Omega_c$) |
| $c''_r$ | transportation cost of route $r$ ($r \in \Omega_o$) |
| $c'''_r$ | transportation cost of route $r$ ($r \in \Omega_s$) |
| $t'_r$ | transportation time of route $r$ ($r \in \Omega_c$) |
| $t''_r$ | transportation time of route $r$ ($r \in \Omega_o$) |
| $t'''_r$ | transportation time of route $r$ ($r \in \Omega_s$) |
| $a'_{ir}$ | 1 if customer $i$ is visited by route $r$; 0 otherwise ($i \in C, r \in \Omega_c$) |
| $a''_{ir}$ | 1 if outlet $i$ is visited by route $r$; 0 otherwise ($i \in C, r \in \Omega_o$) |
| $a'''_{ir}$ | 1 if supplier $i$ is visited by route $r$; 0 otherwise ($i \in C, r \in \Omega_s$) |
| Additional Decision Variables | |
| $x'_r$ | 1 if route $r$ is selected; 0 otherwise ($r \in \Omega_c$) |
| $x''_r$ | 1 if route $r$ is selected; 0 otherwise ($r \in \Omega_o$) |
| $x'''_r$ | 1 if route $r$ is selected; 0 otherwise ($r \in \Omega_s$) |

$$\sum_{r \in \Omega_o} a''_{ir} x''_r = m''_i \quad \forall i \in O \tag{58}$$

$$\sum_{r \in \Omega_s} a'''_{ir} x'''_r = m'''_i \quad \forall i \in S \tag{59}$$

$$\sum_{r \in \Omega_c} x'_r + \sum_{r \in \Omega_o} x''_r + \sum_{r \in \Omega_s} x'''_r \leq |V| \tag{60}$$

$$t'_r x'_r \leq Tcp_{max} \quad \forall r \in \Omega_c \tag{61}$$

$$t''_r x''_r \leq Todp_{max} \quad \forall r \in \Omega_o \tag{62}$$

$$t'''_r x'''_r \leq Tsd_{max} \quad \forall r \in \Omega_s \tag{63}$$

$$Tcp_{max} + Todp_{max} + Tsd_{max} \leq T_{max} \tag{64}$$

Objective function (56) minimizes the total transportation and operational costs. All mandatory visited nodes are visited, as required in Constraints (57)–(59). Constraint (60) limits the number of selected routes (i.e., it does not exceed the number of available vehicles). Constraints (61)–(64) ensure that the time horizon is not exceeded. In order to speed up the calculations for the set partitioning, we warm-start the CPLEX solver with the $Sol^*$ obtained from the first phase.

### 4.2.4 ALNS operators

This section explains the list of ALNS operators in greater detail. In total, there are six DESTROY and nine REPAIR operators, respectively. The DESTROY operators are actually created to select which nodes to remove from the current solution in a better way than to

remove nodes randomly. Subsequently, REPAIR operators are implemented to re-insert back those removed nodes in better positions. Each operator has a different criterion to choose nodes.

A brief overview of why those operators are implemented in ALNS is illustrated in Fig. 4. A good combination of the selected operators within one iteration may lead to a better solution. It may reduce the number of vehicles used from three vehicles to two vehicles (see number of routes between S1 S2 S3 S4 S5). It may also reduce the overlap routes between C3 and C4. These two factors (number of vehicles used and travel distance) are the costs employed to evaluate the quality of the solution.

**Random removal ($R_1$)** randomly remove a node from $Sol_0$. It introduces randomness in the search process.

**Worst removal ($R_2$)** remove a node that has the $x^{th}$ highest total cost difference between including and excluding this node in or from its current route. Every time $R_2$ is applied, $x$ is decided by (65), where $y_1 \sim U(0, 1)$, $p = 3$, and $\xi$ is the number of candidate nodes that is formally formulated in (66) (Case 1).

$$x = \lceil y_1^p \times \xi \rceil \tag{65}$$

$$\xi = \begin{cases} Case1 : |C| + |S| - \text{REMOVEDNODES}, & \text{for } R_2 \\ Case2 : |C| + |S| - \text{REMOVEDNODES} - 2, & \text{for } R_4, R_5 \\ Case3 : \text{REMOVEDNODES}, & \text{for } I_9 \end{cases} \tag{66}$$

**Route removal ($R_3$)** randomly select a vehicle and remove its visited nodes. The purpose is to reconstruct the route sequence in a selected vehicle. When a vehicle visits less than $\pi$ nodes, then all nodes visited by this particular vehicle are removed; otherwise, only a subset of random nodes is removed until $\pi$ is reached.

**Node pair removal ($R_4$)** remove a pair of nodes that has the $x^{th}$ highest arc cost between them. Here, $x$ is determined by (65), while $\xi$ follows (66) case 2. The idea is to remove two adjacent nodes with a high arc cost from $Sol_0$, such that when REPAIR reinserts them back into $Sol_0$, they can be located in better, probably separated, positions.

**Worst pair removal ($R_5$)** similar to $R_2$, but $R_5$ chooses a pair of nodes instead of only one node. The underlying difference between $R_4$ and $R_5$ is that $R_4$ only focuses on the arc cost

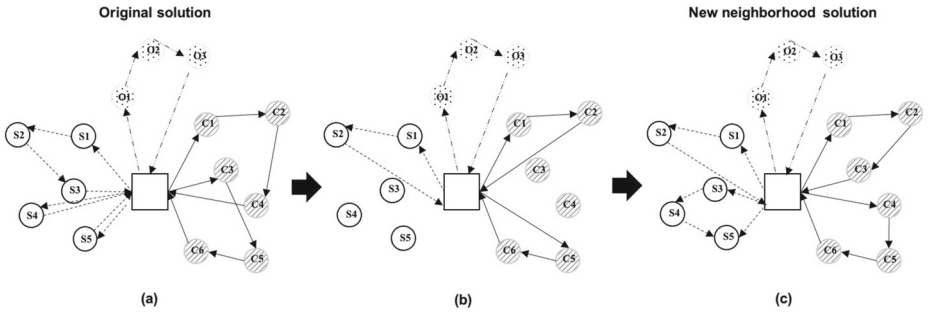


**Fig. 4** Example on how ALNS finds a neighborhood solution: **a** original solution, **b** incomplete solution after applying destroy operator, **c** new neighborhood solution after applying repair operator

between two nodes, while $R_5$ considers the overall costs. The value of $x$ is determined by (65), while $\xi$ is determined by (66) (Case 2).

**Shaw removal ($R_6$)** remove a node that is highly related with other removed nodes such that it is easier to replace the positions of one another during the repair process. Let us define node $i$ as the last removed node and node $j$ as the next candidate to be removed. The relatedness value of node $j$ ($\varphi_j$) to node $i$ is calculated by (67), where $\phi_1$ to $\phi_3$ are weights given to each of the related components in terms of travel distance, travel time, and node position ($l_{ij} = -1$ if nodes $i$ and $j$ are in the same vehicle; 1 otherwise). It means that the lower the $\varphi_j$ is, the more related node $j$ to $i$ is. $R_6$ is started by randomly selecting a node to be removed, setting this node as $i$, and then calculating $\varphi_j$ for the remaining nodes in $Sol_0$. Next, node $j$ with the lowest $\varphi_j$ is selected and removed from $Sol_0$. We implement $\phi_1 = \phi_2 = \phi_3 = \frac{1}{3}$.

$$\varphi_j = \begin{cases} \phi_1 e'_{ij} + \phi_2 t'_{ij} + \phi_3 l_{ij}, & \text{if } i \in C \\ \phi_1 e''_{ij} + \phi_2 t''_{ij} + \phi_3 l_{ij}, & \text{if } i \in O \\ \phi_1 e'''_{ij} + \phi_2 t'''_{ij} + \phi_3 l_{ij}, & \text{if } i \in S \end{cases} \tag{67}$$

**Greedy insertion ($I_1$)** insert a node to a position with the lowest insertion cost (i.e., total cost ($TC$) difference between after and before inserting a node into a particular position).

***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 $TC_k(j)$). Here, $k$ is equal to 2 for $I_2$, 3 for $I_3$, and 4 for $I_4$. The node with the largest regret value (see (68)) is then inserted in its best position.

$$\underset{j \in \text{REMOVEDNODES}}{\text{argmax}} \left\{ \sum_{i=2}^{k} (TC_i(j) - TC_1(j)) \right\} \tag{68}$$

**Greedy insertion with noise function ($I_5$)** an extension of $I_1$ by introducing a noise function to the objective function value (69) when calculating the total cost after inserting a node into a particular position. Here, $\bar{e}$ is the maximum transportation cost between nodes (problem-dependent), $\mu$ is a noise parameter (set to 0.1 in our case), and $y_2 \sim U(-1, 1)$. This noise function introduces randomness in the process.

$$TC_{new} = TC + \bar{e} \times \mu \times y_2 \tag{69}$$

***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 (69) when calculating $TC_i$ $\forall i \in \{1, ..., k\}$ in (68). This noise function also introduces randomness in the process.

**GRASP insertion** ($I_9$): similar to $I_1$, but instead of choosing the position of the node with the lowest insertion cost, $I_9$ chooses the insertion position of a node that has the $x^{th}$ lowest insertion cost. By doing so, other nodes may have a chance to be selected. Here, $x$ is determined by (65), while $\xi$ is determined by (66) (Case 3).

## 5 Computational results

This section presents the experiments and the analysis of the results. We first describe the benchmark instances as well as the experimental set-up of this study in Section 5.1. Section 5.2 summarizes the parameter setting used for the proposed matheuristic. Section 5.3 presents the experimental results on the given benchmark instances and a thorough analysis. The results of the newly proposed larger instances are also presented. The importance of solving the set partitioning formulation is discussed in Section 5.4. Section 5.5 analyzes the performance of the ALNS operators.

### 5.1 Benchmark instances

Widjaja et al. [13] introduced two sets of benchmark VRP-RCD instances. Both sets consist of 30 problem instances with 15 (Set 1) and 40 (Set 2) nodes, respectively. These instances were initially adopted from the available benchmark VRPCD instances [4] by adding several parameters that are related to the VRP-RCD, such as:

- number of outlets ($|O|$)
- outlets' transportation cost ($e''_{ij}$), time ($t''_{ij}$), and demand ($d''_{ik}$)
- amount of customers and outlets' returned products ($r'_{ik}$, $r''_{ik}$)
- percentage of defective products ($p_k$)

We also introduce a new set (Set 3) with larger VRP-RCD instances of 65 nodes by applying the above-mentioned modifications to a set of VRPCD instances. Table 5 summarizes the parameters of all sets. We make these instances and our solutions available online on https://www.mech.kuleuven.be/en/cib/op/opmainpage#section-50.

The mathematical model is solved by optimization software CPLEX 12.9.0.0, limited to three hours for solving each instance. Our proposed matheuristic is coded in C++ with an additional CPLEX solver to solve the set partitioning formulation. The matheuristic is run five times for each instance. All experiments are executed on a computer with Intel Core i7-8700 CPU @ 3.20 GHz processor, 32.0 GB RAM.

### 5.2 Parameter setting

We perform a standard full factorial design to determine the parameter values of six parameters used in the matheuristic. For each parameter, three different values are considered, as summarized in Table 6. In total, there are 729 combinations.

The experiments are conducted on 10 randomly selected instances from Sets 1 to 3, and the average costs and CPU times are recorded. For each combination, we calculate the average cost and average CPU time over all test instances. However, before taking the average, those values are first normalized since each instance has its own range of cost. Figure 5 plots the trade-off between solution quality (cost) and CPU time. This figure also gives us some sense of which parameters affect cost and CPU time the most.

**Table 5** VRP-RCD parameter values

| | Set 1 | Set 2 | Set 3 |
|---|---|---|---|
| $\|S\|$ | 4 | 7 | 12 |
| $\|C\|$ | 6 | 23 | 38 |
| $\|O\|$ | 5 | 10 | 15 |
| $\|V\|$ | 10 | 20 | 30 |
| $q$ | 70 | 150 | 150 |
| $c$ | 1 | 1 | 1 |
| $H$ | 1000 | 1000 | 1000 |
| $T_{max}$ | 960 | 960 | 960 |
| $e'_{ij}, e''_{ij}, e'''_{ij}$ | U ∼ (48,560) | U ∼ (48,480) | U ∼ (48,560) |
| $t'_{ij}, t''_{ij}, t'''_{ij}$ | U ∼ (20,200) | U ∼ (20,100) | U ∼ (20,200) |
| $\sum_{k\in S} d''_{ik}$ | U ∼ (5,50) | U ∼ (5,20) | U ∼ (5,30) |
| $\sum_{k\in S} r'_{ik}, \sum_{k\in S} r''_{ik}$ | U ∼ (5,50) | U ∼ (5,20) | U ∼ (5,30) |
| $p_k$ | U ∼ (0,0.05) | U ∼ (0,0.05) | U ∼ (0,0.05) |

Figure 5 shows that the stopping criterion ($\theta$) and number of iterations in each temperature ($\eta_{SA}$) have a huge impact on both cost and CPU time, compared to any other parameters. The higher the values are set to these parameters, the better is the cost we may get. However, the CPU time is also greatly impacted. We hence decide to choose the middle value for both parameters.

From Fig. 5a and d we observe that setting the initial temperature ($T_0$) and the reaction factor ($\gamma$) with high values has a positive impact on minimizing both total cost and CPU time. Setting $\alpha$ (see Fig. 5b) to 0.9 clearly reduces the CPU time, but results in the worst total cost. Hence, we choose 0.95, although it results in a slight increment on CPU time. A trade-off between solution quality and CPU time must be made on deciding the value of $\eta_{ALNS}$ (see Fig. 5e). The low level (e.g., 100) might save CPU time, but the solution quality is worse compared to the high level (300). For this case, we prioritize solution quality over CPU time. Hence, for the rest of our experiments, we set the following parameter values: $T_0 = 20$, $\alpha = 0.95$, $\theta = 50$, $\gamma = 0.9$, $\eta_{ALNS} = 300$, and $\eta_{SA} = (|S| + |C| + |O|) \times 2$.

### 5.3 Solving benchmark VRP-RCD instances

Optimization software, CPLEX, is able to obtain optimal solutions for Set 1 instances, while for instances in Sets 2 and 3, we only report the best solution found, because CPLEX

**Table 6** Matheuristic parameter values

| Parameter | Value |
|---|---|
| $T_0$ | 5, 10, and 20 |
| $\alpha$ | 0.85, 0.9, and 0.95 |
| $\theta$ | 10, 50, and 100 |
| $\gamma$ | 0.7, 0.8, and 0.9 |
| $\eta_{ALNS}$ | 100, 200, and 300 |
| $\eta_{SA}$ | $(\|S\| + \|C\| + \|O\|) \times 1$, $(\|S\| + \|C\| + \|O\|) \times 2$, and $(\|S\| + \|C\| + \|O\|) \times 3$ |

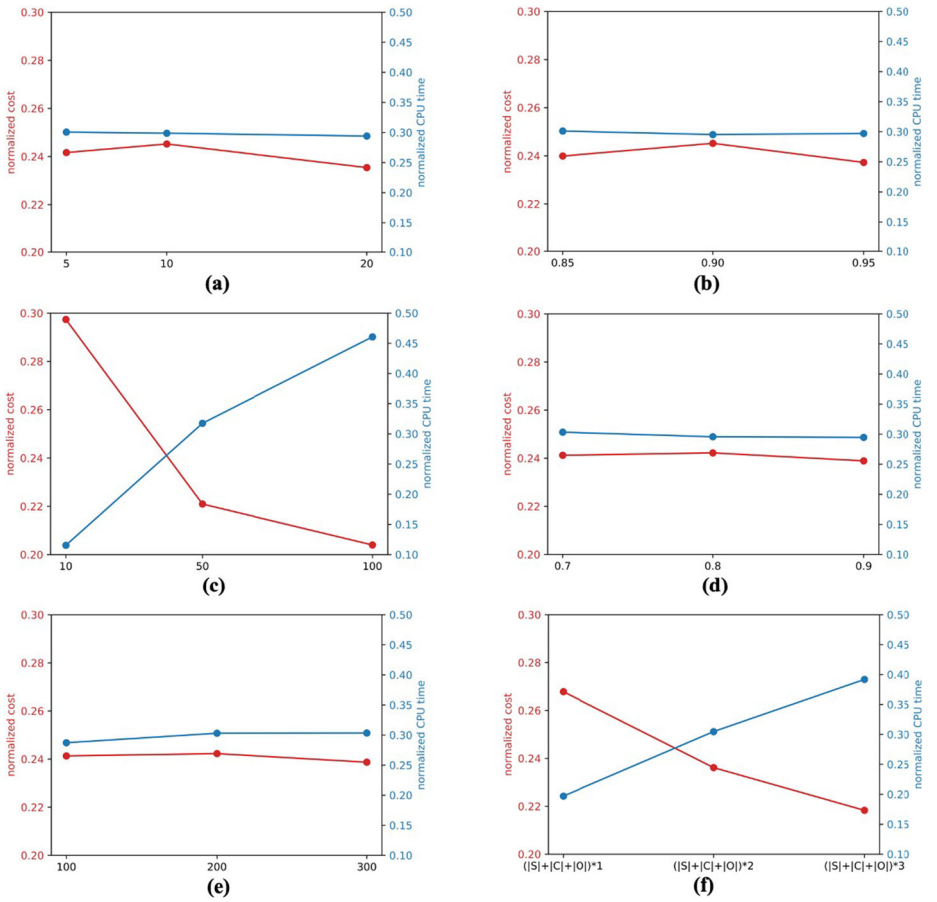


**Fig. 5** Cost and CPU time comparison in each parameter value: **a** $T_0$, **b** $\alpha$, **c** $\theta$, **d** $\gamma$, **e** $\eta_{ALNS}$, and **f** $\eta_{SA}$

is unable to solve them optimally within three hours of CPU time. In some instances, CPLEX is even unable to find any feasible solution, which we mark as "-". For the proposed matheuristic, we present the average objective function value over five runs (total cost, $\overline{TC}$) with the respective gap towards CPLEX results (following (70)). The best solution from all five runs is also included, as indicated by $\text{TC}_{best}$.

$$Gap(\%) = \frac{\overline{TC}_{matheuristic} - TC_{CPLEX}}{TC_{CPLEX}} \times 100 \tag{70}$$

In order to evaluate the performance of the proposed matheuristic, we compare the results against those of: 1) a pure ALNS algorithm, 2) ALNS without the adaptive scheme to select operators but with an SA acceptance criteria (namely LNS-SA), and 3) ALNS-SA of [16]. These three algorithms are also run five times with the same CPU time as the matheuristic for a fair comparison. The gap between the matheuristic and baseline algorithm results is calculated by (71). The CPU times of CPLEX and all algorithms (matheuristic, ALNS, LNS-SA, and ALNS-SA) are also summarized for further assessing the performance of our

proposed matheuristic. All results are summarized in Tables 7, 8, and 9 for Sets 1, 2, and 3, respectively. The best or lowest cost obtained for each instance is indicated in **bold**.

$$Gap(\%) = \frac{\overline{TC}_{matheuristic} - \overline{TC}_{baseline}}{\overline{TC}_{baseline}} \times 100 \tag{71}$$

For Set 1 (refer to Table 7), the proposed matheuristic is able to find all optimal solutions obtained by CPLEX within very short CPU times. Furthermore, we note the consistency of the matheuristic's performance when obtaining these solutions, as it always finds the optimal solution in each run. On the other hand, ALNS, LNS-SA, and ALNS-SA are only able to find the optimal solutions of 28, 27, and 25 instances, respectively.

When we consider a larger set of instances (Set 2 in Table 8), CPLEX is unable to get any optimal solutions within three hours for these instances. On the contrary, all algorithms (ALNS, LNS-SA, ALNS-SA, and matheuristic) are able to obtain better solutions than CPLEX for all instances within less than half a minute on average, with matheuristic results being the best among all. It outperforms ALNS, LNS-SA, and ALNS-SA by 0.71%, 0.5%, and 0.44%, respectively, on average.

In terms of the best solution found ($TC_{best}$) within five runs, LNS-SA is able to find the most $TC_{best}$, which implicitly shows us that giving chances to the previously-less-successful-operators to generate new solutions could be beneficial, followed by the proposed matheuristic, ALNS, and ALNS-SA, respectively. In some instances, ALNS-SA obtains better solutions than those of the matheuristic. The reason is that the stopping criteria for the ALNS-SA inside the matheuristic are not used as the stopping criteria for the pure ALNS-SA (which is time). Therefore, the pure ALNS-SA generates more routes than the routes available for the set partitioning in the matheuristic. Some of the ALNS results are also better than those of the matheuristic, which shows us that it might be beneficial to always generate or explore a new neighborhood solution from the best solution found so far (*Sol**), instead of the starting point of each iteration (*Sol'*).

Finally, for the largest instance set (Set 3 in Table 9), CPLEX is unable to provide any feasible solution within a given three hours of CPU time. ALNS, LNS-SA, and ALNS-SA provide reasonably good solutions that are further improved by the matheuristic by 1.18%, 0.67%, and 0.62%, respectively, on average. Comparing the best solution found within five runs ($TC_{best}$), all algorithms provide almost similar numbers of $TC_{best}$, which are 9, 6, 8, and 8 instances for ALNS, LNS-SA, ALNS-SA, and matheuristic, respectively. In this instance set, all algorithms are able to provide these solutions with less than 10 seconds of computational time.

These results conclude that designing a matheuristic is clearly beneficial for obtaining a better solution quality. On average, our proposed matheuristic is able to obtain the best solutions compared to all baseline algorithms. This shows the ability of the matheuristic to provide stable and robust results over its runs. By using a heuristic scheme to perform the column generation, the search process can be done in much shorter running time compared to the branch-and-cut process as in CPLEX. However, the matheuristic can only prove the optimality over all observed columns found in the first phase, unlike CPLEX, which is able to prove it among all possible solutions.

### 5.4 Importance of solving the set partitioning formulation

We analyze the importance of implementing the set partitioning formulation (phase 2) in the proposed matheuristic. For each instance, we calculate the improvement made by solving phase 2 of the matheuristic, compared to solving phase 1 only. We therefore present the

**Table 7** Results on Set 1

| Instance | CPLEX | ALNS | | LNS-SA | | ALNS-SA [16] | | Matheuristic | | Gap (%) towards | | | | CPU time (s) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TC | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | CPLEX | ALNS | LNS-SA | ALNS-SA [16] | CPLEX | Algorithms |
| 1 | **10982.0** | **10982.0** | **10982.0** | **10982.0** | **10982.0** | **10982.0** | **10982.0** | **10982.0** | **10982.0** | 0.00 | 0.00 | 0.00 | 0.00 | 81.98 | 0.30 |
| 2 | **8304.0** | **8304.0** | **8304.0** | **8304.0** | **8304.0** | **8304.0** | **8304.0** | **8304.0** | **8304.0** | 0.00 | 0.00 | 0.00 | 0.00 | 21.18 | 0.28 |
| 3 | **10076.0** | 10304.0 | 10304.0 | 10304.0 | 10304.0 | 10304.0 | 10304.0 | **10076.0** | **10076.0** | 0.00 | −2.21 | −2.21 | −2.21 | 50.54 | 0.20 |
| 4 | **10753.0** | 10881.4 | **10753.0** | 10795.8 | **10753.0** | **10753.0** | **10753.0** | **10753.0** | **10753.0** | 0.00 | −1.18 | −0.40 | 0.00 | 38.52 | 0.24 |
| 5 | **8584.0** | **8584.0** | **8584.0** | **8584.0** | **8584.0** | **8584.0** | **8584.0** | **8584.0** | **8584.0** | 0.00 | 0.00 | 0.00 | 0.00 | 20.97 | 0.24 |
| 6 | **10965.0** | **10965.0** | **10965.0** | **10965.0** | **10965.0** | **10965.0** | **10965.0** | **10965.0** | **10965.0** | 0.00 | 0.00 | 0.00 | 0.00 | 45.93 | 0.17 |
| 7 | **9703.0** | 9733.4 | **9703.0** | 9763.8 | **9703.0** | 9855.0 | 9855.0 | **9703.0** | **9703.0** | 0.00 | −0.31 | −0.62 | −1.54 | 45.68 | 0.23 |
| 8 | **7630.0** | 7868.4 | **7630.0** | 8106.8 | **7630.0** | 8226.0 | 8226.0 | **7630.0** | **7630.0** | 0.00 | −3.03 | −5.88 | −7.25 | 5.04 | 0.22 |
| 9 | **9519.0** | **9519.0** | **9519.0** | **9519.0** | **9519.0** | **9519.0** | **9519.0** | **9519.0** | **9519.0** | 0.00 | 0.00 | 0.00 | 0.00 | 13.63 | 0.18 |
| 10 | **9486.0** | **9486.0** | **9486.0** | **9486.0** | **9486.0** | **9486.0** | **9486.0** | **9486.0** | **9486.0** | 0.00 | 0.00 | 0.00 | 0.00 | 24.41 | 0.19 |
| 11 | **10581.0** | **10581.0** | **10581.0** | 10585.0 | **10581.0** | 10589.0 | **10581.0** | **10581.0** | **10581.0** | 0.00 | 0.00 | −0.04 | −0.08 | 74.32 | 0.21 |
| 12 | **11381.0** | 11847.8 | 11571.0 | 11457.0 | **11381.0** | **11381.0** | **11381.0** | **11381.0** | **11381.0** | 0.00 | −3.94 | −0.66 | 0.00 | 24.01 | 0.23 |
| 13 | **9432.0** | 9516.4 | **9432.0** | 9600.8 | **9432.0** | **9432.0** | **9432.0** | **9432.0** | **9432.0** | 0.00 | −0.89 | −1.76 | 0.00 | 7.69 | 0.24 |
| 14 | **9428.0** | 9649.6 | **9428.0** | 9705.0 | 9705.0 | 9705.0 | 9705.0 | **9428.0** | **9428.0** | 0.00 | −2.30 | −2.85 | −2.85 | 23.28 | 0.19 |
| 15 | **8993.0** | **8993.0** | **8993.0** | **8993.0** | **8993.0** | **8993.0** | **8993.0** | **8993.0** | **8993.0** | 0.00 | 0.00 | 0.00 | 0.00 | 17.43 | 0.15 |
| 16 | **9591.0** | **9591.0** | **9591.0** | **9591.0** | **9591.0** | **9591.0** | **9591.0** | **9591.0** | **9591.0** | 0.00 | 0.00 | 0.00 | 0.00 | 33.46 | 0.26 |
| 17 | **10049.0** | **10049.0** | **10049.0** | **10049.0** | **10049.0** | **10049.0** | **10049.0** | **10049.0** | **10049.0** | 0.00 | 0.00 | 0.00 | 0.00 | 5874.90 | 0.16 |
| 18 | **9375.0** | **9375.0** | **9375.0** | **9375.0** | **9375.0** | **9375.0** | **9375.0** | **9375.0** | **9375.0** | 0.00 | 0.00 | 0.00 | 0.00 | 9372.26 | 0.20 |
| 19 | **8012.0** | **8012.0** | **8012.0** | 8032.8 | **8012.0** | 8074.6 | 8038.0 | **8012.0** | **8012.0** | 0.00 | 0.00 | −0.26 | −0.78 | 23.42 | 0.22 |

**Table 7** (continued)

| Instance | CPLEX | ALNS | | LNS-SA | | ALNS-SA [16] | | Matheuristic | | Gap (%) towards | | | | CPU time (s) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TC | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | CPLEX | ALNS | LNS-SA | ALNS-SA [16] | CPLEX | Algorithms |
| 20 | **10881.0** | **10881.0** | **10881.0** | **10881.0** | **10881.0** | **10881.0** | **10881.0** | **10881.0** | **10881.0** | 0.00 | 0.00 | 0.00 | 0.00 | 18.83 | 0.17 |
| 21 | **8235.0** | **8235.0** | **8235.0** | **8235.0** | **8235.0** | **8235.0** | **8235.0** | **8235.0** | **8235.0** | 0.00 | 0.00 | 0.00 | 0.00 | 22.67 | 0.22 |
| 22 | **8875.0** | **8875.0** | **8875.0** | **8875.0** | **8875.0** | 8970.2 | **8875.0** | **8875.0** | **8875.0** | 0.00 | 0.00 | 0.00 | −1.06 | 13.25 | 0.22 |
| 23 | **8728.0** | 9061.6 | **8728.0** | 9145.0 | 9145.0 | 8894.8 | **8728.0** | **8728.0** | **8728.0** | 0.00 | −3.68 | −4.56 | −1.88 | 866.68 | 0.25 |
| 24 | **11719.0** | **11719.0** | **11719.0** | **11719.0** | **11719.0** | **11719.0** | **11719.0** | **11719.0** | **11719.0** | 0.00 | 0.00 | 0.00 | 0.00 | 28.07 | 0.18 |
| 25 | **10686.0** | **10686.0** | **10686.0** | **10686.0** | **10686.0** | **10686.0** | **10686.0** | **10686.0** | **10686.0** | 0.00 | 0.00 | 0.00 | 0.00 | 21.81 | 0.14 |
| 26 | **9042.0** | **9042.0** | **9042.0** | **9042.0** | **9042.0** | **9042.0** | **9042.0** | **9042.0** | **9042.0** | 0.00 | 0.00 | 0.00 | 0.00 | 60.50 | 0.23 |
| 27 | **10545.0** | **10545.0** | **10545.0** | **10545.0** | **10545.0** | **10545.0** | **10545.0** | **10545.0** | **10545.0** | 0.00 | 0.00 | 0.00 | 0.00 | 23.81 | 0.20 |
| 28 | **10636.0** | **10636.0** | **10636.0** | **10636.0** | **10636.0** | **10636.0** | **10636.0** | **10636.0** | **10636.0** | 0.00 | 0.00 | 0.00 | 0.00 | 38.19 | 0.19 |
| 29 | **9130.0** | **9130.0** | **9130.0** | **9130.0** | **9130.0** | **9130.0** | **9130.0** | **9130.0** | **9130.0** | 0.00 | 0.00 | 0.00 | 0.00 | 64.07 | 0.21 |
| 30 | **9700.0** | 9782.2 | **9700.0** | **9700.0** | **9700.0** | **9700.0** | **9700.0** | **9700.0** | **9700.0** | 0.00 | −0.84 | 0.00 | 0.00 | 30.11 | 0.16 |
| Avg | | | | | | | | | | 0.00 | −0.61 | −0.64 | −0.59 | 566.22 | 0.21 |

**Table 8** Results on Set 2

| Instance | CPLEX | ALNS | | LNS-SA | | ALNS-SA [16] | | Matheuristic | | Gap (%) towards | | | | CPU time (s) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TC | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | CPLEX | ALNS | LNS-SA | ALNS-SA [16] | CPLEX | Algorithms |
| 1 | 30988.0 | 12728.8 | 12293.0 | 12421.6 | **12176.0** | 12635.0 | 12310.0 | 12383.0 | 12262.0 | −60.04 | −2.72 | −0.31 | −1.99 | 10800 | 14.28 |
| 2 | 19270.0 | 12589.8 | 12477.0 | 12528.0 | **12257.0** | 12605.4 | 12479.0 | 12660.0 | 12422.0 | −34.30 | 0.56 | 1.05 | 0.43 | 10800 | 28.43 |
| 3 | 14062.0 | 11942.8 | 11653.0 | 11663.8 | 11537.0 | 11581.0 | 11354.0 | 11522.0 | **11211.0** | −18.06 | −3.52 | −1.22 | −0.51 | 10800 | 5.27 |
| 4 | 28668.0 | 12176.6 | 11875.0 | 12277.6 | 11791.0 | 12169.8 | 11872.0 | 11949.4 | **11552.0** | −58.32 | −1.87 | −2.67 | −1.81 | 10800 | 2.47 |
| 5 | 13945.0 | 12027.8 | 11915.0 | 11929.6 | **11687.0** | 11895.8 | 11849.0 | 11993.2 | 11767.0 | −14.00 | −0.29 | 0.53 | 0.82 | 10800 | 36.09 |
| 6 | 11837.0 | 10678.0 | 10552.0 | 10749.4 | 10448.0 | 10780.2 | 10537.0 | 10681.0 | **10313.0** | −9.77 | 0.03 | −0.64 | −0.92 | 10800 | 16.71 |
| 7 | 27184.0 | 12435.2 | 12239.0 | 12365.0 | **12182.0** | 12447.2 | 12235.0 | 12461.0 | 12349.0 | −54.16 | 0.21 | 0.78 | 0.11 | 10800 | 23.57 |
| 8 | 16111.0 | 12679.8 | 12503.0 | 12508.6 | **12304.0** | 12631.4 | 12476.0 | 12586.8 | 12361.0 | −21.87 | −0.73 | 0.63 | −0.35 | 10800 | 24.20 |
| 9 | 31137.0 | 13321.0 | 13170.0 | 13132.6 | **12782.0** | 13305.4 | 13145.0 | 13354.8 | 13064.0 | −57.11 | 0.25 | 1.69 | 0.37 | 10800 | 34.40 |
| 10 | 16107.0 | 12296.6 | 12173.0 | 12129.2 | **11855.0** | 12317.6 | 12248.0 | 12390.6 | 12187.0 | −23.07 | 0.76 | 2.16 | 0.59 | 10800 | 16.91 |
| 11 | 17817.0 | 12109.0 | 11917.0 | 12052.0 | **11839.0** | 12103.8 | 11986.0 | 12193.4 | 11993.0 | −31.56 | 0.70 | 1.17 | 0.74 | 10800 | 11.39 |
| 12 | 23026.0 | 12751.4 | 12641.0 | 12662.2 | 12518.0 | 12344.6 | **12030.0** | 12352.8 | 12184.0 | −46.35 | −3.13 | −2.44 | 0.07 | 10800 | 28.67 |
| 13 | 24684.0 | 12765.4 | 12043.0 | 12606.8 | 12221.0 | 12267.6 | 11983.0 | 12311.2 | **11949.0** | −50.12 | −3.56 | −2.34 | 0.36 | 10800 | 25.09 |
| 14 | 20889.0 | 12218.0 | 12000.0 | 12870.4 | 12229.0 | 12469.8 | 12103.0 | 12225.8 | **11988.0** | −41.47 | 0.06 | −5.01 | −1.96 | 10800 | 10.48 |
| 15 | 16497.0 | 12560.6 | 12147.0 | 12434.4 | 12205.0 | 12318.2 | **12102.0** | 12302.6 | 12161.0 | −25.43 | −2.05 | −1.06 | −0.13 | 10800 | 7.94 |
| 16 | 22017.0 | 12122.6 | **11697.0** | 11958.4 | 11708.0 | 11978.8 | 11786.0 | 11851.0 | 11706.0 | −46.17 | −2.24 | −0.90 | −1.07 | 10800 | 11.30 |
| 17 | 16256.0 | 12711.0 | 12544.0 | 12809.8 | 12626.0 | 12954.8 | 12792.0 | 12875.0 | **12475.0** | −20.80 | 1.29 | 0.51 | −0.62 | 10800 | 23.70 |
| 18 | 29582.0 | 12688.6 | 12448.0 | 12784.4 | **12364.0** | 12827.8 | 12496.0 | 12547.6 | 12458.0 | −57.58 | −1.11 | −1.85 | −2.18 | 10800 | 13.22 |
| 19 | 17653.0 | 12229.2 | **12113.0** | 12304.4 | 12206.0 | 12294.6 | 12215.0 | 12260.0 | 12179.0 | −30.55 | 0.25 | −0.36 | −0.28 | 10800 | 30.59 |

**Table 8** (continued)

| Instance | CPLEX | ALNS | | LNS-SA | | ALNS-SA [16] | | Matheuristic | | Gap (%) towards | | | | CPU time (s) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TC | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | CPLEX | ALNS | LNS-SA | ALNS-SA [16] | CPLEX | Algorithms |
| 20 | 27966.0 | 12868.4 | 12640.0 | 12742.0 | **12407.0** | 12812.4 | 12640.0 | 12768.8 | 12620.0 | −54.34 | −0.77 | 0.21 | −0.34 | 10800 | 15.22 |
| 21 | 25482.0 | 12240.2 | **11870.0** | 12223.4 | 12102.0 | 12340.0 | 12262.0 | 12356.6 | 12203.0 | −51.51 | 0.95 | 1.09 | 0.13 | 10800 | 17.64 |
| 22 | 15263.0 | 12708.6 | **12361.0** | 12726.6 | 12389.0 | 12784.0 | 12534.0 | 12722.6 | 12460.0 | −16.64 | 0.11 | −0.03 | −0.48 | 10800 | 20.58 |
| 23 | 22747.0 | 12062.0 | **11646.0** | 11980.0 | 11771.0 | 12114.8 | 11715.0 | 11813.8 | 11668.0 | −48.06 | −2.06 | −1.39 | −2.48 | 10800 | 6.45 |
| 24 | 14834.0 | 12179.6 | 11995.0 | 12144.8 | **11923.0** | 12375.2 | 12220.0 | 12336.0 | 12012.0 | −16.84 | 1.28 | 1.57 | −0.32 | 10800 | 23.80 |
| 25 | 26117.0 | 12747.4 | 12496.0 | 12687.8 | 12515.0 | 12825.4 | 12686.0 | 12576.2 | **12276.0** | −51.85 | −1.34 | −0.88 | −1.94 | 10800 | 13.43 |
| 26 | 28020.0 | 13044.6 | 12885.0 | 13316.2 | 12985.0 | 12868.8 | 12750.0 | 12882.8 | **12595.0** | −54.02 | −1.24 | −3.25 | 0.11 | 10800 | 25.13 |
| 27 | 26497.0 | 12980.8 | 12780.0 | 13053.4 | 12644.0 | 12849.0 | 12661.0 | 12852.2 | **12631.0** | −51.50 | −0.99 | −1.54 | 0.02 | 10800 | 4.98 |
| 28 | 29305.0 | 13102.2 | **12673.0** | 13291.8 | 13078.0 | 13109.8 | 12926.0 | 13162.4 | 13031.0 | −55.08 | 0.46 | −0.97 | 0.40 | 10800 | 32.24 |
| 29 | 21786.0 | 12546.6 | **12145.0** | 12417.8 | 12165.0 | 12301.8 | 12160.0 | 12389.0 | 12149.0 | −43.13 | −1.26 | −0.23 | 0.71 | 10800 | 18.08 |
| 30 | 17600.0 | 12347.4 | **11934.0** | 12343.0 | 12025.0 | 12506.2 | 12478.0 | 12419.2 | 12213.0 | −29.44 | 0.58 | 0.62 | −0.70 | 10800 | 25.36 |
| Avg | | | | | | | | | | −39.11 | −0.71 | −0.50 | −0.44 | 10800 | 18.92 |

**Table 9** Results on Set 3

| Instance | CPLEX | ALNS | | LNS-SA | | ALNS-SA [16] | | Matheuristic | | Gap (%) towards | | | | CPU time (s) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TC | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | CPLEX | ALNS | LNS-SA | ALNS-SA [16] | CPLEX | Algorithms |
| 1 | TIME | 29239.8 | 28250.0 | 28499.8 | **27526.0** | 29694.6 | 28332.0 | 28559.6 | 27793.0 | – | −2.33 | 0.21 | −3.82 | 10800 | 5.45 |
| 2 | TIME | 29170.6 | **27926.0** | 29444.6 | 28396.0 | 28692.6 | 27931.0 | 28628.2 | 28277.0 | – | −1.86 | −2.77 | −0.22 | 10800 | 3.26 |
| 3 | TIME | 26775.4 | 26006.0 | 26815.6 | 26139.0 | 26133.0 | **24581.0** | 26129.0 | 25218.0 | – | −2.41 | −2.56 | −0.02 | 10800 | 3.21 |
| 4 | TIME | 28645.0 | 27363.0 | 27748.4 | **26449.0** | 28373.8 | 27399.0 | 27271.0 | 26782.0 | – | −4.80 | −1.72 | −3.89 | 10800 | 3.51 |
| 5 | TIME | 28412.0 | **27265.0** | 28469.6 | 27573.0 | 29006.0 | 28459.0 | 28702.0 | 27444.0 | – | 1.02 | 0.82 | −1.05 | 10800 | 7.25 |
| 6 | TIME | 28140.0 | 27547.0 | 27587.0 | **26487.0** | 27658.6 | 27047.0 | 27583.0 | 26831.0 | – | −1.98 | −0.01 | −0.27 | 10800 | 8.18 |
| 7 | TIME | 28664.4 | 27968.0 | 28130.4 | 27547.0 | 28312.4 | **27507.0** | 28326.2 | 27756.0 | – | −1.18 | 0.70 | 0.05 | 10800 | 9.86 |
| 8 | TIME | 26225.0 | **25072.0** | 26156.8 | 25458.0 | 26371.0 | 25232.0 | 26851.8 | 25691.0 | – | 2.39 | 2.66 | 1.82 | 10800 | 17.78 |
| 9 | TIME | 26064.0 | 25409.0 | 26315.8 | **25079.0** | 26495.8 | 25458.0 | 26211.4 | 25339.0 | – | 0.57 | −0.40 | −1.07 | 10800 | 10.69 |
| 10 | TIME | 29243.4 | **28012.0** | 30176.8 | 29525.0 | 29882.8 | 28327.0 | 28977.6 | 28182.0 | – | −0.91 | −3.97 | −3.03 | 10800 | 6.40 |
| 11 | TIME | 27659.0 | **26300.0** | 27574.2 | 27080.0 | 27950.2 | 26901.0 | 28084.4 | 27831.0 | – | 1.54 | 1.85 | 0.48 | 10800 | 16.18 |
| 12 | TIME | 25742.4 | **25013.0** | 26045.2 | 25450.0 | 26060.2 | 25628.0 | 26208.8 | 25225.0 | – | 1.81 | 0.63 | 0.57 | 10800 | 10.03 |
| 13 | TIME | 25495.8 | 24313.0 | 24994.8 | **24067.0** | 25349.6 | 25010.0 | 25327.4 | 24691.0 | – | −0.66 | 1.33 | −0.09 | 10800 | 6.83 |
| 14 | TIME | 27152.6 | 26380.0 | 27402.2 | 26407.0 | 27111.2 | 26193.0 | 27009.8 | **26171.0** | – | −0.53 | −1.43 | −0.37 | 10800 | 7.87 |
| 15 | TIME | 31554.4 | 29734.0 | 30488.8 | 29793.0 | 30352.4 | 29694.0 | 29164.8 | **28222.0** | – | −7.57 | −4.34 | −3.91 | 10800 | 3.96 |
| 16 | TIME | 28579.0 | 27623.0 | 27863.4 | 27070.0 | 28308.8 | 27535.0 | 27971.8 | **26630.0** | – | −2.12 | 0.39 | −1.19 | 10800 | 3.62 |
| 17 | TIME | 28608.8 | **26963.0** | 28929.8 | 28113.0 | 28308.0 | **26963.0** | 28236.4 | 27708.0 | – | −1.30 | −2.40 | −0.25 | 10800 | 5.82 |
| 18 | TIME | 26880.6 | 26171.0 | 26893.0 | 26081.0 | 26906.0 | **25872.0** | 27167.2 | 26715.0 | – | 1.07 | 1.02 | 0.97 | 10800 | 10.74 |
| 19 | TIME | 28895.4 | 27736.0 | 29089.8 | 27038.0 | 28387.8 | **26819.0** | 28786.6 | 26911.0 | – | −0.38 | −1.04 | 1.40 | 10800 | 11.77 |
| 20 | TIME | 26435.0 | **25217.0** | 26641.8 | 25953.0 | 26706.6 | 25746.0 | 26459.0 | 25242.0 | – | 0.09 | −0.69 | −0.93 | 10800 | 3.46 |
| 21 | TIME | 26648.0 | 24905.0 | 26223.8 | 24872.0 | 27148.0 | 26108.0 | 25556.6 | **24687.0** | – | −4.10 | −2.54 | −5.86 | 10800 | 8.28 |

**Table 9** (continued)

| Instance | CPLEX | ALNS | | LNS-SA | | ALNS-SA [16] | | Matheuristic | | Gap (%) towards | | | | CPU time (s) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TC | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | $\overline{TC}$ | $TC_{best}$ | CPLEX | ALNS | LNS-SA | ALNS-SA [16] | CPLEX | Algorithms |
| 22 | TIME | 24970.8 | 23759.0 | 25138.8 | **23087.0** | 24976.4 | 24360.0 | 25187.4 | 24561.0 | – | 0.87 | 0.19 | 0.84 | 10800 | 6.05 |
| 23 | TIME | 28848.8 | 27044.0 | 27670.0 | 26913.0 | 28186.6 | 27776.0 | 27610.4 | **26910.0** | – | −4.29 | −0.22 | −2.04 | 10800 | 12.23 |
| 24 | TIME | 24740.8 | **23663.0** | 25013.0 | 24102.0 | 24232.6 | 23827.0 | 24699.2 | 23723.0 | – | −0.17 | −1.25 | 1.93 | 10800 | 13.56 |
| 25 | TIME | 28255.0 | 27386.0 | 28221.2 | 27219.0 | 27789.8 | **26879.0** | 28089.8 | 27351.0 | – | −0.58 | −0.47 | 1.08 | 10800 | 6.21 |
| 26 | TIME | 27964.6 | 26773.0 | 27904.0 | 26997.0 | 27960.8 | 27003.0 | 27156.2 | **26572.0** | – | −2.89 | −2.68 | −2.88 | 10800 | 3.78 |
| 27 | TIME | 27353.8 | 25638.0 | 26600.4 | 25631.0 | 25912.2 | **25012.0** | 26040.4 | 25811.0 | – | −4.80 | −2.11 | 0.49 | 10800 | 7.22 |
| 28 | TIME | 28035.2 | 26958.0 | 27353.0 | 26604.0 | 27388.4 | **26259.0** | 27628.8 | 26736.0 | – | −1.45 | 1.01 | 0.88 | 10800 | 17.58 |
| 29 | TIME | 29570.4 | 28834.0 | 30154.0 | 29508.0 | 29744.4 | 29020.0 | 30074.8 | **27938.0** | – | 1.71 | −0.26 | 1.11 | 10800 | 3.70 |
| 30 | TIME | 27928.8 | 27077.0 | 27862.8 | 26945.0 | 27674.8 | 27108.0 | 27852.8 | **26708.0** | – | −0.27 | −0.04 | 0.64 | 10800 | 7.59 |
| Avg | | | | | | | | | | –d | −1.18 | −0.67 | −0.62 | 10800 | 8.07 |

trade-off between these two aspects and summarize them in Table 10. The improvement is calculated by (72). The average improvement (%) for each set of instances, as well as the computational time of solving phase 1 and CPU time of solving both phases 1 and 2 (1+2), is presented.

$$Improvement\,(\%) = \frac{(\overline{TC}_{phase1+2} - \overline{TC}_{phase1})}{\overline{TC}_{phase1}} \times 100 \tag{72}$$

Solving the set partitioning formulation is remarkably important towards solution quality, because we notice that the improvement is increasing when we solve larger instances. Furthermore, phase 1 (ALNS) alone might not be able to find optimal solutions for Set 1. This shows us that some of the observed routes that are ignored in the ALNS search are indeed those good routes that form an optimal solution. However, solving the set partitioning formulation obviously requires another additional CPU time. Hence, if one only cares about the CPU time, then solving phase 1 of our matheuristic is good enough to provide a reasonably good solution. We also evaluate the importance of the set partitioning implicitly by comparing its performance to an ALNS given equal CPU time as the matheuristic in Section 5.3. It can be concluded that solving the set partitioning formulation leads to better solution quality when using equal CPU time.

### 5.5 Analysis of operators

This section analyzes the importance of operators used in the first phase of the proposed matheuristic. The frequencies of selecting operator $j$, $count_j$, and successful selection (e.g., better solutions), $success_j$, are recorded. For example, within an iteration, $R_2$ is selected to remove $\pi$ nodes from the solution, and afterwards $I_5$ is selected to reinsert $\pi$ nodes back into the solution. In this case, each $count_{R_2}$ and $count_{I_5}$ are increased by one. If the newly generated solution (we call it a neighborhood solution) improves either $Sol'$ or $Sol^*$, then $success_{R_2}$ and $success_{I_5}$ are increased by one. When the matheuristic is terminated, the success rate of each operator is calculated by (73).

$$\text{Operator } j \text{ success rate } (\%) = \frac{success_j}{count_j} \times 100 \quad \forall j \in R \cup I \tag{73}$$

Figure 6 plots the success rates, which are divided by an average per set and an average of all instances in all sets. It illustrates that the operator performance is similar for different sets. We conclude that $I_9$ is far less likely to successfully generate better solutions. Here,

**Table 10** Improvement made by solving the set partitioning formulation

| | Improvement (%) | CPU time phase 1 | CPU time phase 1+2 |
|---|---|---|---|
| Set 1 | −0.91 | 0.11 | 0.21 |
| Set 2 | −0.95 | 0.74 | 18.92 |
| Set 3 | −1.31 | 2.50 | 8.07 |
| Average | −1.06 | 1.11 | 9.07 |

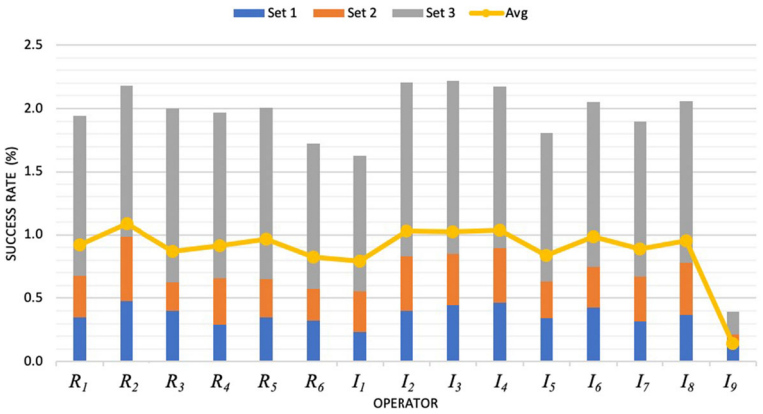


**Fig. 6** Operators' success rates

$R_6$, $I_1$, $I_5$, and $I_9$ have somewhat lower success rates compared to the other operators, while $R_2$, $R_5$, $I_2$, $I_3$, $I_4$, $I_6$, and $I_8$ have higher success rates. This denotes that a simpler algorithm might be designed by employing a subset of the current operators rather than utilizing all of them.

In order to verify the above idea, we introduce two variants of our matheuristic: variant (ii), where $R_6$, $I_1$, $I_5$, and $I_9$ are excluded; and variant (iii), where only $R_2$, $R_5$, $I_2$, $I_3$, $I_4$, $I_6$, and $I_8$ are considered. These two variants are compared against the original matheuristic that employs all 15 operators: variant (i). The difference of objective function values obtained is calculated by (74).

$$Gap(\%) = \frac{\overline{TC}_{variant(i)} - \overline{TC}_{variant(ii)}(or\,\overline{TC}_{variant(iii)})}{\overline{TC}_{variant(ii)}(or\,\overline{TC}_{variant(iii)})} \times 100 \tag{74}$$

Table 11 summarizes the results for all scenarios in terms of solution quality and CPU time. It is interesting to see that a lesser amount of operators used implies a worse performance of the matheuristic. It seems that less successful operators are still required in order to explore the entire solution space. Furthermore, we observe that the number of operators employed does not have a significant impact on the CPU time.

**Table 11** Comparison between employing all operators and removing some operators

| Instance | $\overline{TC}$ | | | Gap (%) | | Average CPU time(s) | | |
|---|---|---|---|---|---|---|---|---|
| | (i) | (ii) | (iii) | (ii) | (iii) | (i) | (ii) | (iii) |
| Set 1 | 9700.7 | 9716.6 | 9719.7 | −0.17 | −0.20 | 0.21 | 0.19 | 0.19 |
| Set 2 | 12372.7 | 12429.6 | 12485.6 | −0.46 | −0.91 | 18.92 | 17.95 | 15.56 |
| Set 3 | 27385.1 | 27464.5 | 27498.6 | −0.28 | −0.40 | 8.07 | 10.22 | 12.39 |
| Average | 16486.2 | 16536.9 | 16568.0 | −0.30 | −0.51 | 9.07 | 9.45 | 9.38 |

## 6 Conclusion and future work

This research studies the integration of the vehicle routing problem and cross-docking to handle reverse product flow, namely the vehicle routing problem with reverse cross-docking (VRP-RCD). Our study is motivated by the recent trend whereby optimizing reverse logistics is a source of profitability and competitiveness for various industries. The main objective is to minimize the overall cost in the distribution systems, such as the transportation costs and the fixed vehicle costs.

We propose herein a matheuristic algorithm based on an adaptive large neighborhood search (ALNS) algorithm and a set partitioning formulation. ALNS focuses on generating a list of feasible candidate routes. A set of 15 operators is employed for generating neighborhood solutions. A set partitioning formulation is then used to find the best combination of routes with respect to the VRP-RCD constraints, for example the limited number of vehicles and the time horizon.

The proposed matheuristic is tested on benchmark VRP-RCD instances and newly generated larger instances. The results show that the matheuristic is able to obtain all optimal solutions for small instances (15 nodes) within much shorter CPU time than CPLEX and outperforms CPLEX and three algorithms (ALNS, LNS-SA, and ALNS-SA) on larger instances with 40 and 65 nodes. Further analysis on the matheuristic properties such as the set partitioning formulation and the operators are also presented. Solving the set partitioning formulation leads to better solution quality, even though it takes more CPU time. In our case, it improves the performance by 1.06% on average. Finally, we illustrate that even operators that are less successful can still be necessary to obtain high quality results.

This research contributes to the literature by designing a matheuristic approach that effectively tackles the VRP-RCD in a four-level supply chain network. This actually provides managerial insights for managing returned products. Future research should focus on the obvious next step, which is the vehicle routing problem with forward and reverse cross-docking. Other considerations from practice could also include: split deliveries, a heterogeneous fleet, and multiple cross-docks. Developing and solving new and larger instances with up to hundreds of nodes can be explored as well, because real-sized problems faced in practice are probably larger than the instances considered in this paper.

## Appendix A: Supplier delivery process constraints

The constraints occurring in the supplier delivery process are formulated as follows.

$$L \sum_{v \in V} \sum_{i \in S \cup 0, i \neq j} x_{ij}^{\prime\prime\prime v} \geq \sum_{i \in O} r_{ij}^{\prime\prime} + \sum_{i \in C} r_{ij}^{\prime} - \sum_{v \in V} \sum_{i \in O} A_{ij}^{\prime\prime v} \quad \forall j \in S \tag{75}$$

$$\sum_{i \in O} r_{ij}^{\prime\prime} + \sum_{i \in C} r_{ij}^{\prime} - \sum_{v \in V} \sum_{i \in O} A_{ij}^{\prime\prime v} \geq \epsilon - L\left(1 - \sum_{v \in V} \sum_{i \in S \cup 0, i \neq j} x_{ij}^{\prime\prime\prime v}\right) \quad \forall j \in S \tag{76}$$

$$\sum_{i \in S} \sum_{j \in S, j \neq i} x_{ij}^{\prime\prime\prime v} \leq L \sum_{j \in S} x_{0j}^{\prime\prime\prime v} \quad \forall v \in V \tag{77}$$

$$\sum_{i \in S \cup 0, i \neq l} x_{il}^{\prime\prime\prime v} = \sum_{j \in S \cup 0, j \neq l} x_{lj}^{\prime\prime\prime v} \quad \forall l \in S, \forall v \in V \tag{78}$$

$$\sum_{v \in V} \sum_{i \in S \cup 0} x_{ij}^{\prime\prime\prime v} \leq 1 \quad \forall j \in S \tag{79}$$

$$\sum_{v \in V} A_j^{\prime\prime\prime v} \geq \left( \sum_{i \in O} r_{ij}^{\prime\prime} + \sum_{i \in C} r_{ij}^{\prime} - \sum_{v \in V} \sum_{i \in O} A_{ij}^{\prime\prime v} \right) - L\left(1 - \sum_{v \in V} \sum_{i \in S \cup 0} x_{ij}^{\prime\prime\prime v}\right) \quad \forall j \in S \tag{80}$$

$$\sum_{v \in V} A_j^{\prime\prime\prime v} \leq \left( \sum_{i \in O} r_{ij}^{\prime\prime} + \sum_{i \in C} r_{ij}^{\prime} - \sum_{v \in V} \sum_{i \in O} A_{ij}^{\prime\prime v} \right) + L\left(1 - \sum_{v \in V} \sum_{i \in S \cup 0} x_{ij}^{\prime\prime\prime v}\right) \quad \forall j \in S \tag{81}$$

$$L \sum_{i \in S \cup 0, i \neq j} x_{ij}^{\prime\prime\prime v} \geq A_j^{\prime\prime\prime v} \quad \forall j \in S, \forall v \in V \tag{82}$$

$$q_0^{\prime\prime\prime v} = \sum_{j \in S} A_j^{\prime\prime\prime v} \quad \forall v \in V \tag{83}$$

$$q_i^{\prime\prime\prime} \geq q_0^{\prime\prime\prime v} - A_i^{\prime\prime\prime v} - L(1 - x_{0i}^{\prime\prime\prime v}) \quad \forall i \in S, \forall v \in V \tag{84}$$

$$q_i^{\prime\prime\prime} \leq q_0^{\prime\prime\prime v} - A_i^{\prime\prime\prime v} + L(1 - x_{0i}^{\prime\prime\prime v}) \quad \forall i \in S, \forall v \in V \tag{85}$$

$$q_j^{\prime\prime\prime} \geq q_i^{\prime\prime\prime} - A_j^{\prime\prime\prime v} - L(1 - x_{ij}^{\prime\prime\prime v}) \quad \forall i, j \in S, \forall v \in V \tag{86}$$

$$q_j^{\prime\prime\prime} \leq q_i^{\prime\prime\prime} - A_j^{\prime\prime\prime v} + L(1 - x_{ij}^{\prime\prime\prime v}) \quad \forall i, j \in S, \forall v \in V \tag{87}$$

$$q_0^{\prime\prime\prime v} \leq q \quad \forall v \in V \tag{88}$$

$$u_j^{\prime\prime\prime} \geq u_i^{\prime\prime\prime} + 1 - |S|\left(1 - \sum_{v \in V} x_{ij}^{\prime\prime\prime v}\right) \quad \forall i, j \in S \tag{89}$$

Constraints (75) and (76) ensure that if either there exist outlets' returned products or there are some returned products from customers that are not sent to any outlet (including the defective products), then the supplier that supplies the product will be visited. Constraint (77) ensures that for every vehicle used in this process, it always starts its trip from the cross-dock. Constraint (78) ensures the outflow and inflow of a vehicle in each supplier node. Constraint (79) ensures that each supplier is visited at most once. The amount of products delivered to each supplier is calculated in constraints (80) and (81), while ensuring no split delivery occurs as in constraint (82). Constraints (83) to (87) track the total load inside a vehicle. Constraint (88) ensures that the total amount of products delivered to all suppliers in a vehicle does not exceed vehicle capacity. Constraint (89) is the sub-tour elimination constraint.

**Acknowledgements** This research is supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant. The work of Vincent F. Yu was partially supported by the Ministry of Science and Technology of Taiwan under grant MOST 108-2221-E-011-051-MY3 and the Center for Cyber-Physical System Innovation from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan.

## References

1. Farahani, R.Z., Rezapour, S., Drezner, T., Fallah, S.: Competitive supply chain network design: an overview of classifications, models, solution techniques and applications. Omega **45**, 92–118 (2014)
2. Rezaei, S., Kheirkhah, A.: Applying forward and reverse cross-docking in a multi-product integrated supply chain network. Prod. Eng. **11**(4–5), 494–509 (2017)
3. Belle, J.V., Valckenaers, P., Cattrysse, D.: Cross-docking: state of the art. Omega **40**(6), 827–846 (2012)
4. Lee, Y.H., Jung, J.W., Lee, K.M.: Vehicle routing scheduling for cross-docking in the supply chain. Comput. Ind. Eng. **51**(2), 247–256 (2006)
5. Liao, C.J., Lin, Y., Shih, S.C.: Vehicle routing with cross-docking in the supply chain. Expert Syst. Appl. **37**(10), 6868–6873 (2010)
6. Yu, V.F., Jewpanya, P., Redi, A.A.N.P.: Open vehicle routing problem with cross-docking. Comput. Ind. Eng. **94**, 6–17 (2016)
7. Gunawan, A., Widjaja, A.T., Vansteenwegen, P., Yu, V.F.: Adaptive large neighborhood search for vehicle routing problem with cross-docking. In: 2020 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2020)
8. Gunawan, A., Widjaja, A.T., Vansteenwegen, P., Yu, V.F.: A matheuristic algorithm for the vehicle routing problem with cross-docking. Appl. Soft Comput., vol. 103 (2021)
9. Gunawan, A., Widjaja, A.T., Vansteenwegen, P., Yu, V.F.: A matheuristic algorithm for solving the vehicle routing problem with cross-docking. In: Kotsireas, I.S., Pardalos, P.M. (eds.) Proceedings of the 2020 Learning and Intelligent Optimization Conference (LION) 2020, Lecture Notes in Computer Science, vol. 12096, pp. 9–15. Springer (2020)
10. Kaboudani, Y., Ghodsypour, S.H., Kia, H., Shahmardan, A.: Vehicle routing and scheduling in cross docks with forward and reverse logistics. Operational Research **20**(3), 1589–1622 (2020)
11. Rogers, D.S., Tibben-Lembke, R.S.: An examination of reverse logistics practices. J. Bus. Logist. **22**(2), 129–148 (2001)
12. Shen, B., Li, Q.: Impacts of returning unsold products in retail outsourcing fashion supply chain: a sustainability analysis. Sustainability **7**(2), 1172–1185 (2015)
13. Widjaja, A.T., Gunawan, A., Jodiawan, P., Yu, V.F.: Incorporating a reverse logistics scheme in a vehicle routing problem with cross-docking network: A modelling approach. In: Proceedings of the 2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA), pp. 854–858 (2020)
14. Ruiz-Benítez, R., Ketzenberg, M., van der Laan, E.A.: Managing consumer returns in high clockspeed industries. Omega **43**, 54–63 (2014)
15. Zuluaga, J.P.S., Thiell, M., Perales, R.C.: Reverse cross-docking. Omega **66**, 48–57 (2017)
16. Gunawan, A., Widjaja, A.T., Vansteenwegen, P., Yu, V.F.: Vehicle routing problem with reverse cross-docking: An adaptive large neighborhood search algorithm. In: Lalla-Ruiz, E., Mes, M., Voß, S. (eds.) Proceedings of the 11th International Conference on Computational Logistics (ICCL) 2020, Lecture Notes in Computer Science, vol. 12433, pp. 167–182. Springer (2020)
17. Gunawan, A., Widjaja, A.T., Gan, B., Yu, V.F., Jodiawan, P.: Vehicle routing problem for multi-product cross-docking. In: Proceedings of the 10th International Conference on Industrial Engineering and Operations Management (IEOM) 2020, pp. 66–77 (2020)
18. Wen, M., Larsen, J., Clausen, J., Cordeau, J.F., Laporte, G.: Vehicle routing with cross-docking. Journal of the Operational Research Society **60**(12), 1708–1718 (2009)
19. Tarantilis, C.D.: Adaptive multi-restart tabu search algorithm for the vehicle routing problem with cross-docking. Optim. Lett. **7**(7), 1583–1596 (2013)
20. Grangier, P., Gendreau, M., Lehuédé, F., Rousseau, L.-M.: A matheuristic based on large neighborhood search for the vehicle routing problem with cross-docking. Computers & Operations Research **84**, 116–126 (2017)
21. Morais, V.W.C., Mateus, G.R., Noronha, T.F.: Iterated local search heuristics for the vehicle routing problem with cross-docking. Expert Syst. Appl. **41**(16), 7495–7506 (2014)

22. Wang, J., Jagannathan, A.K.R., Zuo, X., Murray, C.C.: Two-layer simulated annealing and tabu search heuristics for a vehicle routing problem with cross docks and split deliveries. Computers & Industrial Engineering **112**, 84–98 (2017)
23. Dondo, R., Cerdá, J.: The heterogeneous vehicle routing and truck scheduling problem in a multi-door cross-dock system. Computers & Chemical Engineering **76**, 42–62 (2015)
24. Grangier, P., Gendreau, M., Lehuédé, F., Rousseau, L.-M.: The vehicle routing problem with cross-docking and resource constraints. J. Heuristics, pp. 1–31 (2019)
25. Baniamerian, A., Bashiri, M., Tavakkoli-Moghaddam, R.: Modified variable neighborhood search and genetic algorithm for profitable heterogeneous vehicle routing problem with cross-docking. Appl. Soft Comput. **75**, 441–460 (2019)