Toward a more flexible VRP with pickup and delivery allowing consolidations

Yannis Ancele[a], Minh Hoàng Hà[b], Charly Lersteau[a], Dante Ben Matellini[a], Trung Thanh Nguyen[a,*]

[a] *Liverpool Logistics, Offshore and Marine (LOOM) Research Institute*
Liverpool John Moores University, UK
[b] *ORLab, University of Engineering and Technology, Vietnam National University*

Abstract

One important requirement of modern supply chain management is the frequent exchange of containers via multiple cross-docks which requires spatial and time synchronisations between different types of vehicle. Moreover, as collaborations in logistics between several companies become popular, more flexible and extended models must be solved to consider the different needs of the companies. This is of high importance in a new logistics concept, the Physical Internet, which is expected to considerably improve the way logistics are handled in the current supply chain management.

To optimise the aforementioned requirements, a rich vehicle routing problem with pickup and delivery including numerous attributes is modelled and solved. A mathematical formulation is proposed and implemented in CPLEX to solve the problem. Given the complexity of the problem, solving large instances with exact methods is very time-consuming. Therefore, a multi-threaded meta-heuristic based on Simulated Annealing is developed. A set of new operators coupled with a restart strategy and memory are developed to help improve the performance.

Computational results on a generated data-set showed that the proposed meta-heuristic is superior to the CPLEX solver in terms of solvability and computational time. The proposed meta-heuristic was also compared with the best-known results by current state-of-the-art methods on a classical benchmark on pickup and delivery problems with time windows (with up to 200 customers). The experimental results showed that the proposed method was able to match the best-known results in many of these large scale instances.

Keywords: Vehicle Routing Problem, Pickup and Delivery, Cross-docks, Simulated Annealing, Mixed-Integer Linear Programming, Physical Internet

1. Introduction

As companies seek efficiency and sustainability, optimising the Supply Chain Management (SCM) becomes challenging. During the last decades, many researchers have been developing optimisation and approximation algorithms for vehicle routing problems. The Vehicle Routing Problem (VRP) is a combinatorial optimisation problem that was proposed in the late 1950s and it is still one of the most studied problems in the field of operations research. The great interest in the VRP is due to its practical importance, as well as the difficulty of solving it. The objective of the classical VRP is to deliver a set of customers with known demands on minimum-cost routes originating and terminating at the same depot. The VRP was introduced by Dantzig and Ramser in 1959 [1]. The

*Corresponding author
Email addresses: yannis.ancele@gmail.com (Yannis Ancele), T.T.Nguyen@ljmu.ac.uk (Trung Thanh Nguyen)

Pickup and Delivery Problem (PDP) is a generalisation of the VRP which is about finding optimal routes to satisfy transportation requests. Each request requires both pickup and delivery with precedence constraints. The VRP with cross-dock is also a variant of the classical VRP which contains spatial and load synchronisation constraints. These cross-dock facilities allow products to be transferred and processed. A cross-dock can be considered as a consolidation facility which has short-term storage. The principle of such a system is to unload and sort incoming containers, then to load the outgoing ones on vehicles. This approach differs from the direct-shipping of products in which intermediate trans-shipment points are not solicited. As studies are carried out, emerging methodologies such as cross-docking are being proven to be more suitable for a sustainable SCM.

The Physical Internet (PI-π) is another example of emerging methodologies. The idea behind this concept is to mimic the Digital Internet (DI) into a physical one [2]. The Physical Internet interconnects logistics systems and networks by using a set of world-standard modular containers, interfaces and protocols [3]. Basically, these PI-containers are routed through PI facilities called PI-hubs (which are cross-docks adapted to the PI). Readers can refer to this paper [4] to get more details about PI-hubs and their similarities with cross-docks. In this paper, PI-hubs facilities have been modelled with cross-docks characteristics found in the VRP literature. The PI has already been applied to a few real cases [4]. Also, technical papers which specify various PI-object aspects can be found in the literature [5]. In the end, it is said that this concept could also be applied to human mobility [6]. Just like the PI, City Logistics (CL) is another research area which focuses on the transportation improvement. CL and PI have similarities which allow them to work alongside. One of the first works about their compatibility was done in [6]. In this paper, it is explained that the CL provides the final building blocks for the PI to be complete with regards to transportation in cities.

All those areas have been mostly studied independently, however, technologies like the Internet of Things emphasis the need to interconnect systems in order to work together. Few papers like [7] attempt to unify and solve numerous problems with a single general solution. This kind of unification eases the choice of a solution algorithm from the literature given a particular problem. As problems with cross-docking or even like the PI are given more interest, companies will need to face optimisation problems like the VRP. Therefore, this research aims to solve a rich VRP with pickup and delivery including several attributes at once to fill the gap in the literature. To tackle this problem, a linear programming model is first introduced and solved with CPLEX for comparison purposes on small instances. Then a multi-Threaded Simulated Annealing with Memory (TSAM) algorithm is proposed to handle real-world size instances. Commonly, the VRP assumes a homogeneous fleet of vehicles to serve a set of customers requiring delivery and pickup services with time windows. Sometimes different attributes are considered alongside another one or two. However, in many practical situations, companies need to consider numerous attributes of the VRP at once. Therefore, this paper extends the classical VRP and makes two major contributions. (1) A new model gathering several VRP attributes which are of most important as logistics evolve. The model handles multiple cross-docks in a new and more flexible way while the other attributes have been gathered and combined from the literature. (2) An algorithm with new operators which include generic features as well as problem specificities.

Simulated Annealing (SA) algorithms have been successfully used to solve combinatorial problems such as the VRP. Moreover, SA tested on our selected reference benchmark in [8] yielded good performance for instances containing 100 and 200 customers. Consequently, proposing a new SA to solve this problem and comparing it with

CPLEX and best-known results on large instances from [9] is the adopted methodology in this paper. Researchers, however, can easily integrate any other global search algorithms of their choice with the features described in this paper to handle the proposed model or a variation of it.

The remainder of this paper is organised as follows. Section 2 gives a literature review of different aspects of the problem tackled while section 3 presents a mathematical formulation. Section 4 explains the proposed algorithm to solve the VRP. Section 5 shows a comparative analysis of the meta-heuristic and CPLEX results. The efficiency of the algorithm on a benchmark is also discussed. Finally, section 6 concludes the paper by explaining the contribution and presenting future research directions.

2. Literature review

In terms of methodologies used in this paper, a fundamental building block to optimise transportation logistics, such as PI, is the PDP. Therefore, the most important and relevant attributes of this problem are described below.

In operational research, it is a central problem for logistics as it gives a solution to the transportation process. To help grasp the relevant models and methodologies, numerous reviews can be found such as the work of Silva and Zuluaga [10]. They not only presented a classification of the different attributes of the problem but also provided insights on modelling and solution techniques. When a combination of multiple constraints is being solved, the problem can be categorised as a rich VRP. Caceres-Cruz et al. [11] gave a survey on the rich VRP by summarising problem combinations, constraint definitions and different approaches. Lahyani et al. [12] also worked on the rich VRP by providing a comprehensive and relevant taxonomy alongside its definition. As real-live applications require an increasing number of attributes to be considered simultaneously, researchers tend to design general-purpose solvers. Vidal et al. [7] developed a unified solution framework to tackle a multi-attribute VRP. They demonstrated that such a general method can be efficient for this class of problem. Another attempt to solve a rich VRP with a unified heuristic can be found in [13]. A pickup and delivery model was provided with a robust and self-calibrating framework to solve it.

Different classes of problems arise for the PDP, two of them are simultaneous and mixed PDP. In the simultaneous PDP, pickups and deliveries can be made at the same time. Originally, this variant considers a homogeneous fleet of vehicles to satisfy the customer demands. However, nowadays, there are different types of vehicles available to be used. Wang et al. [14] addressed the VRP in which customers require simultaneous pickup and delivery of goods during specific time windows. They used a parallel Simulated Annealing algorithm to efficiently solve this variant. This problem was also solved in [15] with an adaptive memory framework that generates high-quality solutions by collecting and combining promising solution features. In the mixed PDP, pickups and deliveries can occur in any order on a vehicle route. Wassan and Nagy [16] presented a taxonomy of different problem versions including mixed pickup and delivery. They focused on the back-hauling aspect of the PDP while providing a review of solution methodologies and highlighting issues in the literature. Rich solution frameworks like in [7] also consider this attribute. However, this variant seems to suffer from a lack of published papers even though it has immense practical applicability within logistics.

A commonly used attribute for the VRP is time windows, as shown in [17], it requires that the delivery is made within a specific time window given by the customers. Two categories can be defined: soft time windows which can

be violated while inducing a penalty cost and hard time windows which cannot. Cordeau et al. [18] wrote a survey in which they presented approximation methods and optimal approaches to tackle this variant.

Compatibility or site-dependency constraints can also be added on a VRP model. As shown in [19], this refers to the situation in which a customer must be served from a specific depot, by a specific vehicle or a specific driver. For instance, goods demanded by a customer might require vehicles with special equipment for loading and unloading. Access restrictions regarding the vehicle type can apply in a given area or city. The work of Desrochers et al. [20] also covered this attribute.

Companies that have several depots can solve the multi-depot VRP to satisfy all its customers. In this variant, vehicles can be located at different depots while customers can be assigned to different depots. Montoya-Torres [21] reviewed the state-of-the-art on this variant by considering relevant papers since 1988. They studied several variants and provided a classification for solution approaches dealing with single or multiple objectives. Nagy and Salhi [22] also studied this variant while avoiding the common assumption that pickups and deliveries must be completed in two different stages.

It has been demonstrated that cross-docking strategy plays an important role in goods distribution. In their PDP and VRP with cross-dock models, Nikolopoulou et al. [23] gave a comparison of direct-shipping and cross-docking strategies. They developed a local-search optimisation framework and tested on existing and new benchmark data-sets. It was concluded that depending on the environment and constraints, a cross-docking strategy can outperform the direct-shipping. However, in some cases direct-shipping can still be relevant, this could mean that hybrid methods are necessary. As cross-docking strategies become common, scheduling and door assignment are being solved simultaneously with cross-dock constraints. Enderer et al. [24] proposed two formulations of the problem and compared computational results. They showed that integrated solution approaches can lead to significant savings in costs. Cross-docking might create situations in which vehicles have to wait for empty doors or even products to arrive. Dondo and Cerdá [25] considered this constraint by introducing a mixed-integer linear programming formulation. They also used an integrated solution approach in which the dock door assignment and the truck scheduling at the cross-dock are simultaneously decided. Miao et al. [26] studied the multiple cross-docks where penalty values are added when the time windows are not met. They proved the NP-hard difficulty of the multiple cross-docks problems and therefore designed a hybrid method to solve the problem compared with CPLEX. Although the interest for cross-docking is increasing, only very few papers considered multiple cross-docks constraints. Maknoon and Laporte's paper [27] is one of them, they proposed a mathematical formulation of the problem in which requests have to pass through at least one cross-dock. The efficiency of their proposed adaptive large neighbourhood search heuristic was demonstrated with a comparison with CPLEX. However, their model using heterogeneous vehicles which must start and end their routes at the same assigned cross-dock is not entirely compatible with PI constraints. Their transportation process is divided into two separate shifts (pickup and delivery) which cannot model a mixed pickup and delivery.

In some situations, once the last customer on the route has been visited, the driver does not have to return to the depot. The driver could terminate his route at another depot or even at home. This problem was introduced by Schrage in 1981 and is called open VRP. A few papers have been devoted to this problem. Alinaghian et al. [28] dealt not only with cross-docking but also with open VRP. On top of the capacity of vehicles that is not

completely used, some companies use rental vehicles due to the high cost of purchasing vehicles with high capacity. Therefore, the authors proposed a cross-docking and open-close VRP problem solved by a simulated annealing algorithm. Russell et al. [29] showed relevant applications of the open VRP like newspaper logistics. It was explained that the independent outsourcing characteristic of these processes as independent contractors requires this type of modelling. Yu et al. [30] studied and solved a capacitated homogeneous cross-docking and open VRP with a simulated annealing algorithm. They showed that their proposed method can outperform CPLEX. Atefi et al. [31] also solved this variant while considering a decoupling points strategy which generalises the open VRP as multiple trucks delivery. The first truck performs part of the deliveries, then drops off the load while the second one and others continue from that point onwards. Their decoupling points strategy is similar to our model handling consolidations in a PI environment. On top of open routes, our model also allows vehicles to start and end their routes to different depots.

Although the above VRP attributes have been studied with different combinations, to the best of our knowledge, none has modelled this problem with all the above constraints. Only a few papers proposed to solve the VRP with a single cross-dock and very few solved this problem with multiple cross-docking facilities. Papers like [27], [26], [32], [33] and [34] dealt with multiple cross-docks but modelled the problem differently to this paper. When considering multiple cross-docks, pickups and deliveries were often handled in a less flexible way. They were satisfied in separate stages/vehicles or could not be consolidated within several successive cross-docks before delivery. However, there are real-world problems with all the aforementioned constraints.

Our rich version of the model can arise in many practical applications and therefore contributes toward a more efficient SCM. The PI is one of such examples which uses several types of vehicle and allows load exchanges at multiple facilities to satisfy customers within time windows. Collaboration using external transportation means is another feature of the PI which must be considered. As a vehicle can belong to an external company or even a private owner, it is available for a given period and can have an open route or finish its route at a different depot. One issue cross-docking is tackling is the capacity of vehicles not being entirely used during the deliveries. Having mixed pickups and deliveries is another characteristic of our model that can improve the overall efficiency by introducing some degree of flexibility. The vehicle fulfilment could also be improved by using consolidations during deliveries. This research direction is another motivation for our model to be created and considered in a PI environment. Thus, developing models and algorithms that can deal with all these constraints is of high practical value. This paper fills this gap by solving this challenging PDP with several attributes for real-life size cases.

3. Problem description and formulation

3.1. Context

In this work, we aim at scheduling routes of a set of vehicles to satisfy all the customer requests. The following list of realistic properties are handled: capacitated, heterogeneous, mixed pickup and delivery, multiple depots, open route, different start/end depots, multiple cross-docks, customer time windows, site-dependent. Although this work is not concerned about the scheduling at cross-dock doors, it differs from most of the work studying cross-docking as it gathers numerous VRP attributes and models them with new constraints. As stated by Sarraj et al.

[35], a perceived possible drawback of the PI comes from the shift from direct-shipping to distributed transport involving containers being consolidated in several cross-docks with a possible costs increase. However, they showed that PI scenarios can result in significantly lower overall costs hence making it a profitable alternative. It is also demonstrated that by using consolidations, PI can induce a significant gain regarding transportation fulfilment rate. Many other researchers mentioned the benefit of using cross-docking. Yu et al. [30] studied cross-docking while considering the cost of hiring vehicles. In [36], they discussed the importance of cross-docking for efficient distribution networks as the costs of holding and handling can be lowered in warehouses. Similarly, Chen et al. [37] stated that cross-docking can be an extremely efficient strategy when the inventory costs are high. While showing that cross-docking can increase the cost of vehicle used, Ahmadizar et al. [38] also emphasised the efficiency of consolidation to minimise the total overall costs. Other research proved the reduction of costs due to cross-docking strategies in the SCM [39]. Real-world cases also support those findings [33]. In some scenarios, a direct-shipping strategy can still incur lower costs [23]. However, when P&D pairs are remotely positioned and clustered, cross-docking can reduce the cost compared to the direct-shipping. This is also the case when facing a densely connected network with a many-to-many relation between suppliers and customers. Hence the need for a flexible model capable of handling both strategies which could resemble a many-to-many problem because of standardised PI-containers and consolidation. Several simulations already demonstrated the benefit of the PI when simultaneously considering different cost related to vehicles, truck fulfilment, distance travelled, handling, storage etc. Moreover, as the PI is about collaboration by sharing resources, even if the cost of a company's fleet could increase, it has been shown that the global cost for all the companies collaborating would certainly decrease as they would share the fleet [5, 40]. As a consequence, our model designs a set of routes for vehicles while only considering the cost related to driving time.

In our VRP with cross-docks, a logistics organisation operates with multiple cross-dock facilities O to satisfy a set of customers C calling for transportation requests R. The problem is defined on a graph including a set of nodes $N = C \cup O$ in which for each pair of nodes i and j, there exists an arc (i, j) of driving time d_{ij}. Each customer requests is characterised by a container, an origin and a destination given by two customers $c_1, c_2 \in C$ where the first denotes the pickup point, and the second one denotes the delivery point. A customer request is satisfied when a vehicle picks up the container at the origin and delivers it at the destination. Cross-docks and customers require a service time s_i to handle requests. All requests are associated with their demands/loads $q_r > 0$ and must be satisfied between a time window $[A_i, B_i]$. For a given request r, its pickup and delivery locations are represented by the set $\{H_r^p, H_r^d\}$. Each request has the flexibility to pass through one or multiple cross-docks to be consolidated with others but is not forced to visit any cross-dock if a direct shipping strategy is more efficient. Moreover, requests are not necessarily expected to arrive at the cross-dock simultaneously if an exchange is happening. The organisation uses a set of vehicles K in which each vehicle $k \in K$ has a capacity Q_k, a type w_k and a set of assigned start and end cross-docks $O^k = \{o_s^k, o_e^k\}$. A single route can contain pickup as well as delivery locations but cannot include the same cross-dock more than once, except for the cross-dock depot. Each vehicle k is available within a time window $[E^k, F^k]$ and can only visit nodes for which its type is allowed with $z_i^{w_k}$ being equal to 1. Finally, to handle open routes, a dummy node o_d representing a cross-dock depot is introduced in the graph. It is assumed that each arc connecting the dummy cross-dock to another node of the graph has zero driving time. As a result, any vehicle k with an open route can be addressed by the model with $o_s^k = o_e^k = o_d$.

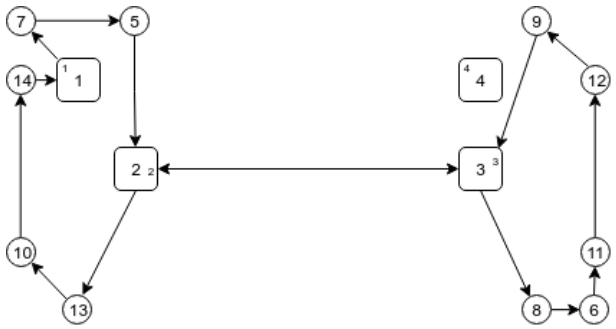


Figure 1: VRP solution with load exchanges

Figure 1 shows an example of a VRP solution which includes 4 cross-docks (squares), 10 customer locations (circles) and 4 vehicles represented with 4 small numbers in the cross-docks. Five requests call for transportation of containers from nodes 5, 7, 9, 11 and 13 to nodes 6, 8, 10, 12 and 14 respectively. In this scenario, pickup and delivery nodes are located in different cities and therefore must be done by different vehicles. Moreover, some constraints force vehicles to return to the depot within time windows. As a result, vehicles are required to transit via the cross-dock facilities to enable consolidation to share the transport work. Vehicle 1 first picks up the containers from nodes 7 and 5 to drop them to cross-dock 2. Then vehicle 2 carries these containers to cross-dock 3 where vehicle 3 picks them to deliver customers 8, 6. Customers 11 and 12 are also satisfied by vehicle 3 during its journey in which the container from customer 9 is brought to the cross-dock 3. After a wait, vehicle 2 can come back to cross-dock 2 with the container to be delivered by vehicle 1 which was waiting before visiting customers 13, 10 and 14. The model gives vehicles the flexibility to mix pickups and deliveries through several cross-docks as for customers 5, 7, 6 and 8 or to directly deliver containers as for customers 13, 14, 11 and 12. Cross-docks can be left unused if necessary as for cross-dock 4 and an alternative scenario could allow vehicle 1 to return to another depot or none after visiting customer 14.

3.2. Mixed-integer linear programming formulation

The decision variable y for request transportation and the Big-M formulation/method are inspired by [27] as this work also considers multiple cross-docks. With M as a large constant slightly greater than the highest value of variables B_i, the model is defined as follows:

$$\textbf{Minimise} \sum_{k \in K} \sum_{i \in N} \sum_{j \in N} x_{ij}^k * d_{ij} \tag{1}$$

subject to:

$$u_j^k \leq B_j \; \forall k \in K, \; \forall j \in C \tag{2}$$

$$v_i^k + M \left(1 - \sum_{j \in N \setminus i} x_{ij}^k \right) \geq A_i + s_i \; \forall k \in K, \; \forall i \in C, \tag{3}$$

$$v_o^k + M\left(1 - \sum_{j \in N \setminus o} x_{oj}^k\right) \geq E^k \; \forall k \in K, \mid o = o_s^k \tag{4}$$

$$u_o^k - M\left(1 - \sum_{j \in N \setminus o} x_{jo}^k\right) \leq F^k \; \forall k \in K, \mid o = o_e^k \tag{5}$$

$$\sum_{k \in K} \sum_{j \in N \setminus i} x_{ij}^k = 1 \; \forall i \in C \tag{6}$$

$$\sum_{j \in N \setminus i} x_{ij}^k \leq z_i^{w_k} \; \forall k \in K, \; \forall i \in N \tag{7}$$

$$\sum_{i \in N} x_{ij}^k = \sum_{i \in N} x_{ji}^k \; \forall k \in K, \; \forall j \in N \setminus O^k \tag{8}$$

$$\sum_{i \in N} x_{ij}^k \leq 1 \; \forall k \in K, \; \forall j \in O \tag{9}$$

$$v_i^k + d_{ij} \leq u_j^k + M\left(1 - x_{ij}^k\right) \; \forall k \in K, \; \forall i, j \in N \tag{10}$$

$$u_j^k + s_j * \sum_{i \in N \setminus j} x_{ij}^k \leq v_j^k \; \forall k \in K, \; \forall j \in N \setminus O^k \tag{11}$$

$$\sum_{r \in R} q_r y_{rij}^k \leq Q_k \; \forall i, j \in N, \; \forall k \in K \tag{12}$$

$$x_{ij}^k \geq y_{rij}^k \; \forall k \in K, \; \forall r \in R, \; \forall i, j \in N \tag{13}$$

$$\sum_{k \in K} \sum_{i \in N} y_{rio}^k \leq 1 \; \forall r \in R, \; \forall o \in O \tag{14}$$

$$\begin{cases} \textbf{if } o_s^k = o_e^k & \\ \sum_{i \in N} x_{ai}^k + \sum_{i \in N} x_{ib}^k = 2 \mid a = o_s^k, b = o_e^k & \\ \textbf{otherwise} & \\ \sum_{i \in N} x_{ai}^k = 1 \mid a = o_s^k & \forall k \in K \\ \sum_{i \in N} x_{ia}^k - x_{aa}^k = 0 \mid a = o_s^k & \\ \sum_{i \in N} x_{bi}^k = 0 \mid b = o_e^k & \\ \sum_{i \in N} x_{ib}^k + x_{aa}^k = 1 \mid a = o_s^k, \; b = o_e^k & \end{cases} \tag{15}$$

$$\sum_{k\in K}\sum_{j\in N} y^k_{raj} = 1 \;\forall r \in R \mid a = H^p_r \tag{16}$$

$$\sum_{k\in K}\sum_{j\in N} y^k_{rja} = 0 \;\forall r \in R \mid a = H^p_r \tag{17}$$

$$\sum_{k\in K}\sum_{j\in N} y^k_{rja} = 1 \;\forall r \in R \mid a = H^d_r \tag{18}$$

$$\sum_{k\in K}\sum_{j\in N} y^k_{raj} = 0 \;\forall r \in R \mid a = H^d_r \tag{19}$$

$$v^k_a \leq u^k_b \;\forall k \in K,\; \forall r \in R \mid a = H^p_r,\; b = H^d_r \tag{20}$$

$$\sum_{i\in N} y^k_{ria} - \sum_{j\in N} y^k_{raj} = 0 \;\forall r \in R,\; \forall k \in K,\; \forall a \in C \backslash H_r \tag{21}$$

$$\sum_{k\in K}\sum_{i\in N} y^k_{rio} = \sum_{k\in K}\sum_{i\in N} y^k_{roi} \;\forall r \in R,\; \forall o \in O \backslash H_r \tag{22}$$

$$u^{k'}_o - M\left(1 - \sum_{i\in N} y^{k'}_{rio}\right) \leq v^k_o + M\left(1 - \sum_{i\in N} y^k_{roi}\right) \forall r \in R,\; \forall o \in O,\; \forall k, k' \in K \tag{23}$$

$$x^k_{ij} \in \{0,1\} \;\forall i, j \in N,\; \forall k \in K \tag{24}$$

$$y^k_{rij} \in \{0,1\} \;\forall r \in R,\; \forall i, j \in N,\; \forall k \in K \tag{25}$$

$$u^k_i \in \mathbb{R} \;\forall i \in N,\; \forall k \in K \tag{26}$$

$$v^k_i \in \mathbb{R} \;\forall i \in N,\; \forall k \in K \tag{27}$$

Eq. (1) is the objective considered in this paper, it is the minimisation of the total driving time of all the vehicles (without waiting times). The objective is subject to constraints 2 - 27. Constraints (2) and (3) enforce that each customer j is available for pickup or delivery between times A_j and B_j. Similarly, constraints (4) and (5) enforce that each vehicle is only available between times E^k and F^k. Constraint (6) imposes that each customer location is served by exactly one vehicle. Constraint (7) ensures that pickups and deliveries are made by allowed vehicles. Constraint (8) means that each vehicle k can only start and end its route at its assigned cross-docks o^k_s and o^k_e, respectively. Constraint (9) forbids vehicles to visit the same cross-dock more than once. Constraints (10) computes the arrival and leaving times at node j which in turn depends on the arrival and leaving times of node

i and the driving time between the two nodes. Given an arrival time, the service time at node j is considered by constraint (11) to calculate the leaving time of a vehicle. Constraint (12) tracks the vehicle load that must respect its capacity. For each request, constraint (13) links its transfer decision to a vehicle route. Constraint (14) prevents requests from visiting a same cross-dock more than once. Constraint (15) forces each vehicle to depart and arrive at its assigned cross-dock depots, but it must not visit the depots more than once ($x^k_{aa} = 1$ if vehicle k is not used). Constraints (16) and (19) allow request pickup H^p_r or delivery H^d_r to be handled by a vehicle. The precedence constraints of pickups and deliveries are checked by constraint (20) as they must be consistent. Constraint (21) forbids a vehicle to drop any request load before reaching the delivery location. Constraint (22) controls the flow of requests entering a cross-dock o which has to leave this cross-dock. Constraint (23) synchronises each vehicle k leaving a cross-dock with another vehicle v carrying its loads to be consolidated. Constraints (24)-(27) define the domain of the decision variables as follows. Variable x^k_{ij} in (24) is the route decision, it is equal to 1 if and only if vehicle k travels from node i to node j, otherwise, x^k_{ij} equals 0. Variable y^k_{rij} in (25) is the transportation decision, it is equal to 1 if customer request r is transported using vehicle k on its route from node i to node j, otherwise, y^k_{rij} equals 0. The variables u^k_i in (26) and v^k_i in (27) represent respectively the arrival and leaving time of vehicle k at node i.

4. Meta-heuristic

As shown in Section 5, the mathematical model in Section 3 cannot be solved exactly for large instances within a reasonable time. Therefore a meta-heuristic approach is considered - a multi-Threaded Simulated Annealing with Memory (TSAM).

4.1. Architecture

The proposed algorithm has been inspired by [8] and extended to tackle the new model. The differences between [8] and TSAM are as follows. First, functions `overall`() 1, `restart`() 2, `neighbour_search`() 5, and `simulated_annealing`() 7 were modified and new functions were introduced. Second, the algorithm was parallelised to better cope with different instance characteristics simultaneously. The parallelisation allows the algorithm to apply traditional operators on a solution while trying to add consolidations without slowing down the overall search. The tabu list was removed and a memory mechanism was added for the threads to communicate and exchange their last created solutions. Third, operator `PD_rearrange`() was removed and new ones were added to handle the model constraints. For example, operator `swap`() 10 is introduced to better handle the "different start/end depots" attribute of the model while operator `consolidation`() 12 handles attribute "multiple cross-docks".

Figure 2 shows the architecture of the algorithm. Three different threads are launched on Algorithm `overall`() 1. Basically, Algorithm `overall`() 1 handles the current solution which will be altered by the other functions. Algorithm `restart`() 2 is used to explore several times a different neighbourhood of the current solution. Compared to Algorithm `random`() 6, Algorithm `neighbour_search`() only returns a solution that is better than the current one. However, they both use a random operator from a list *op_list*. By calling Algorithm `random`() 6, Algorithm `simulated_annealing`() 7 is used to allow the exploration of worse neighbourhoods in order to escape local minima.

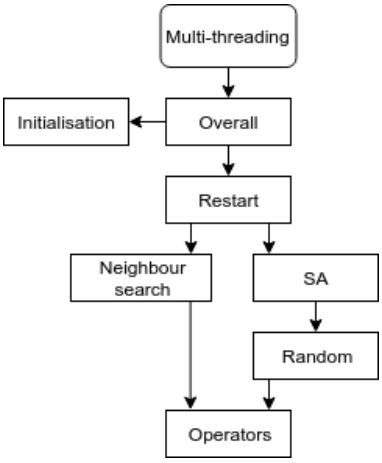


Figure 2: TSAM function architecture

The algorithm uses hierarchical clustering to extract clusters from the instance at hand. The heights of the clusters are used to identify the group of requests that must be consolidated together. If two requests share the same pickup and delivery node clusters (two distinct clusters), then they belong to the same group. Otherwise, they belong to separate groups. The group list *req_groups* is then used in function `PD_consolidate`() to ensure that the operators modify the requests (from the same group) in the same way.

Each solution and its vehicle routes have a flexible size which depends on the number of visited customer locations/nodes. In Figure 3, the vehicle tours/routes are delimited by the departure and arrival cross-docks/depots which are the numbers without any subscript. The ones with subscripts can be the customer or cross-dock nodes. These subscripts are the links between the nodes which represent the request travel paths. In Figure 3, such links are shown with the subscript numbers. Each node has a list of request path IDs which links them to other nodes. Such links are necessary to specify that a certain node must always be in the same vehicle route as another one. Moreover, a link is used as a position constraint. Therefore, a link contains an ID, is associated with two nodes and determines the positioning constraints of these two nodes. In Figure 3, node 1 and node 2 share a link of ID 1. The presence of this link, plus the respective positions of these nodes in the solution representation mean that node 1 must be in the same vehicle route as node 2, which must be positioned somewhere after node 1. This is because the vehicle must pick up the request at node 1 before delivering it to node 2.

7	1_1	2_1	5_3	3_2	4_2	6_3	7	9	9	10	10	8	8
7	5_3	6_3	7	9	3_2	1_1	4_2	2_1	9	10	10	8	8

Figure 3: Solution representation

The TSAM algorithm uses several parameters as follows: $PSMS$ defines the size of the memory for the previous solutions. $RPLI$ defines the number of iterations for the function 12 to change the request path length. $RPLR$ defines the probability for the function 12 to choose between operators `PD_stretch`() 3 and `PD_shrink`() 4. MIT defines the number of iterations before using previous solution in function 1. RIT defines the maximum number of restarts for function 2. $T0$ defines the initial temperature for Algorithm 7. δ defines the size of the temperature step for Algorithm 7.

Here we describe some sub-functions used by the algorithm. Function `cost`() computes the objective value of a solution as defined in Section 3. Function `random_double`() returns a random double value between 0 and 1. Function `random`() picks a random item from the given list. Function `shuffle`() randomly permutes the given list. Function `is_delivery`() returns *true* if the given node is a delivery node. Function `is_hub`() returns *true* if the given node is a cross-dock. Function `remove`() removes the given request from the given node. If the node has no request left, it will be removed from the given solution route. Function `insert`() inserts the given request path ID in the given node if it is not present. Then insert the node in the specified solution route. However, if the node already exists in the route, the function only adds the request path ID to the node. Function `get_best_indices`() returns all the possible indices from the solution route at which the given node can be inserted. The indices are sorted from best to worst giving the resulting route distance. Function `get_best_solution`() returns the best solution found from all the threads. Function `random_previous_solution`() returns randomly a previously found solution. Function `is_valid`() checks if the given solution is valid according to all the attributes defined in Section 3. Function `PD_arrange`() creates a solution where all the P&D pairs are positioned in vehicles of which the start depots are located in the same clusters.

4.2. Algorithm functions

This section describes all the functions of the algorithm and gives some pseudo-codes. Readers can refer to the appendix to get more details for the rest of the pseudo-codes.

Algorithm 1: Overall algorithm

```
1  //Input: problem instance;
2  //Output: the best solution S_b found;
3  S_b ← init_solution();
4  no_progress ← 0;
5  while Termination criterion not reached do
6      S ← get_best_solution() //from shared memory;
7      if no_progress % MIT = 0 then
8          S ← random_previous_solution() //from shared memory;
9      S ← restart(S);
10     if cost(S) < cost(S_b) then
11         S_b ← S;
12         no_progress ← 0;
13     else
14         no_progress ← no_progress + 1;
15 return S_b;
```

Algorithm 1 includes the main steps of the meta-heuristic. It is launched in parallel by three different threads which contain a different operator list *op_list*. Thread 1 has an operator list of `PD_interchange`() and `PD_move`(). Thread 2 has an operator list of `PD_consolidate`(). Thread 3 has an operator list of `PD_swap`() and `PD_exchange`(). Each thread memorises all the solutions found so that they can be re-used if there is no improvement for a long time. This memory is shared between all the threads so that a solution can be modified by all the operators. If the memory size is greater than $PSMS$, a random solution is then removed.

In Algorithm 1, function `restart`() in step 9 is iteratively launched with the best solution or a random previous solution until the termination criterion is reached. While step 6 is used to retrieve the best solution found among all the threads, step 8 get a random solution found by all the threads. Algorithm `init_solution`() in step 3 is used to generate an initial solution based on the insertion heuristic of Solomon. The initialisation function does not use the consolidation operators `PD_stretch`() and `PD_shrink`(). Only customer nodes are handled at this stage as solutions involving load exchange with hubs cannot be found at this stage. As a consequence, the feasible region of instances must include at least one solution without consolidation. To start, one first pair of P&D customers is inserted, then the insertion of each unrouted node is evaluated and compared with the other possible insertions. The one that minimises the additional distance (induced by the insertion) is selected to be included in the partially created route. The function continues inserting P&D pairs in the current route until a constraint is violated, in that case, the insertion is tried in the next route. As P&D nodes must be kept together, in case a delivery node could not be inserted, the function first removes the pickup node and then tries another solution route with both nodes.

Algorithm 2: Restart function

```
1  //Input: a current solution S_c;
2  //Output: the best solution S_b found;
3  S ← neighbor_search(S_c);
4  S_b ← S;
5  while no_progress < RIT do
6      S ← simulated_annealing(S);
7      S ← reorder_routes(S) //re-order routes modified by PD_exchange() and PD_move();
8      S ← neighbor_search(S);
9      if cost(S) < cost(S_b) then
10         S_b ← S;
11         no_progress ← 0;
12     else
13         no_progress ← no_progress + 1;
14 return S_b;
```

Algorithm 2 is used to explore the neighbourhood of a solution several times. Algorithm `simulated_annealing`() 7 in step 6, is used to escape local minima by potentially accepting worse solutions. At each iteration, the temperature is updated to give a probability to accept a solution created by Algorithm `random_solution`() 6. This algorithm allows the global search to make random moves by returning a random solution in the neighbourhood of a given solution. At each iteration, a random operator in *op_list* is applied to the given solution until any solution is found or until the iteration count is greater than RIT. To better explore the neighbour solutions, the proposed approach includes Algorithm `neighbor_search`() 5 which returns a better solution in step 8. At each iteration, a random operator in *op_list* is applied to the current solution until no improvement is made. Function `reorder_routes`() in step 7 is used to re-order the routes modified by operators `PD_exchange`() and `PD_move`(). Each pair of nodes is re-inserted into its route using Solomon's insertion procedure.

Algorithm 3: PD Stretch operator

```
//Input: a current solution S_c, a request r, a set K of vehicles;
//Output: a solution S found;
valid ← false;
while not valid do
    S ← S_c;
    k ← random(K) //get a random vehicle route where r is present;
    v ← random(r, K/k) //v must be different from k and must not contain r;
    pickup ← get_pickup(r, S^k) //get pickup node of request r from route k of solution S;
    delivery ← get_delivery(r, S^k);
    hub_1 ← get_hub(S^k, S^v) //random cross-dock which is not a depot in the given routes;
    hub_2 ← clone(hub_1);
    remove(delivery, r, S^k);
    insert(hub_1, r, S^k);
    insert(hub_2, r, S^v);
    insert(delivery, r, S^v);
    valid ← is_valid(S);
return S;
```

4.3. Algorithm operators

This section describes all the operators implemented in the algorithm and provides pseudo-codes. As before, readers can refer to the appendix to get more details for the rest of the pseudo-codes. Each operator iteratively performs a move and returns a new solution if it is valid. Otherwise, if the iteration count is greater than the number of customer $|N|$, the function stops and returns no solution. Algorithm 8 alters a node position in a vehicle route by removing and inserting it somewhere else in the same vehicle route. Algorithm 9 moves a pair of P&D nodes from a vehicle route to another one. It tries to select vehicles that are already used. Algorithm 10 replaces an entire vehicle route by another one. All the nodes from two routes are exchanged while checking if the new routes do not contain the depots as intermediate cross-docks. This can be useful when the vehicles have different depots. Algorithm 11 selects four P&D nodes from two different vehicle routes and exchanges them. Algorithm 12 changes a request path length by adding or removing an intermediate cross-dock. At each iteration, a request group is randomly selected. Function `PD_stretch()` or `PD_shrink()` is randomly selected (given the probability $RPLR$) to be applied to the request path ID in the group. If the vehicle or the cross-dock count is equal to 1, there are not enough resources to have load exchange, therefore the operator is not used.

7	3_2	4_2	5_3	6_3	7	9	9	10	10	8	1_1	2_1	8
7	3_2	5_3	10_2	6_3	7	9	9	10_2	4_2	10	8	1_1	2_1	8

Figure 4: Stretch operator

7	3_2	5_3	10_2	6_3	7	9_1	2_1	9	10_2	4_2	10	8	1_1	9_1	8
7	3_2	5_3	4_2	6_3	7	9_1	2_1	9	10	10	8	1_1	9_1	8	

Figure 5: Shrink operator

Algorithm 3 inserts intermediate nodes in a request path. Instead of using the direct shipping strategy, a transshipment strategy is applied. Therefore the requested container will be exchanged at a cross-dock. When selecting

Algorithm 4: PD Shrink operator

```
//Input: a solution S_c, a request r, a set K of vehicles;
//Output: a solution S found;
valid ← false;
while not valid do
    S ← S_c;
    k ← random(K) //get a random vehicle route where r is present;
    pickup ← get_pickup(r, S^k);
    hub_1 ← get_delivery(r, S^k);
    if is_hub(hub_1) = false then
        continue;
    v ← get_route(r, k) //get the vehicle route linked to route k via request r;
    hub_2 ← get_pickup(r, S^v);
    delivery ← get_delivery(r, S^v);
    remove(hub_1, r, S^k);
    remove(hub_2, r, S^v);
    remove(delivery, r, S^v);
    insert(delivery, r, S^k);
    valid ← is_valid(S);
return S;
```

v and $hub1$, vehicles and cross-docks from the same cluster as the pickup and delivery nodes are favoured. The function memorises and reuses the selected vehicle route and intermediate cross-dock so that each request path from a group gets the same route and cross-dock. In Figure 4, the solution at the top has been altered and resulted in the solution at the bottom. The request path 2 has been changed and now includes the cross-dock node 10 as an intermediate. This means that vehicle 1 will pick up the container from node 3 to drop it at node 10 so that vehicle 3 could pick it up from its departure node 10 and deliver it at node 4. Algorithm 4, represented in Figure 5, does exactly the opposite to Algorithm 3.

5. Computational results

5.1. Solver configuration

Generator parameters	Values
cross-dock number	1-5
request number	10-50
vehicle number	1-5
vehicle capacity	50-100
request load	5
node time window	0-1000
vehicle time window	0-1000
node service time	1-10
2D coordinates (x,y)	0-100

Table 1: Generator parameters

As this VRP with P&D and multiple cross-docks is a new problem, there is no data-set available. Therefore, data-sets must be generated randomly. Table 1 shows an example of parameters that can be used to generate

instances. Node locations and other instance characteristics are randomly generated with a normal distribution. The time windows are generated as follows. They were first generated as loose constraints and then iteratively tightened until the model became infeasible. In the end, values from the last feasible iteration were saved and used. The ranges for the other parameters are selected in a similar way. There is no predefined unit for the time and the coordinates. Details about the instances are given in Table 3. We also tested the algorithm on a classical benchmark [8] to evaluate its performance on the general VRP.

Algorithm parameters	Values	References
$PSMS$	100	1
$RPLI$	2	12
$RPLR$	0.50	12
RIT	20	2
MIT	5	1
$T0$	50	7
δ	0.75	7

Table 2: Solver parameters

Table 2 presents the parameter values used for the experiments. These parameters values have been identified after a sensitivity analysis to allow the algorithm to provide the best performances. Column "references" indicates where the parameters are mentioned in the paper.

Just like several papers in the literature, a comparative analysis of the proposed algorithm and the proposed mathematical model solved by CPLEX is presented. The model of the CPU used is 'Intel(R) Core(TM) i9-7900X CPU @3.30GHz'. The model is implemented using the CPLEX OPL library and included in a Java framework. The version 12.7 of CPLEX is used with its default configuration and therefore allows parallel computing. Therefore each instance was solved only once by CPLEX. The meta-heuristic algorithm was also implemented and included in a Java framework. The algorithm was launched 30 times for each instance, then the averages were reported.

5.2. CPLEX and meta-heuristic performances

	CPLEX				TSAM			
instance	Objective	Lower bound	Gap	Time (s)	Avg objective	Avg time (s)	Best objective	Best time (s)
d5q50k2c1r10	441.98	441.98	Optimal	107	441.98	0	441.98	0
d5q50k2c1r15		378.46		>10800	558.14	63	557.56	77
d5q50k4c1r30		603.18		>10800	1078.34	101	1010.07	49
d5q50k5c1r50					1535.28	209	1444.22	232
d5q50k2c2r10	589.65	589.65	Optimal	65	589.65	37	589.65	18
d5q50k2c2r15		440.24		>10800	681.04	29	649.43	77
d5q50k4c2r30		801.32		>10800	1224.89	116	1185.26	56
d5q50k5c2r50		973.09		>10800	1739.4	172	1571.92	113
d5q50k5c5r10	447.12	336.28	24.79	10343	407.88	60	398.22	232
d5q50k5c5r15		486.52		>10800	731.77	64	622.19	20
d5q50k5c5r30		592.49		>10800	1022.47	118	900.65	254
d5q50k5c5r50				>10800	1510.3	199	1371.46	237
d5q100k2c1r10	459.73	433.78	5.64	4358	459.73	6	459.73	2
d5q100k2c1r15		441.81		>10800	550.62	22	550.54	50
d5q100k4c1r30		614.89		>10800	1131.31	122	1025.49	79
d5q100k5c1r50				>10800	1622.08	213	1465.62	253
d5q100k2c2r10	605.43	605.43	Optimal	0.5	605.43	0	605.43	0
d5q100k2c2r15	769.08	769.08	Optimal	7	805.68	24	769.09	32
d5q100k4c2r30		745.17		>10800	1066.58	81	1056.99	94
d5q100k5c2r50		1002.97		>10800	1598.83	210	1509.94	278
d5q100k5c5r10	447.12	336.28	24.79	10326	409.8	39	398.22	138
d5q100k5c5r15		485.88		>10800	735.4	67	622.19	111
d5q100k5c5r30		588.83		>10800	1008.49	116	870.56	100
d5q100k5c5r50				>10800	1445.76	220	1325.15	271

Table 3: CPLEX and TSAM comparison

Table 3 shows the results of CPLEX and the meta-heuristic algorithm for several instances of different request quantities. The instance names give the request demand d, the capacity of the vehicle q, the vehicle number k, the cross-dock number c and the request number r. On top of that, there are vehicles with different start and end depots, requests with special vehicle type requirements and vehicle time windows. As CPLEX is an exact solver and gives optimal solutions when feasible, it needs significantly more time to finish the search. Therefore, the termination criteria for the meta-heuristic and CPLEX are set at 5 min and 3 hours, respectively. Columns *Lower bound* and *Gap* give the last infeasible lower bound and Gap found by CPLEX when it ended. If CPLEX was able to find the optimal solution before the end, columns *Lower bound* would give this optimal solution and column *Gap* would contain the word *optimal*. Column *Time* (s) indicates when the solver found the last feasible solution. If the value is > 10800, it indicates that the search needed more than 3 hours to converge. Columns *Avg objective* and *Avg time* (s) give the average of the results over 30 runs while columns *Best objective* and *Best time* (s) give the details of the best solution.

From Table 3 it can be seen that the proposed algorithm outperforms CPLEX. On the one hand, on instances containing more than 30 customers, CPLEX couldn't provide any feasible solutions but gave the lower bound. However, for some instances containing 50 requests and therefore 100 customers, CPLEX could not provide any lower bound therefore, those rows are left blank. It can be noticed that the difficulty can also vary between two instances containing similar characteristics. This is due to the node locations. On the other hand, the proposed

algorithm is not only able to match the results of CPLEX for small instances but is also able to find solutions for large instances. As expected, on average, when the number of requests, vehicles, cross-docks increases or the vehicle capacity decreases, the instance difficulty increases.

5.3. Consolidation performances

	With				Without			
Instances	Distance	Vehicle	Consolidation	Time (s)	Distance	Vehicle	Consolidation	Time (s)
d1q2-5k5c6r2	253.14	3	4	0	284.7	1	0	0
d1q5k5c6r5	267.76	3	10	1	291.71	1	0	0
d1q10k6c4r10	280.74	6	12	3	292.99	1	0	1

Table 4: Consolidation feature results

To study the difference brought by using consolidation centres, we tested TSAM on some special instances with and without the consolidation feature. Table 4 shows the best results of the meta-heuristic algorithm on several instances with different numbers of requests. The instances are clustered so that they represent cities where the deliveries have to be made by special vehicles. Moreover, those cities are linked by highways that can only be used by a specific type of vehicle. Vehicles of this type are not forced to return to the start depot. As a result, using this vehicle would improve the solution cost but require consolidations with the other vehicles. Columns *With* and *Without* give the results of the algorithm with and without the consolidation function, respectively. The distance, the number of vehicles used and the number of consolidations are reported. From Table 4 it can be seen that the consolidation feature is of great importance to find better solutions when cross-docking is involved. Without the consolidation, only one vehicle is used to pick up and deliver all the requests, then it returns to the start depot. However, with the consolidation, several vehicles are used and exchange their loads. This decreases the total distance driven as a specific vehicle is used on the highway and does not need to return to the start depot.

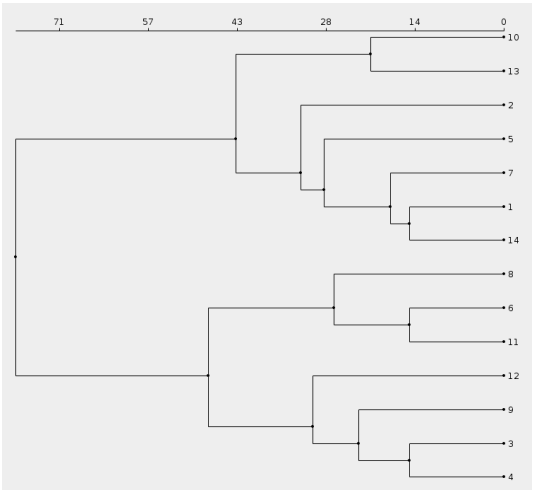


Figure 6: Dendrogram of an instance with clustered nodes

Figure 6 represents the hierarchical clustering of the instance shown in Figure 1. In this dendrogram using an average linkage strategy, the heights reflect the distance between the clusters. In this case, the dendrogram shows that there are two main clusters grouping customers. This clustering method helps the algorithm to group requests

that can be consolidated together for better solutions cost. In this case, since pickup nodes 5 and 7 belong to a different cluster from delivery nodes 6 and 8, the algorithm will try to consolidate their containers.

5.4. Benchmark performances

	Best known		TSAM				
Instances	Distance	Vehicle	Avg distance	Avg time (s)	Best distance	Best vehicle	Best time (s)
lc101s	828.94	10	828.94	0	828.94	10	0
lr101s	1650.8	19	1650.8	66	1650.8	19	17
lrc101s	1708.8	14	1713.15	114	1703.21	15	252
lc1_2_1s	2704.57	20	2704.57	25	2704.57	20	18
lr1_2_1s	4819.12	20	5106.47	845	4873.54	21	884
lrc1_2_1	3606.06	19	3765.98	859	3606.06	19	872
lc109q	1000.60	9	839.25	56	827.82	10	88
lr201	1253.23	4	1339.09	320	1286.83	5	512
lrc_1_2_5s	3715.81	16	4313.25	1043	4044.81	20	1309

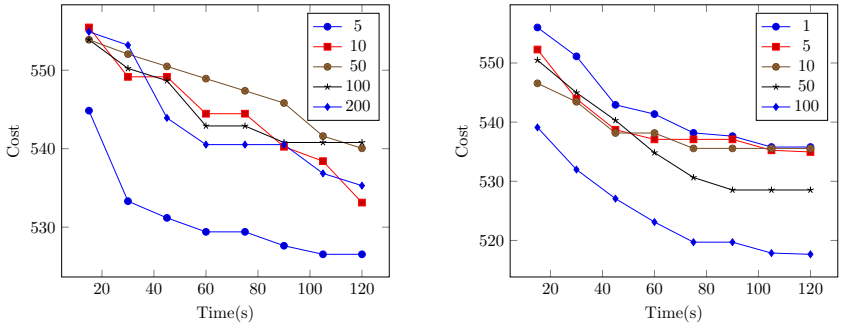
Table 5: Li&Lim benchmark

Table 5 shows the results of the proposed algorithm on some instances of Li&Lim's benchmark [9] to evaluate the performance of TSAM on the general PDP. It should be noted that the results in [9] are continuously updated with the best-known results from the state-of-the-art algorithms in the literature. To test all the different configurations, instances from each group (clustered, random, and random-clustered nodes) for 100 customers and 200 customers were selected. Columns *Best known* give the best-known solution details (taken from [9]) on the selected instances while columns $TSAM$ give the results of the proposed algorithm. Columns *Avg distance* and *Avg time* (s) give the average of the results over 30 runs while columns *Best distance*, *Best vehicle* and *Best time* (s) give the details of the best solution. Those results include the distance driven by all the vehicles, the number of vehicles used and the time at which the solution was found. The termination criterion for the meta-heuristic is set at 25 min.

From Table 5 it can be seen that the proposed algorithm is able to find some of the best-known solutions in a reasonable amount of time. On top of that, the interesting result of instance $lc109$ shows that TSAM can provide solutions which improve the distance compared to the best-known ones, but with a caveat of having to use more vehicles. However, there are instances for which TSAM could only find near-optimal solutions. This can be explained by the objective being different. This type of instance is better solved by approaches which focus on reducing the number of vehicles used. Our method, on the other hand, needs to try several vehicles to find consolidation opportunities as shown in Table 4 while also trying different combinations of vehicles due to the multi-depot constraint.

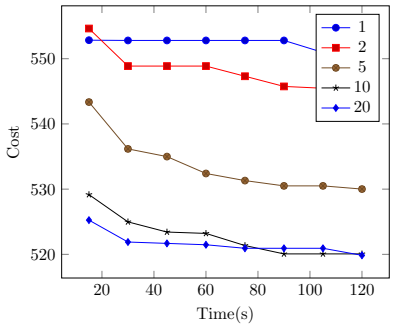
5.5. Sensitivity analysis

Figure 7 shows the convergence of the algorithm with different parameters on the instance $d5q50k2c1r10$. Compared to the one used in Table 3, this instance is more challenging as the algorithm tends to get trapped in local minima more easily with bad parameter values. Those results have been used to set the default values of the parameters in Table 2. In order to analyse the parameter sensitivity, the following reference values have been used: $PSMS = 20$, $MIT = 5$, $T0 = 40$, $delta = 0.95$, $RIT = 5$, $RPLI = 10$, $RPLR = 0.50$. For each figure, a

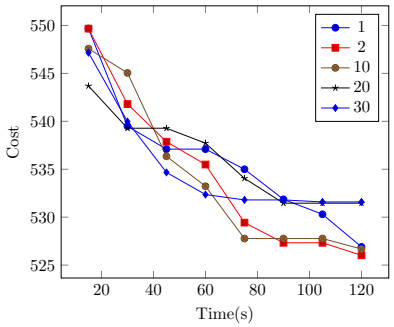


(a) MIT sensitivity analysis plot

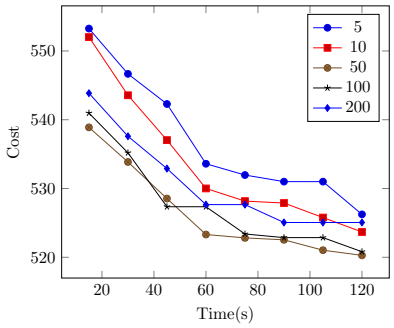
(b) PSMS sensitivity analysis plot



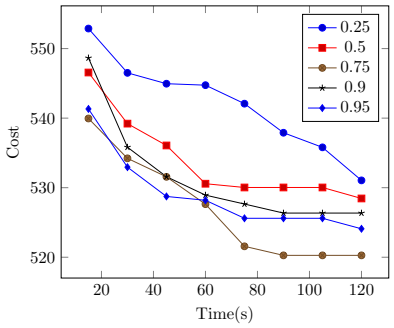
(c) RIT sensitivity analysis plot



(d) RPLI sensitivity analysis plot



(e) T0 sensitivity analysis plot



(f) δ sensitivity analysis plot

Figure 7: Parameters sensitivity analysis

single parameter is changed to 5 different values and the convergence of the algorithm is reported. The termination criterion is set to 2 minutes of running time.

Overall, parameter values which give the best convergences also give the best results at the end of the search, therefore those values are selected as default. $RPLR$ is set at 0.50 as this ratio provides a good trade-off to create solutions with consolidation by adding or removing intermediate cross-docks. A bigger ratio would provide solutions with detours while a smaller ratio would prevent consolidations. It should be noted that those convergences might vary from instance to instance as their characteristics can be different. $RPLI$ could be set to a bigger value to allow a longer search in an infeasible region while using consolidations. $PSMS$ and MIT can be useful to escape local minima as they allow the algorithm to research in previous solution neighbourhoods. Besides, $T0$ and δ can be adjusted to better explore those regions.

6. Conclusion

Cross-docking is known to be one of the most effective strategies in logistics systems to improve the flow of products in supply chains. Therefore, in this paper, a Rich VRP with pickup and delivery and multiple cross-docks is considered. This problem tackles several attributes of the problem to cope with realistic constraints - capacitated, heterogeneous, mixed pickup and delivery, multiple depots, open route, different start/end depots, time windows and site-dependent. A MILP model is designed and solved by CPLEX. Given the high complexity of the considered problem, especially in large scale, a meta-heuristic is also developed. Experiments are conducted using a wide range of generated data-sets that reflect different real-life constraints. Those constraints can force requests to be consolidated in cross-docks during deliveries. It is shown that CPLEX cannot find good solutions in a reasonable amount of time for the biggest instances. However, the proposed algorithm not only outperforms CPLEX in all the benchmark instances but is also able to scale up. Moreover, the proposed algorithm can also match some of the best-known results by state-of-the-art methods on the benchmark of Li&Lim on large instances.

The main contribution is twofold. Firstly, this paper presents a new rich VRP that has not been addressed before. An MILP model is proposed to tackle this problem. Secondly, a multi-threaded simulated annealing algorithm with memory including new operators is introduced to handle real-world size instances.

There are a few directions for further research. Since the problem is new, some standard benchmarks could be created for future comparison purposes. Also, the proposed algorithm could be integrated into a unified solution framework for multi-attribute. Finally, one could extend the problem at hand by considering other constraints such as working hours, rest times for drivers or even multi-objective and dynamism.

Acknowledgment

This work was supported by an LJMU PhD Scholarship, a NRCP grant no. NRCP1617-6-125 delivered by the Royal Academy of Engineering, and an RSSB project no COF-INP-05

References

[1] G. B. Dantzig, J. H. Ramser, The truck dispatching problem, Management Science 6 (1) (1959) 80–91. arXiv: https://doi.org/10.1287/mnsc.6.1.80, doi:10.1287/mnsc.6.1.80.
URL https://doi.org/10.1287/mnsc.6.1.80

[2] R. Sarraj, E. Ballot, S. Pan, B. Montreuil, Analogies between internet network and logistics service networks: challenges involved in the interconnection, Journal of Intelligent Manufacturing 25 (6) (2012) 1207–1219. doi: 10.1007/s10845-012-0697-7.
URL http://dx.doi.org/10.1007/s10845-012-0697-7

[3] B. Montreuil, Physical internet manifesto, in: Transforming the way physical objects are moved, stored, realized, supplied and used, aiming towards greater efficiency and sustainability, 2012.

[4] R. D. Meller, B. Montreuil, C. Thivierge, Z. Montreuil, Functional design of physical internet facilities: A road-based transit center, Tech. rep., Progress in Material Handling Research 2012 (2012).

[5] B. Montreuil, Towards a physical internet: Meeting the global logistics sustainability grand challenge, in: Logistics Res., vol. 3, nos. 2-3, pp. 71-87, 2011, 2011.

[6] T. G. Crainic, B. Montreuil, Physical internet enabled hyperconnected city logistics, Transportation Research Procedia 12 (2016) 383–398. doi:10.1016/j.trpro.2016.02.074.
URL http://dx.doi.org/10.1016/j.trpro.2016.02.074

[7] T. Vidal, T. G. Crainic, M. Gendreau, C. Prins, A unified solution framework for multi-attribute vehicle routing problems, European Journal of Operational Research 234 (3) (2014) 658–673. doi:10.1016/j.ejor.2013.09.045.
URL http://dx.doi.org/10.1016/j.ejor.2013.09.045

[8] H. Li, A. Lim, A metaheuristic for the pickup and delivery problem with time windows, in: Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2001, 2001, pp. 160–167. doi:10.1109/ICTAI.2001.974461.

[9] SINTEF, Li&lim benchmark, https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/, accessed: 2018-09-10.

[10] P. Ballesteros Silva, A. Escobar Zuluaga, Review of state of the art vehicle routing problem with pickup and delivery (vrppd) 34 (2016) 463–482.

[11] J. Caceres-Cruz, P. Arias, D. Guimarans, D. Riera, A. A. Juan, Rich vehicle routing problem: Survey, ACM Comput. Surv. 47 (2) (2014) 32:1–32:28. doi:10.1145/2666003.
URL http://doi.acm.org/10.1145/2666003

[12] R. Lahyani, M. Khemakhem, F. Semet, Rich vehicle routing problems: From a taxonomy to a definition, European Journal of Operational Research 241 (1) (2015) 1–14. doi:10.1016/j.ejor.2014.07.048.

[13] D. Pisinger, S. Ropke, A general heuristic for vehicle routing problems, Computers & Operations Research 34 (8) (2007) 2403 – 2435. doi:https://doi.org/10.1016/j.cor.2005.09.012.
URL http://www.sciencedirect.com/science/article/pii/S0305054805003023

[14] C. Wang, D. Mu, F. Zhao, J. W. Sutherland, A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup–delivery and time windows, Computers & Industrial Engineering 83 (2015) 111 – 122. doi:https://doi.org/10.1016/j.cie.2015.02.005.
URL http://www.sciencedirect.com/science/article/pii/S0360835215000625

[15] E. Zachariadis, C. D. Tarantilis, C. Kiranoudis, An adaptive memory methodology for the vehicle routing problem with simultaneous pick-ups and deliveries, European Journal of Operational Research 202 (2010) 401–411.

[16] N. A. Wassan, G. Nagy, Vehicle routing problem with deliveries and pickups: modelling issues and meta-heuristics solution approaches, International Journal of Transportation 2 (1) (2014) 95–110.

[17] M. M. Solomon, Algorithms for the vehicle routing and scheduling problems with time window constraints, in: Operations Research, no. 254-265, 1985.
URL http://www.jstor.org/stable/170697

[18] J.-F. Cordeau, Q. Groupe d'études et de recherche en analyse des décisions (Montréal, The VRP with time windows, Groupe d'études et de recherche en analyse des décisions Montréal, 2000.

[19] P. Pellegrini, D. Favaretto, E. Moretti, Multiple Ant Colony Optimization for a Rich Vehicle Routing Problem: A Case Study, Springer Berlin Heidelberg, 2007, pp. 627–634.

[20] M. Desrochers, J. K. Lenstra, M. W. P. Savelsbergh, A classification scheme for vehicle routing and scheduling problems, European Journal of Operational Research 46 (3) (1990) 322–332.
URL http://EconPapers.repec.org/RePEc:eee:ejores:v:46:y:1990:i:3:p:322-332

[21] J. R. Montoya-Torres, J. L. Franco, S. N. Isaza, H. F. Jiménez, N. Herazo-Padilla, A literature review on the vehicle routing problem with multiple depots, Computers & Industrial Engineering 79 (2015) 115–129. doi:10.1016/j.cie.2014.10.029.

[22] G. Nagy, S. Salhi, Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries, European Journal of Operational Research 162 (1) (2005) 126–141. doi:10.1016/j.ejor.2002.11.003.

[23] A. I. Nikolopoulou, P. P. Repoussis, C. D. Tarantilis, E. E. Zachariadis, Moving products between location pairs: Cross-docking versus direct-shipping, European Journal of Operational Research 256 (3) (2017) 803–819. doi:10.1016/j.ejor.2016.06.053.

[24] F. Enderer, C. Contardo, I. Contreras, Integrating dock-door assignment and vehicle routing with cross-docking, Computers & Operations Research 88 (Supplement C) (2017) 30 – 43. doi:https://doi.org/10.1016/j.cor.2017.06.018.
URL http://www.sciencedirect.com/science/article/pii/S0305054817301569

[25] R. Dondo, J. Cerdá, A monolithic approach to vehicle routing and operations scheduling of a cross-dock system with multiple dock doors, Computers & Chemical Engineering 63 (2014) 184–205. doi:10.1016/j.compchemeng.2013.12.012.

[26] K. F. Zhaowei Miao, F. Yang, A hybrid genetic algorithm for the multiple crossdocks problem, Mathematical Problems in Engineering (2012).

[27] Y. Maknoon, G. Laporte, Vehicle routing with cross-dock selection, Computers & Operations Research (77) (2017) 254–266. doi:doi:10.1016/j.cor.2016.08.007.

[28] M. Alinaghian, M. R. Kalantari, A. Bozorgi-Amiri, N. G. Raad, A novel mathematical model for cross dock open-close vehicle routing problem with splitting, International Journal of Mathematical Sciences and Computing 2 (3) (2016) 21–31. doi:10.5815/ijmsc.2016.03.02.

[29] R. Russell, W.-C. Chiang, D. Zepeda, Integrating multi-product production and distribution in newspaper logistics, Computers & Operations Research 35 (5) (2008) 1576 – 1588, part Special Issue: Algorithms and Computational Methods in Feasibility and Infeasibility. doi:http://dx.doi.org/10.1016/j.cor.2006.09.002.
URL http://www.sciencedirect.com/science/article/pii/S0305054806002140

[30] V. F. Yu, P. Jewpanya, A. P. Redi, Open vehicle routing problem with cross-docking, Computers & Industrial Engineering 94 (2016) 6–17. doi:10.1016/j.cie.2016.01.018.

[31] R. Atefi, M. Salari, L. C. Coelho, J. Renaud, The open vehicle routing problem with decoupling points, European Journal of Operational Research 265 (1) (2018) 316 – 327. doi:https://doi.org/10.1016/j.ejor.2017.07.033.
URL http://www.sciencedirect.com/science/article/pii/S0377221717306604

[32] M.-C. Chen, Y.-H. Hsiao, H. Reddy, M. K. Tiwari, A particle swarm optimization approach for route planning with cross-docking, in: 7th International Conference on Emerging Trends in Engineering & Technology (ICETET), IEEE, 2015. doi:10.1109/icetet.2015.12.

[33] J. Wang, A. K. R. Jagannathan, X. Zuo, C. C. Murray, Two-layer simulated annealing and tabu search heuristics for a vehicle routing problem with cross docks and split deliveries, Computers & Industrial Engineering 112 (2017) 84–98. doi:10.1016/j.cie.2017.07.031.

[34] A. Ahkamiraad, Y. Wang, Capacitated and multiple cross-docked vehicle routing problem with pickup, delivery, and time windows, Computers & Industrial Engineering 119 (2018) 76 – 84. doi:https://doi.org/10.1016/j.cie.2018.03.007.
URL http://www.sciencedirect.com/science/article/pii/S0360835218300925

[35] R. Sarraj, E. Ballot, S. Pan, D. Hakimi, B. Montreuil, Interconnected logistic networks and protocols: simulation-based efficiency assessment, International Journal of Production Research 52 (11) (2013) 3185–3208. doi:10.1080/00207543.2013.865853.
URL http://dx.doi.org/10.1080/00207543.2013.865853

[36] V. F. Yu, P. Jewpanya, V. Kachitvichyanukul, Particle swarm optimization for the multi-period cross-docking distribution problem with time windows, International Journal of Production Research 54 (2) (2015) 509–525. doi:10.1080/00207543.2015.1037933.

[37] M.-C. Chen, Y.-H. Hsiao, R. H. Reddy, M. K. Tiwari, The self-learning particle swarm optimization approach for routing pickup and delivery of multiple products with material handling in multiple cross-docks, Transportation Research Part E: Logistics and Transportation Review 91 (2016) 208–226. doi:10.1016/j.tre.2016.04.003.

[38] F. Ahmadizar, M. Zeynivand, J. Arkat, Two-level vehicle routing with cross-docking in a three-echelon supply chain: A genetic algorithm approach, Applied Mathematical Modelling 39 (22) (2015) 7065–7081. doi:10.1016/j.apm.2015.03.005.

[39] V. B. Kreng, F.-T. Chen, The benefits of a cross-docking delivery strategy: a supply chain collaboration approach, Production Planning and Control 19 (3) (2008) 229–241.

[40] E. Ballot, O. Gobet, B. Montreuil, Physical internet enabled open hub network design for distributed networked operations, in: Service orientation in holonic and multi-agent manufacturing control, Springer, 2012, pp. 279–292.

Appendix A. Algorithm functions and operators

Algorithm 5: Neighbour Search function

```
1  //Input: a solution S;
2  //Output: the best solution S_b found;
3  S_b ← S;
4  while true do
5      S' ← PD_operate(S_b) //apply a random operator from op_list;
6      if cost(S') < cost(S_b) then
7          S_b ← S';
8      else
9          break;
10 return S_b;
```

Algorithm 6: Random Solution function

```
1  //Input: a solution S;
2  //Output: a solution S' found;
3  it ← 0;
4  S' ← ∅;
5  while S' = ∅ and it < RIT do
6  |   S' ← PD_operate(S);
7  |   it ← it + 1;
8  if S' = ∅ then
9  |   S' ← random_previous_solution();
10 return S';
```

Algorithm 7: Simulated Annealing function

```
1  //Input: a solution S;
2  //Output: a solution S' found;
3  f ← false;
4  t ← T0;
5  while f = false do
6  |   S' ← random_solution(S);
7  |   Δ ← cost(S') − cost(S);
8  |   if Δ ≤ 0 then
9  |   |   p ← 1;
10 |   else
11 |   |   p ← e^{−Δ/T};
12 |   t ← δ * T;
13 |   if random_double() ≤ p then
14 |   |   f ← true;
15 |   else if t < 0.01 then
16 |   |   f ← true;
17 |   |   S' ← S;
18 return S';
```

Algorithm 8: PD Interchange operator

```
1  //Input: a current solution S_c, a set K of vehicles;
2  //Output: a solution S found;
3  valid ← false;
4  while valid = false do
5  |   S ← S_c;
6  |   k ← random(K);
7  |   node ← remove(S^k) //remove a random node;
8  |   if insert(node, S^k) then
9  |   |   valid ← is_valid(S);
10 return S;
```

Algorithm 9: PD Move operator

```
1  //Input: a current solution S_c, a set K of vehicles, a set R of requests;
2  //Output: a solution S found;
3  valid ← false;
4  shuffle(K);
5  r ← random(R);
6  v ← ∅;
7  changed ← 0;
8  while valid = false do
9      S ← S_c;
10     foreach k in K do
11         foreach node in S^k do
12             //cross-docks can contain several r;
13             if contain(node, r) then
14                 if v = ∅ then
15                     v ← random(K/k) //new vehicle route v must be different from k;
16                 remove(node, r, S^k);
17                 if not insert(node, r, S^v) then
18                     break 2 loops;
19                 changed ← changed + 1;
20             if changed = 2 then
21                 valid ← is_valid(S);
22                 break 2 loops;
23 return S;
```

Algorithm 10: PD Swap operator

```
1  //Input: a solution S, a set K of vehicles, a set R of requests;
2  //Output: a solution S' found;
3  valid ← false;
4  while valid = false do
5      S' ← ∅;
6      v_1 ← random(K);
7      v_2 ← random(K/v_1) //vehicle v_2 must be different from v_1;
8      for k in K do
9          if k = v_1 then
10             S'_k ← S_{v_2} //the depots are those from vehicle v_1;
11         else if k = v_2 then
12             S'_k ← S_{v_1} //the depots are those from vehicle v_2;
13         else
14             S'_k ← S_k;
15     valid ← is_valid(S');
16 return S';
```

Algorithm 11: PD Exchange operator

```
1  //Input: a current solution S_c, a set K of vehicles, a set R of requests;
2  //Output: a solution S found;
3  valid ← false;
4  shuffle(K);
5  while valid = false do
6      r_1 ← random(R);
7      r_2 ← random(R/r_1);
8      node_1a ← ∅;
9      node_1b ← ∅;
10     node_2a ← ∅;
11     node_2b ← ∅;
12     S ← S_c;
13     for k in K do
14         for node in S^k do
15             if contain(node, r_1) and (node_1a = ∅ or node_1b = ∅) then
16                 k_1 ← k;
17                 if node_1a = ∅ then
18                     node_1a ← node;
19                 else
20                     node_1b ← node;
21                 remove(node, r_1, S^k);
22             if contain(node, r_2) and (node_2a = ∅ or node_2b = ∅) then
23                 k_2 ← k;
24                 if node_2a = ∅ then
25                     node_2a ← node;
26                 else
27                     node_2b ← node;
28                 remove(node, r_2, S^k);
29     if contain(S^{k_1}, r_2) or contain(S^{k_2}, r_1) then
30         continue;
31     insert(node_1a, r_1, S^{k2});
32     insert(node_1b, r_1, S^{k2});
33     insert(node_2a, r_2, S^{k_1});
34     insert(node_2b, r_2, S^{k_1});
35     valid ← is_valid(S);
36 return S;
```

Algorithm 12: Consolidation function

```
1  //Input: a current solution S_c;
2  //Output: the best solution S_b found;
3  S_b ← ∅;
4  S ← S_c;
5  if random_double() < 0.5 then
6      S ← PD_arrange(S);
7  group ← random(req_groups);
8  while no_progress < RPLI do
9      if random_double() < RPLR then
10         operator ← PD_stretch;
11     else
12         operator ← PD_shrink;
13     cnt ← 0;
14     foreach r in group do
15         S ← operator(S, r) //apply PD_stretch() or PD_shrink();
16     if is_valid(S) and cost(S) < cost(S_b) then
17         S_b ← S;
18         no_progress ← 0;
19         group ← random(req_groups);
20     else
21         no_progress ← no_progress + 1;
22 return S_b;
```