The vehicle routing problem with cross-docking and resource constraints

Philippe Grangier, Michel Gendreau, Fabien Lehuédé, Louis-Martin Rousseau

Journal of Heuristics manuscript No.
(will be inserted by the editor)

The vehicle routing problem with cross-docking and resource constraints

Philippe Grangier, Michel Gendreau, Fabien Lehuédé
and Louis-Martin Rousseau

Received: date / Accepted: date - submitted: August 27, 2019

Abstract In this paper, we propose an extension of the vehicle routing problem with cross-docking that takes into account resource constraints at the cross-dock. These constraints limit the number of docks that can be used simultaneously. To solve this new problem, we adapt a recently proposed matheuristic based on large neighborhood search. In particular, we focus on the feasibility tests for insertions and compare heuristics and constraint programming strategies. Finally, computational experiments on instances adapted from the vehicle routing problem with cross-docking are reported. They give insights on the impact of a limited cross-dock capacity on the routing cost.

1 Introduction

In logistics, cross-docking is a distribution strategy in which goods are brought from suppliers to an intermediate transshipment point, the so-called cross-dock, where they may be transferred (without storing) to another vehicle for delivery. Compared to traditional distribution systems, cross-docking can help reducing delivery costs and delivery lead time, that is why it is used by many companies from different sectors: less-than-truckload (LTL), retail or automotive for example (Van Belle et al., 2012). In the vehicle routing literature, the associated routing problem is called the Vehicle Routing Problem with Cross-Docking (VRPCD) (Wen et al., 2008). It is a variant of the Pickup and Delivery Problem with Transfers (Cortés et al., 2010) with one compulsory transfer point: vehicles start by collecting items, then return to

P. Grangier
Element AI, 6650 rue Saint Urbain, Montreal, QC H2S 3G9, Canada.
E-mail: philippe.grangier@elementai.com

M. Gendreau, L.M. Rousseau
Department of Mathematics and Industrial Engineering and CIRRELT, Ecole Polytechnique de Montréal and CIRRELT, C.P 6079, Succursale Centre-ville, Montreal, QC, H3C 3A7, Canada.
E-mail: {louis-martin.rousseau;michel.gendreau}@cirrelt.net

F. Lehuédé
IMT Atlantique, L2SN, UMR CNRS 6004, 4 Rue Alfred Kastler, 44300 Nantes, France.
E-mail: fabien.lehuede@imt-atlantique.fr

the cross-dock where they unload/reload some items and eventually visit delivery locations. A few authors have proposed methods to solve this problem or variants of it, but to our knowledge, in most models there is no limit on the processing capacities of the cross-dock: as soon as a truck arrives, it immediately undergoes consolidation operations. However in practice, this may not be the case because of limited equipment or workforce, thus trucks may have to wait before being unloaded. In fact, a wide range of the cross-docking literature is dedicated to the scheduling of operations at the cross-dock taking into account the cross-dock capacity. It has recently been pointed out (Buijs et al., 2014; Ladier and Alpan, 2016), that there is a need to consider the synchronization of local and network-wide cross-docking operations, in particular to take into account resource capacity at the cross-dock. To that end, in this paper, we introduce a new variant of the VRPCD in which the number of vehicles that can simultaneously be processed at the cross-dock is limited. We call it the Vehicle Routing Problem with Cross-Docking and Resource Constraints (VRPCD-RC). The dock resource constraint is a *resource synchronization constraint* as defined by Drexl (2012) as vehicles compete to access a scarce resource: the processing capacity of the cross-dock. Very often resource synchronization constraints imply a difficult scheduling problem which is embedded within the vehicle routing problem. VRPCD-RC is no exception and a major contribution of this paper is on the integration of the scheduling problem associated with the dock resource constraints within a recently proposed large neighborhood search (LNS) based matheuristic (Grangier et al., 2017) for the VRPCD.

The remainder of this paper is organized as follows. A literature review is presented in Section 2, while the problem is defined in Section 3. In Section 4, we present the proposed method. Computational results are presented in Section 5, and a conclusion is given in Section 6.

2 Literature review

In this section we review the literature on two related vehicle routing problems: the vehicle routing problem with cross-docking and vehicle routing problems with resource synchronization.

2.1 The vehicle routing problem with cross-docking

Many cross-docking problems have received attention from operations research and management science researchers. For instance: location, assignment of trucks to doors, inner flow optimization or routing (Van Belle et al., 2012). In particular, the vehicle routing problem with cross-docking consists in designing routes to pick up and deliver a set of transportation requests at minimal cost using a single cross-dock. It was introduced by Lee et al. (2006) in a variant which imposes trucks to arrive at the exact same time at the cross-dock. Wen et al. (2008) relaxed this constraint only imposing precedence constraints based on the consolidation decisions and they also added time windows. This is the most studied variant, and it is the one we will refer to as the *vehicle routing with cross-docking* (VRPCD). Several heuristics have been proposed to solve it: based on tabu-search (Wen et al., 2008; Tarantilis, 2012; Nikolopoulou et al., 2017, 2016), iterated local search (Morais et al., 2014) and LNS (Grangier

et al., 2017). Many variants have been studied, with optional cross-dock returns (Petersen and Ropke, 2011; Santos et al., 2013; Nikolopoulou et al., 2017), taking into account the cost of handling (Santos et al., 2011a,b; Enderer et al., 2017), with multiple cross-docks (Ahmadizar et al., 2015; Maknoon and Laporte, 2017; Kroep et al., 2017), with partly subcontracted or left to suppliers routes (Yu et al., 2016; Enderer et al., 2017).

Dondo and Cerdá (2014) considered a case where the number of doors at the cross-dock is fixed and smaller than the number of trucks. In particular each door is modeled individually: a time matrix models the time spent by a truck for moving from an inbound door a to an outbound door b. They solved two randomly generated instances with up to 70 requests with a mathematical model combined with a sweep heuristic.

From a general perspective, the VRPCD can be viewed as a special case of the pickup and delivery problem with transfers with only one compulsory transfer point (Cortés et al., 2010). Guastaroba et al. (2016) released a survey on intermediate facilities in freight transportation. For a general overview of cross-docking and cross-dock related problems we refer the reader to Boysen and Fliedner (2010); Agustina et al. (2010); Van Belle et al. (2012), and Nassief et al. (2018) for a recent article on the dock-door assignment problem.

2.2 Resource synchronization

The expression *resource synchronization* appears in Drexl (2012) as a way to model the following constraint:

> 'The total consumption of a specified resource by all vehicles must be less than or equal to a specified limit.'

Of course, this resource has to be scarce to be constraining, as vehicles *compete* to access it. Such resource constraints arise in many different vehicle routing problems usually when a special infrastructure or equipment is required: a docking station or parking space in airport cargo system (Ebben et al., 2005), a berth in maritime transportation (Gronhaug et al., 2010), a forest loader in forestry (El Hachemi et al., 2010), a pump in ready mix-concrete delivery Schmid et al. (2009), an asphalt paver in public works (Grimault et al., 2014, 2017), intermediate charging stations for electric vehicle (Froger et al., 2017). Limited storage (Ebben et al., 2005) or processing capacities (Hempsch and Irnich, 2008) can also account for resource synchronization constraints. Hempsch and Irnich (2008) also mention a situation where only a fixed number of vehicles (smaller than the total number) can perform *long* routes. When the constrained resource travels to deliver products or services, it corresponds to the definition of *resource constrained routing* given in the recent survey by Paraskevopoulos et al. (2017).

Many approaches have been applied to deal with these resource constraints in vehicle routing problems. Ebben et al. (2005) sequentially inserted requests and check resource constraints in a predefined order. El Hachemi et al. (2010) use a dedicated constraint programming model, later combined with a greedy scheduling heuristic in El Hachemi et al. (2013), to ensure that resource constraints are satisfied during their entire solving process. In ready-mix concrete routing problems as well as in public works routing problem, orders are larger than truck capacity, as such they

have to be split into several delivery operations that should not overlap. Resource synchronization constraints arise at pickup sites or at delivery sites or at both. Asbach et al. (2009), Schmid et al. (2009), Schmid et al. (2010) and Grimault et al. (2017) imposed precedence constraints on the sequencing of operations to handle precedence constraints. Gronhaug et al. (2010) relied on a time-discretized formulation in which resource synchronization constraints are easily expressed. Hempsch and Irnich (2008) proposed a generic modeling for inter-routes constraints via the use of Resource Extension Functions (REF). In particular they focused on the use of REF as efficient feasibility tests in local search based algorithms when the solution is represented by a giant tour. On the other hand, Froger et al. (2017) relax charging station capacity constraints when generating routes via an iterated local search for an electric vehicles routing problem. Only when assembling routes into a solution do they enforce these synchronization constraints in a Benders' like approach. They call this approach *a route-first assemble-second*. Their later part shares some similarity with the approach that is proposed in this paper, however in our method we enforce synchronization constraints even during the route generation phase. Generally speaking, these type of approaches are column generation based matheuristics in the framework of Archetti and Speranza (2014).

From this literature review, it is clear that most of the resource synchronization constraints are in fact complex scheduling problem integrated into a vehicle routing problem. However the precise definition of the scheduling problem depends largely on the vehicle routing problem at stake.

3 The vehicle routing problem with cross-docking: model and resource synchronization constraint

In the VRPCD-RC, we consider a set of transportation requests, each having different origins and destinations. Each vehicle starts from the depot, performs a pickup leg which ends at the cross-dock, then it performs a delivery leg and returns to the depot. At the cross-dock, some transportation requests are exchanged between vehicles, creating precedence constraints between pickup and deliveries legs of different vehicles. We consider the number of doors that can be processed simultaneously a the cross-dock is limited. This new constraint may introduce some additional waiting times for vehicles at the cross-dock (compared to the VRPCD). We consider two cross-dock configurations:

- In the *shared* configuration, the total number of vehicles that can be processed simultaneously is limited to a number S.
- In the *separated* configuration, the unloading capacity is separated from the loading capacity into inbound and outbound doors.

This section presents the vehicle routing problem with cross-docking, and in particular the cross-dock model, as defined by Wen et al. (2008). It also presents in details the two considered cross-dock models.

3.1 The vehicle routing problem with cross-docking

In the VRPCD, we consider a cross-dock c, a set of requests R, and a homogeneous fleet of vehicles V, each of capacity Q and based at o. Each request $r \in R$ has to

be picked up at its pickup location p_r within its pickup time window $[e_{p_r}, l_{p_r}]$, and delivered at its delivery location d_r within its delivery time window $[e_{d_r}, l_{d_r}]$. In the case of early arrival, a vehicle is allowed to wait, but late arrivals are forbidden. We denote by P the set of pickup locations and by D the set of delivery locations.
Each vehicle starts at o, goes to several pickup locations, and then arrives at the cross-dock where it unloads/reloads some requests. It then visits several delivery locations and eventually returns to o. Note that a vehicle must visit the cross-dock even if it does not unload nor reload there. The sequence of operations at the cross-dock is described in Section 3.2. The base of vehicles o and the cross-dock c may be the same location.
The VRPCD is defined on a directed graph $G = (V, A)$, with $G = \{o\} \cup P \cup \{c\} \cup D$ and $A = \{(o, p) | p \in P\} \cup P \times P \cup \{(p, c) | p \in P\} \cup D \times D \cup \{(d, o) | d \in D\} \cup \{(o, c), (c, o)\}$. With each arc $(i, j) \in A$ is associated a travel time $t_{i,j}$ and a travel cost $c_{i,j}$.
Solving the VRPCD involves finding $|V|$ routes, and a schedule for each route, such that the capacity and time-related constraints are satisfied, at minimal routing cost. An arc-based mathematical formulation can be found in Wen et al. (2008).

3.2 Precedence constraints at the cross-dock

Following Wen et al. (2008), if a vehicle k has to unload a set of items R_k^- and reload a set R_k^+ at the cross-dock, the time spent at the cross-dock can be divided into four periods:

- Preparation for unloading. The duration δ_u of this period is fixed.
- Unloading. The duration of this period depends on the quantity of items to unload. For vehicle k the duration is $(\sum_{i \in R_k^-} q_i)/s_u$, where s_u is the unloading speed in quantity per time unit. All unloaded items become available for reloading at the end of this period.
- Preparation for reloading. The duration δ_r of this period is fixed.
- Reloading of requests. The duration of this period depends on the quantity of items to reload. For vehicle k the duration is $(\sum_{i \in R_k^+} q_i)/s_r$, where s_r is the reloading speed in quantity per time unit. All the items for loading must have been unloaded before the beginning of the reloading operation (preemption is not allowed).

Items that are not transferred at the cross-dock remain in the vehicle. Thus, if a vehicle does not unload or reload it need not spend any time at the cross-dock and can leave immediately.

3.3 Resource constraints models at the cross-dock

According to Van Belle et al. (2012), the most common service mode of a cross-dock is called *exclusive*. In an exclusive mode, a dock door is either exclusively dedicated to unloading (inbound operations) or reloading (outbound operations). Such assignment is a decision taken at a strategical or tactical level that cannot be modified in the VRPCD, which is an operational problem. In practice most cross-docks are I-shaped (Van Belle et al., 2012), with inbound doors on one side and outbound doors on the other. The flow of items is thus uni-directional, which is typically easier to manage. This is a common situation but it is not mandatory.

Cross-docks usually have many doors (typically ranging from 40 to 150 according to Van Belle et al. (2012)). However, as mentioned in Hempsch and Irnich (2008) or Li et al. (2004), processing capacities may actually be lower than the number of doors. This comes from a limited workforce or special equipment to move the items within the cross-dock.
We denote by A_I (resp. A_O) the number of inbound doors (resp. outbound) that are being processed simultaneously. Because of the previous two considerations regarding operations at the cross-dock, we will consider two exclusive cross-dock models that integrate resource constraints:

- a case in which the total number of doors (inbound plus outbound) that can be processed simultaneously is limited to a number S, ie $A_I + A_O \leq S$. This is a simple way to model resource constraints due to limited workforce. We refer to this case as *shared*.
- a case in which the number of inbound doors (resp. outbound doors) that can be processed simultaneously is limited to a number I (resp O), ie $A_I \leq I$ (resp. $A_O \leq O$). This is a simple way to model a resource constraints due to special equipment on each side of the cross-dock. We refer to this case as *separated*.

Provided that at least two dock doors can be processed simultaneously, the scheduling problems at the cross-dock are NP-Hard as the scheduling problem $P2||C_{max}$ is included in them. In the rest, we will simply use *dock* to refer to the capacity in the number of docks processed simultaneously.
We call the VRPCD with these dock resource constraints, *vehicle routing problem with cross-docking and resource constraints* (VRPCD-RC). Figure 1 illustrates the sequence of consolidation operations for a vehicle that unloads and reloads items at the cross-dock in the VRPCD-RC. When the vehicle arrives at the cross-dock, it can immediately be prepared for unloading, but it has to wait before being actually unloaded (W_U) because of a lack of available resources (note that such waiting time does not exist in the VRPCD). Once the unloading operation is done, the vehicle can proceed and move to an outbound door. Again the preparation can be performed immediately, but the truck may wait before being actually reloaded (W_R). This waiting time can have two origins: first, not all items are available when it is ready (such situation can also arise in the VRPCD), second, there maybe a lack of available resources (which cannot occur in the VRPCD).

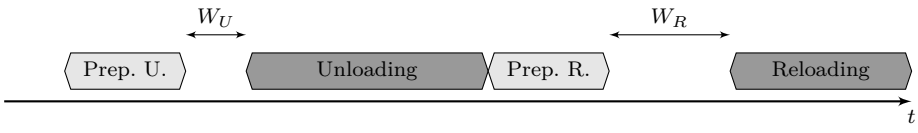


Fig. 1: Time chart for vehicle unloading and reloading at the cross-dock in the VRPCD-RC

4 Matheuristic for the VRPCD-RC

In this section, we present the matheuristic we propose for the VRPCD-RC: Large Neighborhood Search+Set Partitioning and Scheduling (LNS+SPS). It is derived from the method of Grangier et al. (2017) for the VRPCD. We recall the structure of this previous method in Section 4.1, while the overall LNS+SPS method is summarized in Section 4.2. In Section 4.3 we present the destruction and repair operators used in LNS. The resource constraints require special feasibility test for insertions in the repair methods that are detailed in Section 4.4. The set partitioning component called Set Partitioning and Scheduling problem (SPS) is presented in Section 4.5.

4.1 Matheuristic of Grangier et al. (2017) for the VRPCD

In a previous paper (Grangier et al., 2017), we proposed a matheuristic to solve the VRPCD, that is still, to our knowledge, one of the best method for this problem. This method is a LNS with periodic calls to a set partitioning based problem that assembles parts of previously obtained solutions. It is called LNS + SPM (Large Neighborhood Search + Set Partitioning and Matching)

LNS (Shaw, 1998) iteratively destroys (removes several requests from) and repairs (reinserts requests into) the current solution using heuristics. In Grangier et al. (2017), most of these heuristics are adapted to the VRPCD from Pisinger and Ropke (2007). For each request to reinsert, potential transfers lead to quadratic repair possibilities in the number of vehicles (as opposed to linear in the VRP). We only considered transferring a request among vehicles that drive near its pickup or delivery location as a way to fight this growth. This help speeding up the whole method without loss in quality. On top of that, by reusing the technique of Masson et al. (2013) time feasibility of reinsertion could be checked in constant time.

Each time a solution is obtained by applying the destroy+repair process of the LNS, for each route its pickup and delivery parts (called *legs*) are stored in a memory. The SPM component aims at assembling all the legs in memory into a potentially better solution. Due to possible transfers at the cross dock, there exists some precedence constraints between pickup and delivery legs. This makes the SPM a set partitioning problem with additional precedence constraints. It is worth noting that coupling a set partitioning component and a meta-heuristic in a route-first assemble second fashion has proved being successful for many VRP (see e.g. Rochat and Taillard (1995); Archetti et al. (2008); Subramanian et al. (2013); Pillac et al. (2013); Villegas et al. (2013); Mendoza et al. (2015); Froger et al. (2017); Tellez et al. (2018)).

This algorithmic approach has been retained for the VRPCD-RC, in what follows we will detail each part of the algorithm, specifying if it has been reused from the VRPCD or what changes were made for the VRPCD-RC and why.

4.2 Structure of the proposed method

Algorithm 1 presents a sketch of the proposed method: Large Neighborhood Search + Set Partionning and Scheduling. The main component is LNS (l.5-16), which is enhanced by the occasional resolution of an SPS (l.20). When the LNS finds a new

solution, both routes (l. 18) and pickup and delivery part of each route (l. 17) are added to memory components called *pools*. The SPS (Algorithm 2) aims to find the best possible solution from the route in the pool of routes after applying some dominance-based replacement using the pool of legs. This is detailed in Section 4.5.

Result: The best found solution $s^\star$
1 Pool of legs $\mathcal{L} := \emptyset$
2 Pool of routes $\mathcal{K} := \emptyset$
3 Generate an initial solution s
4 $s^\star := s$
5 **while** *stop-criterion not met* **do**
6 $s' := s$
7 **Destroy quantity:** select a number Φ of requests to remove from s'
8 **Operator selection:** select a destruction operator M^- and a repair operator M^+
9 **Destruction :** apply M^- to remove Φ requests from s', and put them in the requests bank of s'
10 **Repair:** apply M^+ to reinsert the requests in the requests bank in s'
11 **if** *acceptance criteria is met* **then**
12 $s := s'$
13 **end**
14 **if** *cost of* s' *is better than cost of* $s^\star$ **then**
15 $s^\star := s'$
16 **end**
17 Add legs of s' to $\mathcal{L}$
18 Add routes of s' to $\mathcal{R}$
19 **if** *set partitioning and scheduling condition is met* **then**
20 Perform SPS($\mathcal{L}$, $\mathcal{K}$, $s^\star$, s');
21 **end**
22 **end**
23 **return** $s^\star$

Algorithm 1: LNS+SPS

4.3 Large neighborhood search

Hereafter, the destruction and repair methods used in the LNS are presented.

4.3.1 Destruction operators

When partially destroying a solution, a destruction method M^- and a number Φ of requests to remove are selected. Unless stated otherwise, this method is reused until Φ is reached. The destruction operators are taken from Grangier et al. (2017).

Random removal: a request is removed at random.

Worst removal: a request with a high *removal gain* is removed, where removal gain is the difference in the cost of the solution with and without the request. The requests are sorted in non-increasing order of removal gain and put in a list N. The request to remove is selected in a randomized fashion as in Ropke and Pisinger (2006): given a parameter p, a random number y between 0 and 1 is drawn. Then the request in position $y^p \times |N|$ is removed.

Historical node-pair removal: each arc $(u, v) \in G$ is associated with the cost of the cheapest solution it appears in (initially this cost is set to infinity). The request that is served using the arcs with the highest associated costs is removed using a randomized selection similar to that of *worst removal*.

Related removals: these methods aim to remove related requests. Let the relatedness of requests i and j be $R(i, j)$. Two distinct relatedness measures are used: distance and time. The distance measure between two requests is the sum of the distance between their pickup points and the distance between their delivery points. The time measure is the sum of the absolute difference between their start of service at their pickup points and the absolute gap between their start of service at their delivery point. In both cases a small $R(i, j)$ indicates a high relatedness. A randomized selection, similar to *worst removal* (albeit with a non decreasing ordering), is performed.

Transfer removal: for each pair of routes (v_i, v_j), with $v_i \neq v_j$ the number of requests transferred from v_i to v_j is computed. Then a roulette-wheel selection is applied on the pairs of routes (the score of a pair being the number of requests transferred), and the requests that are transferred between the routes in the selected pair are removed. If there are less transferred requests than the target number Φ to remove, the rest of the removals is performed with random removal.

4.3.2 Repair operators

In LNS, unplanned requests are stored in a *request bank*. In the following, we describe the operators that are used to reinsert them in an incomplete solution.

Best insertion: among all requests r in the request bank, the request with the cheapest insertion (considering insertions with and without transfer) is selected and inserted at its best position.

Regret Insertion: for each request r in the request bank and for each pair of vehicles (pickup vehicle, delivery vehicle), the cost of its cheapest feasible insertion (if any) is computed. Note that the pickup and delivery vehicles may be the same (in the case of insertion without transfer). Then, with these insertion options, the *k-regret value* of r is defined as $c_r^k = \sum_{i=1}^{k}(f_i - f_1)$, where f_1 is the cost of the cheapest insertion, f_2 is the cost of the second-cheapest insertion and so on, and k is a parameter. The cheapest insertion of the request with the highest regret value is performed.

For repair operators, two facts should be taken into account: first, because of resource constraints, only a limited number of requests will be transferred (some requests that were transferred in the VRPCD may no longer be transferred in the VRPCD-RC). Thus, there is an incentive in creating routes without transfer. Second, as detailed in Section 4.4.2, feasibility tests can no longer be performed in constant time. Thus, repair methods that check many insertions, such as k-regret with a high value of k, should be avoided. As such we use: best insertion, 2-regret, best insertion without transfer and 2-regret without transfer as repair methods. In the variants *without transfer*, only insertions without transfer are considered. This is a difference with Grangier et al. (2017) where all repairs methods were considering transfers and where 3 and 4-regret insertions were computationally affordable.

The initial solution is obtained by applying a 2-regret without transfer.

4.4 Integration of dock resource constraints in LNS

In the repair methods of LNS, we need to ensure that the insertions we consider are feasible both with respect to capacity and time-related constraints (time windows and dock resource). Capacity constraints can easily be checked in constant time. In this section we focus on how to handle dock resource constraints. To that end, we start by presenting the scheduling model associated with dock resource constraints, then the methods we propose to solve it and eventually the general structure of feasibility tests we use for maximal efficiency.

4.4.1 Scheduling problems associated with dock resource constraints

For each route $k \in K$, let a_k be its earliest feasible arrival time at the cross-dock and b_k its latest feasible departure time from the cross-dock. Given an insertion that would insert request r in route k_1 for pickup and route k_2 for delivery, we can compute the new earliest feasible arrival time at the cross-dock of k_1: a'_1, and the new latest feasible departure time of k_2: b'_2 (provided that no time windows violation occurs in the pickup leg of k_1 and in the delivery leg of k_2, in which case we could immediately reject the insertion). Thus, dock resource constraints can be seen as a satisfaction scheduling problem at the cross-dock.
For each route $k \in K$, let T_k be the set of routes that deliver at least one request picked up by k; let t_k^- and t_k^+ be its associated unloading and reloading operations respectively; and let R_k^- and R_k^+ be the sets of requests being unloaded and reloaded respectively. Let $isActive$ be an indicator function such that $isActive(o, h)$ is equal to 1 if and only if task o is being performed at instant h. Let H be the time horizon of the problem.
In the separated case the associated scheduling problem is:

$$\mathrm{t}_{k'}^+.Start \geq t_k^-.End \qquad \forall k \in K, k' \in T_k \qquad (1)$$

$$\mathrm{t}_k^+.Start \geq t_k^-.End + \delta_r \qquad \forall k \in K; s.t.\ R_k^+ \neq \emptyset \qquad (2)$$

$$\mathrm{t}_k^+.Start \geq t_k^-.End \qquad \forall k \in K; s.t.\ R_k^+ = \emptyset \qquad (3)$$

$$\mathrm{t}_k^-.Start \geq a_k + \delta_u \qquad \forall k \in K; s.t.\ R_k^- \neq \emptyset \qquad (4)$$

$$\mathrm{t}_k^-.Start \geq a_k \qquad \forall k \in K; s.t.\ R_{k'}^- = \emptyset \qquad (5)$$

$$\mathrm{t}_k^+.End \leq b_k \qquad \forall k \in K \qquad (6)$$

$$\sum_{k \in K} isActive(t_k^-, h) \leq I \qquad \forall h \in [0, H] \qquad (7)$$

$$\sum_{k \in K} isActive(t_k^+, h) \leq O \qquad \forall h \in [0, H] \qquad (8)$$

In the shared case the associated scheduling problem is:

(1 - 6)

$$\sum_{k \in K} isActive(t_k^-, h) + isActive(t_k^+, h) \leq S \qquad h \in [0, H] \qquad (9)$$

Constraints (1) ensure that all reloading operations that depend on a route k start no earlier than the end of the unloading task associated with k. Constraints (2)

and (3) ensure the delay (preparation) between the unloading and reloading task of a route k respects the model presented in Section 3.2. Constraints (4) and (5) ensure that for each route, its corresponding unloading operation cannot start before the earliest feasible arrival time at the cross-dock. Constraints (6) ensure that for each route, its corresponding reloading operation is done by its latest feasible departure time. Constraints (7 and 8) model the separated case while constraints (9) models the shared case.

Constraints on start and end of some events as well as the *isActive* function are easily modelled respectively with the concept of *IntervalVar* and *CumulExpr* from the OPL modelling language (Van Hentenryck, 1999), which can be used by IBM CP Optimizer (IBM Corporation, 2014). In particular, we recall that (from IBM Corporation (2014)):

> 'An interval variable represents an interval of time during which a task happen, and whose position in time is an unknown of the scheduling problem. An interval is characterized by a start value, an end value and a size. (...) An interval variable can be optional, that is, one can decide not to consider [it] in the solution schedule.'

4.4.2 Proposed methods for the dock resource constraints

To solve the satisfaction scheduling problems of Section 4.4.1 we propose two methods: (1) using a third party CP solver or (2) using scheduling heuristics. When repeatedly calling a third party solver we cannot neglect its overhead (model building, memory allocation of the solver, ...) potentially leading to very high run-times. Scheduling heuristics are potentially faster, we could thus perform more LNS iterations within the same time budget, but they are likely to report false negatives (stating that an insertion is unfeasible although it is actually feasible). This is a common run-time versus solution-quality trade-off situation.

The scheduling heuristics we use are list heuristics: among a list of available tasks (a task is said to be available for scheduling if all its predecessors have already been scheduled) we select one task according to a given criterion and we try to schedule it. If a feasible schedule is found, we update the resource constraints accordingly, then we update the list of available tasks to schedule. We repeat this procedure until no more tasks have to be scheduled. If at one point we cannot schedule a task, we declare this insertion unfeasible according to this scheduling heuristic. The four different selection criteria we use are listed hereafter.

First Come First Served (FCFS): the task with the earliest release date in the list of available tasks is selected.

Earliest Due Date (EDD): the task with the earliest due date in the list of available tasks is selected.

Most Successors First (MSF): the task with the largest number of successors in the list of available tasks is selected. By definition reloading tasks do not have successors, we break ties with the EDD rule.

Shortest Processing Time First (SPTF): the task with the shortest processing time in the list of available tasks is selected.

4.4.3 Checking the feasibility of an insertion in the VRPCD-RC

A necessary condition for an insertion to be feasible in the VRPCD-RC is that it is feasible in the VRPCD. As mentioned in Section 4.1, feasibility tests in the VRPCD can be performed in a constant time. On the other hand, feasibility test with respect to dock resource constraints in the VRPCD-RC cannot be done in a constant time. As a result, we test the feasibility of an insertion as shown on Fig. 2: we start by checking if it is feasible for the VRPCD, if the insertion passes this test, we test it with respect to dock resource constraints using either a CP solver or scheduling heuristics.

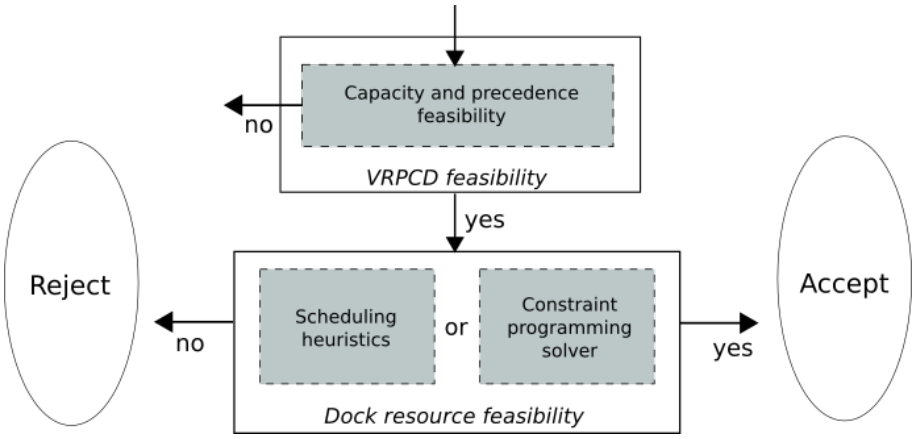


Fig. 2: Logical flow-chart of feasibility tests for the VRPCD-RC

4.5 Set Partitioning and Scheduling

Given a set of routes K, set partitioning and scheduling (SPS) aims to select a subset $\tilde{K}$ of K minimizing cost such that (1) each request is picked up and delivered by exactly one route in $\tilde{K}$ (potentially different routes for the pickup and the delivery) and (2) routes in $\tilde{K}$ can be scheduled at the cross-dock so that time constraints are respected. The overall algorithm is presented in 2. In Section 4.5.1, we present the dominance pre-processing we apply to routes (l. 1-4). The set partitioning (l. 6) is solved using a technique called *branch-and-check*, presented in Section 4.5.2. Its application to the VRPCD-RC is detailed in Section 4.5.3. The post-processing steps aimed at minimizing the volume transferred at the cross-dock in the obtained solution is presented in 4.5.3.

4.5.1 Dominance pre-processing

In the LNS, we stored both the routes and the legs in memory components. The goal of the later is to help improve the routes in K before solving the set partitioning.

Input: pool of legs $\mathcal{L}$, pool of routes $\mathcal{K}$, solution $s^\star$, s' from Algorithm 1
Remove dominated legs from $\mathcal{L}$
for *each route* $k \in \mathcal{K}$ **do**
  Replace, if possible, its pickup leg and/or its delivery leg with a non-dominated equivalent in $\mathcal{L}$
end
Remove dominated routes from $\mathcal{K}$
Solve SPS with all routes in $\mathcal{K}$
if *a new best solution has been found* **then**
  Improve the matching of legs (as in Section 4.5)
  Update $s^\star$
end
if *Set partitioning was not solved to proven optimality* **then**
  Clear $\mathcal{K}$
end

Algorithm 2: Perform SPS

For that we identify non dominated legs. A pickup (resp. delivery) leg l_i is said to be dominated by a leg l_j if and only if: l_i and l_j serve the same set of requests, $c_j < c_i$ and $a_j \leq a_i$ (resp. $b_j \geq b_i$) where c represents the cost of the leg, a represents the arrival time at the cross-dock (resp. b represents the departure time from the cross-dock). Each route in K is improved by substituting its pickup and delivery leg by their non dominated equivalent.

4.5.2 Branch-and-check

We present branch-and-check (Thorsteinsson, 2001) using the following optimization problem:

$$M1: \min c^\intercal x \tag{10}$$
$$Ax \leq b \tag{11}$$
$$H(x, y) \tag{12}$$
$$x \in \{0, 1\}^n \tag{13}$$
$$y \in \mathbb{R}^m \tag{14}$$

Assume that $H(x, y)$ represents a set of constraints that have a limited impact on the LP relaxation and/or are difficult to efficiently model in a MIP, but that can be handled relatively easily by a CP solver. (10), (11) and (13) form a relaxation (M2) of (M1) that can be solved using branch-and-bound. The general principle of branch-and-check is the following. To solve (M1), a branch-and-bound is carried on (M2). Whenever an integral solution of (M2) is found, a CP solver is called to check constraints (12). If they are satisfied, the best solution found so far for (M1) is updated accordingly. Otherwise, this solution is rejected. In both cases the branch-and-bound process continues.

4.5.3 Application of branch-and-check to the VRPCD-RC

For the SPS in the VRPCD-RC, a classical set partitioning problem (SPP) is used as relaxation M2. For each request $r \in R$ and each route $k \in K$, let $\lambda^+_{r,k}$ (resp. $\lambda^-_{r,k}$)

be a binary constant which is equal to 1 if and only if this request is picked up (resp. delivered) by this route. And, for each route, let x_r be a Boolean variable that indicate whether this route is selected. The SPP on routes is then :

$$\min \sum_{k \in K} c_k x_k \tag{15}$$

$$\sum_{k \in K} \lambda^+_{r,k} x_k = 1 \qquad \forall r \in R \tag{16}$$

$$\sum_{k \in K} \lambda^-_{r,k} x_k = 1 \qquad \forall r \in R \tag{17}$$

$$x_k \in \{0,1\} \qquad \forall k \in K \tag{18}$$

The objective (15) is to minimize the cost of the selected routes while constraints (16) (resp. (17)) ensure that each pickup point (resp. delivery point) is covered by exactly one routes.

A solution to the SPP on routes is a solution to the VRPCD-RC if and only if it is possible to find a schedule at the cross-dock such that time related constraints are respected. This is the problem presented in Section 4.4.1. In Grangier et al. (2017), on top of checking synchronization constraints, the set partitioning component was also assembling routes from legs (a process we called *matching*). Preliminary tests showed that, very often, the CP solver could not find any solution within reasonable run-times when asked to perform the matching with the extra resource constraints. Because the set partitioning idea proved very efficient in Grangier et al. (2017), we resort to keeping it but simplifying it to only do the scheduling part.

4.5.4 Improving legs pairing

The SPS approach is efficient to select the best routes, but it may still be possible to reduce the number of transfers at the cross-dock by improving the matching of legs to form routes in the best found solution. This can free processing capacity at the cross-dock and further help the search in subsequent steps. The solution obtained from the SPS is made of a set of pickup legs denoted $\tilde{L_p}$ and a set of delivery legs denoted $\tilde{L_d}$. For each pickup leg $l \in \tilde{L_p}$, let T_l be the set of delivery legs that deliver at least one request picked up by l. If a pickup leg l and a delivery leg l' are matched together to create a route, there exist an associated unloading task $o^-_{ll'}$, with a set of requests $R^-_{ll'}$ being unloaded, and a reloading task $o^+_{ll'}$, with a set of requests $R^+_{ll'}$ being reloaded. These tasks have to be performed if and only if l and l' are in the same route.

Once again, the scheduling related concept used in CP can help with the modelling. Here, we each for each leg, exactly one of the previously described unloading (resp. reloading) task should be performed. This corresponds to alternative activities (Beck and Fox, 1999) and can be modelled using alternative constraints (from IBM Corporation (2014)):

> 'An alternative constraint between an interval variable a and a set of interval variables $b_1, \ldots, b_n$ models an exclusive alternative between $b_1, \ldots, b_n$. If interval a is present, then exactly one of intervals $b_1, \ldots, b_n$ is present and a starts and ends together with this specific interval. Interval a is absent if and only if all intervals in $b_1, \ldots, b_n$ are absent.'

In the shared case, the post-processing is performed by solving the following CP model:

$$\min \sum_{(l,l')\in \tilde{L_p}\times \tilde{L_d}} o^-_{ll'}.IsPresent \times \sum_{r\in R^-_{l,l'}} q_r \tag{19}$$

$$\text{Alternative}(t_l, \{o^-_{ll'}; \forall l' \in \tilde{L_d}\}) \qquad \forall l \in \tilde{L_p} \tag{20}$$

$$\text{Alternative}(t_{l'}, \{o^+_{ll'}; \forall l \in \tilde{L_p}\}) \qquad \forall l' \in \tilde{L_d} \tag{21}$$

$$\text{o}^-_{ll'}.IsOptional \leftarrow True \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \tag{22}$$

$$\text{o}^+_{ll'}.IsOptional \leftarrow True \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \tag{23}$$

$$\text{o}^-_{ll'}.IsPresent \iff o^+_{ll'}.IsPresent \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \tag{24}$$

$$\text{t}_{l'}.Start \geq t_l.End \qquad \forall l \in \tilde{L_p}, l' \in T_l \tag{25}$$

$$\text{o}^+_{ll'}.Start \geq o^-_{ll'}.End + \delta_r \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \; s.t. \; R^+_{ll'} \neq \emptyset \tag{26}$$

$$\text{o}^+_{ll'}.Start \geq o^-_{ll'}.End \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \; s.t. \; R^+_{ll'} = \emptyset \tag{27}$$

$$\text{o}^-_{ll'}.Start \geq a_l + \delta_u \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \; s.t. \; R^-_{ll'} \neq \emptyset \tag{28}$$

$$\text{o}^-_{ll'}.Start \geq a_l \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \; s.t. \; R^-_{ll'} = \emptyset \tag{29}$$

$$\text{o}^+_{ll'}.End \leq b'_l \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \tag{30}$$

$$\sum_{l\in L_p} isActive(t_l, h) \leq I \qquad \forall h \in [0, H] \tag{31}$$

$$\sum_{l'\in L_d} isActive(t'_l, h) \leq O \qquad \forall h \in [0, H] \tag{32}$$

In this model, for each pickup leg l, t_l is an interval variable that represents the associated unloading task that takes place at the cross dock. (19) minimizes the volume transferred at the cross-dock, while constraints (31). Alternative constraints (20) and (22) ensure that for each pickup leg l exactly one unloading task $o_{ll'}$ is scheduled and that it is equal to t_l. The same holds for delivery legs and reloading operations through variables $t_{l'}$ and constraints (21) and (23). Constraints (24) ensure that the unloading operation associated with the matching of pickup leg l and the delivery leg l' in the same vehicle is present if and only if the corresponding reloading operation is present as well. Constraints (25) ensure that all the reloading operations that depend on a pickup leg l start no earlier than the end of the unloading task associated with l. Constraints (26) and (27) ensure that when two legs are packed together, the delay between the two tasks respects the model presented in Section 3.2. Constraints (28) and (29) ensure that for each pickup leg, its corresponding unloading operation cannot start before the earliest feasible arrival time at the cross-dock. Constraints (30) ensure that for each delivery leg, its corresponding reloading operation is done by its latest feasible departure time. Constraints (32- 31) account for the capacity constraints. A similar problem for the separated case can be formulated with an adaptation of (9). With the exception of the capacity constraints, the model is taken from Grangier et al. (2017).

5 Computational experiments

The algorithm is coded in C++ and uses CPLEX and CP Optimizer from IBM ILOG Cplex Optimization Studio 12.6.1 as MIP solver and CP solver, respectively. The experiments were conducted under Linux using an Intel Xeon X7350 @ 2.93 GHz. Only one core is used both by our code and third party solvers. We consider instances proposed by Wen et al. (2008), that range from 50 to 200 requests. They are based on real life data from a Danish logistics company. The termination criterion for all algorithms is based on time: 15 minutes for size 50 instances, 30 minutes for size 100 for instances, 60 minutes for size 150 instances and 120 minutes for size 200 instances. SPS time limit is 180 seconds. These time limits are between two and three times the run-times reported for the VRPCD in Grangier et al. (2017). As in Grangier et al. (2017), the number Φ of requests to remove in the repair phase of the LNS is drawn randomly in the interval [min(30, 10% of $|R|$), max(60, 20% of $|R|$)], acceptance criterion is descent.

5.1 Bound setting for the number of docks

To determine when limiting the number of docks start being a constraint in the VRPCD-RC, we post-processed the solutions obtained in Grangier et al. (2017) for the VRPCD. To that end, for all the ten solutions found for each instance for the VRPCD, we solve optimization versions of the satisfaction scheduling problems introduced in Section 4.4.1. We take as objective: to minimize S in the shared case, and in the separated case, we only consider symmetric configurations where $I = O$ and we minimize I. In Table 1, columns A correspond to the smallest value obtained after two hours of run-time for the CP solver, and columns B correspond to the worst value obtained after five minutes of post-process. For each instance, columns B corresponds to a threshold for the dock value, above which, the VRPCD-RC could be solved as a VRPCD (post-processing the solution for a limited amount of time to satisfy dock resource constraints). As such, in our experiments for the VRPCD-RC, we test dock values up to those reported in columns B.
From this table, we can observe that dock resource constraints arise for dock values that corresponds to approximately 15% of the fleet size in the shared case and 10% of the fleet size in the separated case.

5.2 Parameters tuning

In this section we evaluate and adjust several parameters. We first give some insights on the impact of parameters. We start with the time limit for the CP solver in LNS feasibility tests (called *CP time limit*), then we report the success rate of heuristics. After, we present the influence of the SPS frequency on the quality of solutions. Eventually, we compare the performance of four possible configurations: with CP solver tests/with heuristic tests in LNS, with/without SPS. For tuning, we use instances 50b, 100b, 150b, 200b, and we consider the shared cross-dock configuration case. Parameter tuning has been performed taking an overall time limit for LNS or LNS+SPS (called *LNS time limit*) and comparing the final average cost value over these representative instances.

Instance	Avg. fleet size	Separated		Shared	
		A	B	A	B
50a	14.2	2	2	2	2
50b	16.1	2	2	2	2
50c	16.0	2	2	2	3
50d	15.0	2	2	2	2
50e	16.0	2	2	2	3
100b	31.0	3	3	4	4
100c	31.5	3	3	4	4
100d	29.2	3	3	4	4
100e	32.0	3	3	4	5
150a	45.4	4	5	5	7
150b	46.9	4	5	6	7
150c	45.9	4	5	6	7
150d	45.0	4	5	6	7
150e	46.0	4	5	6	7
200a	62.9	6	7	8	9
200b	62.0	6	6	8	9
200c	61.1	6	7	8	9
200d	62.0	6	7	8	9
200e	62.0	6	7	8	10

Table 1: Dock value obtained when post-processing for each instance all of ten solutions of Grangier et al. (2017). Columns A refer to the best solution (min dock use) obtained after two hours, while columns B correspond to the worst value (max dock use) obtained after five minutes of post-processing.

5.2.1 CP solver time limit in feasibility tests

When checking the feasibility with respect to dock resource constraints, we need to set a time limit after which the CP solver will stop searching and declare the insertion unfeasible. This so-called CP time limit prevents spending a large amount of time checking the feasibility of a single insertion. Four values have been tested and the final result was that a CP-time limit of 0.01 seconds yields better results. To understand this observation, Table 2 provides some insights on the average percentage of unsolved feasibility problems for each evaluated CP time limit.

CP Time Limit (s)	1	0.1	0.01	0.001
Run-time	1	0.33	0.12	0.10
Time limit hit (%)	0.5	12.9	20.1	27.3

Table 2: Comparison of four different time limits for the CP solver, when used as feasibility test in the shared case. Run-time is normalized with 1 representing the run-time for a CP solver time limit of 1s. *Time limit hit* represents the percentage of calls for which the CP solver could not find an answer within the time limit. Figures reported for one thousand LNS iterations, two runs were performed in each case.

5.2.2 Insights on heuristics performance

As mentioned in Section 4.4.2, using CP versus using heuristics for feasibility tests is a quality/run-time trade-off. For each instance in the training set, we performed two runs with LNS, with the stop-criterion set to one thousand iterations. We count how many time insertions were reported feasible by heuristics and how many times they were reported feasible by the CP solver with a time limit of 1 second (according to Table 2, with this CP time limit we can consider that the CP solver give an accurate answer most of the time). On average, 71.1% of feasible insertions are reported

feasible by heuristics. Performing one thousand LNS iterations with feasibility tests performed by heuristics takes only 18.2% of the time taken with CP solver tests (with 0.01s as CP time limit).

5.2.3 Insights on SPS frequency

In Grangier et al. (2017), the SPM was solved twenty times per run (every thousand iterations with a stop-criterion of twenty thousands iterations). In the VRPCD-RC, because feasibility tests are computationally intensive, the stop-criterion is based on time. The number of iterations performed within the LNS time limit depends not only on the size of the instance but also on the dock value. As such, we propose a SPS-criterion based on time. In Table 3 and Table 4, we compare three different settings for the SPS frequency : 10, 20 and 40 calls within the LNS time limit. Accordingly, the number of calls per run is set to 20 for heuristic test and 10 for CP solver test.

Number of calls	10	20	40
Average gap (%)	0	-0.70	-0.29

Table 3: Comparison of the impact of the number of SPS calls per run on the quality of the solution for heuristic feasibility tests. Five runs were performed for each dock value for each instance in the training set. *10 calls* is taken as reference for the gap

Number of calls	10	20	40
Average gap (%)	0	+0.40	+0.68

Table 4: Comparison of the impact of the number of SPS calls per run on the quality of the solution for CP solver tests. Five runs were performed for each dock value for each instance in the training set. *10 calls* is taken as reference for the gap

5.2.4 Performance comparison

On Fig. 3 we present the convergence curves of LNS and LNS+SPS over time for both CP solver and heuristics feasibility tests. From this graph, we can observe two things. First, as in the VRPCD, periodically solving a set partitioning based problem significantly improves performance compared to LNS alone: -6.96% for heuristic test and -7.92% for CP solver test on average. Second, the best performing method is LNS+SPS with heuristic feasibility tests, as it finds solutions that are 1.19% better on average than LNS+SPS with CP solver test. In the rest, we thus use LNS+SPS with heuristic feasibility test.

5.3 Results

Table 5 presents a synthesis of the results in both the shared case and separated cases respectively (detailed results are presented in Appendix A). A dock value of 0

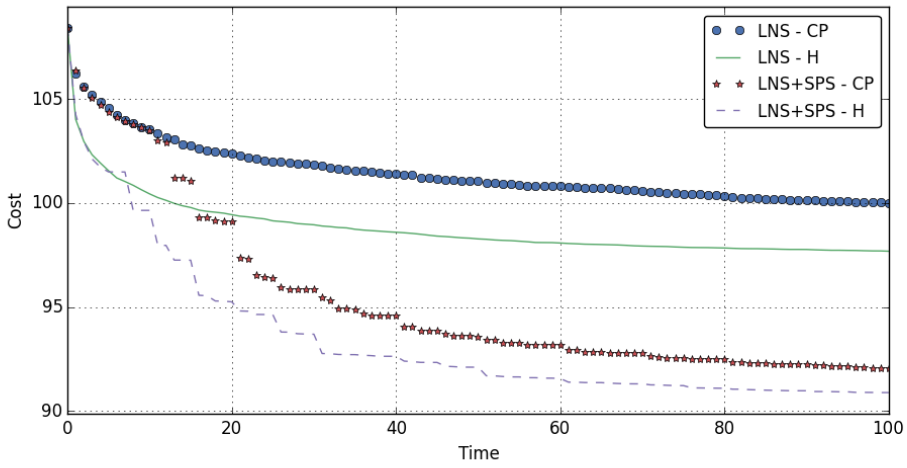


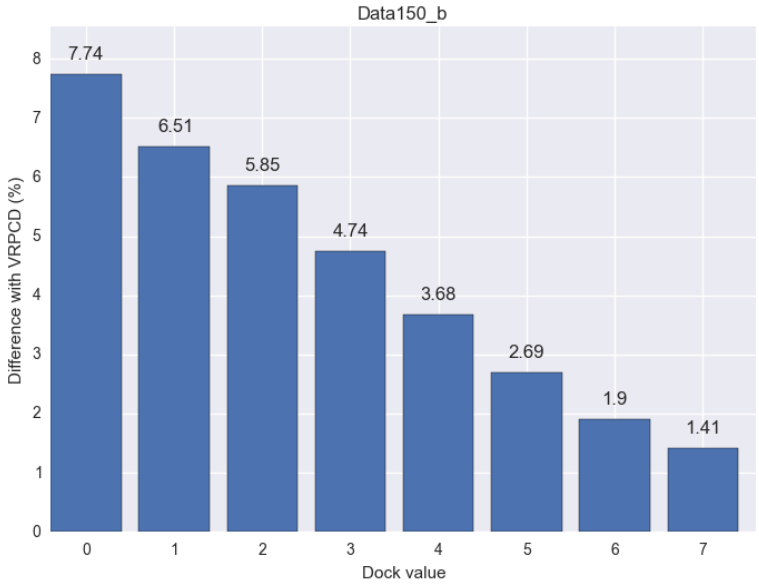
Fig. 3: Comparison of the evolution of the average solution quality over time (in percentage of LNS time limit) for four different configurations: with CP solver/with heuristics tests, with/without SPS. The results aggregate 5 runs for each dock value in the shared case for instances in the training set. They have been normalized, first by instance then by method, with 100 representing the cost at the end of LNS with CP solver tests (LNS-CP)

corresponds to the case where no transfer is allowed, i.e. vehicles have to go back to the cross-dock between pickup and delivery legs but cannot transfer any items. We present this value mostly for comparison purposes. The algorithm shows relatively good performance in terms of stability: the difference between the average value and the best value of the five runs for each instance and each dock value is 0.6% on average and at most 1.4% in the shared case and 0.6% on average and at most 3.2% in the separated case.

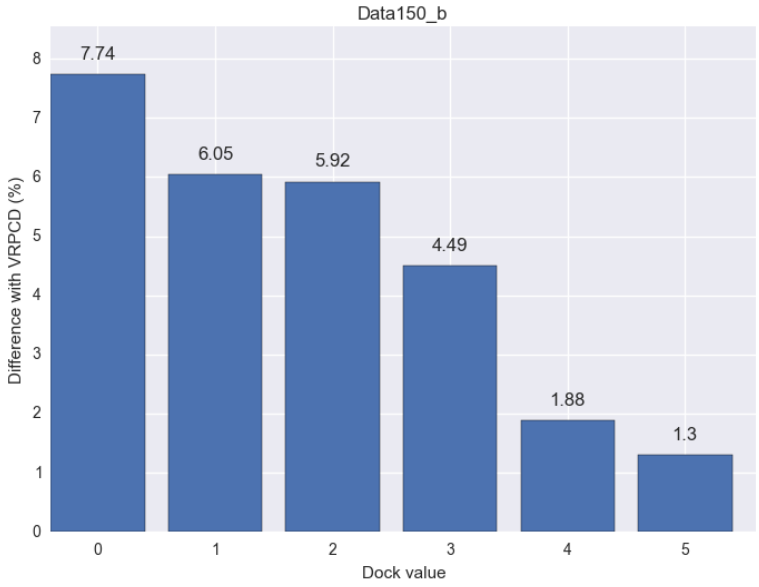
Regarding routing costs: integrating dock resource constraints implies an increase in cost. Comparing the two systems, the shared case costs are slightly smaller than separated costs (e.g, for instance 150b, 5.85% for 2 shared docks compared to 6.05 as shown on Fig. 4). These differences increases with the number of docks.

6 Conclusion

This paper presents an adaptation of the matheuristic proposed in Grangier et al. (2017), which is based on large neighborhood search and periodic calls to a set partitioning based problem, to solve an extension of the VRPCD that includes resource synchronization constraints at the cross-dock. To deal with these constraints, scheduling heuristics and a CP model have been used as feasibility tests for insertions in LNS. In this case, experiments have showed that heuristics are the most efficient compromise in terms of run-time versus solution-quality. The proposed method has



(a) *Shared* cross-dock configuration



(b) *Separated* cross-dock configuration

Fig. 4: Influence of the dock value for instance 150b. Five runs were performed for each dock value. The y-axis represents the average gap with respect to the average value reported in Grangier et al. (2017)

Instance	Docks	Shared		Separated	
		Average	Best	Average	Best
50	0	8.2	8.6	8.2	8.6
	1	3.3	2.9	2.0	1.8
	2	1.3	1.1	0.8	0.8
100	0	8.5	8.6	8.5	8.6
	1	6.2	5.9	5.6	5.2
	2	5.3	4.7	4.7	3.7
	3	3.8	3.1	2.4	1.8
	4	2.4	2.0	-	-
	5	2.3*	1.0*	-	-
150	0	7.8	7.9	7.8	7.9
	1	6.5	6.1	6.4	6.1
	2	6.1	5.6	6.4	5.2
	3	5.1	4.6	4.7	3.7
	4	4.4	4.1	2.9	2.5
	5	3.3	2.9	1.6	1.3
	6	2.2	2.0	-	-
	7	1.7	1.5	-	-
200	0	7.5	7.7	7.5	7.7
	1	6.9	6.7	6.4	6.4
	2	6.3	5.9	5.8	5.4
	3	5.7	5.3	4.9	4.7
	4	5.4	5.1	4.3	4.0
	5	4.9	4.4	3.1	3.0
	6	4.5	4.2	1.8	1.7
	7	3.6	3.4	1.5	1.5
	8	2.7	2.5	-	-
	9	2.0	1.8	-	-
	9	2.2*	2.1*	-	-

Table 5: Gaps (in %) in the shared and separated cross-dock configuration cases with respect to the average and best values reported in Grangier et al. (2017) for the VRPCD. A * indicates that not all instances had to be tested on such cross-dock values, as the best non dock-resource constrained solution did not require that many cross-docks. Detailed results per instances are presented in Tables 6 and 7

been tested on instances from the literature, where it shows an increase in routing costs with the decrease in cross-dock capacity.
Adding resource synchronization constraints has made the problem computationally very challenging: several decision had to be taken (heuristics as feasibility tests, simpler set partitioning component) to limit the run-time. This shows that integrated problems are challenging even for heuristics. From the application point of view, several applications and extension of the proposed method could be envisioned in the field of electric vehicle routing (Schneider et al., 2014; Froger et al., 2017) and city logistics (Grangier et al., 2016).

Acknowledgments

This work was partially supported by the Canadian Natural Science and Engineering Research Council (RGPIN-2015-04696) and by the Fonds de recherche du Québec - Nature et technologies through its Team research Program.

Appendix A Detailed results

Table 6 and Table 7 present the detailed results obtained for each instance. These results highlight the increased complexity induced by integrating dock resource constraints, as there exists for each instance, at maximum dock value, a solution with 0% gap (see Section 5.1). Nevertheless LNS+SPS finds solutions that are 1.6% more expensive in the shared case and 1.5% more expensive in the separated case, which remains satisfactory.

Instance	Dock	Average		Best	
		Value	Gap (%)	Value	Gap (%)
50a	0	6882.4	6.5	6871.93	6.4
	1	6682.72	3.4	6628.05	2.7
	2	6530.81	1.0	6471.48	0.2
50b	0	8027.87	8.1	8021.81	9.6
	1	7609.07	2.4	7541.33	3.0
	2	7481.99	0.7	7470.78	2.0
50c	0	7857.77	7.3	7827.67	7.1
	1	7505.34	2.5	7473.94	2.2
	2	7449.88	1.8	7397.38	1.2
	3	7338.88	0.3	7330.08	0.3
50d	0	7760.32	10.2	7760.11	10.4
	1	7272.57	3.3	7206.48	2.5
	2	7108.91	1.0	7076.13	0.7
50e	0	8157.77	9.1	8156.94	9.4
	1	7843.08	4.9	7772.39	4.3
	2	7616.54	1.8	7554.75	1.4
	3	7537.85	0.8	7473.56	0.3
100b	0	15636.26	8.8	15628.7	8.9
	1	15322.86	6.6	15234.2	6.2
	2	15181.04	5.6	15073.8	5.0
	3	14929.18	3.8	14810.3	3.2
	4	14700.96	2.3	14620.3	1.9
100c	0	14915.92	7.9	14915.6	8.2
	1	14654.16	6.0	14611.5	6.0
	2	14456.6	4.5	14395.0	4.4
	3	14353.9	3.8	14188.2	2.9
	4	14145.78	2.3	14081.3	2.2
100d	0	14860.46	9.3	14832.5	9.2
	1	14424.74	6.1	14338.9	5.6
	2	14316.7	5.3	14207.4	4.6
	3	14070.18	3.5	13976.2	2.9
	4	13866.24	2.0	13774.7	1.5
100e	0	15095.92	8.1	15091.5	8.2
	1	14819.26	6.2	14733.8	5.7
	2	14783.82	5.9	14622.8	4.9
	3	14515.0	4.0	14384.7	3.2
	4	14357.26	2.9	14258.4	2.3
	5	14281.0	2.3	14085.3	1.0
150a	0	20859.22	7.5	20807.7	7.5
	1	20534.8	5.8	20406.4	5.4
	2	20505.68	5.7	20342.5	5.1
	3	20248.54	4.4	20090.1	3.8
	4	20110.14	3.7	19988.6	3.3
	5	19822.08	2.2	19740.0	2.0
	6	19725.04	1.7	19629.7	1.4
	7	19603.78	1.0	19541.6	0.9
150b	0	22272.64	7.7	22236.5	8.0
	1	22018.68	6.5	21934.5	6.6
	2	21882.36	5.9	21825.0	6.0
	3	21651.48	4.7	21585.3	4.9
	4	21432.16	3.7	21360.2	3.8
	5	21227.66	2.7	21097.6	2.5
	6	21065.9	1.9	20986.2	2.0
	7	20963.44	1.4	20899.9	1.5
150c	0	21313.88	7.8	21295.7	8.0
	1	21095.74	6.7	20959.6	6.2
	2	20945.68	5.9	20854.4	5.7
	3	20869.2	5.5	20642.1	4.6
	4	20685.7	4.6	20559.7	4.2
	5	20475.68	3.6	20369.1	3.3
	6	20208.76	2.2	20155.8	2.2
	7	20153.68	1.9	20100.9	1.9
150d	0	21962.58	7.9	21951.3	8.0
	1	21735.46	6.8	21536.0	6.0
	2	21609.34	6.2	21442.6	5.5
	3	21448.08	5.4	21368.0	5.2
	4	21504.76	5.6	21289.8	4.8
	5	21040.52	3.4	20874.7	2.7
	6	20806.38	2.2	20642.2	1.6
	7	20715.78	1.8	20608.5	1.4

Instance	Dock	Average		Best	
		Value	Gap (%)	Value	Gap (%)
150e	0	21036.12	7.9	21019.3	8.1
	1	20827.96	6.8	20703.5	6.4
	2	20774.68	6.6	20587.3	5.9
	3	20599.16	5.7	20320.2	4.5
	4	20398.18	4.6	20341.0	4.6
	5	20346.68	4.4	20245.7	4.1
	6	20073.92	3.0	20025.5	3.0
	7	19915.4	2.2	19773.1	1.7
200a	0	28779.68	7.1	28760.6	7.2
	1	28507.12	6.1	28423.1	6.0
	2	28516.14	6.2	28263.0	5.4
	3	28273.0	5.2	28159.1	5.0
	4	28323.38	5.4	28146.5	5.0
	5	28151.24	4.8	28054.8	4.6
	6	28099.32	4.6	27959.8	4.3
	7	27922.8	3.9	27782.5	3.6
	8	27704.08	3.1	27527.8	2.7
	9	27497.36	2.4	27391.5	2.1
200b	0	29168.18	6.9	29082.8	6.9
	1	28962.52	6.1	28771.7	5.7
	2	28809.76	5.5	28751.4	5.6
	3	28784.88	5.5	28514.1	4.8
	4	28589.7	4.7	28546.4	4.9
	5	28374.36	4.0	28186.6	3.6
	6	28446.44	4.2	28234.3	3.7
	7	28156.7	3.2	28091.2	3.2
	8	27889.54	2.2	27677.2	1.7
	9	27630.92	1.2	27516.8	1.1
200c	0	28153.28	7.9	28119.3	8.5
	1	27996.14	7.3	27819.7	7.3
	2	27744.12	6.4	27555.7	6.3
	3	27629.96	5.9	27381.3	5.6
	4	27534.98	5.5	27207.7	4.9
	5	27522.48	5.5	27315.8	5.4
	6	27236.38	4.4	27135.5	4.7
	7	26921.14	3.2	26820.2	3.4
	8	26685.4	2.3	26570.1	2.5
	9	26583.44	1.9	26477.1	2.1
200d	0	29463.02	7.6	29446.8	7.8
	1	29328.88	7.1	29227.0	6.9
	2	29126.9	6.3	28993.7	6.1
	3	28962.52	5.7	28825.7	5.5
	4	28866.58	5.4	28776.3	5.3
	5	28777.12	5.0	28523.2	4.4
	6	28708.02	4.8	28546.4	4.5
	7	28467.38	3.9	28283.7	3.5
	8	28208.72	3.0	28129.6	2.9
	9	28002.7	2.2	27877.6	2.0
200e	0	28225.92	8.1	28139.3	8.0
	1	28106.28	7.7	28001.5	7.4
	2	27935.84	7.0	27659.2	6.1
	3	27789.42	6.4	27566.0	5.8
	4	27610.22	5.8	27432.2	5.3
	5	27421.46	5.0	27113.3	4.0
	6	27324.54	4.7	27067.2	3.9
	7	27061.8	3.7	26967.5	3.5
	8	26858.48	2.9	26772.0	2.7
	9	26665.24	2.1	26543.9	1.8
	10	26677.62	2.2	26600.4	2.1

Table 6: Average values and best solution found in the shared cross-dock configuration case; LNS+SPS was run five times for each instance. Columns *Gap* refer to the gap to average values and best solutions reported in Grangier et al. (2017) for the VRPCD

Instance	Dock	Average		Best	
		Value	Gap (%)	Value	Gap (%)
50a	0	6882.4	6.5	6871.93	6.4
	1	6614.32	2.3	6554.55	1.5
	2	6574.13	1.7	6545.34	1.4
50b	0	8027.87	8.1	8021.81	9.6
	1	7520.61	1.3	7496.96	2.4
	2	7462.01	0.5	7451.21	1.8
50c	0	7857.77	7.3	7827.67	7.1
	1	7428.2	1.5	7385.09	1.0
	2	7359.79	0.5	7335.97	0.3
50d	0	7760.32	10.2	7760.11	10.4
	1	7169.38	1.8	7127.77	1.4
	2	7078.23	0.5	7054.87	0.4
50e	0	8157.77	9.1	8156.94	9.4
	1	7705.03	3.0	7649.99	2.6
	2	7546.88	0.9	7478.02	0.3
100b	0	15636.26	8.8	15628.7	8.9
	1	15184.1	5.6	15088.4	5.1
	2	15163.32	5.5	14887.9	3.8
	3	14718.0	2.4	14618.6	1.9
100c	0	14915.92	7.9	14915.6	8.2
	1	14484.24	4.7	14360.5	4.2
	2	14470.17	4.6	14313.7	3.8
	3	14130.18	2.2	14029.0	1.8
100d	0	14860.46	9.3	14832.5	9.2
	1	14413.18	6.0	14331.3	5.6
	2	14120.35	3.8	14036.7	3.4
	3	13911.1	2.3	13762.2	1.4
100e	0	15095.92	8.1	15091.5	8.2
	1	14801.8	6.0	14736.4	5.7
	2	14643.54	4.9	14459.3	3.7
	3	14329.12	2.7	14251.0	2.2
150a	0	20859.22	7.5	20807.7	7.5
	1	20482.68	5.6	20397.9	5.4
	2	20430.4	5.3	20298.3	4.9
	3	20161.44	3.9	19866.7	2.6
	4	19818.47	2.1	19683.2	1.7
	5	19709.34	1.6	19599.3	1.2
150b	0	22272.64	7.7	22236.5	8.0
	1	21922.38	6.0	21770.5	5.8
	2	21896.35	5.9	21733.3	5.6
	3	21600.98	4.5	21422.4	4.1
	4	21060.7	1.9	20957.2	1.8
	5	20940.14	1.3	20877.8	1.4
150c	0	21313.88	7.8	21295.7	8.0
	1	21060.12	6.5	20986.5	6.4
	2	21057.4	6.5	20653.7	4.7
	3	20707.24	4.7	20594.7	4.4
	4	20431.26	3.3	20299.3	2.9
	5	20071.94	1.5	19936.7	1.1
150d	0	21962.58	7.9	21951.3	8.0
	1	21714.83	6.7	21549.1	6.1
	2	22019.32	8.2	21327.4	5.0
	3	21424.15	5.2	20967.3	3.2
	4	20964.9	3.0	20784.2	2.3
	5	20666.2	1.5	20553.8	1.2
150e	0	21036.12	7.9	21019.3	8.1
	1	20888.67	7.2	20774.5	6.8
	2	20722.35	6.3	20625.8	6.0
	3	20551.0	5.4	20280.7	4.3
	4	20347.46	4.4	20179.9	3.8
	5	19884.9	2.0	19789.0	1.7
200a	0	28779.68	7.1	28760.6	7.2
	1	28530.67	6.2	28419.3	6.0
	2	28435.3	5.9	28267.4	5.4
	3	28173.58	4.9	28063.0	4.6
	4	27994.62	4.2	27881.1	4.0
	5	27702.8	3.1	27555.8	2.8
	6	27458.1	2.2	27333.0	1.9
	7	27259.12	1.5	27177.7	1.3
200b	0	29168.18	6.9	29082.8	6.9
	1	28890.62	5.8	28695.7	5.4
	2	28784.8	5.5	28526.9	4.8
	3	28450.8	4.2	28329.2	4.1
	4	28200.94	3.3	28054.6	3.1
	5	28049.1	2.8	27940.8	2.7
	6	27653.76	1.3	27581.9	1.3
200c	0	28153.28	7.9	28119.3	8.5
	1	27735.95	6.3	27703.7	6.9
	2	27656.33	6.0	27451.2	5.9
	3	27435.67	5.2	27320.1	5.4
	4	27277.44	4.6	27083.6	4.5
	5	26846.82	2.9	26776.0	3.3
	6	26521.98	1.7	26387.4	1.8
	7	26362.82	1.1	26263.4	1.3
200d	0	29463.02	7.6	29446.8	7.8
	1	29172.35	6.5	29013.9	6.2
	2	28888.03	5.5	28775.9	5.3
	3	28907.9	5.5	28754.6	5.2
	4	28778.5	5.1	28633.4	4.8
	5	28403.72	3.7	28294.1	3.5
	6	27974.1	2.1	27846.6	1.9
	7	27762.08	1.3	27654.5	1.2
200e	0	28225.92	8.1	28139.3	8.0
	1	28046.3	7.4	27972.5	7.3
	2	27683.53	6.0	27555.4	5.7
	3	27360.9	4.8	27186.0	4.3
	4	27229.36	4.3	27025.6	3.7
	5	26943.35	3.2	26823.2	2.9
	6	26593.2	1.9	26530.5	1.8
	7	26687.88	2.2	26593.6	2.0

Table 7: Average values and best solution found in the separated cross-dock configuration case; LNS+SPS was run five times for each instance. Columns *Gap* refer to the gap to average values and best solutions reported in Grangier et al. (2017) for the VRPCD

References

Agustina D, Lee CKM, Piplani R (2010) A review: Mathematical models for cross docking planning. International Journal of Engineering Business Management 2(2):47–54

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

Archetti C, Speranza MG (2014) A survey on matheuristics for routing problem. EURO Journal on Computational Optimization 2:223–246, DOI 10.1016/j.cor.2014.06.008

Archetti C, Speranza MG, Savelsbergh MWP (2008) An optimization-based heuristic for the split delivery vehicle routing problem. Transportation Science 42(1):22–31, DOI 10.1287/trsc.1070.0204

Asbach L, Dorndorf U, Pesch E (2009) Analysis, modeling and solution of the concrete delivery problem. European Journal of Operational Research 193(3):820–835, DOI 10.1016/j.ejor.2007.11.011, URL http://dx.doi.org/10.1016/j.ejor.2007.11.011

Beck JC, Fox MS (1999) Scheduling alternative activities. Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence pp 680–687

Boysen N, Fliedner M (2010) Cross dock scheduling: Classification, literature review and research agenda. Omega 38(6):413–422, DOI 10.1016/j.omega.2009.10.008, URL http://linkinghub.elsevier.com/retrieve/pii/S0305048309000772

Buijs P, Vis IF, Carlo HJ (2014) Synchronization in cross-docking networks: A research classification and framework. European Journal of Operational Research 239(3):593–608, DOI 10.1016/j.ejor.2014.03.012, URL http://alexandria.tue.nl/extra2/766077.pdfhttp://www.sciencedirect.com/science/article/pii/S0377221714002264

Cortés CE, Matamala M, Contardo C (2010) The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. European Journal of Operational Research 200(3):711–724, DOI 10.1016/j.ejor.2009.01.022, URL http://linkinghub.elsevier.com/retrieve/pii/S0377221709000356

Dondo R, Cerdá J (2014) A monolithic approach to vehicle routing and operations scheduling of a cross-dock system with multiple dock doors. Computers & Chemical Engineering 63:184–205, DOI 10.1016/j.compchemeng.2013.12.012, URL http://linkinghub.elsevier.com/retrieve/pii/S0098135413003931

Drexl M (2012) Synchronization in vehicle routing–A survey of VRPs with multiple synchronization constraints. Transportation Science 46(3):297–316, DOI 10.1287/trsc.1110.0400, URL http://transci.journal.informs.org/cgi/doi/10.1287/trsc.1110.0400

Ebben M, van der Heijden M, van Harten A (2005) Dynamic transport scheduling under multiple resource constraints. European Journal of Operational Research 167(2):320–335, DOI 10.1016/j.ejor.2004.03.020, URL http://linkinghub.elsevier.com/retrieve/pii/S0377221704002802

El Hachemi N, Gendreau M, Rousseau LM (2010) A hybrid constraint programming approach to the log-truck scheduling problem. Annals of Operations Research 184(1):163–178, DOI 10.1007/s10479-010-0698-x, URL http://link.springer.

com/10.1007/s10479-010-0698-x

El Hachemi N, Gendreau M, Rousseau LM (2013) A heuristic to solve the synchronized log-truck scheduling problem. Computers and Operations Research 40(3):666–673, DOI 10.1016/j.cor.2011.02.002, URL http://dx.doi.org/10.1016/j.cor.2011.02.002

Enderer F, Contardo C, Contreras I (2017) Integrating dock-door assignment and vehicle routing with cross-docking. Computers and Operations Research

Froger A, Mendoza JE, Jabali O, Laporte G (2017) A Matheuristic for the Electric Vehicle Routing Problem with Capacitated Charging Stations. Research report, Centre interuniversitaire de recherche sur les reseaux d'entreprise, la logistique et le transport (CIRRELT)

Grangier P, Gendreau M, Lehuédé F, Rousseau LM (2016) An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. European Journal of Operational Research 254(1):80–91

Grangier P, Gendreau M, Lehuédé F, Rousseau LM (2017) A matheuristic based on large neighborhood search for the vehicle routing problem with cross-docking. Computers & Operations Research 84:116–126

Grimault A, Lehuédé F, Bostel N (2014) A two-phase heuristic for full truckload routing and scheduling with split delivery and resource synchronization in public works. 2014 International Conference on Logistics Operations Management pp 57–61, DOI 10.1109/GOL.2014.6887418, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6887418

Grimault A, Bostel N, Lehuédé F (2017) An adaptive large neighborhood search for the full truckload pickup and delivery problem with resource synchronization. Computers & Operations Research 88:1–14

Gronhaug R, Christiansen M, Desaulniers G, Desrosiers J (2010) A Branch-and-Price Method for a Liquefied Natural Gas Inventory Routing Problem. Transportation Science 44(3):400–415, DOI 10.1287/trsc.1100.0317

Guastaroba G, Speranza MG, Vigo D (2016) Intermediate facilities in freight transportation planning : a survey. Transportation Science 50(3):763–789

Hempsch C, Irnich S (2008) Vehicle routing problems with inter-tour resource constraints. In: Golden B, Raghavan S, Wasil E (eds) The Vehicle Routing Problem : Latest Advances and New Challenges, Operations Research/Computer Science Interfaces, vol 43, Springer US, Boston, MA, pp 421–444, DOI 10.1007/978-0-387-77778-8, URL http://link.springer.com/10.1007/978-0-387-77778-8

IBM Corporation (2014) IBM ILOG CPLEX Optimization Studio V12.6.1 documentation

Kroep F, Buijs P, Coelho LC, Roodbergen KJ (2017) The Two-period Vehicle Routing Problem with Crossdocking and Transfers. Tech. Rep. CIRRELT-2017-53, URL https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2017-53.pdf

Ladier AL, Alpan G (2016) Cross-docking operations: Current research versus industry practice. Omega 62:145–162, DOI 10.1016/j.omega.2015.09.006, URL http://linkinghub.elsevier.com/retrieve/pii/S0305048315001991

Lee YH, Jung JW, Lee KM (2006) Vehicle routing scheduling for cross-docking in the supply chain. Computers & Industrial Engineering 51(2):247–256, DOI 10.1016/j.cie.2006.02.006, URL http://linkinghub.elsevier.com/retrieve/pii/S036083520600091X

Li Y, Lim A, Rodrigues B (2004) Crossdocking - JIT scheduling with time windows. Journal of the Operational Research Society 55:1342–1351

Maknoon Y, Laporte G (2017) Vehicle routing with cross-dock selection. Computers & Operations Research 77:254–266, DOI 10.1016/j.cor.2016.08.007, URL http://dx.doi.org/10.1016/j.cor.2016.08.007

Masson R, Lehuédé F, Péton O (2013) Efficient feasibility testing for request insertion in the pickup and delivery problem with transfers. Operations Research Letters 41(3):211–215

Mendoza J, Rousseau LM, Villegas JG (2015) A hybrid metaheuristic for the vehicle routing problem with stochastic demand and duration constraint. Journal of Heuristics

Morais VW, Mateus GR, Noronha TF (2014) Iterated local search heuristics for the vehicle routing problem with cross-docking. Expert Systems with Applications 41(16):7495–7506, DOI 10.1016/j.eswa.2014.06.010, URL http://linkinghub.elsevier.com/retrieve/pii/S0957417414003510

Nassief W, Contreras I, Jaumard B (2018) A comparison of formulations and relaxations for cross-dock door assignment problems. Computers and Operations Research 94:76–88, DOI 10.1016/j.cor.2018.01.022, URL https://doi.org/10.1016/j.cor.2018.01.022

Nikolopoulou AI, Repoussis PP, Tarantilis CD, Zachariadis EE (2016) Adaptive memory programming for the many-to-many vehicle routing problem with cross-docking. Operational Research DOI 10.1007/s12351-016-0278-1, URL https://doi.org/10.1007/s12351-016-0278-1

Nikolopoulou AI, Repoussis PP, Tarantilis CD, Zachariadis EE (2017) Moving products between location pairs: Cross-Docking versus Direct-Shipping. European Journal of Operational Research 256:803–819, DOI 10.1016/j.ejor.2016.06.053, URL http://linkinghub.elsevier.com/retrieve/pii/S0377221716304933

Paraskevopoulos DC, Laporte G, Repoussis PP, Tarantilis CD (2017) Resource constrained routing and scheduling: Review and research prospects. European Journal of Operational Research 263(3):737–754

Petersen HL, Ropke S (2011) The pickup and delivery problem with cross-docking opportunity. In: Computational Logistics, Springer, pp 101–113

Pillac V, Guéret C, Medaglia A (2013) A parallel matheuristic for the technician routing and scheduling problem. Optimization Letters 7(7):1525–1535, DOI 10.1007/s11590-012-0567-4

Pisinger D, Ropke S (2007) A general heuristic for vehicle routing problems. Computers & Operations Research 34(8):2403–2435, DOI 10.1016/j.cor.2005.09.012, URL http://linkinghub.elsevier.com/retrieve/pii/S0305054805003023

Rochat Y, Taillard ÉD (1995) Probability diversification and intensification in local search for vehicle routing. Journal of Heuristics 1:147–167

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

Santos FA, Mateus GR, da Cunha AS (2011a) A novel column generation algorithm for the vehicle routing problem with cross-docking. In: Network Optimization - INOC 2011, LNCS, vol 6701, pp 412–425, URL http://link.springer.com/chapter/10.1007/978-3-642-21527-8{_}47

Santos FA, Mateus GR, Salles da Cunha A (2011b) A branch-and-price algorithm for a vehicle routing problem with cross-docking. Electronic Notes in

Discrete Mathematics 37:249–254, DOI 10.1016/j.endm.2011.05.043, URL http://linkinghub.elsevier.com/retrieve/pii/S1571065311000448

Santos FA, Mateus GR, da Cunha AS (2013) The pickup and delivery problem with cross-docking. Computers & Operations Research 40(4):1085–1093, DOI 10.1016/j.cor.2012.11.021, URL http://linkinghub.elsevier.com/retrieve/pii/S0305054812002651

Schmid V, Doerner KF, Hartl RF, Savelsbergh MWP, Stoecher W (2009) A Hybrid Solution Approach for Ready-Mixed Concrete Delivery. Transportation Science 43(1):70–85, DOI 10.1287/trsc.1080.0249

Schmid V, Doerner KF, Hartl RF, Salazar-González JJ (2010) Hybridization of very large neighborhood search for ready-mixed concrete delivery problems. Computers and Operations Research 37(3):559–574, DOI 10.1016/j.cor.2008.07.010, URL http://dx.doi.org/10.1016/j.cor.2008.07.010

Schneider M, Stenger A, Goeke D (2014) The electric vehicle-routing problem with time windows and recharging stations. Transportation Science 48(4):500–520

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: Principles and Practice of Constraint Programming CP98, LNCS, vol 1520, pp 417–431, URL http://link.springer.com/chapter/10.1007/3-540-49481-2{_}30

Subramanian A, Uchoa E, Ochi LS (2013) A hybrid algorithm for a class of vehicle routing problems. Computers and Operations Research 40(10):2519–2531, DOI 10.1016/j.cor.2013.01.013, URL http://dx.doi.org/10.1016/j.cor.2013.01.013

Tarantilis CD (2012) Adaptive multi-restart tabu search algorithm for the vehicle routing problem with cross-docking. Optimization Letters 7(7):1583–1596, DOI 10.1007/s11590-012-0558-5, URL http://link.springer.com/10.1007/s11590-012-0558-5

Tellez O, Vercraene S, Lehuédé F, Péton O, Monteiro T (2018) The fleet size and mix dial-a-ride problem with reconfigurable vehicle capacity. Transportation Research Part C: Emerging Technologies 91:99–123

Thorsteinsson ES (2001) Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In: Walsh T (ed) Principles and Practice of Constraint Programming CP 2001, LNCS, vol 2239, pp 16–30, URL http://www.springerlink.com/index/10RCW2QGYECV41KD.pdf

Van Belle J, Valckenaers P, Cattrysse D (2012) Cross-docking: state of the art. Omega 40(6):827–846, DOI 10.1016/j.omega.2012.01.005, URL http://linkinghub.elsevier.com/retrieve/pii/S0305048312000060

Van Hentenryck P (1999) The OPL Optimization Programming Language. MIT Press

Villegas JG, Prins C, Prodhon C, Medaglia A, Velasco N (2013) A matheuristic for the truck and trailer routing problem. European Journal of Operational Research 230(2):231–244, DOI 10.1016/j.ejor.2013.04.026, URL http://dx.doi.org/10.1016/j.ejor.2013.04.026

Wen M, Larsen J, Clausen J, Cordeau JF, Laporte G (2008) Vehicle routing with cross-docking. Journal of the Operational Research Society 60(12):1708–1718, DOI 10.1057/jors.2008.108, URL http://www.palgrave-journals.com/doifinder/10.1057/jors.2008.108

Yu VF, Jewpanya P, Redi AANP (2016) Open vehicle routing problem with cross-docking. Computers and Industrial Engineering 94:6–17, DOI 10.1016/j.cie.2016.01.018, URL http://dx.doi.org/10.1016/j.cie.2016.01.018