CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Matheuristic Based on Large Neighborhood Search for the Vehicle Routing Problem with Cross-Docking

Philippe Grangier
Michel Gendreau
Fabien Lehuédé
Louis-Martin Rousseau

February 2016

CIRRELT-2016-09

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

A Matheuristic Based on Large Neighborhood Search for the Vehicle Routing Problem with Cross-Docking

Philippe Grangier[1,*], Michel Gendreau[2], Fabien Lehuédé[1], Louis-Martin Rousseau[2]

[1] L'UNAM, École des Mines de Nantes, IRCCyN UMR CNRS 6597, 4 Rue Alfred Kastler, 44307 Nantes Cedex 3, France

[2] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Mathematics and Industrial Engineering, Polytechnique Montréal, P.O. Box 6079, Station Centre-Ville, Montréal, Canada H3C 3A7

Abstract. The vehicle routing problem with cross-docking (VRPCD) consists in defining a set of routes that satisfy transportation requests between a set of pickup points and a set of delivery points. The vehicles bring goods from pickup locations to a cross-docking platform, where the items may be consolidated for efficient delivery. In this paper we propose a new solution methodology for this problem. It is based on large neighborhood search and periodically solving a set partitioning and matching problem with third-party solvers. Our method outperforms existing methods by improving the best known solution in 30 of 35 instances from the literature.

Keywords. Routing, cross-docking, transfers, synchronization, matheuristic.

Acknowledgements. This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) (RGPIN-2015-04696) and by the Fonds de recherche du Québec - Nature et technologies (FRQNT) through its team research Program. This research was performed while the first author was visiting CIRRELT and the Department of Mathematics and Industrial Engineering of Polytechnique Montréal.

---

* Corresponding author: Philippe.Grangier@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2016

1. Introduction

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 to another vehicle for delivery. The transfers are based on consolidation opportunities. There is little or no storage capacity at the cross-dock, so the inventory holding costs are low, and the consolidation process reduces the distribution costs. Cross-docking has been successfully applied to several sectors. The classic example is Walmart: cross-docking is said to have been the key to the growth of the retailer in the 1980s [35].

Problems related to cross-docking include location, assignment of trucks to doors, inner flow optimization, and routing. In particular, the *vehicle routing problem with cross-docking* (VRPCD) consists in designing routes to pick up and deliver a set of goods at minimal cost using a single cross-dock. The trucks bring the goods from the pickup locations to the cross-dock where they can offload some items and load others, and they then make the delivery trips. The exchange of goods at the cross-dock is a consolidation process that aims to minimize the total delivery cost. The VRPCD can be seen as a *pickup and delivery problem with transfers* with a single compulsory transfer point.

In this paper, we propose a matheuristic that relies on large neighborhood search (LNS) to create a pool of routes. These routes are then used in a set partitioning and matching (SPM) problem. This problem is solved using branch-and-check [38], a hybrid method that relies on both a mixed integer programming (MIP) solver and a constraint programming (CP) solver. We present numerical results showing that our method outperforms existing algorithms on most instances.

The remainder of this paper is organized as follows. A literature review is presented in Section 2, and the problem is defined in Section 3. Section 4 discusses the solution methodology, and Section 5 presents the computational results.

2. Literature review

Our literature review will focus on routing. We refer the reader to [1, 5, 39] for a general overview of cross-docks and the related problems. The cross-docking literature is fairly recent, and only a few papers focus on the routing aspect. Lee et al. [16] solved a VRPCD variant in which all the vehicles have to arrive at the cross-dock simultaneously. They proposed an exact formulation and developed a tabu-search heuristic to solve instances with 10, 30, and 50 nodes. This heuristic was improved by Liao et al. [17]. Wen et al. [42] extended the problem by adding time windows on the nodes and removing the requirement for simultaneous arrival at the cross-dock, instead imposing precedence constraints based on the consolidation decisions. They presented an MIP formulation and proposed a tabu search embedded in an adaptive memory procedure. They solved real-world instances with up to 200 requests, and they compared their results to a lower bound obtained by solving two VRPTW problems. These results were improved by Tarantilis [37] using a multi-restart tabu search. Morais et al. [21] developed a method based on an iterative local search and a set partitioning problem

(SPP); they introduced new instances with up to 500 customers. Petersen and Ropke [24] worked with a Danish flower-distribution company on a variant of the VRPCD with time windows, optional cross-dock return, and multiple trips per day; they call this the *vehicle routing problem with cross-docking opportunity*. They created a parallel adaptive LNS (ALNS) to solve instances with between 585 and 982 requests. Santos et al. [30, 32] proposed two branch-and-price approaches for a VRPCD variant in which there is a cost to transfer items at the cross-dock but there are no temporal constraints. They extended their approach [31] to a problem where returns to the cross-dock for consolidation are optional. They refer to this problem as the *pickup and delivery problem with cross-docking* and show that it can reduce the routing cost by between 3.1% and 7.7% on their instances, which are restrictions of those of [42]. Dondo and Cerdá [9] considered a variant of the VRPCD in which they modeled each door at the cross-dock individually (e.g., handling speeds and travel times to other doors), and the number of trucks was greater than the number of doors. They proposed an algorithm based on an MILP formulation and a sweep heuristic, and they solved instances with up to 70 requests. Enderer [11] studied the *dock-door assignment and vehicle routing problem*; it involves only the assignment of trucks to doors and the routing of the deliveries. He proposed and compared several exact and heuristic methods. In a recent survey of synchronization in cross-docking networks, Buijs et al. [6] classified the VRPCD as a network scheduling problem with synchronization.

A related problem is the *pickup and delivery problem with transfers* (PDPT). Heuristics for this problem include ALNS [18] and GRASP+ALNS [27]; an exact branch-and-cut method [7] has also been proposed. The *two-echelon vehicle routing problem* [23], for which collaboration between the two echelons is at the heart of the delivery process, is also related to the VRPCD. In particular, the *two-echelon multiple-trip vehicle routing problem with satellite synchronization* [12] deals with the temporal aspects. Lastly, cross-docking operations correspond to both operation synchronization and resource synchronization as described in [10].

Matheuristic approaches often use one or several (meta)heuristics to generate a pool of routes and then solve an SPP (either during the search or in postprocessing). Examples of the metaheuristics used include local search for the VRP [28], tabu search for the split delivery VRP [3], ALNS for the technician routing problem [25], GRASP and local search procedures for the truck and trailer routing routing problem [41] and the VRP with stochastic demands [20], and iterated local search for seven VRP variants [36]. This last method was applied to the VRPCD in [21]. For more details see the surveys by Doerner and Schmid [8] and Archetti and Speranza [2]; the metaheuristic+SPP technique is classified as a set-covering/SPP approach in the former survey and as a restricted master heuristic in the latter. The LNS+SPM proposed in this paper can also be classified into these categories because of the SPM component. However, the SPM also involves matching and scheduling decisions. It is solved using a *branch-and-check* approach [38], which is a hybrid technique integrating MIP and CP; see Section 4.2.

3. Problem formulation

In this section we present the VRPCD, focusing on the scheduling constraints at the cross-dock.

3.1. Problem statement

In the VRPCD, we consider a cross-dock c, a set of items R, and a homogeneous fleet of vehicles V, each of capacity Q and based at o. Each item $r \in R$ must be collected at its pickup location p_r within the time window $[e_{p_r}, l_{p_r}]$ and delivered to its delivery location d_r within the 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 items. 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 or reload there. The sequence of operations at the cross-dock is described in Section 3.2. The *pickup leg* is the sequence of operations performed by a vehicle between its departure from o to perform pickups and its arrival at the cross-dock. The *delivery leg* is the sequence of operations performed by a vehicle between its departure from the cross-dock to perform deliveries and its return to o at the end of the day.

The VRPCD is defined on a directed graph $G = (V, A)$, with $G = \{o\} \cup P \cup \{c\} \cup D$ and $A = \{(o, p)|p \in P\} \cup P \times P \cup \{(p, c)|p \in P\} \cup \{(c, d)|d \in D\} \cup D \times D \cup \{(d, o)|d \in D\} \cup \{(o, c), (c, o)\}$. 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 a minimal routing cost. An arc-based mathematical formulation can be found in [42].

3.2. Cross-dock operations

Following [42], 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, as shown in Figure 1:

- 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. 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. We do not limit the number of available docks.

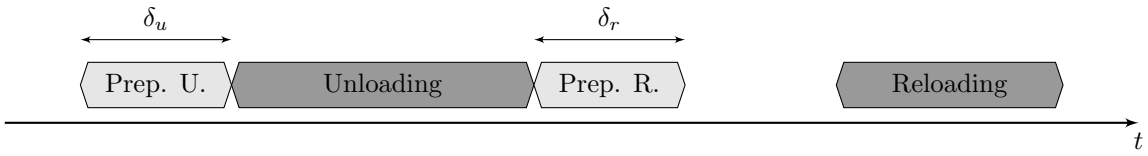

Figure 1: Time chart for vehicle unloading and reloading at the cross-dock. The vehicle must wait to be reloaded because some items are not available when it is ready for reloading.

4. Algorithm

In this section we describe our algorithm for the VRPCD. The main component is LNS, which is enhanced by the occasional solution of an SPM. When the LNS finds a new solution, the legs in this solution are added to a pool of legs that acts as a memory. The SPM is based on an SPP where the set to partition is $P \cup D$, and the candidate partitions are the legs in the pool. Thus, the SPM assists the LNS by selecting good legs that have been previously discovered. We call this LNS+SPM; algorithm 1 presents an outline. Lines 2–15 describe the LNS; see Section 4.1. The SPM is used during the search (line 18); see Section 4.2. The set of legs used in each SPM is managed during the search (line 20).

4.1. Large Neighborhood Search

LNS has been widely used since it was introduced by Shaw [34]. It iteratively destroys (removes requests from) and repairs (reinserts requests into) the current solution using heuristics. Its extension, ALNS [26, 29], selects the destruction and repair methods based on their past success. We select these methods randomly (our method is thus LNS-based) because preliminary experiments suggested that the adaptive layer has an extremely limited impact on the quality of the solutions. Parragh and Schmid [22] likewise choose to use LNS and the SPP for their dial-a-ride problem. We also do not consider noise in the cost function. We now present our destruction and repair methods as well as strategies to reduce the computational time.

4.1.1. Destruction methods

When partially destroying a solution, we select a destruction method M^- and a number Φ of requests to remove. Unless stated otherwise, this method is reused until Φ is reached. We introduce the transfer removals, and the other removal techniques below are inspired by [26].

Random removal:. We randomly remove a request.

Worst removal:. We remove a request with a high removal gain. This is defined as the difference in the cost of the solution with and without the request. We then sort the requests in nonincreasing order of removal gain and place them in a list N. We select the request to remove in a randomized fashion as in [29]: given a parameter p, we draw a random number y between 0 and 1. We then remove the request in position $y^p \times |N|$.

Result: The best found solution $s^\star$

Pool of legs $\mathcal{L} := \emptyset$
Generate an initial solution s
$s^\star := s$
while *stop criterion not met* **do**
    $s' := s$
    Destroy quantity: select a number Φ of requests to remove from s'
    Operator selection: select a destruction method M^- and a repair method M^+
    Destruction: Apply M^- to remove Φ requests from s' and place them in the request bank of s'
    Repair: Apply M^+ to reinsert the requests into the request bank in s'
    if *acceptance criterion is met* **then**
        $s := s'$
    end
    if *cost of s' is better than cost of $s^\star$* **then**
        $s^\star := s'$
    end
    Add legs of s' to $\mathcal{L}$
    if *set partitioning and matching condition is met* **then**
        Perform set partitioning and matching with the legs in $\mathcal{L}$
        Update $s^\star$ and s if a new best solution has been found
        Perform pool management
    end
end
return $s^\star$

Algorithm 1: LNS+SPM

Historical node-pair removal:. Each arc $(u, v) \in G$ is associated with the cost of the cheapest solution in which it appears (initially this cost is set to infinity). We then remove the request that is served using the arcs with the highest associated costs. A randomized selection, similar to that for the worst removal, is performed.

Related removals:. These aim to remove related requests. Let the relatedness of requests i and j be $R(i, j)$. We use two relatedness measures: 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 times at their pickup points and the absolute difference between their start times at their delivery points. In both cases a small $R(i, j)$ indicates a high relatedness. A randomized selection, similar to that for the worst removal (but with a nondecreasing ordering), is performed.

Transfer removal:. For each pair of routes (v_i, v_j), $v_i \neq v_j$, we compute the number of requests transferred from v_i to v_j. We then iteratively apply a roulette-wheel selection on the pairs of routes (the score of a pair being the number of requests transferred), and we remove the requests that are transferred between the routes in the selected pair. If there are fewer transferred requests than the target number Φ to remove, we remove them all and switch to random removal for the rest.

4.1.2. Repair methods

In LNS, the unplanned requests are stored in a *request bank*. We now explain how we insert these requests into a partial solution.

Best insertion:. From the requests r in the request bank, we choose the one with the cheapest insertion cost considering all possible insertions of p_r into pickup legs and d_r into delivery legs.

Regret insertion:. For each request r in the request bank and for each pair of vehicles (pickup vehicle, delivery vehicle), we compute the cost of the cheapest feasible insertion (if any). Note that the pickup and delivery vehicles may be the same (in the case of insertion without transfer). Then, with these insertion options, we compute the *k-regret value* of r, $c_r^k = \sum_{i=1}^{k} f_i - f_1$, 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. We insert the request with the highest regret value.

In algorithm 1 we use 2-regret to generate the inital solution. We use best-insertion, 2-regret, 3-regret, and 4-regret as the repair methods.

4.1.3. Reducing the computational time

In LNS, the repair methods take most of the computational time: more than 96% in [12]. We use two strategies to reduce this time: reducing the size of the neighborhoods and efficient time checking.

Size of the neighborhood:. Because each request can be transferred, the number of candidate insertions for a request is $\Theta(|V|^2)$, which is much larger than for the traditional VRP, where it is only $\Theta(|V|)$. For large instances, it takes considerable time to explore the entire reinsertion neighborhood. This does not necessarily lead to a better solution, because not all the transfer opportunities are worthwhile. Thus, for each request we evaluate all the insertions without transfers, and we consider transfer opportunities for only a subset of g vehicles. For each request r in the request bank we sort the vehicles for pickup (resp. delivery) in nondecreasing order of the cheapest insertion of r in the pickup (resp. delivery) leg. We consider insertion with transfers between the first g vehicles according to this order. A discussion of the choice of g is presented in Section 5.2.2. We use this restricted-neighborhood exploration in both repair methods defined in Section 4.1.2.

Efficient time checking:. If an insertion appears promising (i.e., worth considering with respect to its cost), we need to check that it is feasible. Checking the capacity constraints can easily be done in constant time, but the time constraints are more complex: a straightforward implementation has a linear complexity. Savelsbergh [33] introduced *forward time slacks* that allow us to check in constant time if an insertion is feasible in the VRP with time windows. Masson et al. [19] extended this check to the PDPT. This method has been used several times [12, 13, 18] and has proven successful in significantly reducing the computational time, for example in [12]; see [19] for a detailed description of its implementation. Since the VRPCD can be seen as a PDPT, we reuse it here.

4.2. Set partitioning and matching procedure

In LNS, a solution is rejected solely based on its cost with respect to the best solution so far. A solution that is rejected because it contains bad legs may also contain good legs, which will also be removed. It may also happen that the matching of legs to form routes could be improved (e.g., giving more time flexibility and thus making further improvements easier). We can address these issues by storing the legs found by LNS and using them in an SPP.

The SPM is defined as follows. Given a set of legs L, select a subset of legs $\tilde{L}$ such that (1) each request is picked up (resp. delivered) by exactly one leg in $\tilde{L}$, and (2) the legs in $\tilde{L}$ can be matched to form routes that respect the time constraints. Each leg $l \in L$ has an associated routing cost c_l, and the objective is to minimize the sum of the costs of the selected legs. We now present the branch-and-check method that we use to solve the SPM. We discuss the master problem (set partitioning), the subproblem (matching and scheduling), and the management of the pool of legs.

Our approach is similar to that of Morais et al. [21], who used the SPP as the intensification phase of their SP-ILS. However, there are two main differences: (1) their pool of legs is much smaller (consisting of the legs that appear in their ten best solutions), and (2) when they can not find a feasible matching (they do not discuss their matching procedure), they perform a repair phase by moving requests from one route to another.

4.2.1. Branch-and-check

We present branch-and-check [38] using the following optimization problem:

$$M1: \ \min c^\intercal x \tag{1}$$

$$Ax \leq b \tag{2}$$

$$H(x, y) \tag{3}$$

$$x \in \{0,1\}^n \tag{4}$$

$$y \in \mathbb{R}^m \tag{5}$$

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. (1), (2),

and (4) 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), we carry out a branch-and-bound on (M2). Whenever an integral solution of (M2) is found, we call a CP solver to check constraints (3). If they are satisfied, we update the best solution so far for (M1). Otherwise, we reject this solution. In both cases the branch and bound process continues.

In the SPM, we solve a classical SPP where the set to partition is $P \cup D$, and the candidate partitions are the legs in the pool $\mathcal{L}$. A solution to the SPP is a solution to the VRPCD iff we can match the legs to form a set of routes that respect the time constraints. We determine whether or not this is possible by solving a dedicated matching and scheduling subproblem. Our SPP is the relaxation (M2), and the subproblem checks the constraints $H(x, y)$.

4.2.2. Set partitioning

Consider a set of legs $L = L_p \cup L_d$, where L_p is a set of pickup legs and L_d is a set of delivery legs. For each request $r \in R$, we define a binary constant $\lambda_{r,l}$ that indicates whether or not this request is served by this leg. For each leg $l \in L$ we use a binary variable x_l to determine whether or not it is selected. The SPP is then

$$\min \sum_{l \in L} c_l \times x_l \tag{6}$$

$$\sum_{l \in L_p} \lambda_{r,l} \times x_l = 1 \qquad \forall r \in R \tag{7}$$

$$\sum_{l \in L_d} \lambda_{r,l} \times x_l = 1 \qquad \forall r \in R \tag{8}$$

$$x_l \in \{0, 1\} \qquad \forall l \in L \tag{9}$$

The objective (6) is to minimize the cost of the selected legs, and constraints (7) (resp. (8)) ensure that each pickup point (resp. delivery point) is covered by exactly one leg.

Most MIP solvers use an *incumbent callback* to call an auxiliary subproblem after an integral solution has been found. To save time, we warm-start the MIP solver with the best solution so far ($s^\star$ in algorithm 1).

If the matching and scheduling subproblem is not feasible, one could add some lazy constraints (this is common in branch-and-check). For example, if, for a given pickup leg l and a given delivery leg l' with some requests in common, there is not enough time to perform the associated operations at the cross-dock whether or not they are packed together, we could add $x_l + x'_l \leq 1$ as a lazy constraint in the SPP. We experimented with this technique, but it performed poorly. We identify two reasons for this: (1) the vast majority of the incumbent callbacks are successful, and (2) in CPLEX 12.6.1 (which we use) adding lazy-constraint callbacks disables dynamic search, which seems to reduce CPLEX's computational time on the SPP.

4.2.3. Matching and scheduling subproblem

A solution of the SPP, with a set of pickup legs denoted $\tilde{L_p}$ and a set of delivery legs denoted $\tilde{L_d}$, is a solution to the VRPCD iff there exists a matching of the pickup legs and delivery legs into routes that respect the time constraints. We can compute the earliest feasible arrival time, a_l, of each pickup leg $l \in L_p$ at the cross-dock, and we can compute the latest feasible departure time, $b_{l'}$, of each delivery leg $l' \in L_d$ from the cross-dock. Moreover, for each pickup leg $l \in \tilde{L_p}$ we can determine the set of selected delivery legs T_l that deliver at least one item picked up by l. If we match a pickup leg l and a delivery leg l' to create a route, we can define an associated unloading task $o^-_{ll'}$, with a set of items $R^-_{ll'}$ to unload, and a reloading task $o^+_{ll'}$, with a set of items $R^+_{ll'}$ to reload. These tasks must be performed iff l and l' are in the same route.

The problem is modeled as a constraint satisfaction problem and represented using notation from OPL (Optimization Programming Language [40]). In particular, the model is based on the notion of interval variables [14]. An interval variable represents an unknown interval of time during which a task occurs. The interval has a start value, an end value, and a size, and the associated variable may be optional. We model alternative activities [4] using alternative constraints:

> "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" [14].

We thus consider the following problem:

$$\text{Alternative}(t_l, \{o^-_{ll'}; \forall l' \in \tilde{L_d}\}) \qquad \forall l \in \tilde{L_p} \tag{10}$$

$$\text{Alternative}(t_{l'}, \{o^+_{ll'}; \forall l \in \tilde{L_p}\}) \qquad \forall l' \in \tilde{L_d} \tag{11}$$

$$\text{o}^-_{ll'}.IsOptional \leftarrow True \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \tag{12}$$

$$\text{o}^+_{ll'}.IsOptional \leftarrow True \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \tag{13}$$

$$\text{o}^-_{ll'}.IsPresent \iff o^+_{ll'}.IsPresent \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \tag{14}$$

$$\text{t}_{l'}.Start \geq t_l.End \qquad \forall l \in \tilde{L_p}, l' \in T_l \tag{15}$$

$$\text{o}^+_{ll'}.Start \geq o^-_{ll'}.End + \delta_r \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \; s.t. \; R^+_{ll'} \neq \emptyset \tag{16}$$

$$\text{o}^+_{ll'}.Start \geq o^-_{ll'}.End \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \; s.t. \; R^+_{ll'} = \emptyset \tag{17}$$

$$\text{o}^-_{ll'}.Start \geq a_l + \delta_u \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \; s.t. \; R^-_{ll'} \neq \emptyset \tag{18}$$

$$\text{o}^-_{ll'}.Start \geq a_l \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \; s.t. \; R^-_{ll'} = \emptyset \tag{19}$$

$$\text{o}^+_{ll'}.End \leq b'_l \qquad \forall l \in \tilde{L_p}, \forall l' \in \tilde{L_d} \tag{20}$$

For each pickup leg l we create an interval variable t_l that represents the associated unloading task that takes place at the cross-dock. The alternative constraints (10) and (12) 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 the delivery legs and the reloading operations through variables $t_{l'}$ and constraints (11) and (13). Constraints (14) ensure that the unloading operation associated with the matching of pickup leg l and delivery leg l' in the same vehicle is present iff the corresponding reloading operation is present as well. Constraints (15) 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 (16) and (17) ensure that when two legs are packed together, the delay between the two tasks respects the model presented in Section 3.2. Constraints (18) and (19) ensure that the unloading of each pickup leg does not start before the earliest feasible arrival time at the cross-dock. Constraints (20) ensure that the reloading of each delivery leg is completed by its latest feasible departure time.

4.2.4. Set partitioning and matching criterion and management of pool of legs

When solving the SPP, we consider only the legs in the pool of legs $\mathcal{L}$ (from algorithm 1) that are nondominated. A pickup (resp. delivery) leg l_i is said to be dominated by a leg l_j iff l_i and l_j serve the same set of requests, $c_j < c_i$, and $a_j \leq a_i$ (resp. $b_j \geq b_i$). It is clear that for every solution of the SPP that contains a dominated leg, there exists a solution with a lower cost that does not contain it.

Regarding the management of the pool of legs, we make two observations. First, as the algorithm progresses, it is able to provide the MIP solver with better starting solutions, thus improving its cutoff ability. Therefore, the SPPs tend to be easier to solve at the end of the algorithm than at the beginning. Second, if the MIP has too many legs the solver may fail to improve the initial solution within the time limit. Therefore, every time the SPM is solved, if the solver is able to find an optimal solution and prove its optimality within the time limit, we retain the pool; otherwise we clear it. In practice, this policy tends to empty the pool more frequently at the beginning of the search.

The SPM procedure is called every k_{SPM} iterations; we discuss this value in Section 5.2.3.

5. Computational experiments

The algorithm is coded in C++. We use CPLEX and CP Optimizer from IBM ILOG Cplex Optimization Studio 12.6.1 as the MIP solver and CP solver respectively. The experiments were performed using Linux on an Intel Xeon X5675 @ 3.07 GHz. Just one core is used by our code and the third-party solvers.

5.1. Instances

There are two benchmarks for the VRPCD. The first, which we call the Wen set, was introduced by Wen et al. [42] and contains instances with 50 to 200 requests. It is based on data from a Danish logistics company. The second, which we call the Morais set, was introduced by Morais et al. [21] and contains instances with 200 to 500 requests. It is derived from Gehring and Homberger's instances for the VRPTW. Neither set limits the number of vehicles.

5.2. Parameters

The stopping criterion is set to 20 000 iterations, which is a good compromise between solution quality and computational time. Based on Masson et al. [18] and our preliminary experiments, we choose the number Φ of requests to remove in the repair phase of the LNS randomly from the interval [min(30, 10% of $|R|$), max(60, 20% of $|R|$)].

In the following subsection, we describe the tuning experiments we conducted to set the acceptance criterion, the reduction of the transfer neighborhood, and the SPM period. For the training set, we selected the following instances: 50b, 100b, 150b, and 200b from the Wen set and R1-4-1, R1-6-1, R1-8-1, and R1-10-1 from the Morais set.

5.2.1. Acceptance criterion

We tested three acceptance criteria: descent (a solution is accepted iff it is better than the current best solution), α threshold (a solution is accepted if it is less than α% more expensive than the current best solution), and simulated annealing (as implemented by Ropke and Pisinger [29]). Table 1 compares the performance of the following criteria: descent, 1% threshold, 3% threshold, 10% threshold, and simulated annealing, with descent as the reference. The table shows that all the criteria except the 10% threshold have similar performance at the end of the algorithm. We select descent because it has no parameters.

Accept. Crit.	Descent	1% thresh.	3% thresh.	10% thresh.	Sim. Ann.
Gap (%)	0.00	-0.01	0.04	1.55	0.00

Table 1: Average performance for five acceptance criteria; 10 runs were performed for each instance in the training set, and descent is taken as the reference.

5.2.2. Reduction of the transfer neighborhood

As mentioned in Section 4.1.3, for each request in the request bank we evaluate insertions with transfers between only the g most promising vehicles. We ran tests with a large value of g and we observed that in the best solutions found during the search no vehicle transferred items to more than 10 vehicles. Table 2 gives the average results over five runs on the Morais training instances for different values of g. The percentage gap is similar in each case; this confirms our intuition that only a subset of transfers are worth considering. Figure 2 shows the computational time for different values of g: for R1-10-1 (500 requests) there is a reduction of more than 50% as g decreases. Therefore, we set g to 5.

g	5	10	20	∞
Gap (%)	-0.01	0.00	0.03	0.00

Table 2: Impact of g on solution quality for five runs for each instance in the Morais training set. Infinity is taken as the reference.

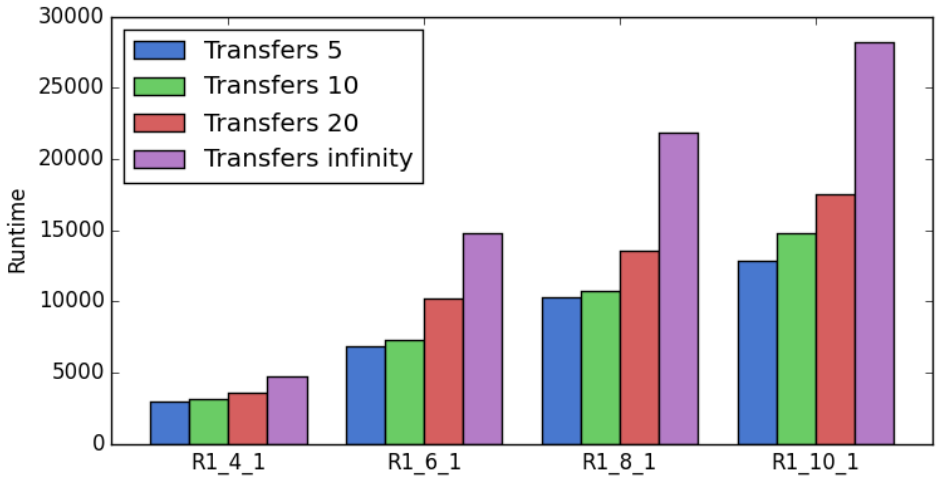

Figure 2: Impact of g on the average computational time (in seconds) for the Morais training set; five runs were performed in each case.

5.2.3. Set partitioning

Table 3 shows the influence of setting the SPM frequency k_{SPM} to 500, 1000, 2500, and 5000 iterations. To maintain a fair balance in the time budget given to the SPM, and thus to see the influence of k_{SPM} on the solution quality and computational time of LNS+SPM, we set the time limits for each call to the SPM to 45, 90, 225, and 450 seconds respectively. We see that frequent calls help to find better solutions, but calling the SPM too often can increase the computational time. We thus set $k_{SPM} = 1000$ with a time limit of 90 seconds.

k_{SPM}	500	1000	2500	5000
Average gap (%)	0	-0.05	0.18	0.70
Average computational time (%)	0	-1.7	10.3	14.3

Table 3: Impact of SPM frequency on solution quality and computational time for five runs for each instance in the training set. The reference is $k_{SPM} = 500$.

5.3. Efficiency of set partitioning and matching

The SPM component is the major contribution of this paper. To assess its efficiency, we compare LNS+SPM and LNS without SPM (we remove lines 16–23 and 23–24 of algorithm 1). Figure 3 shows the convergence curves of LNS and LNS+SPM. On the training set, for 20 000 iterations, LNS finds solutions that are 7.4% more expensive than those of LNS+SPM (4.2% on the Wen training set and 10.6% on the Morais training set). This difference is observed early in the search process. SPM accounts for an increase in the computational time of 20% on average. Thus, SPM is a key component that significantly improves the performance of LNS for the VRPCD. Figure 3 also shows that after a few thousand iterations, LNS alone is

not able to improve the best solution (there are plateaus between calls to SPM). Thus, the LNS contribution is to find good legs that will then be matched by SPM.

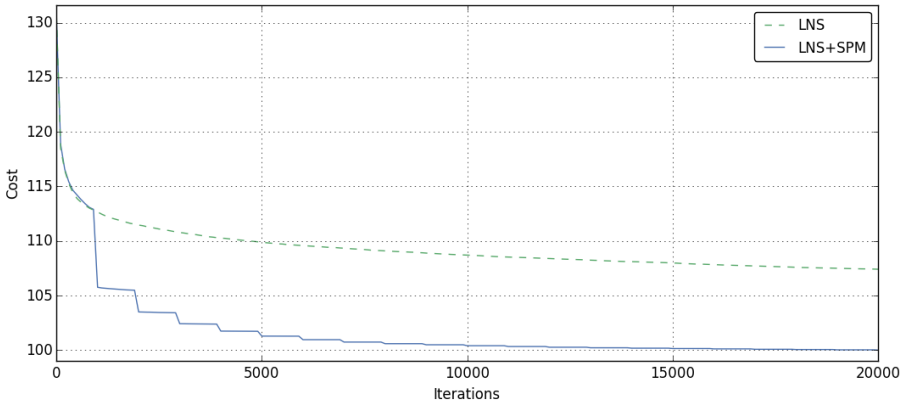

Figure 3: Evolution of the average solution quality for LNS and LNS+SPM; 10 runs were performed for each instance in the training set. The results are normalized, with 100 representing the cost at the end for LNS+SPM.

5.4. Results

In this section we recall the experiments conducted by other authors and report our results.

5.4.1. Existing methods in the literature

There are three methods for the VRPCD: Wen et al. [42], Tarantilis [37], and Morais et al. [21]. Wen et al. [42] designed their method to find good solutions within five minutes of computational time (because of a real-life constraint). They also report results for runs without this time limit, but the design decisions based on the constraint may affect the quality of their results. The methods of [37] and [21] use a time limit of 3000 seconds. Our approach is similar to that of [21] with an emphasis on solution quality rather than computational times.

We evaluated all the methods on the Wen set and only the method of [21] on the Morais set. Wen et al. [42] performed 25 runs for each instance; they report the average and best solutions found within 5 minutes and the best solution found without a time limit. Tarantilis [37] performed 3 runs for each instance with a time limit of 3000 seconds; he reports the best solution found for each instance. Morais et al. [21] performed 40 runs for each instance with a time limit of 3000 seconds for the Wen set and 1200 seconds for the Morais set. They report the best and average solutions for each instance for four variants of their algorithm. We report the results of their SP-ILS variant because it has the best performance.

5.4.2. Best and average results

For each instance LNS+SPM was run ten times. Tables 4 and 5 give our average and best results for the Wen set and compare them to those of the existing methods. The lower bounds are those reported in [42].

We also give the percentage gap between the solution and the lower bound. For the existing methods we give the gaps reported in [21]. Table 6 gives our average and best results for the Morais set and compares them to the results of [21].

Instance	LB	Wen et al.		Morais et al.		LNS+SPM		
		Value	Gap (%)	Value	Gap (%)	Value	Gap (%)	Time (s)
50a	6340.90	6534.2	3.05	6477.72	2.16	**6463.60**	**1.94**	209
50b	7201.89	7504.9	4.21	7443.92	3.36	**7427.45**	**3.13**	545
50c	7241.05	7440.0	2.75	7441.64	2.77	**7320.45**	**1.10**	261
50d	6887.93	7107.6	3.19	7063.17	2.54	**7040.59**	**2.22**	705
50e	7347.54	7629.4	3.84	7514.02	2.27	**7479.04**	**1.79**	272
100b	14200.48	14770.9	4.02	14498.69	2.10	**14376.71**	**1.24**	778
100c	13631.24	14145.0	3.77	13993.00	2.65	**13828.04**	**1.44**	1009
100d	13395.33	13949.6	4.14	13776.76	2.85	**13600.80**	**1.53**	738
100e	13745.60	14396.1	4.73	14159.96	3.01	**13958.75**	**1.55**	712
150a	19012.02	19871.3	4.52	19726.52	3.76	**19401.77**	**2.05**	1911
150b	20371.08	21284.0	4.48	20986.64	3.02	**20672.16**	**1.48**	1959
150c	19419.55	20320.5	4.64	20150.90	3.77	**19771.90**	**1.81**	1862
150d	20013.37	20891.3	4.39	20656.44	3.21	**20356.65**	**1.72**	1760
150e	19141.66	20034.6	4.66	19882.60	3.87	**19493.53**	**1.84**	1525
200a	26538.53	27683.9	4.32	27391.74	3.22	**26863.54**	**1.23**	2993
200b	26722.88	27989.1	4.74	27694.50	3.64	**27295.45**	**2.14**	2986
200c	25607.31	26654.1	4.09	26490.33	3.45	**26087.30**	**1.87**	2835
200d	26969.42	28088.2	4.15	27825.63	3.17	**27394.04**	**1.57**	2774
200e	25776.01	26868.6	4.24	26753.12	3.79	**26108.69**	**1.29**	2691

Table 4: Average values for the Wen set; LNS+SPM was run ten times for each instance with 20 000 iterations as the stopping criterion.

Instance	LB	Wen et al.		Tarantilis		Morais et al.		LNS+SPM		
		Value	Gap (%)	Value	Gap (%)	Value	Gap (%)	Value	Gap (%)	Time (s)
50a	6340.90	6471.9	2.07	**6450.28**	**1.73**	6453.08	1.77	6455.77	1.81	149
50b	7201.89	7410.6	2.9	7428.54	3.15	7434.90	3.24	**7320.77**	**1.65**	240
50c	7241.05	7330.6	1.24	**7311.77**	**0.98**	7317.35	1.05	**7311.77**	**0.98**	120
50d	6887.97	7050.3	2.36	**7021.39**	**1.94**	7035.50	2.14	7028.69	2.04	681
50e	7347.54	7516.8	2.30	**7451.42**	**1.41**	7482.01	1.83	7452.83	1.43	168
100b	14200.48	14526.1	2.29	14405.52	1.44	14441.01	1.69	**14349.60**	**1.05**	828
100c	13631.24	13967.8	2.47	13889.22	1.89	13932.78	2.21	**13784.70**	**1.13**	688
100d	13395.33	13763.3	2.75	**13564.23**	**1.26**	13708.81	2.34	13577.20	1.36	614
100e	13745.60	14212.7	3.4	14059.62	2.28	14122.32	2.74	**13943.10**	**1.44**	601
150a	19012.02	19537.3	2.76	19638.04	3.29	19532.28	2.74	**19358.90**	**1.82**	1886
150b	20371.08	20974.8	2.96	20922.27	2.71	20823.40	2.22	**20581.50**	**1.03**	2093
150c	19419.55	20126.5	3.64	20019.50	3.09	19964.59	2.81	**19726.80**	**1.58**	2187
150d	20013.37	20549.4	2.68	20600.33	2.93	20509.97	2.48	**20318.80**	**1.53**	1660
150e	19141.66	19848.5	3.69	19782.00	3.35	19716.87	3.01	**19449.50**	**1.61**	1546
200a	26538.53	27324.4	2.96	27397.31	3.24	27112.48	2.16	**26816.50**	**1.05**	2988
200b	26722.88	27637.7	3.42	27582.87	3.22	27509.08	2.94	**27215.10**	**1.84**	2804
200c	25607.31	26358.6	2.93	26425.29	3.19	26320.39	2.78	**25926.00**	**1.24**	2376
200d	26969.42	27749.7	2.89	27818.77	3.15	27686.75	2.66	**27328.70**	**1.33**	2457
200e	25776.01	26620.6	3.28	26704.81	3.60	26443.29	2.59	**26063.50**	**1.12**	2240

Table 5: Best solutions found for the Wen set; LNS+SPM was run ten times for each instance with 20 000 iterations as the stopping criterion.

Our method outperforms the existing methods: it has better average results for all the instances, and it improves the best known solutions for 30 of the 35 instances. It improves the best known solutions by 0.78% on average on the Wen set and by 2.16% on average on the Morais set. However, with more time the existing methods may have found better solutions. The computational times for our method are higher than those of Tarantilis [37] and Morais et al. [21]. For each run, we have logged the evolution of the best known solution over time. Thus, we can report the best known solution found by LNS+SPM at any time. Taking

Instance	Morais et al.		LNS+SPM			
			Average		Best solution	
	Average value	Best value	Value	Time (min)	Value	Time (min)
R1-4-1	15530.10	15445.28	**15211.2**	59.6	**15170.0**	56.8
R1-4-2	14996.86	14850.75	**14666.2**	67.1	**14626.9**	57.2
R1-4-3	14414.90	14332.27	**14192.0**	63.6	**14146.6**	54.6
R1-4-4	15622.74	15521.49	**15336.4**	56.8	**15293.3**	56.3
R1-6-1	33776.88	33511.04	**32748.1**	153.4	**32598.1**	154.0
R1-6-2	33744.43	33540.56	**32726.5**	124.3	**32628.7**	123.4
R1-6-3	33478.77	33282.54	**32658.6**	145.9	**32571.5**	129.7
R1-6-4	33606.97	33468.72	**32850.6**	129.7	**32746.3**	139.9
R1-8-1	60611.89	60300.22	**59046.5**	181.3	**58831.1**	186.1
R1-8-2	58420.03	58113.83	**57137.6**	215.0	**56956.4**	179.2
R1-8-3	58859.03	58558.94	**57653.7**	248.2	**57421.7**	224.5
R1-8-4	60834.83	60502.26	**59427.9**	218.6	**59295.3**	233.7
R1-10-1	94687.60	94080.68	**92289.7**	302.5	**91949.2**	280.2
R1-10-2	93718.82	92792.34	**91360.2**	294.4	**91005.8**	297.7
R1-10-3	94200.82	93222.85	**91504.6**	217.1	**91317.0**	221.7
R1-10-4	94795.34	94372.82	**92804.0**	225.8	**92341.1**	227.3

Table 6: Average values and best solution found for the Morais set; LNS+SPM was run ten times for each instance with 20 000 iterations as the stopping criterion.

into account the difference in the speed of the processors used we can compare our method with the existing methods. Following [15], we use CINT2000 and CINT2006[1] to normalize the performance of the processors: [37] used a processor about 2.4 times slower than ours while [21] used a processor about 2.0 times slower than ours. For the Wen set, with comparable computational times, the best values found by LNS+SPM are on average 0.4% better than those in [37], while the average values are on average 1.26% better than those in [21].

6. Concluding remarks

This paper presents a new method for the VRPCD based on LNS and regular calls to an SPP. The SPP component is solved using both an MIP solver and a CP solver. This component helps to find solutions that are significantly better than those obtained by LNS alone. The proposed method has been tested on two benchmarks. It clearly outperforms the existing methods: it improves most of the previously best known results and all the average results. With comparable computational times, it performs better than existing methods that focus on solution quality.

Solving a CP subproblem provides a simple and efficient way to integrate precedence constraints into the SPP. It would be interesting to investigate whether this method could be adapted to solve other VRPs with synchronization-related constraints.

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

[2] C. Archetti, M. G. Speranza, A survey on matheuristics for routing problem, EURO Journal on Computational Optimization 2 (2014) 223–246.

[1] See the Standard Performance Evaluation Corporation website: http://www.spec.org

[3] C. Archetti, M. G. Speranza, M. W. P. Savelsbergh, An optimization-based heuristic for the split delivery vehicle routing problem, Transportation Science 42 (1) (2008) 22–31.

[4] J. C. Beck, M. S. Fox, Scheduling alternative activities, Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence (1999) 680–687.

[5] N. Boysen, M. Fliedner, Cross dock scheduling: Classification, literature review and research agenda, Omega 38 (6) (2010) 413–422.

[6] P. Buijs, I. F. Vis, H. J. Carlo, Synchronization in cross-docking networks: A research classification and framework, European Journal of Operational Research 239 (3) (2014) 593–608.

[7] C. E. Cortés, M. Matamala, C. Contardo, The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method, European Journal of Operational Research 200 (3) (2010) 711–724.

[8] K. F. Doerner, V. Schmid, Survey: Matheuristics for rich vehicle routing problems, in: Hybrid Metaheuristics, vol. 6373 of LNCS, Springer, 2010, pp. 206–221.

[9] 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.

[10] M. Drexl, Synchronization in vehicle routing–A survey of VRPs with multiple synchronization constraints, Transportation Science 46 (3) (2012) 297–316.

[11] F. Enderer, Integrating dock-door assignment and vehicle routing in cross-docking, Ph.D. thesis, Concordia University (2014).

[12] P. Grangier, M. Gendreau, F. Lehuédé, L.-M. Rousseau, An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization, Tech. rep., CIRRELT 2014-33 (2014).

[13] A. Grimault, N. Bostel, F. Lehuédé, An adaptive large neighborhood search for the full truckload pickup and delivery problem with resource synchronization, Tech. rep., Ecole des Mines de Nantes 15/4/AUTO (2015).

[14] IBM Corporation, IBM ILOG CPLEX Optimization Studio V12.6.1 documentation (2014).

[15] G. Laporte, S. Ropke, T. Vidal, Heuristics for the vehicle routing problem, in: P. Toth, D. Vigo (eds.), Vehicle Routing Problems: Problems, Methods, and Applications, 2nd ed., chap. 4, MOS-SIAM, 2014, pp. 87–116.

[16] Y. H. Lee, J. W. Jung, K. M. Lee, Vehicle routing scheduling for cross-docking in the supply chain, Computers & Industrial Engineering 51 (2) (2006) 247–256.

[17] C.-J. Liao, Y. Lin, S. C. Shih, Vehicle routing with cross-docking in the supply chain, Expert Systems with Applications 37 (10) (2010) 6868–6873.

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

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

[20] J. Mendoza, L.-M. Rousseau, J. G. Villegas, A hybrid metaheuristic for the vehicle routing problem with stochastic demand and duration constraint, Journal of Heuristics.

[21] V. W. Morais, G. R. Mateus, T. F. Noronha, Iterated local search heuristics for the vehicle routing problem with cross-docking, Expert Systems with Applications 41 (16) (2014) 7495–7506.

[22] S. N. Parragh, V. Schmid, Hybrid column generation and large neighborhood search for the dial-a-ride problem, Computers and Operations Research 40 (1) (2013) 490–497.

[23] G. Perboli, R. Tadei, D. Vigo, The two-echelon capacitated vehicle routing problem: models and math-based heuristics, Transportation Science 45 (3) (2011) 364–380.

[24] H. L. Petersen, S. Ropke, The pickup and delivery problem with cross-docking opportunity, in: Computational Logistics, Springer, 2011, pp. 101–113.

[25] V. Pillac, C. Guéret, A. Medaglia, A parallel matheuristic for the technician routing and scheduling problem, Optimization Letters 7 (7) (2013) 1525–1535.

[26] D. Pisinger, S. Ropke, A general heuristic for vehicle routing problems, Computers & Operations Research 34 (8) (2007) 2403–2435.

[27] Y. Qu, J. F. Bard, A GRASP with adaptive large neighborhood search for pickup and delivery problems with transshipment, Computers & Operations Research 39 (10) (2012) 2439–2456.

[28] Y. Rochat, É. D. Taillard, Probability diversification and intensification in local search for vehicle routing, Journal of Heuristics 1 (1995) 147–167.

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

[30] F. A. Santos, G. R. Mateus, A. S. da Cunha, A novel column generation algorithm for the vehicle routing problem with cross-docking, in: Network Optimization - INOC 2011, vol. 6701 of LNCS, 2011, pp. 412–425.

[31] F. A. Santos, G. R. Mateus, A. S. da Cunha, The pickup and delivery problem with cross-docking, Computers & Operations Research 40 (4) (2013) 1085–1093.

[32] F. A. Santos, G. R. Mateus, A. Salles da Cunha, A branch-and-price algorithm for a vehicle routing problem with cross-docking, Electronic Notes in Discrete Mathematics 37 (2011) 249–254.

[33] M. W. P. Savelsbergh, Local search in routing problems with time windows, Annals of Operations Research 4 (1) (1985) 285–305.

[34] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, in: Principles and Practice of Constraint Programming CP98, vol. 1520 of LNCS, 1998, pp. 417–431.

[35] G. Stalk, P. Evans, L. E. Schulman, Competing on capabilities: The new rules of corporate strategy, Harvard Business Review 70 (2) (1992) 57–69.

[36] A. Subramanian, E. Uchoa, L. S. Ochi, A hybrid algorithm for a class of vehicle routing problems, Computers and Operations Research 40 (10) (2013) 2519–2531.

[37] C. D. Tarantilis, Adaptive multi-restart tabu search algorithm for the vehicle routing problem with cross-docking, Optimization Letters 7 (7) (2012) 1583–1596.

[38] E. S. Thorsteinsson, Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming, in: T. Walsh (ed.), Principles and Practice of Constraint Programming CP 2001, vol. 2239 of LNCS, 2001, pp. 16–30.

[39] J. Van Belle, P. Valckenaers, D. Cattrysse, Cross-docking: state of the art, Omega 40 (6) (2012) 827–846.

[40] P. Van Hentenryck, The OPL Optimization Programming Language, MIT Press, 1999.

[41] J. G. Villegas, C. Prins, C. Prodhon, A. Medaglia, N. Velasco, A matheuristic for the truck and trailer routing problem, European Journal of Operational Research 230 (2) (2013) 231–244.

[42] M. Wen, J. Larsen, J. Clausen, J.-F. Cordeau, G. Laporte, Vehicle routing with cross-docking, Journal of the Operational Research Society 60 (12) (2008) 1708–1718.