European Journal of Operational Research 277 (2019) 875–885

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Production, Manufacturing, Transportation and Logistics

Probabilistic Tabu Search for the Cross-Docking Assignment Problem

Oualid Guemri[a], Placide Nduwayo[a], Raca Todosijević[a], Saïd Hanafi[a], Fred Glover[b,*]

[a] Université Polythecnique Hauts de France, CNRS UMR 8201 - LAMIH, F-59313, Valenciennes, France
[b] College of Engineering & Applied Science, University of Colorado, Boulder, CO, 80309, USA

ARTICLE INFO

Article history:
Received 16 February 2018
Accepted 20 March 2019
Available online 27 March 2019

Keywords:
Metaheuristics
Probabilistic Tabu Search
Cross-Docking Assignment Problem
Combinatorial Optimization

ABSTRACT

The Cross-Docking Assignment Problem (CDAP) is a challenging optimization problem in supply chain management with important practical applications in the trucking industry. The goal is to assign incoming trucks (outgoing trucks) to inbound (outbound) doors to minimize the material handling cost within a cross-docking platform while respecting the capacity and assignment constraints. A capacity constraint is imposed on each inbound/outbound door and an associated assignment constraint is imposed on each incoming/outgoing truck requiring it to be assigned to only one inbound/outbound door. To solve this NP-hard optimization problem, we develop two novel heuristics based on Probabilistic Tabu Search utilizing a new neighborhood structure applicable both to CDAP and related problems. The proposed heuristics are evaluated on 99 benchmark instances from the literature, disclosing that our approaches outperform recent state-of-the-art approaches by reaching 45 previous best-known solutions and discovering 53 new best-known solutions while consuming significantly less CPU time.

1. Introduction

The Cross-Docking Assignment Problem (CDAP) is an NP-hard combinatorial optimization problem that arises in the operational level of supply chain management. Considering a two-sided cross-docking facility where the inbound doors are at one side and the sets of outbound doors are at the opposite side, the cross-docking policy consists in the following. Fully loaded incoming trucks enter the cross-docking platform and unload goods at inbound doors. After that the goods are immediately sorted and organized according to their destinations, and transferred to outbound doors where they are loaded on outgoing trucks. The goal of the CDAP is to find an optimal assignment of incoming trucks to inbound doors and outgoing trucks to outbound doors so that the material handling cost within the cross-docking platform is minimized. Material handling cost is measured as total weighted traveled distance of devices used to transfer goods between inbound and outbound doors. To be feasible, a solution of the CDAP may be required to satisfy various constraints, typically consisting of capacity and assignment constraints.

In the literature, the CDAP is considered as an instance of an assignment problem (Guignard, Hahn, Pessoa, & da Silva, 2012). Assignment problems are well-studied optimization problems that have given rise to numerous proposals for solution algorithms including both metaheuristics and exact methods (see, e.g., Pentico, 2007). To briefly indicate some of the more salient contributions, variants of assignment problems that have received attention include: The Generalized Assignment Problem (GAP) (Yagiura, Ibaraki, & Glover, 2006), the generalized quadratic assignment problem (Pessoa, Hahn, Guignard, & Zhu, 2010) and the quadratic three-dimensional assignment problem (Hahn et al., 2008). In (Zhu, Hahn, Liu, & Guignard, 2009), the authors observe a relationship between the Generalized Quadratic 3-dimensional Assignment Problem and the CDAP we study here which discloses that the CDAP can be solved as GQ3AP. Tsui and Chang (1990) propose a mathematical formulation for the CDAP which requires that each incoming truck is assigned to only one indoor and each indoor can serve only one incoming truck. Tsui and Chang (1992) proposed a branch & bound method to solve the problem formulated in Tsui and Chang (1990). Zhu et al. (2009) extend the Cross-Docking Assignment Problem presented in Tsui and Chang (1990) by allowing more than one truck to be assigned to a door and by imposing a capacity constraint on the doors. The form of CDAP considered in Zhu et al. (2009) includes the Generalized Assignment Problem as a subproblem and like the GAP problem is NP-hard. Because of its NP-hard character, most of the studies of the CDAP in the literature have been dedicated to developing efficient heuristic solution approaches to cope with large scale instances. In this regard, Guignard et al. (2012) proposed two heuristics to solve the CDAP as defined in Zhu et al. (2009) where the first is a multi-start local search where the authors derived two variants and the second is a

* Corresponding author.
E-mail addresses: oualid.guemri@uphf.fr (O. Guemri), placide.nduwayo@uphf.fr (P. Nduwayo), raca.todosijevic@uphf.fr (R. Todosijević), said.hanafi@uphf.fr (S. Hanafi), glover@opttek.com (F. Glover).

https://doi.org/10.1016/j.ejor.2019.03.030
0377-2217/

metaheuristic called Convex Hull. Nassief et al. (2016) presented a linear mixed integer programming (MIP) formulation and proposed a Lagrangean relaxation algorithm to deal with CDAP as formulated by Zhu et al. (2009). In (Tarhini, Yunis, & Chamseddine, 2016), the authors presented a scatter search and a genetic algorithm to deal with CDAP based on the problem definition of Tsui and Chang (1990). Nassief, Contreras, and Jaumard (2018) presented a study of the standard CDAP (as defined in (Zhu et al., 2009)) with and without the load and unload times where they compared new MIP formulations and LP relaxations. Ladier and Alpan (2016) studied the gap between the academic literature and the industrial applications of cross-docking. For other cross-docking problems and/or literature reviews in cross-docking, we refer the reader to Bellanger, Hanafi, and Wilbaut (2013) and Buijs, Vis, and Carlo (2014), Boysen and Fliedner (2010), Van Belle, Valckenaers, and Cattrysse (2012). In this paper we deal with the realistic variant of the CDAP proposed in Zhu et al. (2009) where the capacity constraint of each inbound/outbound door must be satisfied as well as a constraint requiring each origin/destination to be assigned to only one inbound/outbound door.

Formally the CDAP may be defined in the following way. We are given a set of incoming trucks (treated as origins) M, a set of outgoing trucks (treated as destinations) N, a set of inbound doors I and a set of outbound doors J. When the origin $m \in M$ is assigned to the inbound door $i \in I$ and the destination $n \in N$ is assigned to the outbound door $j \in J$ a material handling cost is incurred. The cost is calculated as the product of $d_{i,j}$ and $f_{m,n}$ where $d_{i,j}$ is the distance between the door i and the door j, and $f_{m,n}$ is the number of pallets to be moved from the origin m to the destination n. The total number of pallets transferred from origin $m \in M$ is $s_m = \sum_{n \in N} f_{m,n}$ and the total number of pallets received at destination $n \in N$ is $r_n = \sum_{m \in M} f_{m,n}$. The capacity of an inbound door $i \in I$ is denoted by S_i and the capacity of an outbound door $j \in J$ is denoted by R_j. The capacity refers to the total number of pallets processed at a door over given time interval (i.e., planning horizon). Using binary variables $x_{m,i}$ and $y_{n,j}$ to indicate whether or not an inbound truck m is assigned to inbound door i, and whether or not an outbound truck n is assigned to outbound door j, respectively, the CDAP may be formulated as the following 0–1 quadratic program:

$$\min f(x,y) = \sum_{m \in M} \sum_{i \in I} \sum_{n \in N} \sum_{j \in J} d_{i,j} f_{m,n} x_{m,i} y_{n,j} \tag{1-a}$$

Subject to:

$$\sum_{i \in I} x_{m,i} = 1 \quad \forall m \in M \tag{1-b}$$

$$\sum_{j \in J} y_{n,j} = 1 \quad \forall n \in N \tag{1-c}$$

$$\sum_{m \in M} s_m x_{m,i} \le S_i \quad \forall i \in I \tag{1-d}$$

$$\sum_{m \in M} r_n y_{n,j} \le R_j \quad \forall j \in J \tag{1-e}$$

$$x_{m,i} \in \{0,1\}, \quad \forall m \in M, \ \forall i \in I \tag{1-f}$$

$$y_{n,j} \in \{0,1\} \quad \forall n \in N, \ \forall j \in J \tag{1-g}$$

The objective function (1-a) minimizes the material handling cost inside the warehouse. The two sets of constraints (1-b) and (1-c) ensure that each origin/destination must be allocated to one and only one inbound/outbound door, respectively. The constraints (1-d) (resp. (1-e)) guarantee that the capacity of each inbound (resp. outbound) door is respected. The last two sets of constraints (1-f) and (1-g) impose binary requirement on decision variables. Some other mixed integer programming formulations of the studied problem may be found in Guignard et al. (2012), Nassief, Contreras, and As' ad (2016) and Gelareh et al. (2018). According to the results reported in Gelareh et al. (2018), instances with up to 15 origins/destinations and 7 indoors/outdoors may be optimally solved by the CPLEX MIP solver within the time limit of two hours. However, the largest instances remain elusive for the CPLEX MIP solver and therefore there is a need for heuristic approaches.

In this work we propose two Probabilistic Tabu Search (PTS) heuristics which differ in the way they construct a candidate list of solutions and accept new incumbent solutions. In addition, we propose a new extension of the swap neighborhood that allows the exchange of more than two elements and we design an efficient heuristic method to explore it. Extensive testing is performed on benchmark instances from the literature to assess the performance of our proposed approaches, showing that our PTS heuristics outperform the previous state-of-the art approaches by reaching 45 previous best-known solutions and discovering 53 new best-known solutions on a set of 99 instances. In addition, the CPU time consumed by our approaches is substantially less than consumed by the previous state-of-the art methods. We also conduct tests to show that our heuristic exploration yields a good trade-off between solution quality and CPU time in comparison with exhaustive exploration of our new neighborhood structure.

The rest of the paper is organized as follows. The next section describes the main ingredients of the proposed heuristics based on Probabilistic Tabu Search, including a procedure for constricting an initial solution, as well as the neighborhood structures used and efficient ways of exploring them. Section 3 presents two Probabilistic Tabu Search heuristics built on the ingredients described in the preceding section and Section 4 is dedicated to computational experiments to assess the merit of the proposed approaches. Finally, Section 5 offers concluding observations and sketches some possible future research directions.

2. Main ingredients of the Probabilistic Tabu Search approaches

In this section we present the main ingredients of our proposed Probabilistic Tabu Search heuristics with multiple neighborhood structures. First, we present the procedure used to generate an initial solution and then we describe neighborhood structures exploited by our PTS heuristics. In addition, we expose the data structures and updating procedures used in our implementation.

A solution of the CDAP is represented by partitions of M and N denoted by X and Y, respectively. Each element X_i of X is a set containing all origins assigned to the inbound door $i \in I$. Similarly, each element Y_j is a set containing all destinations assigned to the outbound door $j \in J$. More formally, if we use binary variables $x_{m,i}$, $y_{n,j}$ defined in the introductory section, the set X_i and Y_j may be stated as:

$$X_i = \{m \in M : \ x_{m,i} = 1\} \text{ and } Y_j = \left\{n \in N : y_{n,j} = 1\right\}.$$

We note that some sets X_i or Y_j can be empty in a feasible solution.

The objective function of a solution (X, Y) may be calculated as

$$f(X,Y) = \sum_{i \in I} \sum_{j \in J} \sum_{m \in X_i} \sum_{n \in Y_j} d_{i,j} f_{m,n}$$

Correspondingly, the cost incurred by assigning origin $m \in M$ to inbound door $i \in I$, for a given partition Y, may be expressed as

$$c^{o}_{m,i}(Y) = \sum_{j \in J} \sum_{n \in Y_j} d_{i,j} f_{m,n} \tag{2-a}$$

and the cost of assigning destination $n \in N$ to outbound door $j \in J$, for a given partition X, may be expressed as

$$c^d_{n,j}(X) = \sum_{i \in I} \sum_{m \in X_i} d_{i,j} f_{m,n} \tag{2-b}$$

The amount of capacity $S(X_i)$ (resp. $R(Y_j)$) consumed at each inbound (resp. outbound) door with respect to the solution *(X, Y)* is expressed as

$$S(X_i) = \sum_{m \in X_i} s_m \quad \forall i \in I$$

$$R(Y_j) = \sum_{m \in Y_j} r_n \qquad \forall j \in J.$$

2.1. Constructive heuristic to generate the initial solution

We use the following procedure to generate an initial solution. First, the procedure sorts the origins so that their numbers of pallets, s_m, are in descending order of size, and then assigns these origins to the inbound doors in a random fashion (Step 6) respecting the capacity constraint of these doors (Step 4). The destinations are then assigned to the outbound doors using a greedy procedure in which the destinations are similarly sorted in descending order according to the total number of pallets $r_{n,}$ they receive (Step 10). After that, one by one the destinations are assigned to the outbound doors following the established order. This latter assignment is accomplished by assigning a destination n to a door j associated with the minimum assignment cost $c^d_{n,j}(X)$, where $c^d_{n,j}(X)$ depends on the given assignment of origins (Steps 11–17). We have found that sorting the origins and destinations in this simple manner greatly enhances the algorithm's ability to find a feasible initial solution that satisfies the doors' capacities, although of course there is no guarantee that this starting solution will be feasible. Namely, some origins (destinations) may remain non-assigned to inbound (outbound) doors. If this happens, non-assigned origins (destinations) are assigned to inbound (outbound) doors in a greedy way so that the violation of the capacity constraints is minimized. To measure the violation of the capacity constraints the following function is used:

$$g(X,Y) = \sum_{i \in I} \max\{0,\ S(X_i) - S_i\} + \sum_{j \in J} \max\{0,\ R(Y_j) - R_j\}.$$

After that, in order to attain feasibility, we launch a Probabilistic Tabu Search (PTS) algorithm, whose steps are given in Section 3. In this case, the PTS considers $g(X,Y)$ as the objective function and may accept also infeasible solutions. Once a feasible solution is found, it is used as an initial solution for the PTS which works only with feasible solutions and uses the CDAP objective function, $f(X,Y)$ (see Section 3 for more details). If the best solution found so far by PTS, denoted (X^*, Y^*), is infeasible, PTS considers $g(X,Y)$ as an objective function. Once feasibility of the modified solution (X^*, Y^*) is attained our approach considers the CDAP objective function $f(X,Y)$. Starting from this point, a candidate list $\mathcal{N}(X, Y)$ is forced to contain only feasible solutions at each subsequent iteration. The procedure is depicted in Algorithm 1. Note that the algorithm verifies the capacity constraints (Steps 4 and 12) using the residual capacities $S(X_i)$ and $R(Y_j)$. The residual capacities, as previously defined, refer to the amount of capacity consumed at the certain door by origins/destinations currently assigned to it.

Algorithm 1 Constructive heuristic.

1. Create empty solution: $X_i = \emptyset$ for all $i \in I$, and $Y_j = \emptyset$ for all $j \in J$;
2. Sort the origins $m \in M$ in descending order of their s_m values;
3. **For** each $m \in M$ **do**
4. Let $I' = \{i \in I : S(X_i) + s_m \le S_i\}$ be the set of indoors that can receive the origin m;
5. **If** $I' \neq \emptyset$ **then**
6. Select randomly an indoor $i \in I'$;
7. $X_i = X_i \cup \{m\}$;
8. **EndIf**
9. **EndFor**
10. Sort the destinations $n \in N$ in descending order of their r_n values;
11. **For** each $n \in N$ **do**
12. Let $J' = \{j \in J : R(Y_j) + r_n \le R_j\}$ be the outdoors that can receive the destination n;
13. **If** $J' \neq \emptyset$ **then**
14. Let $j = argmin\{c^d_{n,j'}(X) : j' \in J'\}$;
15. $Y_j = Y_j \cup \{n\}$;
16. **EndIf**
17. **EndFor**
18. **Return** (X, Y);

2.2. Neighborhood structures and move evaluation

A solution (X, Y) corresponds to a partition of the set of origins M and a partition of set of destinations N, respectively. The moves that define the neighborhood structure consist of transferring a truck from one door to another, and of exchanging two subsets of trucks between two doors. Hence, we define the neighborhood structures of the current solution $\mathcal{N}^\tau(X,Y)$ that affect either the origins ($\tau = o$) or the destinations ($\tau = d$). For each side $\tau \in \{o, d\}$, we denote by $\bar{\tau}$ the opposite side of τ, i.e., if $\tau = o$ then $\bar{\tau} = d$ and vice-versa. Specifically, we divide the moves into the following two types *Shift* moves and *Swap* moves. It is worth mentioning that we consider only feasible moves when defining the neighborhood structure. However, in the exceptional case where the solution returned by the initial solution procedure is not feasible, the algorithm accepts only those moves that decrease infeasibility until a feasible solution is found. Then, only feasible moves are performed.

2.2.1. Shift moves

A shift move transfers a selected truck (origin or destination) from one door to another (inbound door or outbound door). For the origin side $\tau = o$, a solution that is a neighbor of the current solution (X, Y) is obtained by shifting an origin $m \in M$ from its current inbound door i ($m \in X_i$) to another inbound door $i^* \in I - \{i\}$ selected randomly among the k best inbound doors (having the smallest costs $c^o_{m,i^*}(Y)$). More precisely, for each origin $m \in M$, we re-index the inbound doors $i' \in I - \{i\}$ so that $c^o_{m,1}(Y) \le c^o_{m,2}(Y) \le \ldots \le c^o_{m,|I|-1}(Y)$ and let $I^k_m = \{1, \ \ldots, k\}$ be the set identifying the inbound doors $i' \in I - \{i\}$ with the k smallest values $c^o_{m,i'}$. A neighboring solution $(X', Y') \in \mathcal{N}^{o,k}_{Shift}(X,Y)$ is obtained by setting $Y' = Y$, selecting randomly $i^* \in I^k_m$ and for all $i' \in I$ setting

$$X'_{i'} = \begin{cases} X_i - \{m\} & if\ i' = i \\ X_{i^*} + \{m\} & if\ i' = i^* \\ X_i & \text{otherwise} \end{cases} \tag{3-a}$$

Analogously, for the destination side $\tau = d$, a solution in the neighborhood of the current solution (X, Y) is obtained by shifting a destination $n \in N$ from its current outbound door j ($n \in Y_j$) to another outbound door $j^* \in J - \{j\}$ selected randomly among the k best outbound doors (having the smallest costs $c^d_{n,j^*}(X)$). For the sake of completeness, we provide definitions of these moves as well: for each destination $n \in N$, we re-index the outbound doors $j' \in J - \{j\}$ so that $c^d_{n,1}(X) \le c^d_{n,2}(X) \le \ldots \le c^d_{n,|J|-1}(X)$ and let $J^k_n = \{1, \ \ldots, k\}$ be the set identifying the outbound doors $j' \in J - \{j\}$ with the k smallest values $c^d_{n,j'}(X)$. A neighboring solution $(X', Y') \in \mathcal{N}^{d,k}_{Shift}(X,Y)$ is obtained by setting $X' = X$, selecting randomly $j^* \in J_n^k - \{j\}$ and for all $j' \in J$ setting

$$Y'_{j'} = \begin{cases} Y_j - \{n\} & if\ j' = j \\ Y_{j^*} + \{n\} & if\ j' = j^* \\ Y_j & \text{otherwise} \end{cases} \tag{3-b}$$

Remark 1. if $k=1$, the origin m (resp. the destination n) is transferred to the best inbound (resp. outbound) door, while if $k=|I|-1$ (resp. $k=|J|-1$) it is transferred to a randomly selected inbound (resp. outbound) door.

2.2.2. Swap moves

A swap move consists of exchanging trucks between two different doors. In our implementation, we consider two groups of swap moves: *elementary swap moves* and *multiple swap moves*. An elementary swap move consists of exchanging two different trucks between two different doors, while a multiple swap move consists of exchanging two subsets of trucks between two different doors.

Formally, for the origin side $\tau = o$, a neighborhood solution $(X', Y') \in \mathcal{N}_{Swap}^{o,p,q}(X, Y)$ is obtained by setting $Y' = Y$, choosing $i, i' \in I$ with $i \neq i'$, selecting $P \subseteq X_i$ such that $|P| = p$ and $Q \subseteq X_{i'}$ such that $|Q| = q$ and for all $h \in I$ setting

$$X'_h = \begin{cases} X_h - P + Q & if\ h = i \\ X_h - Q + P & if\ h = i' \\ X_h & \text{otherwise} \end{cases} \tag{4-a}$$

Similarly, for the destination side $\tau = d$, a neighboring solution $(X', Y') \in \mathcal{N}_{Swap}^{d,p,q}(X, Y)$ is obtained by setting $X' = X$, choosing $j, j' \in J$ with $j \neq j'$, selecting $P \subseteq Y_j$ such that $|P| = p$ and $Q \subseteq Y_{j'}$ such that $|Q| = q$ and for all $h \in J$ setting

$$Y'_h = \begin{cases} Y_h - P + Q & if\ h = j \\ Y_h - Q + P & if\ h = j' \\ Y_h & \text{otherwise} \end{cases} \tag{4-b}$$

Remark 2. The elementary swap moves can be derived from the above definition by choosing $p=1$ and $q=1$.

The set of neighboring solutions generated by swap moves that affect sets X_i and $X_{i'}$ (resp. Y_j and $Y_{j'}$) will be denoted by $\mathcal{N}_{X_{Swap}}^{o,p,q}(X_i,\ X_{i'})$ (resp. $\mathcal{N}_{Y_{Swap}}^{d,p,q}(Y_j,\ Y_{j'})$). Using these definitions we have $\mathcal{N}_{Swap}^{o,p,q}(X, Y) = \bigcup_{i,i' \in I,\ i \neq i'} \mathcal{N}_{X_{Swap}}^{o,p,q}(X_i,\ X_{i'})$ and similarly $\mathcal{N}_{Swap}^{d,p,q}(X, Y) = \bigcup_{j,j' \in J,\ j \neq j'} \mathcal{N}_{Y_{Swap}}^{d,p,q}(Y_j,\ Y_{j'})$. In Section 2.2.4, we describe an efficient procedure to explore the swap neighborhood.

2.2.3. Data structures for evaluation of moves and their updates

To efficiently evaluate each move presented in the preceding section we use auxiliary data structures. By move evaluation here we mean the change in the objective function caused by executing a certain move on a current solution. Here we present only a method for efficiently evaluating the shift moves, since each swap move (elementary or multiple) can be easily transformed into a set of shift moves.

From Eq. (1-a) the objective function value of a solution (X, Y) can be expressed as

$$f(X, Y) = \sum_{i \in I} \sum_{j \in J} \sum_{m \in X_i} \sum_{n \in Y_j} d_{i,j} f_{m,n}$$

Using Eq. (2-a), this can be rewritten as

$$f(X, Y) = \sum_{i \in I} \sum_{m \in X_i} c^o_{m,i}(Y). \tag{5-a}$$

Or equivalently by Eq. (2-b):

$$f(X, Y) = \sum_{j \in J} \sum_{n \in Y_j} c^d_{n,j}(X). \tag{5-b}$$

Again, we differentiate shift moves that affect origin-inbound door assignments ($\tau = o$) and those that affect destination-outbound door assignments ($\tau = d$).

First consider a shift move on origin side ($\tau = o$), that transfers an origin $m \in M$ from its current inbound door i ($m \in X_i$) to another inbound door $i^* \in I_m^k - \{i\}$. The objective function change produced by this shift move is given by

$$\Delta^o(m, i, i^*) = f\left(X', Y\right) - f(X, Y).$$

Using Expressions (4-a) and (5-a) we obtain

$$\Delta^o(m, i,\ i^*) = c^o_{m,i^*}(Y) - c^o_{m,i}(Y) \tag{6-a}$$

Next consider a shift move on the destination side ($\tau = d$), that transfers a destination $n \in N$ from its current inbound door j ($n \in Y_j$) to another inbound door $j^* \in J_n^k - \{j\}$. The objective function change produced by this shift move is given by

$$\Delta^d(n, j, j^*) = f\left(X, Y'\right) - f(X, Y).$$

Similarly, using Expressions (4-b) and (5-b) we obtain

$$\Delta^d(n, j, j^*) = c^d_{n,j^*}(X) - c^d_{n,j}(X). \tag{6-b}$$

As consequence of Expressions (6-a) and (6-b), the shift move can be evaluated in constant time $O(1)$, if we make reference to the two matrices $c^o_{m,i}(Y)$ and $c^d_{n,j}(X)$. Hence, to achieve this constant time computation of the objective function change $\Delta^o(m, i,\ i^*)$ and $\Delta^d(n, j, j^*)$, we need to update the two matrices $c^o_{m,i}$ and $c^d_{n,j}$ after each shift move.

Let $c'^o_{m,i}(Y)$ (resp. $c'^d_{n,j}(X)$) be the value of entry (m, i) (resp. (n, j)) in the matrix $c^o_{m,i}(Y)$ (resp. $c^d_{n,j}(X)$) after a shift move. Observe from Eq. (2-a) and (2-b) that provide the definitions of $c^o_{m,i}(Y)$ and $c^d_{n,j}(X)$ respectively, that the execution of a shift move on the side $\tau = o$ affects the matrix $c^d_{n,j}(X)$ and vice versa. More precisely, after a shift move on the side $\tau = o$, we have

$$c'^d_{n,j}\left(X'\right) = \sum_{i \in I} \sum_{m \in X'_i} d_{i,j} f_{m,n}.$$

Using Expression (3-a), we obtain

$$c'^d_{n,j}\left(X'\right) = c^d_{n,j}(X) + f_{m,n}(d_{i^*,j} - d_{i,j}). \tag{7-a}$$

Similarly, after a shift move on the side $\tau = d$, we have

$$c'^o_{m,i}\left(Y'\right) = \sum_{j \in J} \sum_{n \in Y'_j} d_{i,j} f_{m,n}.$$

Using Expression (3-b), we obtain

$$c'^o_{m,i}\left(Y'\right) = c^o_{m,i}(Y) + f_{m,n}(d_{i,j^*} - d_{i,j}). \tag{7-b}$$

As a consequence, the complexity of updating Δ^o after a shift move on the side $\tau = d$ is $O(|N| \times |J|)$ and the complexity of updating Δ^d after a shift move on the side $\tau = o$ is $O(|M| \times |I|)$.

2.2.4. Efficient exploration of the swap neighborhood

The complexity of the neighborhood $\mathcal{N}_{Swap}^{o,p,q}(X, Y)$ is $O\left(\sum_{i,\ i' \in I,\ i \neq i'} \binom{|X_i|}{p}\binom{|X_{i'}|}{q}\right)$. Consequently, the exhaustive exploration of the union of neighborhoods $\mathcal{N}_{Swap}^{o,p,q}(X, Y)$, $1 \le p \le |X_i|$ and $1 \le q \le |X_{i'}|$, which we denote by $\mathcal{N}_{Swap}^{o}(X, Y)$, has complexity $O(\sum_{i,i' \in I,\ i \neq i'} 2^{|X_i|+|X_{i'}|})$. However, if each solution in $\mathcal{N}_{Swap}^{o}(X, Y)$ is feasible, then the best solution in this neighborhood can be found by an exploration of smaller complexity, as we demonstrate in the following proposition.

Proposition. The best solution within the union of swap neighborhoods $\mathcal{N}_{Swap}^{o}(X, Y)$ can be determined with time complexity $O(\sum_{i,\ i' \in I,\ i \neq i'} |X_i| + |X_{i'}|)$ if all solutions in $\mathcal{N}_{Swap}^{o}(X, Y)$ are feasible.

Proof. Consider two sets X_i and $X_{i'}$ and define $X_i^{imp} = \{m \in X_i : \Delta^o(m, i, i') < 0\}$ and $X_{i'}^{imp} = \{m' \in X_{i'} : \Delta^o(m', i', i) < 0\}$. By these definitions the best multiple swap move that affects sets X_i and $X_{i'}$ is one that exchanges sets X_i^{imp} and $X_{i'}^{imp}$. Denote the solution obtained from such a swap move by $(X^{i,i'}, Y)$. The generation of this solution requires $O(|X_i| + |X_{i'}|)$ operations, since sets X_i^{imp} and $X_{i'}^{imp}$ may be generated in linear time complexity $O(|X_i|)$ and $O(|X_{i'}|)$, respectively. Consequently, the best solution in the neighborhood $N^o_{Swap}(X, Y)$, *i.e.*, $(X^*, Y^*) = argmin\{f(X^{i,i'}, Y) : i, i' \in I, i \neq i'\}$ may be found with complexity $O(\sum_{i,\ i' \in I,\ i \neq i'} |X_i| + |X_{i'}|)$. □

The preceding result does not hold if there is an infeasible solution in the neighborhood $\mathcal{N}^o_{Swap}(X, Y)$. This can be demonstrated by a small example involving only 3 incoming trucks m_1, m_2 and m_3 with loads $s_{m_1} = 3$, $s_{m_2} = 5$ and $s_{m_3} = 8$, respectively. Suppose we have only two incoming doors i_1 and i_2 both with capacity $S_{i_1} = S_{i_2} = 10$. Further, in the solution (*X*, *Y*), assume trucks m_1 and m_2 with loads 3 and 5 are assigned to the first incoming door i_1 and truck m_3 with load 8 is assigned to the door i_2. Then the neighborhood $\mathcal{N}^o_{Swap}(X, Y)$ contains both feasible and infeasible solutions with respect to the capacity constraints. Let $\Delta^o(m_1, i_1, i_2) > 0$, $\Delta^o(m_2, i_1, i_2) < 0$ and $\Delta^o(m_3, i_2, i_1) < 0$. Then if we use the procedure from the preceding proposition only the move that exchanges trucks m_2 and m_3 between doors will be considered as a potential improving move, but this move is infeasible. Consequently, the current solution (*X*, *Y*) would be the best solution. However, in the case that $\Delta^o(m_1, i_1, i_2) + \Delta^o(m_2, i_1, i_2) + \Delta^o(m_3, i_2, i_1) < 0$, a swap move that exchanges trucks m_1 and m_2 from one side with a truck m_3 from the other side is an improving move. So, the procedure used in the preceding proposition may fail to reach the best solution if there is an infeasible solution in the neighborhood $\mathcal{N}^o_{Swap}(X, Y)$.

However, to avoid exhaustive exploration of the neighborhood $\mathcal{N}^o_{Swap}(X, Y)$, which may be time consuming due to its large cardinality, but to be still able to find a near best solution (i.e. a solution with a quality near to the quality of the best solution), we propose the following heuristic exploration of the neighborhood $\mathcal{N}^o_{Swap}(X, Y)$. We consider two sets X_i and $X_{i'}$ and sort the origins in X_i (resp. $X_{i'}$) in ascending order with respect to $\Delta^o(m, i, i')$ (resp. $\Delta^o(m', i', i)$). Represent the established order by $X_i = \{m_1,\ m_2, \ldots,\ m_{|X_i|}\}$ and $X_{i'} = \{m'_1,\ m'_2, \ldots,\ m'_{|X_{i'}|}\}$, respectively. Then the procedure tries to find the best improving move by exchanging sets $L = \{m_1,\ m_2, \ldots,\ m_p\}$, $1 \le p \le |X_i|$ and $' = \{m'_1,\ m'_2, \ldots,\ m'_q\}$ $1 \le q \le |X_{i'}|$. The steps of the procedure are given in Algorithm 2. As will be shown in the computational results section, this procedure is able to find a solution which is the best solution or close to the best solution, while taking much smaller CPU time than exhaustive exploration. Henceforth, when we speak of the swap neighborhood we refer to the set of solutions inspected by the procedure in Algorithm 2.

Algorithm 2 Exploration of swap neighborhood ($o,\ X_i, X_{i'}$).

1. Sort the origins $m \in X_i$ in ascending order of the values $\Delta^o(m, i, i')$;
2. Sort the origins $m' \in X_{i'}$ in ascending order of the values $\Delta^o(m', i', i)$;
3. Set $N^o_{X_{Swap}}(X_i, X_{i'}) = \emptyset$; and $L = \emptyset$;
4. **For** each $m \in X_i$ **do**
5. $L = L + \{m\}$; $L' = \emptyset$;
6. **For** each $m' \in X_{i'}$ **do**
7. $L' = L' + \{m'\}$;
8. **If** $S(X_i) + S(L') - S(L) \le S_i$ and $S(X_{i'}) + S(L) - S(L') \le S_{i'}$ **then**
9. $(X',\ Y') = (X, Y)$;
10. $X'_i = X'_i + L' - L$;
11. $X'_{i'} = X'_{i'} + L - L'$;
12. $\mathcal{N}^o_{X_{Swap}}(X_i,\ X_{i'}) = \mathcal{N}^o_{X_{Swap}}(X_i,\ X_{i'}) + \{(X',\ Y')\}$;
13. **EndIf**
14. **EndFor**
15. **EndFor**
16. **Return** $\mathcal{N}^o_{X_{Swap}}(X_i,\ X_{i'})$;

Hence, the complexity of the procedure that explores the entire swap neighborhood of a solution (*X*, *Y*) on the side $\tau = o$ is $O(\sum_{i,\ i' \in I,\ i \neq i'} |X_i||X_{i'}| + |X_i|\log|X_i| + |X_i{}'|\log|X_{i'}|)$.

Remark 3. If there is no infeasible solution in the swap neighborhood of the current solution the heuristic procedure described in Algorithm 2 and the exhaustive exploration procedure return the same solution at the output.

Analogous results hold for the exploration of the neighborhood $\mathcal{N}^d_{Swap}(X, Y)$ and we will not bother to describe them.

3. Probabilistic Tabu Search

In this section we present the Probabilistic Tabu Search approaches we use to solve the CDAP. Probabilistic Tabu Search is a variant of the metaheuristic Tabu Search introduced by Glover (1986). The main steps of the PTS procedure for solving the CDAP are presented in Algorithm 3. Starting from an initial solution, PTS is run until a predefined stopping criterion is met. The procedure presented in Algorithm 1 is used to generate an initial solution and afterward using the following function to evaluate visited solutions At each iteration, our PTS approach constructs a candidate list $\mathcal{N}$(*X*, *Y*), selects a solution from it to be new incumbent solution, updates the tabu list *TL*, auxiliary data structures $c^o_{m,i}(Y)$ and $c^d_{n,j}(X)$ (explained in the preceding section) and the best solution found so far. To construct a candidate list $\mathcal{N}$(*X*, *Y*) and select a new incumbent solution we propose two approaches which lead to two different variants of PTS which we denote PTS1 and PTS2. In both variants the tabu list (*TL*) (referred to as short term memory in the original tabu search approach) is managed in the simplest way. The old incumbent solution (*X*, *Y*) is added to the tabu list and if the size of the list is greater than *l*, the oldest solution in *TL*, added before the *l* most recent iterations, is deleted.

Algorithm 3 Probabilistic Tabu Search: general framework.

1. Generate an initial solution (*X*, *Y*) using the procedure in Algorithm 1;
2. Assign any non-assigned trucks in a greedy way using the function *g*(*X*,*Y*);
3. Set $(X^*, Y^*) = (X, Y)$; $TL = \emptyset$;
4. **While** a stopping criterion is not met **do**
5. **If** (X^*, Y^*) is feasible **then** *F*(*X*, *Y*)=*f*(*X*,*Y*);
6. **Else** *F*(*X*, *Y*)=g(*X*,*Y*);
7. $\mathcal{N}(X, Y)$ = Construct_candidate_list(*X*, *Y*, *TL*);
8. (X, Y) = Select_solution ($\mathcal{N}(X, Y)$, *F*(*X*, *Y*));
9. Update matrices $c^o_{mi}(Y)$ and $c^d_{nj}(X)$;
10. Update tabu list *TL*;
11. $(X'', Y'') = argmin\{F(X', Y') : (X', Y') \in \mathcal{N}(X, Y)\}$;
12. $(X^*, Y^*) = argmin\{F(X'', Y''), F(X^*, Y^*)\}$;
13. **EndWhile**
14. **Return** (X^*, Y^*);

3.1. Probabilistic Tabu Search: Variant 1

The first PTS variant, denoted PTS1, constructs a candidate list $\mathcal{N}$(*X*, *Y*) by Algorithm 4. The procedure first selects side $\tau \in \{o, d\}$ at random. After that, it constructs a candidate list of size μ, selecting half of the solutions from the shift neighborhoods $\mathcal{N}^{\tau,k}_{Shift}(X, Y)$ and half of the solutions from the swap neighborhood $\mathcal{N}_{Z^\tau_{Swap}}(Z_i,\ Z_{i'})$ (where $\mathcal{N}_{Z^\tau_{Swap}}(Z_i,\ Z_{i'})$ corresponds either to $\mathcal{N}_{X^o_{Swap}}(X_i,\ X_{i'})$ or $\mathcal{N}_{Y^d_{Swap}}(Y_j,\ Y_{j'})$ depending on the chosen side τ). Solutions from the neighborhoods are chosen based on a random variable *p* generated in [0, 1]: if $p \in [0, 0.6]$ the best solution

Algorithm 4 Candidate list construction in PTS1.

Function Construct_candidate_list ((*X, Y*), μ, *b, TL*)
1. $\mathcal{N}(X,Y) = \emptyset$;
2. Select a side $\tau \in \{o, d\}$ at random;
3. **For** 1 to $\mu/2$ **do**
4. $p = random(0,1)$;
5. **If** $p \in [0, 0.6]$ **then** $k^* = 1$;
6. **If** $p \in]0.6, 0.8]$ **then** $k^* = b$;
7. **If** $p \in]0.8, 1]$ and $\tau = o$ **then** $k^* = |I|$;
8. **If** $p \in]0.8, 1]$ and $\tau = d$ **then** $k^* = |J|$;
9. Select a random solution $(X', Y') \in \mathcal{N}^{\tau,k^*}_{Shift}(X,Y) - TL$;
10. $\mathcal{N}(X,Y) = \mathcal{N}(X,Y) + (X', Y')$;
11. EndFor
12. **For** 1 to $\mu/2$ **do**
13. **If** $\tau = o$ **then** $(Z, Z') = (X_i, X_{i'})$, $i \neq i'$, $(X_i, X_{i'})$ chosen at random;
14. **If** $\tau = d$ **then** $(Z, Z') = (Y_j, Y_{j'})$, $j \neq j'$, $(Y_j, Y_{j'})$ chosen at random;
15. $p = random(0,1)$;
16. **If** $p \in [0, 0.6]$ **then** Select the best solution $(X', Y') \in N^o_{Z_{Swap}}(Z, Z') - TL$;
17. **If** $p \in]0.6, 0.8]$ **then** Among *b* best select a random $(X', Y') \in N^o_{Z_{Swap}}(Z, Z') - TL$;
18. **If** $p \in]0.8, 1]$ **then** Select a random solution $(X', Y') \in N^o_{Z_{Swap}}(Z, Z') - TL$;
19. $\mathcal{N}(X,Y) = \mathcal{N}(X,Y) + (X', Y')$;
20. **EndFor**
21. **Return** $\mathcal{N}(Y,Y)$;

from the neighborhood is chosen, if $p \in]0.6, 0.8]$ a solution among the *b* best ones is chosen, and finally if $\boldsymbol{p} \in]0.8, 1]$ a random solution is chosen. The procedure considers solutions to be admissible only if they are not in the tabu list *TL*.

In our implementation, we do not use any auxiliary data structure or procedure to avoid repetition of solutions in the candidate list. The reason is that the size of the candidate list is chosen to be much smaller than the size of the pool of candidate solutions (see the computational results in Section 4) and therefore the probability of having repeated solutions is very small. Moreover, the use of an auxiliary data structure or procedure would slow down the proposed heuristics.

To choose a new incumbent solution, PTS1 uses the procedure in Algorithm 5, which selects the best solution in the candidate list as the new incumbent.

Algorithm 5 Solution selection in PTS1.

Procedure Select_solution($\mathcal{N}$(*X, Y*), *F*(*X, Y*))
1. $(X'', Y'') = \text{argmin}\{F(X', Y') : (X', Y') \in \mathcal{N}(X,Y)\}$
2. **Return** (X'', Y'');

3.2. Probabilistic Tabu Search: Variant 2

The second Probabilistic Tabu Search variant, denoted PTS2, constructs a candidate list $\mathcal{N}(X,Y)$ by Algorithm 6 below. The procedure first chooses a side $\tau \in \{o, d\}$, at random, as a basis for building the candidate list. Then it adds to the candidate list solutions from the neighborhood $\mathcal{N}^{\tau,1}_{Shift}(X,Y)$. If there is no improving solution available to be added, it proceeds by adding solutions from the neighborhood $\mathcal{N}^{\tau,1,1}_{Swap}(X, Y)$. If still no improving solutions exist to be added, it adds to the candidate list the best solutions from the swap neighborhood $\mathcal{N}^{\tau}_{Swap}(X, Y)$, which corresponds either to $\mathcal{N}^{o}_{Swap}(X, Y)$ or $\mathcal{N}^{d}_{Swap}(X, Y)$ depending on the chosen side τ. As in the first variant, the procedure considers solutions to be admissible only if they are not in the tabu list *TL*.

To select a new incumbent solution, PTS2 uses Algorithm 7. The procedure first sorts the neighboring solutions of the solution (*X*, *Y*) in increasing order with respect to the objective function (Step 1). Then if the set of the improving neighboring solutions $\mathcal{N}^*(X,Y)$ is not empty we set $\eta^* = \min(|\mathcal{N}^*(X,Y)|, b)$ and

Algorithm 6 Candidate list construction in PTS2.

Procedure Construct_candidate_list(*X, Y, TL*)
1. $\mathcal{N}(X,Y) = \emptyset$;
2. Select a side $\tau \in \{o, d\}$ at random;
3. $\mathcal{N}(X,Y) = \mathcal{N}^{\tau,1}_{Shift}(X,Y) - TL$;
4. **If** no improving solution is available in $\mathcal{N}(X,Y)$ **then**
5. $\mathcal{N}(X,Y) = \mathcal{N}(X,Y) + \mathcal{N}^{\tau,1,1}_{Swap}(X, Y) - TL$;
6. **Endif**
7. **If** no improving solution is available in $\mathcal{N}(X,Y)$ **then**
8. **// multiple swap moves**
9. **If** $\tau = o$ **then** $L = \{(X_i, X_{i'}) : i, i' \in I, i \neq i'\}$;
10. **If** $\tau = d$ **then** $L = \{(Y_j, Y_{j'}) : j, j' \in J, j \neq j'\}$;
11. **For** each pair $(Z, Z') \in L$ **do**
12. Select the best solution $(X', Y') \in \mathcal{N}^{\tau}_{Z_{Swap}}(Z, Z') - TL$;
13. $\mathcal{N}(X,Y) = \mathcal{N}(X,Y) + \{(X', Y')\}$;
14. EndFor
15. EndIf
16. **Return** $\mathcal{N}(X,Y)$;

otherwise set $\eta^* = b$. After this, in Step 5 the procedure selects at random one of the η^* best solutions in the candidate list to be new incumbent solution. This means that in the case where improving solutions exist the choice is made among at most *b* best improving solutions. On the other hand, if there is no improving solutions, the choice is made among exactly the *b* best (non-improving solutions) in the candidate list.

Algorithm 7 Solution selection in PTS2.

Procedure Select_solution ($\mathcal{N}(X,Y)$, *F*(*X, Y*), *b*)
1. Sort solutions in $\mathcal{N}(X,Y)$ in increasing order with respect to the function *F*(*X, Y*), i.e., $F(X'^1, Y'^1) \leq F(X'^2, Y'^2) \leq \ldots \leq F(X'^\eta, Y'^\eta)$, where $\eta = |\mathcal{N}(X, Y)|$;
2. Let $\mathcal{N}^*(X,Y) = \{(X'^k, Y'^k) : F(X'^k, Y'^k) < F(X,Y)\}$
3. **If** $\mathcal{N}^*(X,Y) \neq \emptyset$ **then** $\eta^* = \min(|\mathcal{N}^*(X,Y)|, b)$;
4. **Else** $\eta^* = b$;
5. Select a solution (X'', Y'') randomly from the set $\{(X'^1, Y'^1), (X'^2, Y'^2), \ldots, (X'^{\eta^*}, Y'^{\eta^*})\}$
6. **Return** (X'', Y'');

4. Computational Results

In this section, we first compare the results of exhaustive and heuristic exploration of the swap neighborhood. The goal is to show that our heuristic exploration yields a good trade-off between solution quality and CPU time in comparison with exhaustive exploration. Following this, we compare our methods with the state-of-the art methods from the literature. Our approaches were implemented in Java and executed on a PC with 16 gigabytes of RAM and using an Intel Xeon E3-1505 M v5 processor with 2.80 gigahertz. For testing purposes, two benchmark data sets were used: the first one proposed in Guignard et al. (2012) and the second proposed subsequently by the same authors.

Guignard et al. (2012) generated the set of instances in the following way. The number of origins is equal to the number of destinations and is chosen from the set {8, 9, 10, 11, 12, 15, 20}. Similarly, the number of inbound doors is equal to the number of outbound doors and is selected from the set {4, 5, 6, 7, 10}. The distance between two doors is chosen from the interval [8, 8+|*I*|] depending on the position of the two doors in the facility. The distance between two doors that are face-to-face is set to 8, while at each successive door the distance is incremented by one unit. The flow matrix is generated by setting the values of 25% of the elements of matrix $f_{m,n}$ to be random integers from the interval [10, 50], and setting the values of remaining elements to 0. This is done so that each destination receives products from at least one origin and each origin sends products to at least one destination. All doors are given the same capacity determined as follows as a

Table 1
Heuristic Vs. Exhaustive exploration of swap neighborhood.

Data Set	Heuristic			Exhaustive			%dev	#same
	CPU(ms)	Value	#solutions	CPU(ms)	Value	#solutions		
SetA	0.00	11,419.04	65.84	0.40	11,416.80	331.76	0.020	33
SetB	3.27	562,505.60	2542.98	11,960.76	562,459.90	51,537,734.50	0.008	38

specified fraction of total incoming flow and the number indoors increased by the capacity slack. The capacity slack is calculated as c% of the fraction of total incoming flow and the number indoors, where $c \in \{5, 10, 15, 20, 30\}$.

Later, the authors generated a new set of large-scale instances in the same manner. In the newly generated instances, the number of origins/destinations is chosen from {25, 50, 75, 100} and the number of indoors/outdoors is chosen from {10, 20, 30, 43}. The first set of test problems is referred to as "SetA" and contains 50 instances, while the second (large-scale) set is denoted "SetB" and contains 49 instances. The name of each instance has the format 00 $\times$ 00S00, where the first 00 refers to the number of origins/destinations, the second 00 after x refers to the number of inbound/outbound doors and the last 00 after S is the slack. For example, the instance name 8×4S30 refers to an instance with eight origins, eight destinations, four inbound doors, four outbound doors and slack equal to 30%.

4.1. Comparison of exhaustive and heuristic exploration of swap neighborhood

In order to highlight the advantage of using our proposed heuristic exploration of the swap neighborhood presented in Algorithm 2, as contrasted to exhaustive exploration, we perform the following test. On each test instance we generate an initial feasible solution using Algorithm 1 and perform heuristic and exhaustive exploration of the swap neighborhood starting from this solution. For comparison purposes we store the best solution value found, the CPU time consumed (in milliseconds) and the number of solutions evaluated by both approaches. Table 1 presents the averages of these values over the SetA and SetB instances (Columns 'CPU', 'value' and '#solutions'). In addition, we report the average percentage deviations of solution values found by heuristic exploration from those found by exhaustive exploration (Column '% dev.'), and the number of instances in each data sets where heuristic and exhaustive exploration reach the same value (Column '#same.'). The outcomes show that the heuristic exploration is significantly faster than the exhaustive one as a result of evaluating significantly fewer solutions (as expressed in the proposition of Section 2.2.4). Despite evaluating fewer solutions, it is able to find solutions whose quality is only slightly worse than that obtained by exhaustive exploration, as evidenced by the fact that the average percentage deviations are 0.02% and 0.008% on setA and setB, respectively. In addition, it should be emphasized that on 71 out of 99 instances these two approaches return the same solution as final.

4.2. Comparison with the methods from the literature

As a basis for comparison, we refer to the following four leading heuristics from the literature: two local search based heuristics, named LS1 and LS2, the Convex Hull Relaxation (CHR) heuristic proposed in Guignard et al. (2012) and the Lagrangean relaxation (LR) heuristic proposed in Nassief et al. (2016).

After some tuning, the parameters of our algorithms are set in the following way. Both PTS1 and PTS2, use a stopping criterion that limits the number of iterations performed. For both methods, the limiting number is set to 10^5 on SetA and to 2×10^5 on SetB. The parameter b of the selection procedures is set to 3 and the size of the tabu list (TL) is set to $(|M|+|N|+|I|+|J|)/16$. For PTS1, the size μ of the neighborhood $\mathcal{N}(X, Y)$ is set to $(|M|+|N|)/2$. On each instance, our PTS heuristics are executed 10 times using different random seeds.

In Tables 2 and 3, we compare the results of PTS1 and PTS2 on SetA with the best-known solution (BKS) values reported in Guignard et al. (2012) and Nassief et al. (2016). The BKS values in Table 2 are found by the LS1, LS2, CHR, CPLEX and LR heuristics, while the BKS values in Table 3 are found by LS1 and LS2. Tables 2 and 3 provide summary results over test classes while detailed results may be found in the Appendix. By convention, the test class is formed by Instances with the same number of origins/destinations and inbound/outbound doors. Therefore, the headings of Tables 2 and 3 are defined as follows. The number of origins/destinations and inbound/outbound doors in each class is given in the first column in the form $|N| \times |I|$. The second column is dedicated to BKS values. In columns two and three we present the CPU time needed for CPLEX to solve the recent best MIP formulations for CDAP, where column two (Column 'MIP1') is taken from Nassief et al. (2016) and column three (Column 'MIP2') is taken from Gelareh et al. (2018). Remaining columns report the results of our heuristics. On each instance our heuristics were executed 10 times recording the best solution value and the average solution value found in 10 runs, and the average CPU time spent in solving the instance. The averages of these values over the instances from the same test class are reported in Columns 'Best', 'Avg.', and 'CPU', respectively.

In Table 4, we report the total number of instances, over each data set, where the first approach in the comparison provides better, equal or worse solutions than the second approach in the comparison. For example, under the header PTS1 vs BKS, we provide the total numbers of instances where PTS1 offers better (Columns 'Best'), equal (Columns 'Equal'), and worse (Columns 'Worse') solution than BKS.

From the results presented for SetA, we see that both algorithms, PTS1 and PTS2, only fail to reach the best-known solution value previously reported in the literature on a single instance (i.e., 20×10S15), while they both establish new best-known solution values for four instances. Regarding CPU-time, we observe that PTS2 is more than 2 times faster on average than PTS1. However, the average solution values found by PTS1 are in general better than those of PTS2 and sometimes better than the previously found best-known solution values (see Table 5 in the Appendix, for the instances 20×10S5, 20×10S10, 20×10S20, 20×10S30). Furthermore, the results reported in Gelareh et al. (2018) and Nassief et al. (2016) show that instances with up to 15 origins/destinations and 7 indoors/outdoors are optimally solved by the CPLEX MIP solver within the maximum of 492 seconds. On these instances, the PTS algorithms were able to reach all optimal solutions in less than 8 seconds. Moreover, for instances with 20 origins/destinations and 10 indoors/outdoors, CPLEX did not reach optimal solutions in two hours while the PTS algorithms provide better results in less than 10 seconds. This comparison with CPLEX results indicates the merit of using Probabilistic Tabu Search for solving hard optimization problem such as CDAP.

Table 2
Summary results on “SetA” instances.

$\|N\|x\|I\|$	BKS	MIP1	MIP2	PTS1			PTS2		
		CPU(s)	CPU(s)	Best	Avg.	CPU(s)	Best	Avg.	CPU(s)
8 × 4	5120.8	0.35	0.15	5120.8	5120.8	3.50	5120.8	5120.8	1.10
9 × 4	5978.2	0.54	0.21	5978.2	5978.2	4.04	5978.2	5978.2	1.03
10 × 4	6319.8	0.93	0.38	6319.8	6319.8	4.61	6319.8	6319.8	1.10
10 × 5	6427.8	3.17	1.00	6427.8	6427.8	4.20	6427.8	6427.8	1.46
11 × 5	7555.6	4.24	1.64	7555.6	7555.6	4.73	7555.6	7555.9	1.55
12 × 5	7970.2	14.32	3.67	7970.2	7970.2	5.29	7970.2	7970.2	1.53
12 × 6	10,449.8	80.48	26.09	10,449.8	10,449.8	4.84	10,449.8	10,453.1	2.06
15 × 6	13,756.4	622.78	132.93	13,756.4	13,756.4	6.55	13,756.4	13,773.0	2.29
15 × 7	14,688.8	3843.62	1044.73	14,688.8	14,688.8	6.13	14,688.8	14,703.0	2.81
20 × 10	29,171.4	7200.00	7200.00	29,151.2	29,157.8	7.89	29,151.2	29,342.0	5.17
Average	**10,743.88**	**768.48**	**745.35**	**10,741.86**	**10,742.53**	**5.18**	**10,741.86**	**10,764.39**	**2.01**

Table 3
Summary results on “SetB” instances.

$\|N\|x\|I\|$	BKS	PTS1			PTS2		
		Best	Avg.	CPU(s)	Best	Avg.	CPU(s)
25 × 10	48,446.0	48,268.0	48,287.1	21.68	48,280.0	48,341.0	12.58
25 × 20	51,741.0	51,533.0	51,618.4	19.81	51,562.0	52,334.2	28.51
50 × 10	187,945.4	187,395.0	187,551.5	72.99	187,469.0	188,446.8	26.26
50 × 20	230,622.2	229,566.2	230,233.4	48.71	231,038.0	233,009.4	49.41
50 × 30	264,322.3	262,510.0	263,745.3	46.43	265,606.0	268,292.8	84.85
50 × 43	330,661.0	330,285.0	332,378.3	47.46	335,036.0	341,233.4	121.07
75 × 10	431,150.2	429,874.2	430,538.9	172.98	430,845.2	432,162.1	44.84
75 × 20	513,604.6	511,545.2	512,406.4	89.49	514,356.6	518,236.2	73.03
75 × 30	608,476.0	605,108.0	606,868.6	83.37	611,928.0	616,832.8	111.24
100 × 10	756,508.0	754,670.6	755,528.7	352.63	755,725.0	757,630.9	68.74
100 × 20	933,612.6	929,704.0	931,873.7	153.61	934,695.4	939,120.4	103.30
100 × 30	1,113,857.0	1,102,169.2	1,105,648.0	130.49	1,114,188.4	1,122,055.4	143.05
Average	**504,253.51**	**501,137.59**	**502,307.05**	**117.41**	**504,369.71**	**507,363.03**	**70.51**

Table 4
Comparison of methods in terms of solution quality.

Data Set	PTS1 vs BKS			PTS2 vs BKS			PTS1 vs PTS2		
	Better	Equal	Worse	Better	Equal	Worse	Better	Equal	Worse
SetA	4	45	1	4	45	1	0	50	0
SetB	49	0	0	27	0	22	43	3	3
All	**53**	**45**	**1**	**31**	**45**	**23**	**43**	**53**	**3**

On the other hand, on SetB, PTS1 outperforms the state-of-the-art methods, LS1 and LS2, in finding the best-found solution. On several instances even the average solution values reported by PTS1 are better than the best solution values found by LS1 and LS2 (see Table 6 in the Appendix). Comparing the best solutions found by PTS1 and PTS2, we see that PTS2 is better than PTS1 on 3 instances, ties with PTS1 on 3 instances, while on the remaining 43 instances PTS1 is better than PTS2. Comparing the average solution values of PTS1 and PTS2, we see that PTS1 outperforms PTS2 on 48 out of 49 instances (see Table 6 in the Appendix). We also see that PTS2 performs very well on instances with 10 indoors/outdoors where it obtains results very close to those of PTS1, while consuming very little CPU time compared to PTS1. In addition, compared to the BKS method, PTS2 provides better solutions on 27 instances out of 49 instances.

Previous findings indicate that PTS1 outperforms the other approaches in terms of solution quality. In order to check if this superior performance is significant or not, we perform the Wilcoxon signed rank test (Wilcoxon, 1945). The resulting *p*-values of 1.1101×10^{-9}, $1.1101 \times 10^{-7,}$ and 6.7951×10-9 when comparing PTS1 vs LS1, PTS1 vs LS2, and PTS1 vs PTS2, respectively, reveal the significance of the differences that establish the superiority of PTS1.

Comparing the running times of PTS1 and PTS2, we see that: PTS2 is better on instances with 10 doors (instances with reduced neighborhood structure); the methods are very similar on instances with 20 doors (where sometimes PTS1 is better than PTS2); PTS1 in general outperforms PTS2 in terms of running time on instances with 30 doors and on one instance with 43 doors, even though the neighborhood considered in PTS1 is much larger than the neighborhood considered in PTS2. On the other hand, the running times of our methods are much better than those of LS1 and LS2 (see Table 6 in the Appendix).

In the light of these results, we conclude that PTS2 is more suitable for small problems, while PTS1 is more suitable for large problems. We conjecture that the good performance of PTS1 on large instances may be explained by the fact that it explores a smaller neighborhood than PTS2, which enables it to achieve a better tradeoff between intensification and diversification than PTS2 within the same amount of time.

5. Conclusion

In this study, two Probabilistic Tabu Search heuristics have been presented to tackle the Cross-Docking Assignment Problem (CDAP) which has important applications in supply chain management. The main differences between these methods are embodied in the ways they generate candidate lists and accept solutions in each iteration. Our methods are implemented in an innovative manner using a new large swap neighborhood structure that is useful for solving either the CDAP or similar problems. In addition, a supporting heuristic is proposed to explore the large swap neighborhood efficiently. The merit of our proposed algorithms is assessed by comparing them with the most effective methods from the literature on two established benchmark data sets. Computational tests disclose that our approaches significantly outperform the previously proposed methods by reaching 45 previous best-known solutions and establishing 53 new best-known solutions over the full set of 99 instances. In addition, the CPU time consumed by our approaches is substantially less than consumed by the previous state-of-the art methods.

Future work will examine ways to exploit advantages of each of the proposed PTS heuristics by combining their best features within one schema to yield other variants for solving CDAP. We plan to deal with extended versions of CDAP by considering the arrival and departure times of trucks within the context of truck scheduling. Finally, we envision that benefits will accrue from future work that studies the impact of cross-docking facilities on vehicle scheduling in a supply chain by combining CDAP with the vehicle routing problem.

Acknowledgments

This research has been supported in part by the Key Laboratory of International Education Cooperation of Guangdong University of Technology. We are indebted to Monique Guignard for providing us with benchmark instances as well as with results of two local search heuristics on SetB instances.

Appendix

In Tables 5 and 6, we compare results found by our approaches with those reported in the literature by other methods. For PTS1 and PTS2 we report the best solution value (Column ‘Best’) and the average solution value found in 10 runs (Column ‘Avg.’) as well the average CPU time (Column ‘CPU’). In Table 5, together with the best known solutions (Column ‘BKS’), we report the CPU time needed for CPLEX to solve the recent best MIP formulations for CDAP. The first one (Column ‘MIP1’) is taken from Nassief et al. (2016) and the second one (Column ‘MIP2’) is taken from Gelareh et al. (2018). The results reported show that instances with up to 15 origins/destinations and 7 indoors/outdoors are optimally solved by CPLEX MIP solver within the maximum time limit of 492 seconds. On these instances, PTS algorithms were able to reach all optimal solutions in less than 8 seconds. Moreover, for instances with 20 origins/destinations and 10 indoors/outdoors, CPLEX did not reach optimal solutions in two hours while the PTS algorithms provide better results in less than 10 seconds. This comparison with CPLEX results, underscores the value of using Probabilistic Tabu Search for solving hard optimization problem such as CDAP.

In Table 6, The results are compared with those of the LS1 and LS2 heuristics proposed by Guignard et al. (2012), which are the only two methods executed on the SetB instances so far. For LS1 and LS2 we report the best solution value (Column ‘Cost’) and the normalized CPU time. Since LS1 and LS2 were executed on a machine with an AMD Phenom 9600 processor with 2.31 gigahertz, a machine with different characteristics than our machine, we normalize their running times using the approach described in Dongarra (2014) and data from http://www.cpubenchmark.net/. All comparisons were made according to the Passmark CPU Score (PCPUS). The running times were normalized by using our machine as the reference point, i.e., Norm.Time(Algo) = PCPUS (AMD Phenom 9600) × Time(Algo) /PCPUS(Intel Xeon E3-1505 M), where Algo refers to LS1 & LS2. The Passmark CPU Scores of the processors AMD Phenom 9600 and Intel Xeon E3-1505 M are 2303 and 8978, respectively.

In each table, the boldfaced values correspond to the values that are equal or better than current BKS values, while underlined values denote the new BKS values established by our PTS. The values in italics correspond to the optimal solution values as reported in Gelareh et al. (2018).

Table 5
Detailed results on “SetA” instances.

Instance	BKS	MIP1	MIP2	PTS1			PTS2		
		CUP(s)	CPU(s)	Best	Avg	CPU (s)	Best	Avg	CPU (s)
8 × 4S5	*5174*	0.25	0.28	**5174**	**5174.0**	3.87	**5174**	**5174.0**	1.69
8 × 4S10	*5169*	0.25	0.22	**5169**	**5169.0**	3.68	**5169**	**5169.0**	1.12
8 × 4S15	*5112*	0.22	0.23	**5112**	**5112.0**	3.46	**5112**	**5112.0**	0.98
8 × 4S20	*5086*	0.1	0.16	**5086**	**5086.0**	3.29	**5086**	**5086.0**	0.87
8 × 4S30	*5063*	0.25	0.19	**5063**	**5063.0**	3.20	**5063**	**5063.0**	0.83
9 × 4S5	*6047*	0.42	0.23	**6047**	**6047.0**	4.50	**6047**	**6047.0**	1.25
9 × 4S10	*6027*	0.39	0.3	**6027**	**6027.0**	4.11	**6027**	**6027.0**	1.09
9 × 4S15	*5976*	0.28	0.14	**5976**	**5976.0**	3.93	**5976**	**5976.0**	0.92
9 × 4S20	*5937*	0.37	0.12	**5937**	**5937.0**	3.90	**5937**	**5937.0**	0.97
9 × 4S30	*5904*	0.42	0.23	**5904**	**5904.0**	3.74	**5904**	**5904.0**	0.90
10 × 4S5	*6518*	0.78	0.53	**6518**	**6518.0**	5.10	**6518**	**6518.0**	1.38
10 × 4S10	*6325*	0.45	0.33	**6325**	**6325.0**	4.87	**6325**	**6325.0**	1.09
10 × 4S15	*6296*	0.42	0.25	**6296**	**6296.0**	4.55	**6296**	**6296.0**	0.98
10 × 4S20	*6267*	0.53	0.31	**6267**	**6267.0**	4.40	**6267**	**6267.0**	0.97
10 × 4S30	*6193*	0.47	0.3	**6193**	**6193.0**	4.13	**6193**	**6193.0**	1.07
10 × 5S5	*6616*	1.62	0.84	**6616**	**6616.0**	4.54	**6616**	**6616.0**	1.80
10 × 5S10	*6476*	1.41	0.78	**6476**	**6476.0**	4.45	**6476**	**6476.0**	1.51
10 × 5S15	*6397*	1.09	0.78	**6397**	**6397.0**	4.14	**6397**	**6397.0**	1.37
10 × 5S20	*6342*	0.94	0.94	**6342**	**6342.0**	4.02	**6342**	**6342.0**	1.32
10 × 5S30	*6308*	0.84	0.5	**6308**	**6308.0**	3.86	**6308**	**6308.0**	1.31
11 × 5S5	*7812*	3,00	1.88	**7812**	**7812.0**	5.23	**7812**	**7812.0**	2.12
11 × 5S10	*7572*	1.92	1.66	**7572**	**7572.0**	4.95	**7572**	**7572.0**	1.61
11 × 5S15	*7535*	2.03	1.64	**7535**	**7535.0**	4.76	**7535**	**7535.0**	1.36
11 × 5S20	*7439*	1.34	0.86	**7439**	**7439.0**	4.47	**7439**	**7439.0**	1.29
11 × 5S30	*7420*	1.42	0.97	**7420**	**7420.0**	4.23	**7420**	7421.6	1.36

(*continued on next page*)

Table 5 (*continued*)

Instance	BKS	MIP1	MIP2	PTS1			PTS2		
		CUP(s)	CPU(s)	Best	Avg	CPU (s)	Best	Avg	CPU (s)
12 × 5S5	*8072*	3.27	3.3	**8072**	**8072.0**	5.81	**8072**	**8072.0**	1.76
12 × 5S10	*7978*	2.98	2.73	**7978**	**7978.0**	5.53	**7978**	**7978.0**	1.58
12 × 5S15	*7939*	2.55	2.28	**7939**	**7939.0**	5.36	**7939**	**7939.0**	1.48
12 × 5S20	*7939*	3.09	2.24	**7939**	**7939.0**	5.06	**7939**	**7939.0**	1.43
12 × 5S30	*7923*	5.98	3.22	**7923**	**7923.0**	4.68	**7923**	**7923.0**	1.40
12 × 6S5	*10,891*	35.92	4.98	**10,891**	**10,891.0**	5.36	**10,891**	**10,891.0**	2.72
12 × 6S10	*10,456*	8.87	12.05	**10,456**	**10,456.0**	5.13	**10,456**	**10,456.0**	1.98
12 × 6S15	*10,362*	8.33	7.05	**10,362**	**10,362.0**	4.89	**10,362**	10,378.5	1.82
12 × 6S20	*10,312*	6.70	9.45	**10,312**	**10,312.0**	4.61	**10,312**	**10,312.0**	1.92
12 × 6S30	*10,228*	8.92	26.95	**10,228**	**10,228.0**	4.22	**10,228**	**10,228.0**	1.84
15 × 6S5	*13,927*	107.78	92.03	**13,927**	**13,927.0**	7.32	**13,927**	**13,927.0**	2.47
15 × 6S10	*13,803*	112.56	24.39	**13,803**	**13,803.0**	7.02	**13,803**	13,810.7	2.20
15 × 6S15	*13,765*	158.27	96.47	**13,765**	**13,765.0**	6.54	**13,765**	13,792.3	2.23
15 × 6S20	*13,720*	112.75	68.5	**13,720**	**13,720.0**	6.17	**13,720**	13,750.0	2.31
15 × 6S30	*13,567*	149.90	26.39	**13,567**	**13,567.0**	5.68	**13,567**	13,585.2	2.22
15 × 7S5	*15,054*	492,00	245.17	**15,054**	**15,054.0**	6.90	**15,054**	15,063.1	3.36
15 × 7S10	*14,810*	313.52	208.47	**14,810**	**14,810.0**	6.49	**14,810**	14,843.1	2.70
15 × 7S15	*14,657*	303.81	283.13	**14,657**	14,657.2	6.16	**14,657**	14,658.2	2.68
15 × 7S20	*14,514*	259.12	62.94	**14,514**	**14,514.0**	5.80	**14,514**	14,537.6	2.72
15 × 7S30	*14,409*	306.20	70.95	**14,409**	**14,409.0**	5.31	**14,409**	14,413.2	2.61
20 × 10S5	29,933	7200.00	7200.00	**29,907**	**29,909.6**	9.16	**29,907**	30,004.4	6.39
20 × 10S10	29,286	7200.00	7200.00	**29,236**	**29,253.3**	8.49	**29,236**	29,567.3	5.01
20 × 10S15	***29,134***	7200.00	7200.00	29,135	29,135.5	7.77	29,135	29,345.7	4.81
20 × 10S20	28,963	7200.00	7200.00	**28,945**	**28,951.2**	7.18	**28,945**	29,051.5	4.81
20 × 10S30	28,541	7200.00	7200.00	**28,533**	**28,539.6**	6.85	**28,533**	28,741.3	4.82
AVG	**10,743.88**	**768.48**	**745.35**	**10,741.86**	**10,742.53**	**5.18**	**10,741.86**	**10,764.39**	**2.01**

Table 6
Detailed results on "SetB" instances.

Instance	LS1		LS2		PTS1			PTS2		
	Cost	CPU(s)	Cost	CPU(s)	Best	AVG	CPU(s)	Best	AVG	CPU(s)
25 × 10S5	49,144	49.76	49,335	50.79	**49,013**	**49,014.8**	26.11	**49,013**	**49,013.0**	13.81
25 × 10S10	48,949	29.76	48,941	33.09	**48,672**	**48,699.3**	23.49	**48,740**	**48,869.5**	12.50
25 × 10S15	48,556	31.29	48,504	33.60	**48,407**	**48,415.6**	20.08	**48,407**	**48,477.0**	12.10
25 × 10S20	48,215	21.80	48,235	23.09	**47,934**	**47,949.3**	19.11	**47,926**	**47,977.6**	12.24
25 × 10S30	47,480	19.75	47,426	20.78	**47,314**	**47,356.5**	19.61	**47,314**	**47,368.1**	12.25
25 × 20S30	51,921	46.43	51,741	51.30	**51,533**	**51,618.4**	19.81	**51,562**	52,334.2	28.51
50 × 10S5	191,773	1036.07	191,788	1153.30	**191,160**	**191,241.3**	78.29	**191,186**	192,350.7	26.58
50 × 10S10	189,409	1371.08	189,833	1362.87	**189,166**	189,478.5	66.16	189,573	190,316.3	26.19
50 × 10S15	188,006	862.66	188,264	800.33	**187,315**	**187,417.9**	69.16	**187,377**	188,834.9	26.45
50 × 10S20	186,800	988.61	186,578	956.55	**186,085**	**186,246.2**	72.28	**185,975**	186,788.2	26.24
50 × 10S30	183,961	423.76	184,013	401.96	**183,249**	**183,373.7**	79.04	**183,234**	**183,943.9**	25.85
50 × 20S5	238,048	1814.59	239,673	1874.62	**237,656**	238,244.5	59.28	239,835	241,379.3	50.66
50 × 20S10	235,178	2020.58	234,807	1985.69	**233,341**	**234,045.4**	44.56	235,326	236,932.9	51.59
50 × 20S15	230,758	2004.67	230,666	1921.30	**229,638**	**230,429.2**	45.58	**230,375**	233,110.0	48.04
50 × 20S20	227,698	1836.91	227,883	1879.75	**226,448**	**227,134.2**	45.65	228,332	230,201.4	50.93
50 × 20S30	221,892	1831.01	222,060	1783.04	**220,748**	**221,313.8**	48.48	**221,322**	223,423.5	45.85
50 × 30S15	275,973	1532.68	275,121	1782.79	**273,166**	**274,455.9**	50.52	276,938	278,798.1	91.23
50 × 30S20	264,199	632.82	263,790	783.91	**261,725**	**263,099.9**	43.55	264,729	267,802.5	86.29
50 × 30S30	254,056	481.99	254,795	503.28	**252,639**	**253,680.2**	45.23	255,151	258,277.8	77.02
50 × 43S30	332,318	1789.71	330,661	1893.34	**330,285**	332,378.3	47.46	335,036	341,233.4	121.07
75 × 10S5	440,248	2337.89	440,420	2423.05	**439,055**	**439,478.3**	158.11	**440,181**	441,088.8	44.82
75 × 10S10	435,985	2268.11	435,970	2354.30	**434,275**	**435,134.7**	157.91	**435,595**	437,051.9	44.88
75 × 10S15	431,405	2162.17	431,686	2185.77	**430,005**	**430,781.3**	171.16	**430,663**	432,183.0	45.28
75 × 10S20	427,468	2250.16	427,522	2084.70	**426,385**	**427,176.8**	179.44	**427,168**	428,920.9	44.74
75 × 10S30	420,851	1732.51	420,660	1704.04	**419,651**	**420,123.3**	198.28	**420,619**	421,566.1	44.48
75 × 20S5	531,762	2096.76	532,873	2243.49	**529,131**	**529,964.2**	86.09	532,968	535,724.6	72.92
75 × 20S10	523,447	2377.65	521,970	2479.23	**520,127**	**521,708.1**	86.09	524,001	526,829.3	72.91
75 × 20S15	514,760	2508.21	514,981	2466.14	**512,170**	**512,788.7**	87.11	515,098	519,040.7	73.93
75 × 20S20	506,346	2210.40	506,114	2512.06	**504,502**	**505,141.8**	90.33	506,313	511,407.4	72.29
75 × 20S30	493,417	2449.47	493,977	2527.96	**491,796**	**492,429.4**	97.83	**493,403**	498,179.1	73.09
75 × 30S10	636,697	1769.45	634,304	1792.02	**630,259**	**631,982.2**	78.17	637,957	644,569.3	113.86
75 × 30S15	620,356	1919.00	618,688	1921.56	**613,001**	**614,840.5**	81.81	621,149	626,365.0	112.66
75 × 30S20	600,422	2230.92	601,199	2151.91	**599,329**	**601,011.5**	84.85	605,055	610,181.4	114.44
75 × 30S30	580,490	2369.18	582,766	2349.43	**577,843**	**579,640.3**	88.66	583,551	586,215.3	103.99
100 × 10S5	773,971	1429.05	773,498	1369.54	**771,172**	**771,976.4**	319.05	**771,368**	774,128.9	68.89
100 × 10S10	764,866	1288.74	763,908	1384.93	**762,282**	**763,320.3**	311.69	**763,469**	765,036.1	69.55
100 × 10S15	757,159	1496.00	757,046	1368.00	**755,040**	**755,955.7**	348.72	**756,236**	758,154.4	68.04
100 × 10S20	750,658	1321.06	750,394	1323.62	**748,611**	**749,327.1**	367.73	**750,047**	751,747.3	68.52
100 × 10S30	738,033	1122.26	737,694	1253.34	**736,248**	**737,063.8**	415.95	**737,505**	739,087.7	68.69

(*continued on next page*)

Table 6 (*continued*)

Instance	LS1		LS2		PTS1			PTS2		
	Cost	CPU(s)	Cost	CPU(s)	Best	AVG	CPU(s)	Best	AVG	CPU(s)
100 × 20S5	966,474	1262.06	970,189	1154.32	**961,900**	**964,422.8**	142.32	968,861	973,939.2	104.34
100 × 20S10	951,882	1147.65	949,715	1267.19	**945,835**	**948,003.6**	145.08	952,317	958,369.8	102.84
100 × 20S15	935,443	1294.64	936,227	1284.38	**931,525**	**933,518.5**	151.57	**935,246**	940,053.2	102.65
100 × 20S20	921,746	1266.68	922,768	1334.65	**916,505**	**918,597.7**	157.15	**920,851**	923,581.8	102.52
100 × 20S30	894,685	1424.95	896,656	1367.74	**892,755**	894,825.9	171.95	896,202	899,657.8	104.15
100 × 30S5	1,170,457	722.86	1,167,044	700.03	**1,154,077**	**1,159,790.8**	121.82	**1,164,617**	1,174,570.3	141.40
100 × 30S10	1,145,700	1065.31	1,142,881	1104.81	**1,127,161**	**1,130,487.3**	126.65	**1,140,965**	1,149,830.1	145.87
100 × 30S15	1,113,552	1149.70	1,119,040	1184.08	**1,103,176**	**1,105,138.4**	130.40	1,116,441	1,123,978.2	146.01
100 × 30S20	1,093,126	1264.11	1,096,146	1232.82	**1,081,933**	**1,086,086.7**	133.10	1,093,174	1,100,436.0	143.75
100 × 30S30	1,052,682	1292.33	1,057,544	1271.81	**1,044,499**	**1,046,736.6**	140.48	1,055,745	1,061,462.5	138.23
AVG	**504,253.51**	**1388.88**	**504,448.86**	**1410.05**	**501,137.59**	**502,307.05**	**117.41**	**504,369.71**	**507,363.03**	**70.51**

References

Bellanger, A., Hanafi, S., & Wilbaut, C. (2013). Three-stage hybrid-flowshop model for cross-docking. *Computers & Operations Research, 40*(4), 1109–1121.

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

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

Dongarra, J. J. (2014). *Performance of various computers using standard linear equations software. CS - 89 - 85*. University of Manchester.

Gelareh, S., Glover, F., Guemri, O., Hanafi, S., Nduwayo, P., & Todosijevic, R. (2018). A comparative study of formulations for the Cross-dock Door Assignment Problem. *Omega* (Accepted December 6th). doi:10.1016/j.omega.2018.12.004.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research, 13*(5), 533–549.

Guignard, M., Hahn, P. M., Pessoa, A. A., & da Silva, D. C. (2012). Algorithms for the cross-dock door assignment problem. In *Proceedings of the fourth international workshop on model-based metaheuristics*.

Hahn, P. M., Kim, B. J., Stuetzle, T., Kanthak, S., Hightower, W. L., Samra, H., Ding, Z., & Guignard, M. (2008). The quadratic three-dimensional assignment problem: Exact and approximate solution methods. *European Journal of Operational Research, 184*(2), 416–428.

Ladier, A.-L., & Alpan, G. (2016). Cross-docking operations: Current research versus industry practice. *Omega, 62*, 145–162.

Nassief, W., Contreras, I., & As'ad, R. (2016). A mixed-integer programming formulation and Lagrangean relaxation for the cross-dock door assignment problem. *International Journal of Production Research, 54*(2), 494–508.

Nassief, W., Contreras, I., & Jaumard, B. (2018). A comparison of formulations and relaxations for cross-dock door assignment problems. *Computers & Operations Research, 94*, 76–88.

Pentico, D. W. (2007). Assignment problems: A golden anniversary survey. *European Journal of Operational Research, 176*(2), 774–793.

Pessoa, A. A., Hahn, P. M., Guignard, M., & Zhu, Y. R. (2010). Algorithms for the generalized quadratic assignment problem combining Lagrangean decomposition and the Reformulation-Linearization Technique. *European Journal of Operational Research, 206*(1), 54–63.

Tarhini, A. A., Yunis, M. M., & Chamseddine, M. (2016). Natural Optimization Algorithms for the Cross-Dock Door Assignment Problem. *IEEE Transactions on Intelligent Transportation Systems, 17*(8), 2324–2333.

Tsui, L. Y., & Chang, C. H. (1990). A microcomputer based decision support tool for assigning dock doors in freight yards. *Computers & Industrial Engineering, 19*(1–4), 309–312.

Tsui, L. Y., & Chang, C. H. (1992). An optimal solution to a dock door assignment problem. *Computers & Industrial Engineering, 23*(1–4), 283–286.

Van Belle, J., Valckenaers, P., & Cattrysse, D. (2012). Cross-docking: State of the art. *Omega, 40*(6), 827–846.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin, 1*(6), 80–83.

Yagiura, M., Ibaraki, T., & Glover, F. (2006). A path relinking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research, 169*(2), 548–569.

Zhu, Y.-R., Hahn, P. M., Liu, Y., & Guignard, M. (2009). New approach for the cross-dock door assignment problem. In *Proceedings of the XLI Brazilian symposium on operations research*.