

Scheduling incoming and outgoing trucks at cross docking terminals to minimize the storage cost

Ruslan Sadykov*

September 16, 2011

Abstract

Cross docking terminals allow companies to reduce storage and transportation costs in a supply chain. At these terminals, products of different types from incoming trucks are unloaded, sorted, and loaded to outgoing trucks for a delivery. When the designated outgoing truck is not immediately available for some products, they are temporarily stocked in a small storage area available at the terminal.

This paper focuses on the operational activities at a cross docking terminal with two doors: one for incoming trucks and another for outgoing trucks. We consider the truck scheduling problem with the objective to minimise the storage usage during the product transfer inside the terminal. We show that this problem is NP-hard in the strong sense even if there are only two product types. For a special case with fixed subsequences of incoming and outgoing trucks, we propose a dynamic programming algorithm which is the first polynomial algorithm for this case. The results of numerical tests of the algorithm on randomly generated instances are also presented.

Keywords: Logistics; Cross docking; Truck scheduling; Storage cost; Dynamic programming

1 Introduction

Cross docking terminal is a distribution center carrying a considerably reduced amount of stock in contrast to traditional warehouses. Incoming shipments delivered by incoming trucks are unloaded, sorted and loaded onto outgoing trucks, which forward the shipments to the respective locations within the distribution system. Compared to traditional warehousing, a cost intensive storage and retrieval of goods is eliminated by a synchronization of inbound and outbound flows. An additional advantage of cross docking is efficient usage of truck capacity (i.e. full loads) and the implementation of a good scheduling system [1].

In this paper, we consider a simplified cross docking terminal with one receiving, one shipping door and a storage, see Figure 1. The incoming trucks

*INRIA Bordeaux — Sud-Ouest, France, e-mail: Ruslan.Sadykov@inria.fr

arrive at the receiving door and the outgoing trucks arrive to the shipping door. Once docked, the products of an incoming truck are unloaded and transferred to the designated outgoing truck. This transfer can be immediate, if the truck is at the shipping door, or delayed. In the latter case, a small intermediate storage is used to stock the products until the arrival of the designated outgoing truck. Once an outgoing (incoming) truck has been completely loaded (unloaded), it is removed from the dock, replaced by another truck and the course of action repeats.

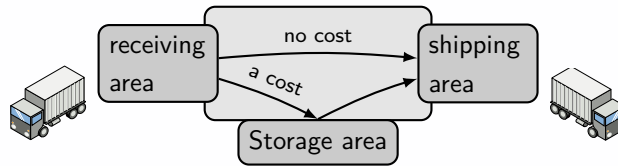


Figure 1: The cross docking terminal

The efficiency of such a system depends on the appropriate coordination of inbound and outbound flows. This paper deals with the truck scheduling problem which comprises the sequencing of arrivals (and departures) of incoming and outgoing trucks. This sequencing should reduce the delay of shipments at the cross docking terminal. In particular, storage usage should be limited, as it increases the duration and the resource consumption (i.e. manpower) of the product transfer.

There is a decent number of papers appeared recently which are devoted to cross dock scheduling. Boysen and Flidner [2] reviewed this research direction. There are several papers which dealt with truck scheduling problems at a cross docking terminal with two doors. Yu and Egbelu [9] developed a model for scheduling incoming and outgoing trucks to minimise the makespan (i.e. the maximum completion time) and proposed several heuristic algorithms for it. A simpler but similar problem was considered by Boysen, Flidner and Scholl [3]. They have proposed some lower bounds and an exact decomposition approach for it. Chen and Lee [5] formulated the truck scheduling problem to minimize the makespan as a flow-shop problem on two machines. They proved that the latter is NP-hard and proposed a branch-and-bound algorithm for its exact resolution.

Maknoon et al. [7] concentrated on the objective function which minimizes the storage cost and the special case of fixed subsequences of incoming and outgoing trucks. Two exact but exponential algorithms have been proposed. Labri et al. [6] studied a similar problem but with truck replacement costs in addition to the storage cost. In particular, they have also studied the special case with a fixed sequence of incoming trucks. Under the assumption that the products are loaded to outgoing trucks according to first-in first-out policy, the problem is shown to be polynomially solvable. Briskorn and Leung [4]

considered the truck scheduling problem at a cross docking terminal with one door (where both incoming and outgoing trucks are docked) to minimize the maximum lateness. In particular, they showed that the special case with fixed subsequences of incoming and outgoing trucks can be solved in linear time.

In this work, which is an extended version of [8], we consider the truck scheduling problem at a cross docking terminal with two doors with the objective to minimize the storage cost. In Section 2, we formally describe the problem and formulate it as a mixed integer program for the first time. We show in Section 3 that even a quite restricted version of the problem is NP-hard in the strong sense. Section 4 is devoted to the special case of the problem with fixed subsequences of incoming and outgoing trucks, which was introduced in [7]. We prove for that this special case can be solved in polynomial time and propose a dynamic programming algorithm for it. Finally, we test this algorithm on a set of randomly generated instances. Conclusions are drawn in Section 5.

2 Problem description and mathematical program

We now formally define the problem we consider. A set of n incoming trucks should be unloaded at the single receiving door, and a set of m outgoing trucks should be loaded at the single shipping door of the cross docking terminal. Each incoming truck is loaded with units of different product types $t \in T$. The number of units of product type t contained in an incoming truck I_i is denoted by a_{it} . Each outgoing truck O_o is to be loaded with a predetermined number of units b_{ot} of product type t , for each $t \in T$. Let T_o be the set of product types demanded by O_o , i.e. $T_o = \{t \in T : b_{ot} > 0\}$. Let also q be the maximum number of product types demanded by a single outgoing truck, i.e. $q = \max_{1 \leq o \leq m} |T_o|$. An intermediate storage of capacity D is available, where product units can be temporarily stored. One unit of product type $t \in T$ occupies a volume d_t in the storage.

Once an incoming (outgoing) truck arrives to the receiving (shipping) door, it should be fully unloaded (loaded) before its departure. Product units can be unloaded from an incoming truck either to the storage or directly to an outgoing truck currently present at the shipping door. Similarly, product units can be loaded to an outgoing truck either from the storage or directly from an incoming truck currently present at the receiving door.

Potentially, an incoming truck may unload product units to several outgoing trucks, and vice versa. As an example, consider the following departure sequence of trucks:

$$I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow O_1 \rightarrow O_2 \rightarrow I_4 \rightarrow O_3 \rightarrow O_4 \rightarrow I_5 \rightarrow I_6 \rightarrow O_5 \quad (1)$$

In Figure 2, we graphically represent departure sequence (1). For each truck, a rectangle represents the time interval during which the truck is at the corresponding door. Arcs represent a possible “flow” of product units. They can

be directly unloaded from a truck to another if both trucks are at the doors at the same moment. For example, truck I_4 may unload product units directly to trucks O_1 , O_2 , and O_3 . Also, only truck I_5 may unload product units directly to truck O_4 .

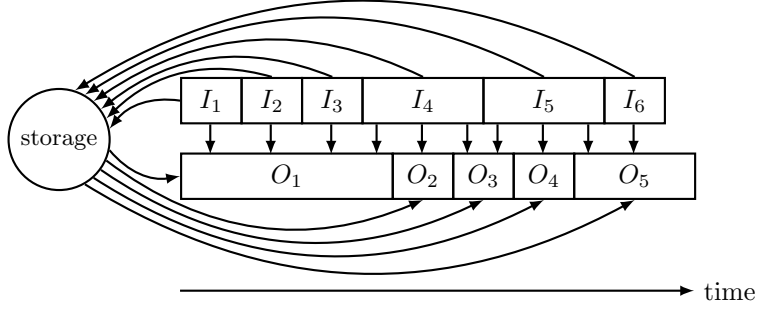


Figure 2: Graphical representation of departure sequence (1)

The problem we consider determines the departure sequence of trucks and the flow of product units, i.e. where each product unit is unloaded, and, if it is unloaded to the storage, to which outgoing truck it is loaded afterwards. Each time a unit of product type t is unloaded to the storage, a cost c_t is paid. The objective is to minimize the total storage cost.

We now formulate the problem as a mixed integer program (MIP), in which the following variables are used.

x_{ip} a binary variable which takes value 1 if incoming truck I_i is at position p in the departure sequence, 0 otherwise

y_{op} a binary variable which takes value 1 if outgoing truck O_o is at position p in the departure sequence, 0 otherwise

z_{io} a binary variable which takes value 1 if truck I_i may unload product units directly to truck O_o , 0 otherwise

f_{tio} number of units of product type t unloaded from I_i directly to O_o

f_{tip}^\downarrow number of units of product type t unloaded from truck I_i to the storage if truck I_i is at position p in the departure sequence, 0 otherwise

f_{top}^\uparrow number of units of product type t loaded to truck O_o from the storage if truck O_o is at position p in the departure sequence, 0 otherwise

st_p number of units of product type t stored in the storage after the truck at position p in the departure sequence left the corresponding door

$$\min \sum_{t \in T} \sum_{i=1}^n \sum_{p=1}^{n+m} c_t \cdot f_{tip}^\downarrow \quad (2)$$

$$\text{s.t.} \quad \sum_{p=1}^{n+m} x_{ip} = 1 \quad \forall i = 1, \dots, n \quad (3)$$

$$\sum_{p=1}^{n+m} y_{op} = 1 \quad \forall o = 1, \dots, m \quad (4)$$

$$\sum_{i=1}^n x_{ip} + \sum_{o=1}^m y_{op} = 1 \quad \forall p = 1, \dots, n+m \quad (5)$$

$$z_{io} \leq 3 - \sum_{p'=1}^{p-1} x_{ip'} - \sum_{\substack{1 \leq o' \leq m: \\ o' \neq o}} y_{o'p} - \sum_{p'=p+1}^{n+m} y_{op'} \quad \forall i, o, p \quad (6)$$

$$z_{io} \leq 3 - \sum_{p'=1}^{p-1} y_{op'} - \sum_{\substack{1 \leq i' \leq n: \\ i' \neq i}} x_{i'p} - \sum_{p'=p+1}^{n+m} x_{ip'} \quad \forall i, o, p \quad (7)$$

$$f_{tio} \leq \min\{a_{it}, b_{ot}\} \cdot z_{io} \quad \forall i, o, t \in T \quad (8)$$

$$f_{tip}^\downarrow \leq a_{it} \cdot x_{ip} \quad \forall i, p, t \quad (9)$$

$$f_{top}^\uparrow \leq b_{ot} \cdot y_{op} \quad \forall o, p, t \quad (10)$$

$$\sum_{o=1}^m f_{tio} + \sum_{p=1}^{n+m} f_{tip}^\downarrow = a_{it} \quad \forall i, t \quad (11)$$

$$\sum_{i=1}^n f_{tio} + \sum_{p=1}^{n+m} f_{top}^\uparrow = b_{ot} \quad \forall o, t \quad (12)$$

$$s_{tp} = s_{t,p-1} + \sum_{i=1}^n f_{tip}^\downarrow - \sum_{o=1}^m f_{top}^\uparrow \quad \forall t, p \quad (13)$$

$$s_{tp} \geq 0 \quad \forall t, p \quad (14)$$

$$\sum_{t \in T} d_t \cdot s_{tp} \leq D \quad \forall p \quad (15)$$

Objective function (2) minimizes the storage cost. Constraints (3) and (4) ensure that each truck is assign to exactly one position in the departure sequence, whereas constraints (5) ensure that each position is assigned to exactly one truck. Constraints (6) and (7) which link variables x and y with variables z are based on the following observation.

Proposition 1 *Incoming truck I_i cannot unload product units directly to outgoing truck O_o if and only if*

- either there exists an outgoing truck $O_{o'}$, $o' \neq o$, such that, in the departure sequence, $I_i \rightarrow O_{o'} \rightarrow O_o$,
- or there exists an incoming truck $I_{i'}$, $i' \neq i$, such that, in the departure sequence, $O_o \rightarrow I_{i'} \rightarrow I_i$.

Proof. Sufficiency is trivial to verify. Necessity can be shown easily by observing that, in the departure sequence, either $I_i \rightarrow O_o$ or $O_o \rightarrow I_i$, and by considering these two cases. \square

Constraints (8), (9), and (10) impose upper bounds on the flow of product units. Constraints (11) and (12) guarantee that every incoming (outgoing) truck is fully unloaded (loaded). (13) are the flow conservation constraints for the storage. Constraints (14) and (15) impose the lower and upper bounds on the storage level.

3 NP-hardness proof

In this section, we show that our problem is NP-hard in the strong sense even for the case in which

- each incoming truck supplies product units of at most two types,
- each outgoing truck demands product units of at most one type,
- all the storage costs are unitary,
- the storage capacity is unlimited.

We will perform a reduction from the 3-partition problem.

Remember that, in the 3-partition problem, we are given an integer B and a set of $3n$ integers r_1, r_2, \dots, r_{3n} such that $\sum_{i=1}^{3n} r_i = Bn$ and $B/4 < r_i < B/2$ for each i . We need to decide whether there exists a partition of the set of indexes $\{1, 2, \dots, 3n\}$ into n sets $\{A_1, A_2, \dots, A_n\}$ such that $\sum_{i \in A_j} r_i = B$, $\forall j = 1, \dots, n$. Note that, if such a partition exists, each subset A_j contains exactly 3 indexes.

Given an instance of the 3-partition problem, we now define the corresponding instance of our cross docking problem. There are $3n$ incoming, $4n$ outgoing trucks ($3n$ of the first type, n of the second type) and two product types. The supplies and demands are the following:

$$\begin{aligned}
 a_{i1} &= 1, & i &= 1, \dots, 3n, \\
 a_{i2} &= 2n + r_i, & i &= 1, \dots, 3n, \\
 b_{i1} &= 1, & i &= 1, \dots, 3n, \\
 b_{i2} &= 0, & i &= 1, \dots, 3n, \\
 b_{i1} &= 0, & i &= 3n + 1, \dots, 4n, \\
 b_{i2} &= 6n + B, & i &= 3n + 1, \dots, 4n.
 \end{aligned}$$

Proposition 2 *There exists a 3-partition if and only if there is a solution to the corresponding instance of the cross docking problem in which at most n product units are unloaded to the storage.*

Proof. Suppose that there exists a 3-partition $\{A_1, A_2, \dots, A_n\}$, where $A_j = \{i_{j1}, i_{j2}, i_{j3}\}$. Then, the trucks are sequenced in n groups. Group j , $1 \leq j \leq n$, includes set A_j of incoming trucks and outgoing trucks O_{2j-1} , O_{2j} and O_{3n+j} . The departure sequence of group j is the following

$$O_{2j-1} \rightarrow I_{i_{j1}} \rightarrow I_{i_{j2}} \rightarrow O_{3n+j} \rightarrow I_{i_{j3}} \rightarrow O_{2j}.$$

As $r_{i_{j1}} + r_{i_{j2}} + r_{i_{j3}} = B$, the incoming trucks unload $6n + B$ units of product type 2 directly to truck O_{3n+j} , $I_{i_{j1}}$ unloads one unit of product type 1 directly to O_{2j-1} , and $I_{i_{j3}}$ unloads one unit of product type 1 directly to O_{2j} . Only one unit of product type 1 is unloaded to the storage from $I_{i_{j2}}$. This product flow is represented in Figure 3. As there are n groups, n product units in total are unloaded to the storage and then loaded to outgoing trucks O_{2n+1}, \dots, O_{3n} , which are sequenced at the end.

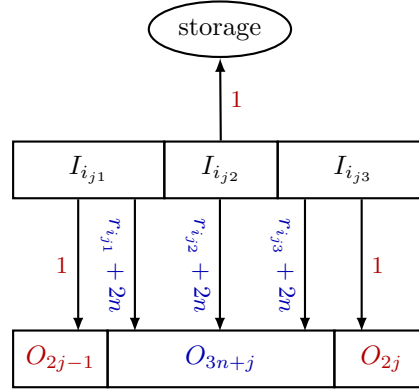


Figure 3: Product flow within group j of trucks

Suppose now there is a solution in which at most n products are unloaded to the storage. Then, every outgoing truck O_o , $o = 3n + 1, \dots, 4n$ must be loaded with product units directly from at least 3 incoming trucks. Otherwise it would be loaded with at least $2n$ units of product type 2 from the storage. Therefore, at least one unit of product type 1 is unloaded to the storage while each such truck O_o is being loaded, and the total number of units of product type 1 unloaded to the storage is at least n , meaning that all units of product type 2 should be unloaded directly to outgoing trucks. Then, each incoming truck can unload units of product type 2 directly to exactly one outgoing truck. Otherwise, between two outgoing trucks of type 2, only one outgoing truck can be loaded with a unit of product type 1 directly from an incoming truck, and the total number of units of product type 1 unloaded to the storage would exceed n .

We conclude that there should exist a partition of incoming trucks into triples $\{A_1, A_2, \dots, A_n\}$ such that $\sum_{i \in A_j} r_i = B$. Otherwise there would exist a triple A_j such that $r_{i_{j1}} + r_{i_{j2}} + r_{i_{j3}} < B$, and the outgoing truck which is loaded from the incoming trucks in A_j would need to be loaded with at least one unit of product type 2 from the storage. \square

4 Special case with fixed subsequences

Let the subsequences of incoming and outgoing trucks are fixed. For convenience, we renumber incoming and outgoing trucks according to these subsequences:

$$I_1 \rightarrow I_2 \rightarrow \dots \rightarrow I_n \text{ and } O_1 \rightarrow O_2 \rightarrow \dots \rightarrow O_m$$

Complexity of this special case, introduced in [7], has not been known to our knowledge. Here we propose a polynomial dynamic programming algorithm for it.

In the rest of the paper, we will use an equivalent objective which is the maximization of the total cost of product units unloaded directly to an outgoing truck.

4.1 Preliminary observations

For each departure sequence of trucks we define unique *direct first* product flow: each time trucks I_i and O_o are at the doors, for each $t \in T$, I_i unloads directly to O_o as many units of product type t as possible, i.e. the minimum between the number of units of product type t still available in I_i and the number of units of product type t which are still demanded by O_o .

The following important fact serves as the base for the algorithm.

Proposition 3 *There exists an optimal solution with a direct first product flow.*

Proof. Suppose, in an optimal solution, the product flow is not direct first. Then, let (I_i, O_o) be the first couple of trucks at the doors at the same time, such that u units of a product type t can be unloaded from I_i directly to O_o but they are “saved” for consequent outgoing truck(s). Then these u units have to be loaded to O_o from the storage. In the modified product flow, these u units are unloaded directly from I_i to O_o , and product units, which are loaded to consequent outgoing truck(s) directly from I_i in the original product flow, are loaded from the storage. The cost of the modified solution does not increase.

Applying this modification a finite number of times, we obtain an optimal solution with a direct first product flow. \square

By Proposition 3, the search for an optimum solution can be limited to the set of direct first product flows. By the definition, each direct first product flow is characterized by a departure sequence of trucks.

Note that some departure sequences are infeasible. First, when outgoing truck O_o departs, it is loaded with the missed product units from the storage.

Suppose that incoming truck I_i is at the door. Then, there is enough product units in the storage to complete the demand of O_o if and only if

$$\forall t \in T, \quad \sum_{k=1}^i a_{kt} \geq \sum_{j=1}^o b_{jt}. \quad (16)$$

For a given \bar{i} , let $lo(\bar{i})$ be the maximum o such that (16) is satisfied.

Second, when incoming truck I_i departs, it unloads to the storage the product units which were not unloaded directly to outgoing trucks. Suppose that outgoing truck O_o is at the door. Then, there is enough capacity in the storage to receive the surplus of I_i if and only if

$$\sum_{t=1}^T d_t \cdot \left(\max \left\{ 0, \sum_{k=1}^i a_{kt} - \sum_{j=1}^o b_{jt} \right\} \right) \leq D. \quad (17)$$

For a given \bar{o} , let $li(\bar{o})$ be the maximum i such that (17) is satisfied.

In the following, we present a dynamic programming algorithm which finds a feasible departure sequence of trucks which leads to an optimum direct first product flow.

4.2 Dynamic programming states

In our dynamic programming algorithm, there are two sets of states: S^{out} and S^{inc} .

Consider the set of the following departure sequences of trucks:

$$\dots, O_{o-1}, I_i, \dots$$

Such sequences create the situation in which truck I_i has departed, truck I_{i+1} is going to arrive, and truck O_o , arrived after the departure of truck I_{i-1} , is now at the shipping door. This means that O_o could be loaded with product units only from truck I_i . In a state $S^{out}(i, o, \{f_t\}_{t \in T_o})$, which corresponds to this situation, O_o is loaded with f_t units of product type t directly from I_i .

Consider the set of the following departure sequences of trucks:

$$\dots, I_{i-1}, O_o, \dots$$

Such sequences create the situation in which truck O_o has departed, truck O_{o+1} is going to arrive, and truck I_i , arrived after the departure of truck O_{o-1} , is at the shipping door. This means that I_i could unload product units only to O_o . In a state $S^{inc}(i, o, \{f_t\}_{t \in T_o})$, which corresponds to this situation, I_i unloads f_t units of product type t directly to O_o .

To simplify the presentation, when there is no ambiguity, we will use the shortened notations $S^{inc}(i, o, f)$, $S^{out}(i, o, f)$.

One could think of using simpler two-parameter dynamic programming states $S^{inc}(i, o)$ and $S^{out}(i, o)$. However, these two parameters do not suffice to fully

describe the situation. For a state $S^{inc}(i, o)$, we can calculate the total number of units of each product type in incoming truck I_i and the storage, but we cannot uniquely determine the distribution of product units between truck I_i and the storage. The similar observation holds for a state $S^{out}(i, o)$. The following example illustrates this observation in details.

Consider two partial departure sequences or trucks:

$$I_1 \rightarrow O_1 \rightarrow I_2 \rightarrow I_3 \rightarrow O_2 \rightarrow I_4 \rightarrow O_3, \quad (18)$$

$$I_1 \rightarrow O_1 \rightarrow I_2 \rightarrow O_2 \rightarrow I_3 \rightarrow I_4 \rightarrow O_3. \quad (19)$$

These sequences create the situation which corresponds to two-parameter state $S^{inc}(5, 3)$.

We will denote u units of product type t as $u^{(t)}$. Let incoming truck I_1 supplies $3^{(1)}$ and $3^{(2)}$, I_2 supplies $2^{(1)}$ and $2^{(2)}$, I_3 supplies $3^{(3)}$, I_4 supplies $3^{(2)}$, I_5 supplies $5^{(3)}$. Let also each outgoing truck O_o , $o = 1, 2, 3$, demands $5^{(o)}$. In Figure 4, we represent direct first product flows which correspond to the both partial departure sequences. Let $c_2 = 1$ and $c_3 = 2$. Sequence (18) creates the situation which corresponds to three-parameter state $S^{inc}(5, 3, \{5\})$ with partial cost 9 and nothing left in I_5 . Sequence (19) creates the situation which corresponds to three-parameter state $S^{inc}(5, 3, \{2\})$ with partial cost 6. This cost can be increased by up to 6 because of 3 units of product type 3 left in I_5 , as we do not know whether they will be unloaded to the storage or directly to an outgoing truck. Therefore, non of the states $S^{inc}(5, 3, \{5\})$ and $S^{inc}(5, 3, \{2\})$ is dominated, which shows that the use of two-parameter states does not permit to find the optimum departure sequence of trucks.

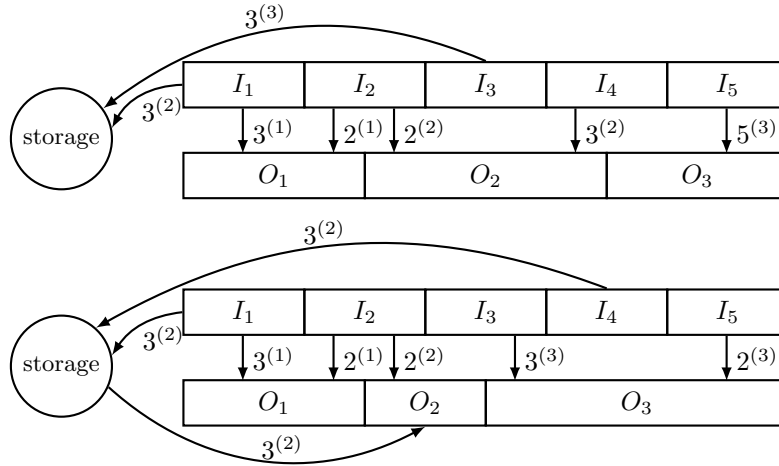


Figure 4: Direct first product flows which correspond to partial departure sequences (18) and (19)

To clarify the presentation, we show in Figure 5 the underlying directed

graph of the dynamic programming algorithm. Each “square node” (i, o) collects set of states $S^{out}(i, o, f)$. Each “circle node” (i, o) collects set of states $S^{inc}(i, o, f)$. From a state $S^{out}(i, o, f)$, we can go to a state $S^{inc}(i', o, f')$, $i' > i$. From a state $S^{inc}(i, o, f)$, we can go to a state $S^{out}(i, o', f')$, $o' > o$. The path shown in Figure 5 corresponds to the departure sequence

$$\begin{aligned}
 & I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow O_1 \rightarrow O_2 \rightarrow I_4 \rightarrow O_3 \rightarrow O_4 \rightarrow I_5 \rightarrow \\
 & \rightarrow I_6 \rightarrow \cdots \rightarrow I_{n-1} \rightarrow O_6 \rightarrow \cdots \rightarrow O_{m-1} \rightarrow I_n \rightarrow O_m.
 \end{aligned}$$

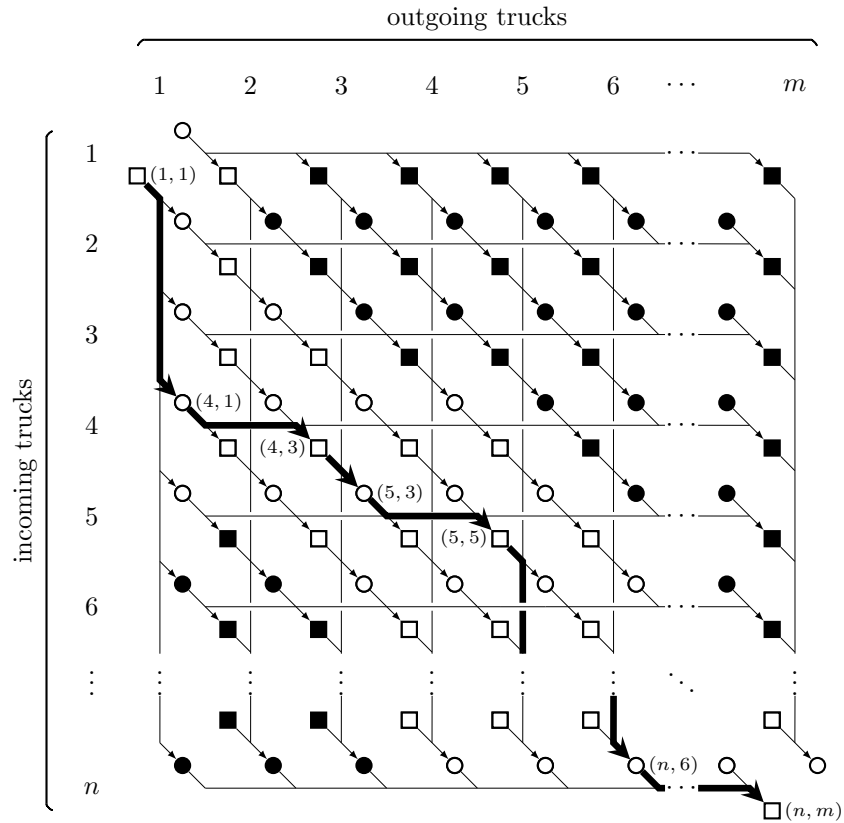


Figure 5: The underlying directed graph for the dynamic programming algorithm

Every product flow corresponds to exactly one sequence of states or a state path. A *direct first path* is a path which corresponds to a direct first product flow. In the following, we will use the same notation P for a path and the product flow which corresponds to it. A *direct first state* is a state which is contained in at least one direct first path.

By the inequalities (16) and (17),

- a direct first state $S^{out}(i, o, f)$ is feasible if and only if $i \leq li(o)$ and $o - 1 \leq lo(i)$;
- a direct first state $S^{inc}(i, o, f)$ is feasible if and only if $i - 1 \leq li(o)$ and $o \leq lo(i)$.

Possible situation of infeasible direct first states is shown in Figure 5. Sets of infeasible direct first states are shown as black nodes.

For each state $S^{out}(i, o, f)$ and $S^{inc}(i, o, f)$, we keep and update the objective function value of the best path to this state. Let $V^{out}(i, o, f)$ and $V^{inc}(i, o, f)$ be these values. To solve the problem, we need to find the best path terminating at a state $S^{out}(n, m, f)$ or $S^{inc}(n, m, f)$.

In a state $S^{out}(i, o, f)$ or $S^{inc}(i, o, f)$, by definition, we have

$$0 \leq f_t \leq \min\{a_{it}, b_{ot}\}, \quad \forall t \in T_o.$$

Let $AB = \max_{i,o,t} \min\{a_{it}, b_{ot}\}$, then the overall number of states

$$|S| = \sum_{i=1}^n \sum_{o=1}^m \prod_{t \in T_o} (\min\{a_{it}, b_{ot}\} + 1) = O(nm \cdot AB^q).$$

This number is a pseudopolynomial of the number of trucks and an exponential of q . Now we are going to prove that the number of direct first states is polynomial.

To do it, we will need the following lemma. But first we introduce some additional notations. Let $\mathcal{P}^{out}(i, o, f)$ and $\mathcal{P}^{inc}(i, o, f)$ be the sets of paths which contain states $S^{out}(i, o, f)$ and $S^{inc}(i, o, f)$ respectively.

Lemma 1

1. For any two direct first paths $P' \in \mathcal{P}^{out}(i, o, f')$ and $P'' \in \mathcal{P}^{out}(i, o, f'')$, $f' \neq f''$, if, for some type $t' \in T_o$, $f'_{t'} < f''_{t'}$, then $f'_t \leq f''_t$ for all types $t \in T_o$.
2. Analogously, for any two direct first paths $P' \in \mathcal{P}^{inc}(i, o, f')$ and $P'' \in \mathcal{P}^{inc}(i, o, f'')$, $f' \neq f''$, if, for some type $t' \in T_o$, $f'_{t'} < f''_{t'}$, then $f'_t \leq f''_t$ for all types $t \in T_o$.

Proof. We will prove this lemma by induction.

Suppose that claim 2 of the lemma is true for $i = i^*$ and for any $o < o^*$. We will prove claim 1 of the lemma for $i = i^*$ and $o = o^*$. Without loss of generality, let $P' \in \mathcal{P}^{inc}(i^*, o', \bar{f}')$, $o' < o^*$, and $P'' \in \mathcal{P}^{inc}(i^*, o'', \bar{f}'')$, $o'' < o^*$.

As $f'_{t'} < f''_{t'}$, in product flow P' , truck I_{i^*} unloads directly to O_{o^*} less units of product type t' than in product flow P'' . Therefore, I_{i^*} transfers directly to trucks O_o , $o' \leq o < o^*$, more units of product type t' in product flow P' than to trucks O_o , $o'' \leq o < o^*$, in product flow P'' . As P' and P'' are direct first product flows, there are two possible cases:

- $o' = o''$ and $\bar{f}'_{t'} > \bar{f}''_{t'}$. Then, as $P' \in \mathcal{P}^{inc}(i^*, o', \bar{f}')$ and $P'' \in \mathcal{P}^{inc}(i^*, o', \bar{f}'')$, by induction, $\bar{f}'_t \geq \bar{f}''_t$ for all types $t \in T_{o'}$.
- $o' < o''$.

In both cases, for every $t \in T_{o^*}$, truck I_{i^*} cannot not transfer less units of product type t to all trucks O_o , $o' \leq o < o^*$, in product flow P' , than to all trucks O_o , $o'' \leq o < o^*$, in product flow P'' , as the both product flows are direct first. Therefore, $f'_t \leq f''_t$ for all $t \in T_{o^*}$.

Analogously, if we suppose that claim 1 is true for all $i < i^*$ and for $o = o^*$, we can prove claim 2 for $i = i^*$ and $o = o^*$.

As the base of the induction for claim 1 we can take the case $i^* = 2$. In this case, we can only have $o' < o''$, as, for any fixed o^* , there is only one direct first path which contains a state $S^{inc}(2, o^*, f)$.

Analogously, as the base of the induction of claim 2, we can take the case $o^* = 2$. \square

Proposition 4 *The overall number of direct first states is $O(qnm^2)$.*

Proof. In a state $S^{out}(i, o, f)$ or $S^{inc}(i, o, f)$, we will call a value f_t , $t \in T_o$, *canonical*, if it is at its bounds: $f_t = 0$ or $f_t = \min\{a_{it}, b_{ot}\}$. For a fixed product type t^* , there are at most $2nm$ canonical values f_{t^*} in all states.

Consider the direct first product flows in $\mathcal{P}^{out}(i', o', f')$. Any such product flow is in $\mathcal{P}^{inc}(i'', o', f'')$ for some $i'' > i'$, where

$$f''_t = \max \left\{ 0, \min \left\{ a_{i''t}, b_{o't} - f'_t - \sum_{k=i'+1}^{i''-1} a_{kt} \right\} \right\}, \quad \forall t \in T_{o'}.$$

The next observation is very important. Suppose that truck $O_{o'}$ is loaded with f_{t^*} units of product type t^* directly from a truck I_i , $i > i'$, and $0 < f_{t^*} < \min\{a_{it^*}, b_{ot^*}\}$. As the product flow is direct first, this can happen only if $O_{o'}$ becomes full for units of product type t^* . Therefore, $O_{o'}$ cannot be loaded with units of product type t^* directly from subsequent incoming trucks. We conclude that, for a fixed product type t^* , there is at most one direct first path in $\mathcal{P}^{out}(i', o', f')$ which contains a state $S^{inc}(i'', o', f'')$, $i'' > i'$, with a non-canonical value $f_{i''t^*}$.

Consider now the direct first product flows in $\mathcal{P}^{inc}(i', o', f')$. Any such product flow is in $\mathcal{P}^{out}(i', o'', f'')$ for some $o'' > o'$, where

$$f''_t = \max \left\{ 0, \min \left\{ b_{o''t}, a_{i't} - f'_t - \sum_{j=o'+1}^{o''-1} b_{jt} \right\} \right\}, \quad \forall t \in T.$$

Again, suppose that truck $I_{i'}$ unloaded f_{t^*} units of product type t^* directly to a truck O_o , $o > o'$, and $0 < f_{t^*} < \min\{a_{it^*}, b_{ot^*}\}$. As the product flow is direct first, this can happen only if $I_{i'}$ unloaded directly to O_o all available units of

product type t^* . Therefore, $I_{i'}$ cannot unload any more units of product type t^* directly to subsequent outgoing trucks. We conclude that, for a fixed product type t^* , there is at most one direct first path in $\mathcal{P}^{inc}(i', o', f')$ which contains a state $S^{out}(i', o'', f'')$, $o'' > o'$, with a non-canonical value f_{t^*}'' .

We can say that, for a fixed t^* , each canonical value f_{t^*} in a direct first state $S^{out}(i, o, f)$ “begets” at most one non-canonical value f_{t^*}' in a direct first state $S^{inc}(i', o, f')$, $i' > i$, which itself can beget one non-canonical value f_{t^*}'' in a direct first state $S^{out}(i', o'', f'')$, $o'' > o$, and so on. Therefore, for fixed t^* and o^* , each canonical value f_{t^*} “begets” at most one non-canonical value f_{t^*} in direct first states $S^{out}(i, o^*, f)$. Therefore, the total number of different values f_{t^*} in these states is $O(nm)$. Similarly, for fixed t^* and o^* , the total number of different values f_{t^*} in states $S^{inc}(i, o^*, f)$ is also $O(nm)$.

We fix now values i^* and o^* , and define the following lexicographic order for direct first states $S^{out}(i^*, o^*, f)$. A state $S^{out}(i^*, o^*, f')$ is lexicographically smaller than a state $S^{out}(i^*, o^*, f'')$, $f'' \neq f'$, if and only if $f'_t \leq f''_t$ for all $t \in T_{o^*}$. It is always possible to compare in this way two direct first states, as we cannot have $f'_{t'} < f''_{t'}$ and $f'_{t''} > f''_{t''}$ for any two types $t', t'' \in T_{o^*}$ by Lemma 1.

In order to pass from a state $S^{out}(i^*, o^*, f)$ to the lexicographically next state, at least one of the values f_t , $t \in T_{o^*}$ should be increased. Therefore, the total number of direct first states $S^{out}(i^*, o^*, f)$ does not exceed the sum, for every type $t \in T_{o^*}$, of the number of different values f_t in these states. Analogously, the same holds for the direct first states $S^{inc}(i^*, o^*, f)$.

Consequently, for any fixed value o^* , the total number of direct first states $S^{out/inc}(i, o^*, f)$ is $O(qnm)$. From this we conclude that the overall number of direct first states is $O(qnm^2)$. \square

4.3 The algorithm

The idea of the algorithm is simple. We consider direct first states in a topological order. From each state, we make all possible moves to other direct first states. A complication consists in the fact that we do not know a priori which states are direct first and which are not. Therefore, the states are created dynamically. Each time we move to a state $S^{inc/out}(i, o, f)$, we verify whether this state has been visited before. If yes, we retrieve it and update the corresponding best value $V^{inc/out}(i, o, f)$. If not, we create the state and store the best value.

We now estimate the complexity of checking whether a state has been created. We will denote it as ρ . From the proof of Proposition 4, remember that, given fixed values i^* and o^* , the number of states $S^{inc/out}(i^*, o^*, f)$ is $O(qnm)$. As these states can be lexicographically ordered, the storage and the search of these states can be done using a binary tree. The lexicographic comparison between two states can be done in $O(q)$ time. Therefore, to check whether a state $S^{inc/out}(i, o, f)$, has been already created and retrieve the best value for it, we need $\rho = O(q \log(qnm))$ operations.

We now describe in details how the moves are done in the algorithm.

Suppose we are in a state $S^{inc}(i, o, f)$. From this state it is only possible to

move to a state $S^{out}(i, o', f')$, where $o < o' \leq lo(i) + 1$. When we make such a move, truck I_i transfers directly to every truck O_j , $o < j \leq o'$, as much products as possible.

```

1 for t = 1 to T do r[t] ← ait;
2 for t ∈ To do r[t] ← ait - ft;
3 v ← Vinc(i, o, f);
4 for j ← o + 1 to lo(i) + 1 do
5   for t ∈ Tj do
6     dt[t] ← min{r[t], bjt};
7     v ← v + ct · dt[t];
8   if i ≤ li(j) then
9     if state Sout(i, j, {dt[t]}t∈Tj) does not exist then
10      create it: Vout(i, j, {dt[t]}t∈Tj) ← -∞;
11     if v > Vout(i, j, {dt[t]}t∈Tj) then
12      Vout(i, j, {dt[t]}t∈Tj) ← v;
13   for t ∈ Tj do r[t] ← r[t] - dt[t];

```

Algorithm 1: Algorithm to make moves from a state $S^{inc}(i, o, f)$.

The formal procedure for making moves from a state $S^{inc}(i, o, f)$ is presented in Algorithm 1. The complexity of this procedure is $O(m(q + \rho))$.

Suppose now we are in a state $S^{out}(i, o, f)$. From this state it is only possible to move to a state $S^{inc}(i', o, f')$, where $i < i' \leq li(o) + 1$. When we make such a move, truck O_o receives directly from every truck I_k , $i < k \leq i'$, as much products as possible.

```

1 for t ∈ To do r[t] ← ft;
2 v ← Vout(i, o, f);
3 for k ← i + 1 to li(o) + 1 do
4   for t ∈ To do
5     dt[t] ← min{bot - r[t], akt};
6     v ← v + ct · dt[t];
7   if o ≤ lo(k) then
8     if state Sinc(k, o, {dt[t]}t∈To) does not exist then
9      create it: Vinc(k, o, {dt[t]}t∈To) ← -∞;
10    if v > Vinc(k, o, {dt[t]}t∈To) then
11     Vinc(k, o, {dt[t]}t∈To) ← v;
12   for t ∈ To do r[t] ← r[t] + dt[t];

```

Algorithm 2: Algorithm to make moves from a state $S^{out}(i, o, f)$.

The formal procedure for making moves from a state $S^{inc}(i, o, f)$ is presented in Algorithm 2. The complexity of this procedure is $O(n(q + \rho))$.

```

1 for  $t \in T_1$  do  $dt[t] \leftarrow \min\{a_{1t}, b_{1t}\}$ ;
2  $v \leftarrow \sum_{t \in T_1} c_t \cdot dt[t]$ ;
3  $V^{inc}(1, 1, \{dt[t]\}_{t \in T_1}) \leftarrow v$ ;
4 run Algorithm 1 for the state  $S^{inc}(1, 1, \{dt[t]\}_{t \in T_1})$ ;
5  $V^{out}(1, 1, \{dt[t]\}_{t \in T_1}) \leftarrow v$ ;
6 run Algorithm 2 for the state  $S^{out}(1, 1, \{dt[t]\}_{t \in T_1})$ ;
7 for  $i \leftarrow 1$  to  $n$  do
8   for  $o \leftarrow 1$  to  $m$  do
9     run Algorithm 1 for all created states  $S^{inc}(i, o, f)$ ;
10  for  $o \leftarrow 1$  to  $m$  do
11    run Algorithm 2 for all created states  $S^{out}(i, o, f)$ ;
12 return  $\max\{V^{out}(n, m, f), V^{inc}(n, m, f)\}$ 

```

Algorithm 3: The full algorithm

In the full algorithm, presented as Algorithm 3, we look through all the created states and make all possible moves from them as described above. To obtain an optimal product flow, it suffices to store, for each state, along the value V , the previous state on the path which gives this value. At the end of the algorithm, the best path and the corresponding product flow can be obtained by backtracking from the state $S^{out}(n, m, f)$ or $S^{inc}(n, m, f)$.

Proposition 5 *The complexity of the dynamic programming algorithm is $O(q^2nm^2(n + m) \log(qnm))$.*

Proof. To obtain the complexity C_{alg} of the algorithm, it suffices, for each group of states S^{out} and S^{inc} , to multiply the number of states by the complexity of the procedure which makes all the moves from a state:

$$\begin{aligned}
C_{alg} &= O(n(q + \rho)) \cdot O(qnm^2) + O(m(q + \rho)) \cdot O(qnm^2) \\
&= O(qnm^2(n + m)(q + \rho)) = O(q^2nm^2(n + m) \log(qnm)).
\end{aligned}$$

□

At the end, we present a dominance rule which speeds up the algorithm. It is quite easy to see that state $S^{out}(i, o, f')$ dominates state $S^{out}(i, o, f'')$ if $f'_t \leq f''_t, \forall t \in T_o$, and $V^{out}(i, o, f') \geq V^{out}(i, o, f'')$. Indeed, when we are in state $S^{out}(i, o, f')$, for each product type t , more units can be potentially unloaded directly to an outgoing truck, and the total cost of the product units already unloaded directly to an outgoing truck is larger. The same holds for states S^{inc} . In practice, making only moves from non-dominated states decreases significantly the running time of the algorithm.

n	a_{\max}	$q = 1$		$q = 2$		$q = 4$		$q = 8$	
		$ S $	RT	$ S $	RT	$ S $	RT	$ S $	RT
100	10	12.7	0.01	15.2	0.02	20.1	0.02	30.3	0.05
100	1000	13.4	0.01	18.3	0.02	24.2	0.03	35.5	0.06
200	10	65.9	0.11	92.6	0.18	140.9	0.35	225.5	0.77
200	1000	77.0	0.13	107.4	0.19	167.9	0.40	286.2	0.92
400	10	311.8	1.19	448.3	1.82	740.0	3.75	1'259.3	8.62
400	1000	364.6	1.37	532.5	2.09	877.2	4.22	1'548.5	10.05
800	10	1'257.4	14.29	2'089.8	20.80	3'546.2	38.60	6'102.8	82.19
800	1000	1'626.4	15.97	2'444.2	22.53	4'175.1	41.56	7'477.3	93.52

Table 1: Results of the numerical tests

4.4 Numerical tests

We have tested our dynamic programming algorithm on randomly generated instances to see how it scales in practice.

The test instances were generated in the following way. All the data are integer. The values c_t are uniformly distributed in the range $[1, 10]$. The capacity of the temporary storage is infinite. Number of incoming and outgoing trucks are equal: $n = m$. The number of types $|T| = 10q$. The values a_{it} are uniformly distributed in the range $[1, a_{\max}]$. The values b are generated in such a way that, for each o , at most q values b_{ot} are non-zero and

$$\sum_{i=1}^n a_{it} = \sum_{o=1}^m b_{ot}, \quad \forall t \in T. \quad (20)$$

Let

$$b_t^{av} = \frac{\sum_{i=1}^n a_{it}}{m \cdot q / |T|}.$$

Then, for each t , values b_{ot} which are randomly chosen to be non-zero are uniformly distributed in the range $[\frac{1}{3}b_t^{av}, \frac{5}{3}b_t^{av}]$, and slightly adjusted to satisfy the equality (20).

The following parameters values were used:

$$\begin{array}{ll} n & 100, 200, 400, 800 \\ q & 1, 2, 4, 8 \\ a_{\max} & 10, 1000 \end{array}$$

For each triple of parameters (n, q, a_{\max}) , 10 instances were generated.

The algorithm was implemented in the $C++$ programming language. The experiments were done on a workstation with an Intel Xeon X5460 3.16 GHz processor using a single thread (no parallelization).

In Table 1, we present the results of the numerical tests. In this table, $|S|$ is the average number of created states in thousands, and RT is the average running time in seconds. The algorithm solved all the instances in a reasonable time. On average,

- when the number of trucks doubles, the running time of the algorithm becomes 11.3 times larger;
- when value q doubles, the running time of the algorithm becomes 1.9 times larger.

5 Conclusion

In this paper, we considered a trucks scheduling problem at a cross docking terminal with two doors and the objective to minimize the storage cost. We showed that even a quite restricted version of this problem is NP-hard in the strong sense. Also, we presented a polynomial dynamic programming algorithm for the special case of the problem with fixed subsequences of incoming and outgoing trucks. This algorithm allows us to determine the computational complexity of this special case for the first time.

Numerical tests showed that the dynamic programming algorithm can be used in practice for solving instances with $n = m \leq 800$ and $q \leq 8$ in a reasonable time. When $n \leq 200$, the algorithm terminates in less than 1 second.

One interesting direction for a future research is to try to find a linear programming formulation for the special case studied here. Such a formulation is likely to exist, since this special case was shown to be polynomially solvable. A linear programming formulation would help a lot in developing methods for solving the general problem.

References

- [1] Uday M. Apte and S. Viswanathan. Effective cross docking for improving distribution efficiencies. *International Journal of Logistics Research and Applications*, 3(3):291–302, 2000.
- [2] Nils Boysen and Malte Fliedner. Cross dock scheduling: Classification, literature review and research agenda. *Omega*, 38(6):413 – 422, 2010.
- [3] Nils Boysen, Malte Fliedner, and Armin Scholl. Scheduling inbound and outbound trucks at cross docking terminals. *OR Spectrum*, 32:135–161, 2010.
- [4] Dirk Briskorn and Joseph Leung. Branch and bound algorithms for minimizing maximum lateness of trucks at a transshipment terminal. *Optimization Online*, 2677, 2010.
- [5] Feng Chen and Chung-Yee Lee. Minimizing the makespan in a two-machine cross-docking flow shop problem. *European Journal of Operational Research*, 193(1):59 – 72, 2009.
- [6] Rim Larbi, Gülgün Alpan, Pierre Baptiste, and Bernard Penz. Scheduling cross docking operations under full, partial and no information on inbound arrivals. *Computers and Operations Research*, 38(6):889 – 900, 2011.

- [7] Mohammad Yousef Maknoon, Pierre Baptiste, and Oumar Kone. Optimal loading and unloading policy in cross docking platform. In *Proceedings of 13th IFAC Symposium on Information Control Problems in Manufacturing*, pages 1263–1268, Moscow, Russia, June 2009.
- [8] Ruslan Sadykov. A polynomial algorithm for a simple scheduling problem at cross docking terminals. Research Report RR-7054, INRIA, 2009.
- [9] Wooyeon Yu and Pius J. Egbelu. Scheduling of inbound and outbound trucks in cross docking systems with temporary storage. *European Journal of Operational Research*, 184(1):377 – 396, 2008.