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

Ruslan Sadykov*

June 14, 2012

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 delivery. If 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 one for outgoing trucks. We consider the truck scheduling problem with the objective to minimize 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 terminals are distribution centers 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, the cost intensive storage and retrieval of goods is eliminated by the synchronization of inbound and outbound flows. An additional advantage of cross docking is the efficient usage of truck capacity (i.e. full loads) through the implementation of a good scheduling system [1].

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

As described in [3], such a scheduling system should synchronize incoming and outgoing truckloads so that the intermediate storage inside the terminal is kept low and on-time deliveries are ensured. The process of transshipment of goods in cross docking terminals can be subdivided into the tasks of unloading incoming trucks and loading outgoing trucks, which are typically separated by a time lag for material handling inside the terminal. These tasks are processed by the terminal's "dock doors", which can process one truck a time and are assumed to be equipped with loading equipment and workers. As cost related to customer satisfaction incurred by delayed deliveries is hard to quantify accurately, a time related surrogate objective is often a better choice for cross docking scheduling.

In this paper, we consider a simplified cross docking terminal with one receiving door, one shipping door and the storage area, see Figure 1. The incoming trucks arrive at the receiving door and the outgoing trucks arrive at the shipping door. Once docked, the products of an incoming truck are unloaded and transfered to the designated outgoing truck. This transfer can be immediate, if the correct truck is at the shipping door, or delayed. In the latter case, a small intermediate storage area 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.

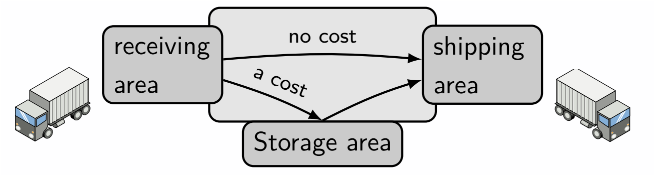

Figure 1: The cross docking terminal

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 some situations, storage usage can have a great impact on this delay, as it increases the duration and the resource consumption (i.e. manpower) of the product transfer. Thus storage usage should be limited in this case.

A number of recent articles have been devoted to cross dock scheduling. Boysen and Fliedner [3] 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 [11] 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 et al. [4]. They proposed lower bounds and an exact decomposition approach for the problem. Chen and Lee [6] 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. Boloori Arabani et al. [2] considered the truck scheduling problem with the total earliness-tardiness objective, and proposed three meta heuristic algorithms for it. Another meta heuristic algorithm was suggested by Forouharfard and Zandieh [7] for the same problem but with the objective to minimize the number of products that pass through the storage area.

Maknoon et al. [9] 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. Larbi et al. [8] 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 a first-in first-out policy, the problem is shown to be polynomially solvable. Briskorn and Leung [5] 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 [10], we consider the truck scheduling problem at a cross docking terminal with two doors with the objective to minimize the storage cost. We are mainly interested in theoretical properties of this problem. We prove that even a quite restricted version of the problem is NP-hard in the strong sense. Then, we show that the special case of the problem with fixed subsequences of incoming and outgoing trucks is polynomial. This is a rare example of non-trivial polynomial special cases of cross docking scheduling problems.

The paper is organized as follows. In Section 2, we formally describe the problem and formulate it as a mixed integer program. We prove in Section 3 that the problem is NP-hard. Section 4 is devoted to the special case of the problem mentioned above. We show that this special case, introduced in [9], 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 formulation

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 b_{ot} of units of product type t, for each $t \in T$. Let T_o be the set of product types demanded by O_o, i.e. $T_o = \{t \in T : \ b_{ot} > 0\}$. Also let q be the maximum number of product types demanded by a single outgoing truck, i.e. $q = \max_{1 \le o \le m} |T_o|$, where $|T_o|$ is the cardinality of set T_o. An intermediate storage of capacity D is available, i.e. at most D product units can be stored there.

Product units can be unloaded from an incoming truck either to storage or directly to the outgoing truck currently present at the shipping door. Similarly, product units can be loaded to an outgoing truck either from storage or directly from the incoming truck currently present at the receiving door.

The following assumptions are made about the model.

- All trucks are available at time 0.
- Product units can be loaded from trucks and unloaded to trucks in any sequence.
- Preemption is not allowed, i.e. once a truck arrives at a door, it should be fully unloaded or loaded before its departure.
- Initial storage is zero.
- For each product type the total number of incoming product units should be equal to the total number of outgoing ones:

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

- There is no "idle time" at the doors, i.e. once an incoming or outgoing truck departs, the next one arrives immediately.

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:

$$I_1 \to I_2 \to I_3 \to O_1 \to O_2 \to I_4 \to O_3 \to O_4 \to I_5 \to I_6 \to O_5 \tag{2}$$

In Figure 2, we graphically represent this departure sequence. 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.

The problem we consider determines the departure sequence of trucks and the flow of product units, i.e. where each product unit is unloaded to, and, if it is unloaded to storage, to which outgoing truck it is loaded afterwards. Each

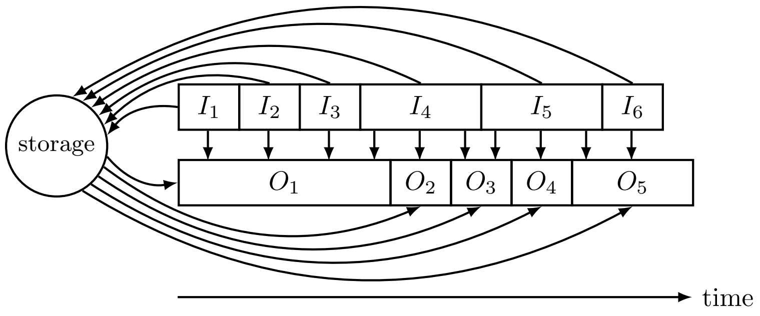

Figure 2: Graphical representation of departure sequence (2)

time a unit of product type t is unloaded to 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_{iot} number of units of product type t unloaded from I_i directly to O_o

$f_{it}^{\downarrow}$ number of units of product type t unloaded from truck I_i to storage

$f_{ot}^{\uparrow}$ number of units of product type t loaded to truck O_o from storage

s_{pt} number of units of product type t stored in the storage area after the truck at position p in the departure sequence leaves the corresponding door

$$\min \sum_{t \in T} \sum_{i=1}^{n} c_t \cdot f_{it}^{\downarrow} \tag{3}$$

$$\text{s.t.} \quad \sum_{p=1}^{n+m} x_{ip} = 1 \qquad \forall i = 1, \dots, n \tag{4}$$

$$\sum_{p=1}^{n+m} y_{op} = 1 \qquad \forall o = 1, \dots, m \tag{5}$$

$$\sum_{i=1}^{n} x_{ip} + \sum_{o=1}^{m} y_{op} = 1 \qquad \forall p = 1, \ldots, n+m \tag{6}$$

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

$$z_{io} \leq 3 - \sum_{p'=1}^{p-1} y_{op'} - \sum_{\substack{1 \leq i' \leq n: \\ i' \neq i}} x_{i'p} - \sum_{p'=p+1}^{n+m} x_{ip'} \qquad \forall i, o, p \tag{8}$$

$$0 \leq f_{iot} \leq \min\{a_{it}, b_{ot}\} \cdot z_{io} \qquad \forall i, o, t \in T \tag{9}$$

$$\sum_{o=1}^{m} f_{iot} + f_{it}^{\downarrow} = a_{it} \qquad \forall i, t \tag{10}$$

$$\sum_{i=1}^{n} f_{iot} + f_{ot}^{\uparrow} = b_{ot} \qquad \forall o, t \tag{11}$$

$$s_{pt} = s_{p-1,t} + \sum_{i=1}^{n} x_{ip} \cdot f_{it}^{\downarrow} - \sum_{o=1}^{m} y_{op} \cdot f_{ot}^{\uparrow} \qquad \forall t, p \tag{12}$$

$$\sum_{t \in T} s_{pt} \leq D \qquad \forall p \tag{13}$$

$$s_{0t} = 0 \qquad \forall t \tag{14}$$

$$s_{pt} \geq 0 \qquad \forall t, p \tag{15}$$

The objective function (3) minimizes the storage cost. Constraints (4) and (5) ensure that each truck is assigned to exactly one position in the departure sequence, whereas constraints (6) ensure that each position is assigned to exactly one truck. Constraints (7) and (8), 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 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. □

Constraints (9) impose bounds on the direct flow of product units. Constraints (10) and (11) guarantee that every incoming (outgoing) truck is fully unloaded (loaded). Constraints (12) are the flow conservation constraints for the storage. Note that they can be linearized if needed. We present them in a non-linear form for simplicity. Constraints (13), (14), and (15) impose 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.

Recall that, in the 3-partition problem, we are given an integer B and a set of $3n$ integers $r_1, r_2, \ldots, r_{3n}$ 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 $\{1, 2, \ldots, 3n\}$ into n sets $\{A_1, A_2, \ldots, A_n\}$ such that $\sum_{i \in A_j} r_i = B$, $\forall j = 1, \ldots, n$. 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 trucks, $4n$ outgoing trucks ($3n$ of the first type, n of the second type) and two product types. The supplies and demands are the following:

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

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 storage.*

Proof. Necessity. Suppose that there exists a 3-partition $\{A_1, A_2, \ldots, A_n\}$, where $A_j = \{i_{j1}, i_{j2}, i_{j3}\}$. Then, the trucks are sequenced in n groups. Group j, $1 \le j \le n$, 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

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

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 $O_{2n+1}, \ldots, O_{3n}$, which are sequenced at the end.

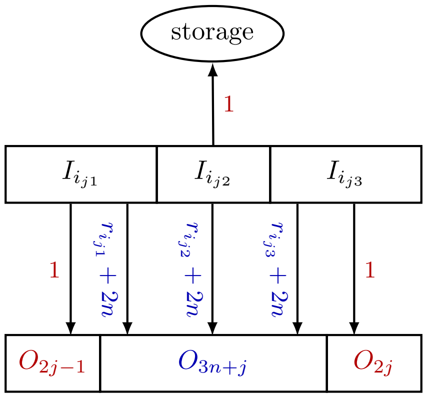

Figure 3: Product flow within group j of trucks

Sufficiency. Suppose now there is a solution in which at most n products are unloaded to the storage. Then, every outgoing truck O_o of type 2, i.e. $o = 3n+1, \ldots, 4n$, must be loaded with product units directly from at least three incoming trucks. Otherwise it would be loaded with at least $2n$ units of product type 2 from the storage. Therefore, when an outgoing truck of type 2 loads product units directly from its second incoming truck, one unit of product type 1 is unloaded to the storage. Thus, the number of product units of type 1 unloaded to the storage is exactly n. Consequently, in the sequence, between two outgoing trucks of type 2, there should be two trucks of type 1, which load their products units directly from incoming trucks. This means that each incoming truck can unload product units of type 2 directly to only one outgoing truck. We conclude that there should exist a partition of incoming trucks into triples $\{A_1, A_2, \ldots, A_n\}$ 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. □

4 Special case with fixed subsequences

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

$$I_1 \to I_2 \to \cdots \to I_n \text{ and } O_1 \to O_2 \to \cdots \to O_m$$

To our knowledge, the complexity of this special case, introduced in [9], is not known. 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. The equivalence between two objective functions can be seen from the fact that the total costs of product units unloaded to storage and unloaded directly to an outgoing truck sum up to a constant.

4.1 Preliminary observations

For each departure sequence of trucks we define the 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 pair 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 storage. In the modified product flow, these u units are unloaded directly from I_i to O_o. Outgoing truck(s), loaded with these u units directly from I_i in the original product flow, replace these units by ones from storage in the modified flow. 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. □

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

Note that some departure sequences are infeasible. First, when outgoing truck O_o departs, it is loaded with the missing product units from the storage. Suppose that incoming truck I_i is at the door. Then, there are enough product units in storage to complete the demand of O_o if and only if

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

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

Second, when incoming truck I_i departs, it unloads to 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

$$\sum_{t=1}^{T} \max\left\{0, \sum_{k=1}^{i} a_{kt} - \sum_{j=1}^{o} b_{jt}\right\} \le D. \tag{17}$$

For a given $\bar{o}$, let $li(\bar{o})$ be the maximum index 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 optimal direct first product flow.

4.2 Dynamic programming states

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

Consider the set of the following partial departure sequences of trucks:

$$\ldots, O_{o-1}, I_i. \tag{18}$$

Such sequences create the situation in which truck I_i has departed, truck I_{i+1} is going to arrive, and truck O_o, which arrived after the departure of truck I_{i-1}, is now at the shipping door. This means that O_o can be loaded with product units only from truck I_i (at the moment of departure of 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 partial departure sequences of trucks:

$$\ldots, I_{i-1}, O_o. \tag{19}$$

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

A move from a state $S^{in}(i, o, f)$ can be done only to a state $S^{out}(i, o', f')$, $o' > o$. Such a move corresponds to the continuation

$$O_{o+1}, \ldots, O_{o'-1}, I_i$$

of one of partial departure sequences (19). Under the direct first product flow assumption, vector f' and the cost of such a move can be uniquely determined, i.e. they depend only on i, o, f, and o':

$$f'_t = \min\left\{b_{o't},\ \max\left\{0,\ a_{it} - \tilde{f}_t - \sum_{j=o+1}^{o'-1} b_{jt}\right\}\right\}, \quad \forall t \in T_{o'}, \tag{20}$$

where

$$\tilde{f}_t = \begin{cases} f_t, & t \in T_o, \\ 0, & t \notin T_o, \end{cases}$$

and the cost equals to

$$\sum_{t \in T} c_t \cdot \min \left\{ a_{it} - \tilde{f}_t,\ \sum_{j=o+1}^{o'} b_{jt} \right\}. \tag{21}$$

A move from a state $S^{out}(i, o, f)$ can be done only to a state $S^{in}(i', o, f')$, $i' > i$. Such a move corresponds to the continuation

$$I_{i+1}, \ldots, I_{i'-1}, O_o$$

of one of partial departure sequences (18). Under the direct first product flow assumption, vector f' and the cost of such a move can be uniquely determined, i.e. they depend only on i, o, f, and i':

$$f'_t = \min \left\{ a_{i't},\ \max \left\{ 0,\ b_{ot} - f_t - \sum_{k=i+1}^{i'-1} a_{kt} \right\} \right\}, \quad \forall t \in T_o, \tag{22}$$

and the cost equals to

$$\sum_{t \in T_o} c_t \cdot \min \left\{ b_{ot} - f_t,\ \sum_{k=i+1}^{i'} a_{kt} \right\}. \tag{23}$$

Remark 1. One could think of using simpler two-parameter dynamic programming states $S^{in}(i, o)$ and $S^{out}(i, o)$. However, these two parameters do not suffice to fully describe the situation. For a state $S^{in}(i, o)$, we can calculate the total number of units of each product type in incoming truck I_i and in the storage area, but we cannot uniquely determine the distribution of product units between truck I_i and 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 of trucks:

$$I_1 \to O_1 \to I_2 \to I_3 \to O_2 \to I_4 \to O_3, \tag{24}$$

$$I_1 \to O_1 \to I_2 \to O_2 \to I_3 \to I_4 \to O_3. \tag{25}$$

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

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 (24) creates the situation which corresponds to three-parameter state $S^{in}(5, 3, \{5\})$ with partial cost 9 and nothing left in I_5. Sequence (25) creates the situation which corresponds to three-parameter state $S^{in}(5, 3, \{2\})$ with partial cost 6. This cost can be potentially increased by 6 because of 3 units of product type 3 left in I_5, as later they may be unloaded directly to an outgoing truck. Therefore, non of the states $S^{in}(5, 3, \{5\})$ and $S^{in}(5, 3, \{2\})$ is dominated, which shows that the use of two-parameter states does not permit one to find an optimal departure sequence of trucks.

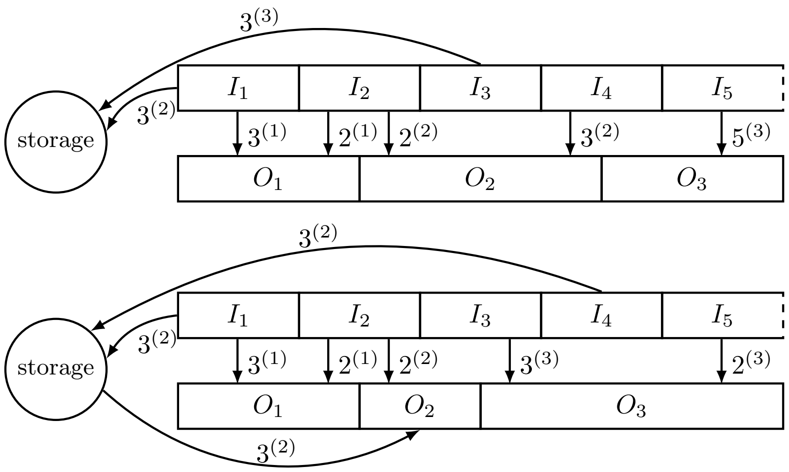

Figure 4: Direct first product flows which correspond to partial departure sequences (24) and (25)

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 $S^{in}(i, o, f)$. The path shown in Figure 5 corresponds to departure sequence

$$I_1 \to I_2 \to I_3 \to O_1 \to O_2 \to I_4 \to O_3 \to O_4 \to I_5 \to$$

$$\to I_6 \to \cdots \to I_{n-1} \to O_6 \to \cdots \to O_{m-1} \to I_n \to O_m.$$

Recall that, for each feasible departure sequence, there are many possible product flows, each of which corresponds to a different sequence of three-parameters states or a states path. Exactly one of these product flows is direct first. The states path this direct flow product flow corresponds to we will call *direct first path.* The cost of such a path can be computed using formulae (21) and (23). In the following, we will use the same notation P for a product flow and the states path which corresponds to it. A *direct first state* is a state which is contained in at least one direct first path.

By inequalities (16) and (17),

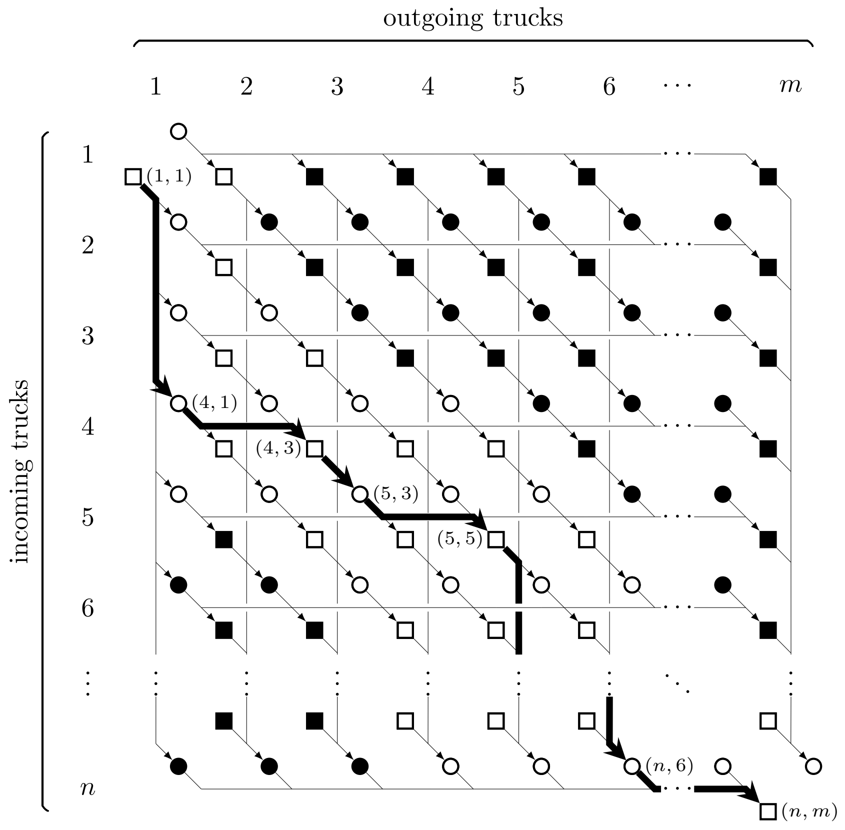

Figure 5: The underlying directed graph for the dynamic programming algorithm

- 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 $S^{in}(i, o, f)$ is feasible if and only if $i - 1 \leq li(o)$ and $o \leq lo(i)$.

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

In a state $S^{out}(i, o, f)$ or $S^{in}(i, o, f)$, 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 $|S|$ can be estimated as

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

as $q = \max_{o=1}^{m} |T_o|$. This number is pseudopolynomial if q is fixed, and exponential otherwise. Now we are going to prove that the number of direct first states is polynomial.

For this, we will need some additional notations. Let $\mathcal{P}^{out}(i,o,f)$ and $\mathcal{P}^{in}(i,o,f)$ be the sets of paths which contain states $S^{out}(i,o,f)$ and $S^{in}(i,o,f)$ 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 $P' \in \mathcal{P}^{in}(i,o,f')$ and $P'' \in \mathcal{P}^{in}(i,o,f'')$, $f' \neq f''$, 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 $P' \in \mathcal{P}^{in}(i^*, o', \bar{f}')$, $o' < o^*$, and $P'' \in \mathcal{P}^{in}(i^*, o'', \bar{f}'')$, $o'' < o^*$.

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 $P' \in \mathcal{P}^{in}(i^*, o', \bar{f}')$ and $P'' \in \mathcal{P}^{in}(i^*, o', \bar{f}'')$, 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 $S^{in}(2, o^*, f)$.

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

Proposition 4. *The overall number of direct first states is $O(qnm\min\{n,m\})$.*

Proof. In a state $S^{out}(i,o,f)$ or $S^{in}(i,o,f)$, we will call a value f_t, $t \in T_o$, *canonical*, if it is at its bounds: $f_t = 0$ or $f_t = \min\{a_{it}, b_{ot}\}$.

Consider set $\mathcal{P}^{out}(i,o,f)$ of direct first product flows. Any such product flow is in $\mathcal{P}^{in}(i',o,f')$ for some $i' > i$, where f' is computed according to (22). Suppose that $0 < f'_t < \min\{a_{i't}, b_{ot}\}$ for a product type $t \in T_o$. As the product flow is direct first, this can happen only if O_o becomes full for units of product type t when receiving product units from $I_{i'}$. Therefore, O_o cannot be loaded with units of product type t directly from subsequent incoming trucks. We conclude that, for a each product type $t \in T_o$, there is at most one direct first path in $\mathcal{P}^{out}(i,o,f)$ which contains a state $S^{in}(i',o,f')$, $i' > i$, with a non-canonical value f'_t.

Consider now set $\mathcal{P}^{in}(i,o,f)$ of direct first product flows. Any such product flow is in $\mathcal{P}^{out}(i,o',f')$ for some $o' > o$, where f' is computed according to (20). Suppose that $0 < f'_t < \min\{a_{it}, b_{o't}\}$ for a product type $t \in T_o$. As the product flow is direct first, this can happen only if I_i unloads 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 each product type $t \in T_o$, there is at most one direct first path in $\mathcal{P}^{in}(i,o,f)$ which contains a state $S^{out}(i,o',f')$, $o' > o$, with a non-canonical value f'_t.

We can say that, each canonical value f_t, $t \in T_o$, in a direct first state $S^{out}(i,o,f)$ "begets" at most one non-canonical value f'_t in a direct first state $S^{in}(i',o,f')$, $i' > i$, 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. Thus, each canonical value f_t, $t \in T_o$, in a direct first state $S^{out}(i,o,f)$ "begets" at most $2 \cdot \min\{n,m\}$ non-canonical values in all states. The same holds for each canonical value f_t, $t \in T_o$, in a direct first state $S^{in}(i,o,f)$. Therefore, for fixed i^* and o^*, in all states $S^{in/out}(i^*,o^*,t)$, all canonical values f_t, $t \in T_{o^*}$, "beget" $O(q\min\{n,m\})$ non-canonical values in all states. Consequently, the total number of non-canonical values f_t for all product types in all states is $O(qnm{\cdot}\min\{n,m\})$. This dominates the total number of canonical values which is $O(qnm)$.

We fix again values i^* and o^*, and define the following lexicographic order for direct first states $S^{out}(i^*,o^*,f)$. State $S^{out}(i^*,o^*,f')$ is lexicographically smaller than 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 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 total number of values f_t for all product types $t \in T_{o^*}$. Analogously, the same holds for the direct first states $S^{in}(i^*,o^*,f)$.

Consequently, the total number of direct first states does not exceed the total number of values f_t for all product types in all states, i.e. $O(qnm \cdot \min\{n,m\})$. □

4.3 The algorithm

Given the description of the states, the dynamic programming algorithm is straightforward. We consider the direct first states in a topological order. From each state, we make all possible moves along direct first paths. 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 $S^{in/out}(i,o,f)$, we verify whether this state has been visited before. If yes, we retrieve it and update the corresponding best value $V^{in/out}(i,o,f)$. If not, we create the state and store the best value.

We now estimate the complexity (denoted as ρ) of checking whether a state has been created. Given fixed values i^* and o^*, for each $t \in T_{o^*}$, the number of different values f_t in states $S^{in/out}(i^*,o^*,f)$ is $O(nm)$, as each canonical value f_t in other states "begets" at most one value f_t in states $S^{in/out}(i^*,o^*,f)$. Similarly to the proof of Proposition 4, it can be then shown that the number of direct first states $S^{in/out}(i^*,o^*,f)$ 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 $S^{in/out}(i,o,f)$, has been already created and retrieve the best value for it, we need $\rho = O\big(q\log(qnm)\big)$ operations.

Recall that from a state $S^{in}(i,o,f)$ 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.

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

```
1  for t = 1 to $T$ do  r[t] ← $a_{it}$;
2  for t ∈ $T_o$ do  r[t] ← $a_{it} - f_t$;
3  v ← $V^{in}(i,o,f)$;
4  for j ← $o+1$ to $lo(i)+1$ do
5      for t ∈ $T_j$ do
6          dt[t] ← min{r[t], $b_{jt}$};
7          v ← v + $c_t$ · dt[t];
8      if $i \leq li(j)$ then
9          if state $S^{out}(i, j, \{dt[t]\}_{t\in T_j})$ does not exist then
10             create it: $V^{out}(i, j, \{dt[t]\}_{t\in T_j}) \leftarrow -\infty$;
11         if v > $V^{out}(i, j, \{dt[t]\}_{t\in T_j})$ then
12             $V^{out}(i, j, \{dt[t]\}_{t\in T_j})$ ← v;
13     for t ∈ $T_j$ do r[t] ← r[t] − dt[t];
```

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

From a state $S^{out}(i,o,f)$ it is only possible to move to a state $S^{in}(i',o,f')$, 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.

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

```
for t ∈ $T_o$ do r[t] ← $f_t$;
v ← $V^{out}(i, o, f)$;
for k ← $i + 1$ to $li(o) + 1$ do
    for t ∈ $T_o$ do
        dt[t] ← $\min\{b_{ot} - r[t], a_{kt}\}$;
        v ← v + $c_t \cdot dt[t]$;
    if $o \leq lo(k)$ then
        if state $S^{in}(k, o, \{dt[t]\}_{t \in T_o})$ does not exist then
            create it: $V^{in}(k, o, \{dt[t]\}_{t \in T_o}) \leftarrow -\infty$;
        if v > $V^{in}(k, o, \{dt[t]\}_{t \in T_o})$ then
            $V^{in}(k, o, \{dt[t]\}_{t \in T_o}) \leftarrow$ v;
    for t ∈ $T_o$ do r[t] ← r[t] + dt[t];
```

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

In the complete procedure, presented as Algorithm 3, we look through all the created states and make all possible moves from them as described above. To obtain an optimum 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 which gives the best solution.

```
for t ∈ $T_1$ do dt[t] ← $\min\{a_{1t}, b_{1t}\}$;
v ← $\sum_{t \in T_1} c_t \cdot dt[t]$;
$V^{in}(1, 1, \{dt[t]\}_{t \in T_1}) \leftarrow$ v;
$V^{out}(1, 1, \{dt[t]\}_{t \in T_1}) \leftarrow$ v;
for i ← 1 to $n$ do
    for o ← 1 to $m$ do
        run Algorithm 1 for all created states $S^{in}(i, o, f)$;
    for o ← 1 to $m$ do
        run Algorithm 2 for all created states $S^{out}(i, o, f)$;
return $\max\{V^{out}(n, m, f), V^{in}(n, m, f)\}$
```

Algorithm 3: The full algorithm

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

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

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

□

Finally, 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^{in}. In practice, making moves only from non-dominates states decreases significantly the running time of the algorithm.

4.4 Numerical tests

We have tested our dynamic programming algorithm on randomly generated instances of the following sizes:

$$\begin{aligned} n = &\quad 100,\ 200,\ 400 \\ m = &\quad 100,\ 200,\ 400 \\ |T| = &\quad 5,\ 10,\ 20 \\ q = &\quad 1,\ 2,\ 4 \end{aligned}$$

In all test instances, all data are integer, and the capacity of the temporary storage is infinite. Two types of instances were generated.

In the instances of type 1, values c_t are uniformly distributed in interval $[1,10]$, and values a_{it} are uniformly distributed in interval $[0, a_{\max}]$, where $a_{\max} = 100$. We fix this parameter, as preliminary experiments showed that the computation time of the algorithm weakly depends on $a_{\max}$: if $a_{\max}$ is multiplied by 100, computation time increases only by 10%.

In the instances of type 2, product types are divided into three equal groupes: with small, medium and large product units. Number of product types in each group is almost equal. Values c_t and a_{it} are generated in the following way:

small product units	$a_{it} \in [0,5]$	$c_t \in [25,100]$
medium product units	$a_{it} \in [5,25]$	$c_t \in [5,25]$
big product units	$a_{it} \in [25,100]$	$c_t \in [1,5]$

In the instances of both types, values b_{ot} were generated in such a way that, for each o, at most q values are non-zero and condition (1) is satisfied. Let b_t^{av} be the average value for b_{ot}:

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

	$m = 100$		$m = 200$		$m = 400$	
	$\|S\|$	RT	$\|S\|$	RT	$\|S\|$	RT
$n = 100$	24.2	0.03	58.0	0.10	125.4	0.26
$n = 200$	49.5	0.09	121.8	0.26	269.9	0.73
$n = 400$	97.9	0.26	242.4	0.74	554.2	2.20

Table 1: Numerical results for different number of incoming and outgoing trucks

	$\|T\| = 5$		$\|T\| = 10$		$\|T\| = 20$	
	$\|S\|$	RT	$\|S\|$	RT	$\|S\|$	RT
$q = 1$	119.6	0.31	100.2	0.27	70.6	0.19
$q = 2$	174.0	0.48	168.5	0.48	141.8	0.46
$q = 4$	236.4	0.70	270.7	0.88	261.4	0.92

Table 2: Numerical results for different number of types and different values q

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

For each quadruple of parameters $(n, m, |T|, q)$ and each instance type, 10 instances were generated. Thus, the total number of generated instances is 1620.

The algorithm was implemented using the programming language *C++*. The experiments were done on a portable computer with an Intel Core i7 2.0 GHz processor using a single thread (no parallelization) and 8Gb of Ram.

Computational results showed that both the number of created states and the solution time for the instances of different types are similar. These statistics are 10% larger for the instances of type 1. This shows that instances in which the costs of product types depend on their size are not harder than “non-correlated” instances.

Results for different instance sizes are presented in Tables 1 and 2. Here $|S|$ is the number of created states, and RT is the running time in seconds. Each number in these tables is aggregate for 180 instances with the same characteristics: either with the same number of incoming and outgoing trucks in Table 1, or with the same number of types and the same value q in Table 2. These results show that the dynamic programming algorithm is very fast for the generated test instances. Moreover, it scales well: multiplication of parameter n or m by two increases the running time only by a factor of three, and multiplication of parameter q by two at most doubles the running time. Dependence on parameter $|T|$ is more complex, as the theoretical complexity of the algorithm does not depend on it. When parameter q is small, increasing $|T|$ decreases the running time, while the latter does not much depend on $|T|$ when $q = 4$. The easiest instances are those with a large ratio $|T|/q$. In this case, for a fixed product type, few trucks demand units of this type, i.e. there are less decisions to make.

5 Conclusion

In this paper, we considered the truck scheduling problem at a cross docking terminal with two doors and the objective of minimizing the storage cost. We showed that even a quite restricted version of this problem in 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 is very fast for generated test instances with up to 400 incoming and 400 outgoing trucks, 20 product types and $q = 4$. Also, it scales well when increasing the instance size.

The contribution of the paper is mainly theoretical. However, some practical interest in the dynamic programming algorithm comes from its potential use in heuristics, in which the search neighbourhood is defined in terms of the truck sequences. Thus, an important direction for a future research is to try to extend the dynamic programming algorithm proposed here to more practical situations. The most natural extension concerns the case with multiple doors. Here the truck sequence should include not only departures but also arrivals of trucks. An open question is whether the special case with fixed incoming and outgoing truck sequences of the multiple doors problem is still polynomial or not.

The general case, in which the sequence of trucks is a part of the decision, is also worthy of investigation. Note that the “direct first” property is valid for the general case. If one finds additional structural properties of the problem, this will contribute to development of an efficient exact enumeration-based algorithm for the problem and its more realistic extensions (multiple doors, different arriving times and departures due dates of trucks).

Acknowledgement

The author would like to thank all the referees for the constructive remarks which allowed him to significantly improve the presentation of the paper.

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] A. Boloori Arabani, S. Fatemi Ghomi, and M. Zandieh. A multi-criteria cross-docking scheduling with just-in-time approach. *The International Journal of Advanced Manufacturing Technology*, 49:741–756, 2010.

[3] Nils Boysen and Malte Fliedner. Cross dock scheduling: Classification, literature review and research agenda. *Omega*, 38(6):413 – 422, 2010.

[4] Nils Boysen, Malte Fliedner, and Armin Scholl. Scheduling inbound and outbound trucks at cross docking terminals. *OR Spectrum*, 32:135–161, 2010.

[5] Dirk Briskorn and Joseph Leung. Branch and bound algorithms for minimizing maximum lateness of trucks at a transshipment terminal. Optimization Online, 2677, 2010.

[6] 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.

[7] S. Forouharfard and M. Zandieh. An imperialist competitive algorithm to schedule of receiving and shipping trucks in cross-docking systems. *The International Journal of Advanced Manufacturing Technology*, 51:1179–1193, 2010.

[8] 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.

[9] 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.

[10] Ruslan Sadykov. A polynomial algorithm for a simple scheduling problem at cross docking terminals. Research Report RR-7054, INRIA, 2009.

[11] 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.