Cross docking scheduling to minimize the storage cost: a polynomial special case

Ruslan Sadykov

INRIA Bordeaux — Sud-Ouest, 351 cours de la Libération, 33405 Talence, France

Abstract

At cross docking terminals, products from incoming trucks are sorted according to their destinations and transferred to outgoing trucks using a small temporary storage. Such terminals allow companies to reduce storage and transportation costs in a supply chain. This paper focuses on the operational activities at cross docking terminals.

We consider the trucks scheduling problem with the objective to minimise the storage usage during the product transfer. We show that a simplification of this NP-hard problem in which the departure sequences of incoming and outgoing trucks are fixed is polynomially solvable and propose a dynamic programming algorithm for it. The results of numerical tests of the algorithm on randomly generated instances are also presented.

Keywords: Cross docking, scheduling, complexity, 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].

Email address: Ruslan.Sadykov@inria.fr (Ruslan Sadykov)
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 loaded by products of different types arrive at the receiving door where the products are unloaded. Each truck leaves the platform when it is fully unloaded. An unloaded product can be transferred directly to the shipping door if the outgoing truck currently staying there demands the products of this type and does not yet have enough of them. Otherwise, the unloaded product is transferred to the temporary storage. Each outgoing truck can leave the door only if it is fully loaded by products moved either directly from incoming trucks or from the storage. The objective is to find the best arrival (or departure) order for the whole set of incoming and outgoing trucks to increase the efficiency of the cross docking terminal.

![Figure 1: The cross docking platform](image)

There are several papers in the literature which dealt with this scheduling problem. Yu and Egbelu [2] 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, Fliedner and Scholl [3]. They have proposed some lower bounds and an exact decomposition approach for it. Maknoon, Baptiste and Kone [4] concentrated on the objective function which minimizes the storage cost in a simplified setting in which the sequences of incoming and outgoing trucks are fixed and any outgoing truck demands the products of only one type.

We now formally define the problem we consider. The number of incoming and outgoing trucks are, respectively, \( n \) and \( m \). The number of different product types is \( T \). We will denote the \( i \)-th incoming truck as \( I_i \) and the \( o \)-th outgoing truck as \( O_o \). Each incoming truck \( I_i \) supplies \( a_{it} \) products of type \( t \). Each outgoing truck \( O_o \) demands \( b_{ot} \) products of type \( t \). Let also \( T_o \) be the set of product types demanded by \( O_o \), i.e. \( T_o = \{ t : b_{ot} > 0 \} \). We suppose that each outgoing truck demands products of at most \( q \) different
types, i.e. \( q = \max_{1 \leq o \leq m} | T_o | \). For each type \( t \), the total number of supplied products of this type should be not less than the total number of demanded products of this type, otherwise at least one outgoing truck would not be able to depart fully loaded.

A product of type \( t \) can be moved directly from an incoming truck at the receiving door to an outgoing truck \( O_o \) at the shipping door if the number of products of type \( t \) which are already in \( O_o \) is less than \( b_{ot} \). Otherwise the product should be transferred to the temporary storage at a cost \( c_t \). One product of type \( t \) occupies a volume \( d_t \) in the storage. The total volume of products in the storage should not exceed its total capacity which is \( D \). The problem consists in finding a policy for unloading and loading products such that the total cost is minimized. For convenience, in the rest of the paper, we will use an equivalent objective which is the maximization of the total cost of the directly transferred products. This problem is NP-hard in the strong sense even for a very restricted case, as shown in Appendix Appendix A.

Note that the objective function can be interpreted differently. We can see the storage cost as the difference between the time needed to transfer a product directly from an incoming to an outgoing truck and via the intermediate storage. So, our objective function is equivalent to the total processing time. Although this objective seems similar to the makespan objective used in [3] and [2], the problems studied there are quite different. The main difference is that, in our case, concurrent operations do not allow one to save in cost. In contrast to this, for example in [2], it is more advantageous to discharge a product from an incoming truck to the storage and load a product from the storage onto an outgoing truck at the same time than to do these operations consecutively. As a result, transferring a product via the storage does not always increase the objective function. In other words, our objective suits the case when the workforce is versatile, whereas the objective used in [3] and [2] suits the case, when the the workforce is specialized to either loading or unloading.

In the rest of the paper we concentrate on the problem in which the departure sequences of incoming and outgoing trucks are fixed. Formally this means that, for each \( i \in \{1, \ldots, n - 1\} \), the truck \( I_i \) departs before the arrival of \( I_{i+1} \), and for each \( o \in \{1, \ldots, m - 1\} \), the truck \( O_o \) departs before the arrival of \( O_{o+1} \).

Note that the problems considered in [3] and [2] become easy and polynomially solvable by a linear program when the sequences of incoming and outgoing trucks are fixed. However, our problem with fixed sequences is not
as trivial as it can appear. For the first outgoing truck it is indeed trivial: it just waits until all demanded products are brought and then departs immediately. However, for subsequent outgoing trucks there is a choice: either to take products from the storage and pay a cost or wait for other incoming trucks and receive more products directly.

In [4], Maknoon at al. considered our problem with some restrictions including \( q = 1 \), but left the complexity question open. In this paper, we answer this question by presenting a polynomial dynamic programming algorithm. This work is an updated version of the research report [5].

2. Preliminary observations

We begin with a very important fact which serves as the base for the algorithm.

**Proposition 1.** There exists an optimal policy in which, each time trucks \( I_i \) and \( O_o \) are at the doors, for each \( t \), \( I_i \) transfers directly to \( O_o \) as many products of type \( t \) as possible, i.e. the minimum between the number of products of type \( t \) still available in \( I_i \) and the number of products of type \( t \) which are still demanded by \( O_o \).

**Proof.** It is easy to see that this proposition is correct. Suppose, in an optimal policy, \( I_i \) can transfer \( z \) products of type \( t \) to \( O_o \) but does not do it and “saves” them for consequent outgoing truck(s). Then \( O_o \) is obliged to take these \( z \) products from the storage. In the modified policy, these \( z \) products are transferred directly from \( I_i \) to \( O_o \), and products, transferred directly from \( I_i \) to consequent outgoing truck(s) in the original policy, are taken from the storage. The cost of the modified policy which comply with the proposition do not increase.

We will call a policy which complies with Proposition 1 *direct first*. Each direct first policy is characterized only by a departure order of all trucks. Remember that the departure order is fixed within the set of incoming or outgoing trucks but not fixed within the set of all trucks.

As an example, consider the following departure order of trucks:

\[
I_1, I_2, I_3, O_1, O_2, I_4, O_3, O_4, I_5, I_6, \ldots, I_{n-1}, O_6, \ldots, O_{m-1}, I_n, O_m. \tag{1}
\]

In the policy which corresponds to this order, each incoming truck can directly transfer products only to certain outgoing trucks. This is illustrated
in Figure 2. In this figure, a square \((i,o)\) is marked if only if truck \(I_i\) can directly transfer products to truck \(O_o\).

![Figure 2: Possible direct transfers of products in the policy corresponding to the departure order (1)](image)

Note that some departure orders lead to infeasible direct first policies. First, when outgoing truck \(O_o\) departs, it takes the missed products from the storage. Suppose that incoming truck \(I_i\) is at the door. Then, there is enough products in the storage to complete the demand of \(O_o\) if and only if

\[
\forall t \in T, \quad \sum_{k=1}^{i} a_{kt} \geq \sum_{j=1}^{o} b_{jt}.
\] (2)

For a given \(i\), let \(lo(i)\) be the maximum \(o\) such that \(2\) is satisfied.

Second, when incoming truck \(I_i\) departs, it puts to the storage the products which were not transferred directly. 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

\[
\sum_{t=1}^{T} d_t \cdot \left( \max \left\{ 0, \sum_{k=1}^{i} a_{kt} - \sum_{j=1}^{o} b_{jt} \right\} \right) \leq D.
\] (3)
For a given $o$, let $li(\bar{o})$ be the maximum $i$ such that 3 is satisfied.

In the next two sections, we present a dynamic programming algorithm which finds the best feasible direct first policy.

3. Dynamic programming states

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

Consider one of the following departure orders of trucks:

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

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

Consider now one of the following departure orders of trucks:

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

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

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

To clarify the presentation, we present in Figure 3 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 3 corresponds to the departure order (1).

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

By the inequalities (2) and (3),
Figure 3: 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 $S_{inc}(i, o, f)$ 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 3. 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 $S^{\text{out}}(i, o, f)$ or $S^{\text{inc}}(i, o, f)$, by the definition, we have

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

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 $\mathcal{P}^{\text{out}}(i, o, f)$ and $\mathcal{P}^{\text{inc}}(i, o, f)$ be the sets of paths which contain states $S^{\text{out}}(i, o, f)$ and $S^{\text{inc}}(i, o, f)$ respectively.

**Lemma 1.**

1. For any two direct first paths $P' \in \mathcal{P}^{\text{out}}(i, o, f')$ and $P'' \in \mathcal{P}^{\text{out}}(i, o, f'')$, $f' \neq f''$, 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 $P' \in \mathcal{P}^{\text{inc}}(i, o, f')$ and $P'' \in \mathcal{P}^{\text{inc}}(i, o, f'')$, $f' \neq f''$, 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 $P' \in \mathcal{P}^{\text{inc}}(i^*, o', f')$, $o' < o^*$, and $P'' \in \mathcal{P}^{\text{inc}}(i^*, o^*, f'')$, $o'' < o^*$.

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

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

- $o' < o''$. 

8
In both cases, for every \( t \), truck \( I_i \) cannot not transfer more products of type \( t \) to trucks \( O_o \), \( o'' \leq o < o^* \), in policy \( P' \) than to all trucks \( O_o \), \( o'' \leq o < o^* \), in policy \( P'' \), as the both policies are direct first. Therefore, \( f''_t \leq f''_t \) for all types \( t \in T_{o^*} \).

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 \( S_{\text{inc}}(2, o^*, f_t) \).

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

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

**Proof.** In a state \( S^{\text{out}}(i, o, f) \) or \( S^{\text{inc}}(i, o, f) \), 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 2 \( nm \) canonical values \( f_t \) in all states.

Consider the direct first policies in \( P^{\text{out}}(i', o', f') \). Any such policy is in \( P^{\text{inc}}(i'', o', f'') \) for some \( i'' > i' \), where

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

The next observation is very important. Suppose that truck \( O_o \) received directly \( f_{t^*} \) products of type \( t^* \) from a truck \( I_i \), \( i > i' \), and \( 0 < f_{t^*} < \min\{a_{it^*}, b_{ot^*}\} \). As the policy is direct first, this can happen only if \( O_o \) becomes full for products of type \( t^* \). Therefore, \( O_o \) can receive only zero products of type \( t^* \) from subsequent incoming trucks. We conclude that, for a fixed product type \( t^* \), there is at most one direct first policy in \( P^{\text{out}}(i', o', f') \) such that this policy contains a state \( S^{\text{inc}}(i'', o', f'') \), \( i'' > i' \), with a noncanonical value \( f''_{t^*} \).

Consider now the direct first policies in \( P^{\text{inc}}(i', o', f'') \). Any such policy is in \( P^{\text{out}}(i', o'', f'') \) for some \( o'' > o' \), where

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

Again, suppose that truck \( I_i \) shipped directly \( f_{t^*} \) products of type \( t^* \) to a truck \( O_o \), \( o > o' \), and \( 0 < f_{t^*} < \min\{a_{it^*}, b_{ot^*}\} \). As the policy is direct
first, this can happen only if \( I_{i'} \) ships to \( O_o \) all available products of type \( t^* \). Therefore, \( I_{i'} \) can ship only zero products of type \( t^* \) to subsequent outcoming trucks. We conclude that, for a fixed product type \( t^* \), there is at most one direct first policy in \( P^{inc}(i', o', f') \) such that this policy 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 \( f_{t^*} \) in a direct first state \( S^{inc}(i, o^*, f) \) “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 \( S^{out/in}(i, o^*, f) \) is \( O(qnm) \). From this we conclude that the overall number of direct first states is \( O(qnm^2) \).

4. 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 $\rho$. From the proof of Proposition 2, 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.

```plaintext
1 for t = 1 to T do r[t] ← a_{rt};
2 for t ∈ T_o do r[t] ← a_{rt} - f_{t};
3 v ← V^{inc}(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} \cdot dt[t];
8     if i ≤ 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            $V^{out}(i, j, \{dt[t]\}_{t \in T_j}) \leftarrow v$;
13     for t ∈ T_j 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 $i < i' \leq lo(o) + 1$. 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.
for \( t \in T \) do 
\( r[t] \leftarrow f_t; \)

\( v \leftarrow V_{\text{out}}(i, o, f); \)

for \( k \leftarrow i + 1 \) to \( li(o) + 1 \) do

for \( t \in T \) do

\( dt[t] \leftarrow \min\{b_{\alpha t} - r[t], a_{kt}\}; \)

\( v \leftarrow v + c_t \cdot dt[t]; \)

if \( o \leq lo(k) \) then

if state \( S_{\text{inc}}(k, o, \{dt[t]\}_{t \in T}) \) does not exist then

create it: \( V_{\text{inc}}(k, o, \{dt[t]\}_{t \in T}) \leftarrow -\infty; \)

if \( v > V_{\text{inc}}(k, o, \{dt[t]\}_{t \in T}) \) then

\( V_{\text{inc}}(k, o, \{dt[t]\}_{t \in T}) \leftarrow v; \)

for \( t \in T \) do 
\( r[t] \leftarrow r[t] + dt[t]; \)

\[ \text{Algorithm 2: Algorithm to make moves from a state } S_{\text{out}}(i, o, f). \]

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

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 policy, 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 policy can be obtained by backtracking from the state \( S_{\text{out}}(n, m, f) \) or \( S_{\text{inc}}(n, m, f) \).

**Proposition 3.** The complexity of the dynamic programming algorithm is \( O(q^2nm^2(n + m) \log(qnm)) \).

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

\[
C_{\text{alg}} = O(n(q + \rho)) \cdot O(qnm^2) + O(m(q + \rho)) \cdot O(qnm^2) = O(qnm^2(n + m)(q + \rho)) = O(q^2nm^2(n + m) \log(qnm)).
\]

At the end, we present a dominance rule which speeds up the algorithm. It is quite easy to see that state \( S_{\text{out}}(i, o, f') \) dominates state \( S_{\text{out}}(i, o, f'') \).
for $t \in T_1$ do $\mathbf{d}_t[t] \leftarrow \min\{a_{1t}, b_{1t}\}$;  
2 $v \leftarrow \sum_{t \in T_1} c_t \cdot \mathbf{d}_t[t]$; 
3 $V^{inc}(1,1,\{\mathbf{d}_t[t]\}_{t \in T_1}) \leftarrow v$; 
4 run Algorithm 1 for the state $S^{inc}(1,1,\{\mathbf{d}_t[t]\}_{t \in T_1})$; 
5 $V^{out}(1,1,\{\mathbf{d}_t[t]\}_{t \in T_1}) \leftarrow v$; 
6 run Algorithm 2 for the state $S^{out}(1,1,\{\mathbf{d}_t[t]\}_{t \in T_1})$; 
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

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 the $S^{out}(i,o,f')$, for each type $t$, more products can be potentially transferred directly, and the total cost of the product already transferred directly is larger. The same holds for the states $S^{inc}$. In practice, making only moves from non-dominates states decreases significantly the running time of the algorithm.

5. 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 $[1,a_{\text{max}}]$. The values $b$ are generated is such a way that, for each $o$, at most $q$ values $b_{ot}$ are non-zero and

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

(4)
Let
\[ b_{it}^{av} = \frac{\sum_{i=1}^{n} a_{it} m \cdot q / |T|}{|S|}. \]

Then, for each \( t \), values \( b_{it} \) which are randomly chosen to be non-zero are uniformly distributed in the range \( \left[ \frac{1}{3} b_{it}^{av}, \frac{2}{3} b_{it}^{av} \right] \), and slightly adjusted to satisfy the equality (4).

The following parameters values were used:

\[
\begin{align*}
    & n & 100, 200, 400, 800 \quad & q & 1, 2, 4, 8 \quad & a_{\text{max}} & 10, 1000 \\
\end{align*}
\]

For each triple of parameters \((n, q, a_{\text{max}})\), 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.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( a_{\text{max}} )</th>
<th>( q = 1 )</th>
<th>( q = 2 )</th>
<th>( q = 4 )</th>
<th>( q = 8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>10</td>
<td>12.7</td>
<td>15.2</td>
<td>20.1</td>
<td>30.3</td>
</tr>
<tr>
<td>100</td>
<td>1000</td>
<td>13.4</td>
<td>18.3</td>
<td>24.2</td>
<td>35.5</td>
</tr>
<tr>
<td>200</td>
<td>10</td>
<td>65.9</td>
<td>92.6</td>
<td>140.9</td>
<td>225.5</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>77.0</td>
<td>107.4</td>
<td>167.9</td>
<td>286.2</td>
</tr>
<tr>
<td>400</td>
<td>10</td>
<td>311.8</td>
<td>448.3</td>
<td>740.0</td>
<td>1'259.3</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>364.6</td>
<td>532.5</td>
<td>877.2</td>
<td>1'548.5</td>
</tr>
<tr>
<td>800</td>
<td>10</td>
<td>1'257.4</td>
<td>2'089.8</td>
<td>3'546.2</td>
<td>6'102.8</td>
</tr>
<tr>
<td>800</td>
<td>1000</td>
<td>1'626.4</td>
<td>2'444.2</td>
<td>4'175.1</td>
<td>7'477.3</td>
</tr>
</tbody>
</table>

Table 1: Results of the numerical tests
6. Conclusion

In this paper, we presented a polynomial dynamic programming algorithm for the scheduling problem with fixed sequences of incoming and outgoing trucks to minimise the storage cost in cross docking terminal. This algorithm allows us to determine the computational complexity of the problem for the first time.

Numerical tests showed that the 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 problem. Such a formulation is likely to exist, since the problem was shown to be polynomially solvable. A linear programming formulation would help a lot in developing methods for solving more practical generalisation of the problem in which the truck sequences are not fixed.

Appendix A. NP-hardness proof

Here we consider the general problem in which the sequences of trucks are not fixed. We show the this problem is NP-hard in the strong sense even for the case in which

- each incoming truck supplies products of at most two types,
- each outgoing truck demands products of at most one type,
- all the storage costs are unitary,
- and 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 \( r_1, r_2, \ldots, r_{3n} \) 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, \( 4n \)
outgoing trucks (3n of the first type, n of the second type) and two types of products. The supplies and demands are the following:

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

We claim that there exists a 3-partition if and only if at most \(n\) products are transferred via the storage.

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 \(j\), \(1 \leq j \leq n\), includes set \(A_j\) of incoming trucks and outgoing trucks \(O_{2j-1}, O_{2j}\) and \(O_{3n+j}\). The departure order of group \(j\) is the following

\[
O_{2j-1}, I_{i_{j1}}, I_{i_{j2}}, O_{3n+j}, I_{i_{j3}}, O_{2j}.
\]

As \(r_{i_{j1}} + r_{i_{j2}} + r_{i_{j3}} = B\), the incoming trucks transfer \(6n + B\) products of type 2 directly to truck \(O_{3n+j}\), \(I_{i_{j1}}\) transfers 1 product of type 1 directly to \(O_{2j-1}\), and \(I_{i_{j3}}\) transfers 1 product of type 1 directly to \(O_{2j}\). Only 1 product of type 1 is transferred to the storage from \(I_{i_{j2}}\). These transfers are depicted in Figure A.4. As there are \(n\) groups, \(n\) products in total are transferred to the storage and then put to the outgoing trucks \(O_{2n+1}, \ldots, O_{3n}\), which are sequenced at the end.

Suppose now there is a sequence of trucks such that at most \(n\) products are transferred through the storage. Then, every outgoing truck \(O_i\), \(i = 3n + 1, \ldots, 4n\) must receive products directly from at least 3 incoming trucks. Otherwise it would receive from the storage at least \(2n\) products of type 2. Therefore, at least 1 product of type 1 goes to the storage while each such truck \(O_i\) is supplied, and the total number of products of type 1 transferred through the storage is at least \(n\), meaning that all products of type 2 should be transferred directly. Then, each incoming truck can transfer directly products of type 2 to exactly one outgoing truck. Otherwise, between two outgoing trucks of type 2, only one outgoing truck can receive a product of type 1 directly, and the total number of products of type 1 transferred through the storage would exceed \(n\). 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 $r_{ij1} + r_{ij2} + r_{ij3} < B$, and the outgoing truck which is supplied by the incoming trucks in $A_j$ would need to take at least one product of type 2 from the storage.

References


