Solving a scheduling problem at cross docking terminals

Ruslan Sadykov

EURO XXIV
Lisbon, July 12, 2010
Contents

Introduction

The problem

The algorithm

Numerical experiments

Appendix
Cross docking

- Product of several types should be delivered from several production/distribution points to several costumers.
- The cross docking terminal serves to reallocate goods according to their destinations (costumers) in order to reduce the transportation costs.
Cross-docking scheduling problem

- If a product unit goes to the storage, a cost should be paid.
- An incoming (outgoing) truck leaves the door only if it is fully unloaded (loaded).
- We need to schedule the sequences of incoming and outgoing trucks and obtain a product transfer policy which minimizes the cost.
Negative result

The problem is **NP-hard in the strong sense** even if

- There is only one receiving door and one shipping door.
- Incoming trucks supply products of at most 2 types.
- Outgoing trucks demand products of one type.
- Storage costs are unitary.
- Storage capacity is unlimited.

**Exact methods**

- Yu and Egbelu (2008)
- Boysen, Fliender, and Scholl (2010)
Contents

Introduction

The problem

The algorithm

Numerical experiments

Appendix
Cross docking scheduling problem: notations

- \( n \) incoming trucks, \( m \) outgoing trucks (\( m = n \) for simplicity)
- \( T \) different products types
- An incoming truck \( I_i \) supplies \( a_{it} \) units of product type \( t \)
- An outgoing truck \( O_o \) demands \( b_{ot} \) units of product type \( t \)
- Each outgoing truck demands products of at most \( q \) types
- The cost of storing one unit of product type \( t \) is \( c_t \)
- The volume of a unit of product type \( t \) is \( d_t \)
- Storage capacity is \( D \).
Cross docking scheduling problem: special case

- There is only one receiving door and one shipping door.
- The sequences of incoming and outgoing trucks are fixed:

\[
\begin{array}{cccccc}
I_1 & I_2 & I_3 & I_4 & \cdots & I_n \\
\downarrow & \downarrow & \downarrow & \downarrow & \quad & \downarrow \\
O_1 & I_4 & O_2 & \cdots & O_{n-1} & I_n \\
& O_2 & \cdots & O_{n-1} & O_n & \\
\end{array}
\]

- We need to find
  - an aggregate sequence of truck arrivals/departures,
  - a product transfer policy,

which minimize the storage cost (maximizes the weighted number of product units transferred directly).

- Introduced by Maknoon, Baptiste, and Kone (2009) \((q = 1)\).
A dominance rule

Observation
There exists an optimal policy in which, each time trucks $I_k$ and $O_j$ are at the doors, for each $t$, $I_k$ transfers directly to $O_j$ as many products of type $t$ as possible.

Consequence

- We call a policy complying with the observation direct first.
- For each departure sequence of trucks, there is exactly one direct first policy.
Dynamic programming states: first group

\( S^{out}(i, o, f) \) — departure sequence is

\[ \ldots, O_{o-1}, I_i, \ldots \]

\( f = \{ f_t \}_{t \in T_o}, \quad 0 \leq f_t \leq \min\{a_{it}, b_{ot}\}, \)

\( f_t \) — number of products of type \( t \) transferred from \( I_i \) to \( O_o \)

\( I_{i+1} \)

\( I_i \)

\( O_o \)

\( f = \{ f_t \}_{t \in T_o} \)

Cross-docking platform

Receiving area

Shipping area

Storage area
Dynamic programming states: second group

\[ S^{inc}(i, o, f) \] — departure sequence is

\[ \ldots, I_{i-1}, O_o, \ldots \]

\[ f = \{f_t\}_{t \in T_o}, \quad 0 \leq f_t \leq \min\{a_{it}, b_{ot}\}, \]

\[ f_t \] — number of products of type \( t \) transferred from \( I_i \) to \( O_o \)

Cross-docking platform diagram with receiving area, storage area, and shipping area.
The underlying graph for the dynamic programming

outgoing trucks

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>...</th>
<th>n</th>
</tr>
</thead>
</table>

incoming trucks

- \((1, 1)\)
- \((4, 1)\)
- \((4, 3)\)
- \((5, 3)\)
- \((5, 5)\)
- \((n, 6)\)
- \((n, m)\)

I

<table>
<thead>
<tr>
<th></th>
<th>I_1</th>
<th>I_2</th>
<th>I_3</th>
<th>I_4</th>
<th>I_5</th>
<th>I_6</th>
<th>...</th>
<th>I_n</th>
</tr>
</thead>
</table>

O

<table>
<thead>
<tr>
<th></th>
<th>O_1</th>
<th>O_2</th>
<th>O_3</th>
<th>O_4</th>
<th>O_5</th>
<th>O_6</th>
<th>...</th>
<th>O_n</th>
</tr>
</thead>
</table>
In a state $S(i, o, f)$, for every type $t$,

$$0 \leq f_t \leq \min\{a_{it}, b_{ot}\}.$$

Then, the overall number of states is a pseudo-polynomial of $n$ and an exponential of $q$:

$$|S| = \sum_{i=1}^{n} \sum_{o=1}^{n} \prod_{t: b_{ot} > 0} (\min\{a_{it}, b_{ot}\} + 1) = O(n^2 \cdot AB^q),$$

where $AB = \max_{i, o, t} \min\{a_{it}, b_{ot}\}$.

But the number of direct first states (which correspond to a direct first policy) is polynomial.
Complexity of the dynamic programming algorithm

Theorem
The total number of the direct first states $S^{out}$ is $O(qn^3)$ (same holds for $S^{inc}$).

- Complexity of checking whether a state $S(i, o, f)$ has been already visited is $O(q \log(qn^2)) = \rho$.
- Complexity of making all moves from a state $S(i, o, f)$ is $O(n(q + \rho)) = O(nq \log n)$.

Theorem
The complexity of the dynamic programming algorithm is

$$O(q^2 n^4(q + \log n))$$
Contents

Introduction

The problem

The algorithm

Numerical experiments

Appendix
Test instances

Parameters

- $n = 100, 200, 400, 800$
- $q = 1, 2, 4, 8$
- $|T| = 10q$
- $a_{it} \in U[1, 1000]$
- $c_t \in U[1, 10]$
- storage capacity is unlimited

Number of instances

10 instances generated for each pair $(n, q)$
Numerical results

<table>
<thead>
<tr>
<th>S</th>
<th>— number of the created states, in thousands</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT</td>
<td>— average running time, in seconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>q = 1</th>
<th></th>
<th>q = 2</th>
<th></th>
<th>q = 4</th>
<th></th>
<th>q = 8</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S</td>
<td>RT</td>
<td>S</td>
<td>RT</td>
<td>S</td>
<td>RT</td>
<td>S</td>
<td>RT</td>
</tr>
<tr>
<td>100</td>
<td>13</td>
<td>0.01</td>
<td>18</td>
<td>0.02</td>
<td>24</td>
<td>0.03</td>
<td>36</td>
<td>0.06</td>
</tr>
<tr>
<td>200</td>
<td>77</td>
<td>0.13</td>
<td>107</td>
<td>0.19</td>
<td>168</td>
<td>0.40</td>
<td>286</td>
<td>0.92</td>
</tr>
<tr>
<td>400</td>
<td>365</td>
<td>1.37</td>
<td>533</td>
<td>2.09</td>
<td>877</td>
<td>4.22</td>
<td>1’549</td>
<td>10.05</td>
</tr>
<tr>
<td>800</td>
<td>1’626</td>
<td>15.97</td>
<td>2’444</td>
<td>22.53</td>
<td>4’175</td>
<td>41.56</td>
<td>7’477</td>
<td>93.52</td>
</tr>
</tbody>
</table>

When $n$ doubles, running time is $11.3$ times larger on average.

When $q$ doubles, running time is $1.9$ times larger on average.
Conclusions and perspectives

Conclusion

- We presented a polynomial dynamic programming algorithm for the problem.
- Note that the complexity question was open (even for $q = 1$).

Perspectives

- Linear Programming formulation?
Contents

Introduction

The problem

The algorithm

Numerical experiments

Appendix
Number of different values for $f_t$

- Value $f_t$ is **canonical** in a state $S(i, o, f)$ if
  \[ f_t \in \{0, \min\{a_{it}, b_{ot}\}\}. \]

- From any state $S_{out}(i, o, f)$, we can pass to at most one direct first state $S_{inc}(i', o, f')$, $i' > i$, with a non-canonical value $f_t$.

- From any state $S_{inc}(i, o, f)$, we can pass to at most one direct first state $S_{out}(i, o'', f'')$, $o'' > o$, with a non-canonical value $f_t$.

- Therefore, any state with a canonical value $f_t$ “generates” at most $2n$ direct first states with non-canonical values $f_t$.

- Then, the number of different values for $f_t$ in all direct first states is $O(n^3)$. 
Lemma
For fixed $i^*$ and $o^*$, there are no two direct first states $S^{out}(i^*, o^*, f')$ and $S^{out}(i^*, o^*, f'')$ such that $f'_{t_1} < f''_{t_1}$ and $f'_{t_2} > f''_{t_2}$.

Consequence
For fixed $i^*$ and $o^*$, direct first states $S^{out}(i^*, o^*, f)$ can be lexicographically ordered:

$$S^{out}(i^*, o^*, f') \prec S^{out}(i^*, o^*, f'') \iff f'_t \leq f''_t, \forall t.$$ 

Theorem
The total number of the direct first states $S^{out}$ is $O(qn^3)$ (same holds for $S^{inc}$).