A Column Generation Algorithm for Slab Stack Shuffling (SSS) Problem

Reporter: Xu Cheng
Supervisor: Lixin Tang

The Logistics Institute, Northeastern University, Shenyang, China
http://tli.neu.edu.cn
1. Introduction

2. Literature Review

3. Description of The SSS Problem

4. A Column Generation Algorithm for the SSS

5. Experiment Results
1. Introduction

Steel-making → Continuous Casting → Slab yard → Hot-rolling → Final product → Delivery

Slab yard serves as the storage buffer between continues-casting and hot-rolling, plays a key role in coordinating the product temple between CC and HR.
1. Introduction

(a) conveyor

(b) Stack height

Barrier slabs
(slabs to be shuffled)

Target slab
1. Introduction

- Target slab
  - Carry to the charging roller
- Barrier slab
- Carry onto the conveyor
- On top of the stack
1. Introduction

Order Demands

Items 1, 2, 3, \ldots, M

Candidate Slabs

Slab 1

Slab 2

Slab 3

Slab \( j \)

Slab \( n \)
2. Literature Review

SSS in Steel Industry

- Parallel GA (*Singh et al.*, *IJPE*, 2004)
- Segmented Dynamic Programming-Based Heuristic (*Tang et al.*, *COR*, 2010)

Related Works in Container Terminal

- ILP Model (*Liu et al.*, *NRL*, 2009)
3. Description of The SSS Problem

Parameters:

- $i$ \quad Index of rolling item, $i \in \{1, 2, \ldots, M\}$, which is the set of rolling items in hotrolling schedule;
- $S_i$ \quad Slab family for the $i$th rolling item, $i = 1, 2, \ldots, M$, and $S = \bigcup_{i=1}^{M} S_i$ denotes the set of candidate slabs for all the items;
- $\Phi$ \quad Slab set, $\Phi = \{1, 2, \ldots, j, \ldots, n\}$;
- $\varphi_j$ \quad The stack that slab $j$ initially stored in;
- $D_j$ \quad The number of slabs above slab $j$ in $\varphi_j$.

Decision variables:

$$X_{ij} = \begin{cases} 1, & \text{if slab } j \text{ is selected for the } i\text{th item,} \\ 0, & \text{otherwise.} \end{cases}$$

$S_{ij}$ \quad Number of shuffles occur if slab $j$ is selected for the $i$th item.
\[(IP_1) \quad \text{Min} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} S_{ij} X_{ij} \quad \text{for} \ i = 1, 2, \ldots, M \quad \text{(1)}\]

\[\text{s. t.} \quad \sum_{j \in S_i} X_{ij} = 1 \quad \text{for} \ i = 1, 2, \ldots, M \quad \text{(2)}\]

\[\sum_{i=1}^{M} X_{ij} \leq 1 \quad \text{for} \ \forall \ j \in S \quad \text{(3)}\]

\[X_{ij} \in \{0, 1\} \quad \text{for} \ i = 1, 2, \ldots, M, \ j \in S \quad \text{(4)}\]

where,

\[S_{ij} = \begin{cases} 
D_j - 1, & \text{if } R = \emptyset; \\
D_j - \max D_j - 1, & \text{if } R \neq \emptyset; 
\end{cases} \quad \text{(5)}\]

in which \[R = \left\{ j' \mid \phi_j = \phi_{j'}, D_j > D_{j'}, X_{ij'} = 1, i \in M, j' \in \Phi \right\}\]
Definition 1. For any item \( i, \ i \in \{1, 2, \ldots, M\} \), and slab \( j \in S_i \), we call the binary combination \((j, i)\) a candidate match.

It is clear that for two different candidate matches \((j, i)\) and \((j', i')\), if \(i = i'\) or \(j = j'\), then they cannot be adopted simultaneously.

Definition 2. Two different candidate matches \((j, i)\) and \((j', i')\) are compatible with each other iff \(i \neq i'\) and \(j \neq j'\); otherwise, they are incompatible with each other.

Let \(Sc(\tau) = \{(j, i) \mid \varphi_j = \tau, j \in S_i \text{ and } i = 1, 2, \ldots, M\}\) denote the set of candidate matches involving stack \(\tau\).
Definition 3. A match set $s \subseteq Sc(\tau)$ is called a stack-scheme with respect to stack $\tau$ iff $s \neq \emptyset$ and any two candidate matches in $s$ are compatible with each other.

For example in Figure 3, three candidate slabs $(j_1, j_2, j_3)$ are in a stack and related to four items in stack $\tau$, such as $i_2, i_3, i_4, i_5, i_7$. $Sc(\tau) = \{(j_1, i_2), (j_1, i_4), (j_1, i_5), (j_2, i_2), (j_2, i_3), (j_3, i_2), (j_3, i_4), (j_3, i_7)\}$. According to Definition 3, Candidate match sets $\{(j_1, i_2), (j_2, i_2)\}$, $\{(j_2, i_3), (j_3, i_2)\}$ and $\{(j_1, i_5), (j_2, i_3), (j_3, i_4)\}$ are all stack-schemes with respect to the stack $\tau$, but $\{(j_1, i_2), (j_2, i_2)\}$ is not due to the incompatibility between item $i_2$ which is can not be matched to two slabs.
• The net shuffle number in a stack is independent with the situation in other stacks
• The needed shuffle number associated within a stack-scheme can be individually calculated.
Based on the notations defined above and from the perspective of individual stack, the original model IP_1 can be reformulated as a master problem and a set of subproblems by Danzig-Wolf decomposition.

Each subproblem is to make the optimal stack-scheme decision for each stack by enumerate all its stack-schemes.

The master problem is to decide which stack-scheme will be taken as the final optimal scheme.

Model IP-2 is the master problem of Danzig-Wolf decomposition.
4. A CG Algorithm for the SSS

IP_2:

Parameters:
- $\Gamma$: Set of stacks;
- $\omega_{rs}$: The shuffle number of stack-scheme $s$ of stack $\tau$;
- $\varphi_j$: The stack that slab $j$ initially stored in;
- $D_j$: The number of slabs above slab $j$ in $\varphi_j$;

$\gamma = \bigcup_{\tau \in \Gamma} Sc(\tau)$: The whole set of candidate stack-schemes;

Variables:
- $x_{\tau s} = \begin{cases} 1, & \text{stack-scheme } s \text{ of stack } \tau \text{ is adopted,} \\ 0, & \text{otherwise.} \end{cases}$
- $\mu_{t rs} = \begin{cases} 1, & \text{stack-scheme } s \text{ of stack } \tau \text{ includes a match corresponding to item } i, \\ 0, & \text{otherwise.} \end{cases}$
4. A CG Algorithm for the SSS

\[(\text{IP-2}) \quad \text{Max} \quad \sum_{\tau \in \Gamma} \sum_{s \in S(\tau)} \omega_{\tau s} x_{\tau s}, \quad (6)\]

\[\text{s.t.} \quad \sum_{\tau \in \Gamma} \sum_{s \in S(\tau)} \mu_{i\tau s} x_{\tau s} = 1, \quad \forall i = 1, 2, \ldots M \quad \pi_i \quad (7)\]

\[\sum_{s \in \varphi(\tau)} x_{\tau s} \leq 1, \quad \forall \tau \in \Gamma \quad \nu_{\tau} \quad (8)\]

\[x_{\tau s} \in \{0, 1\}, \quad \forall \tau \in \Gamma, s \in \varphi(\tau) \quad (9)\]

Using the notations above, the reduced cost \(\bar{c}_{\tau s}\) of scheme \(s\) in stack \(\tau\) is given below:

\[\bar{c}_{\tau s} = \omega_{\tau s} - \pi_i \mu_{i\tau s} - \nu_{\tau} \quad (10)\]
4. A CG Algorithm for the SSS

• Given an initial feasible solution, IP_2 is known as the restricted master problem in the column generation context. In a column generation method, the subproblem must be able to find stack-schemes of each stack that have negative reduced cost with regard to a given dual solution to the linear relaxation of the restricted master problem.
4. A CG Algorithm for the SSS

• Subproblem

Find a stack-scheme for each stack with negative or the most negative value of reduced cost. The schemes with negative reduced cost are added to the master problem.

Then a new iteration starts by solving the relaxation of the new master problem and the lower bound is updated to accelerate the search process of the branch-and-bound tree.
• At each node of the branch-and-bound tree, we take the column generation procedure iteratively for solving the linear relaxation of the restricted master problem to get its lower bound.

• At each iteration process, linear relaxation of master problem restricted to a subset of stack-schemes is solved to get dual solutions.

• Given the dual variables, CG is taken until there is no negative column to add.
For testing performance of the proposed algorithm on examples which are of various kinds of scales, 3 groups of instances under different problem configurations are randomly generated. The randomly generated instance has 60 (group 1 and 2) or 90 (group 3) items separately, which is quite the same or even larger than that of practical data.

### Table 1: Experiment results on Random Generated Data

<table>
<thead>
<tr>
<th>Group</th>
<th>Items</th>
<th>Test index</th>
<th>Schemes</th>
<th>Obj</th>
<th>LB</th>
<th>Gap(%)</th>
<th>CPU(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>60</td>
<td>1</td>
<td>1399</td>
<td>35</td>
<td>35</td>
<td>0</td>
<td>14.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1128</td>
<td>53</td>
<td>53</td>
<td>0</td>
<td>13.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1659</td>
<td>26</td>
<td>26</td>
<td>0</td>
<td>14.4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>10520</td>
<td>56</td>
<td>56</td>
<td>0</td>
<td>22.8</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>2</td>
<td>13890</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>28.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>32878</td>
<td>81</td>
<td>81</td>
<td>0</td>
<td>42.0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1678</td>
<td>56</td>
<td>56</td>
<td>0</td>
<td>25.1</td>
</tr>
<tr>
<td>3</td>
<td>90</td>
<td>2</td>
<td>3098</td>
<td>62</td>
<td>62</td>
<td>0</td>
<td>24.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>5003</td>
<td>39</td>
<td>39</td>
<td>0</td>
<td>23.0</td>
</tr>
</tbody>
</table>
For testing the practical value of the algorithm proposed, experiments are also tested on real data collected from a large scale steel-iron enterprise in China which is shown in Table 2.

<table>
<thead>
<tr>
<th>Test index</th>
<th>Initial Obj</th>
<th>Obj</th>
<th>LB</th>
<th>Gap (%)</th>
<th>CPU (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0.6</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>34</td>
<td>34</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>0</td>
<td>1.1</td>
</tr>
<tr>
<td>6</td>
<td>48</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>30</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>14</td>
<td>14</td>
<td>0</td>
<td>0.5</td>
</tr>
</tbody>
</table>
Thank You!