A branch-and-price algorithm to solve the molten iron allocation problem in iron and steel industry

Lixin Tang\textsuperscript{a,}*, Gongshu Wang\textsuperscript{a}, Jiyin Liu\textsuperscript{b}

\textsuperscript{a}The Logistics Institute, Northeastern University, Shenyang, China
\textsuperscript{b}Business School, Loughborough University, Loughborough, Leicestershire LE11 3TU, UK

Available online 20 December 2005

Abstract

The molten iron allocation problem (MIAP) is to allocate molten iron from blast furnaces to steel-making furnaces. The allocation needs to observe the release times of the molten iron defined by the draining plan of the blast furnaces and the transport time between the iron-making and steel-making stages. Time window constraints for processing the molten iron must be satisfied to avoid freezing. The objective is to find a schedule with minimum total weighted completion time. This objective reflects the practical consideration of improving steel-making efficiency and reducing operation cost caused by the need for reheating. Such a problem can be viewed as a parallel machine scheduling problem with time windows which is known to be NP-hard. In this paper, we first formulate the molten iron allocation problem as an integer programming model and then reformulate it as a set partitioning model by applying the Dantzig–Wolfe decomposition. We solve the problem using a column generation-based branch-and-price algorithm. Since the subproblem of column generation is still NP-hard, we propose a state-space relaxation-based dynamic programming algorithm for the subproblem. Computational experiments demonstrate that the proposed algorithm is capable of solving problems with up to 100 jobs to optimality within a reasonable computation time.

© 2005 Elsevier Ltd. All rights reserved.

Keywords: Molten iron allocation; Integer programming; Column generation; Branch-and-price; State-space relaxation; Dynamic programming

1. Introduction

In iron and steel production the iron- and steel-making stages must be well coordinated to achieve high productivity and low-energy consumption. In the iron-making stage, molten iron is smelted from iron ore, limestone and bituminous coal in blast furnaces and then poured into pots carried on torpedo cars. These torpedo cars are hauled by engines through a rail-track network to steel-making plants where the molten iron is used for steel-making. Fig. 1 shows the layout of the rail tracks for molten iron transportation between the iron- and steel-making plants in Shanghai Baoshan Iron and Steel Complex (Baosteel). Before the molten iron is puddled, it needs to undergo a pretreatment. The process of molten iron pretreatment at Baosteel plants is shown in Fig. 2.
Fig. 1. Layout of the molten iron transportation track.

Fig. 2. Flow of molten iron and torpedo cars.
Molten iron scheduling is concerned with the allocation of molten iron to steel-making furnaces and scheduling of the transportation activities between the iron-making and steel-making stages. The whole problem is very complex and therefore often decomposed and solved in the following three steps. First, pots of the molten iron from blast furnaces are allocated to the steel-making furnaces. Next, engines are allocated to the molten iron transportation requests resulted from the molten iron allocation. Finally, transportation routes for the engines and torpedo cars are established. The first step, molten iron allocation, is to determine the allocation of the molten iron from the blast furnaces to the steel-making furnaces with minimal total weighted pretreatment completion time. In the molten iron allocation decision several practical factors such as the temperature reduction of molten iron caused by waiting, the availability of steel-making equipment, the continuous production of the blast furnaces and the transport times must be considered. Insufficient allocation of molten iron will cause the steel-making operation to shut down and the downstream production will be afflicted with significant cost penalties. Exceeding allocation of molten iron, on the other hand, will result in full torpedo cars queuing up in front of the steel-making furnaces, increasing their turnaround times and leaving fewer torpedo cars available for the draining of the blast furnaces. With too few torpedo cars available, the production rate of the blast furnaces will have to be reduced, degrading the molten iron and in the worst case causing catastrophic damage and lengthy down time. In addition, if the temperature of the molten iron is allowed to drop below a certain point due to long waiting, it will need to be reheated, which will increase cost. Even worse, if the delay exceeds a certain number of hours, the molten iron will freeze, destroying the torpedo car. Therefore optimizing the allocation of the molten iron from the blast furnaces to the steel-making furnaces can improve the overall efficiency and reduce energy consumption in the integrated system.

At present the scheduling of molten iron in Baosteel has achieved great success with the help of computer-aided automation, while molten iron allocation, the most critical part of the process, is still accomplished by manual operation. Baosteel is equipped with 3 blast furnaces for iron-making, 5 converters and 1 electric arc furnace for steel-making. Generally, Baosteel dispatchers use the following scheme for molten iron allocation. Molten iron drained from blast furnaces #1 and #2 is assigned to converters #1, #2 and #3; and molten iron drained from blast furnace #3 is assigned to converters #4, #5 and the electric arc furnace. If blast furnaces #1 and #2 cannot supply enough molten iron to converters #1, #2 and #3, blast furnace #3 also supplies molten iron to them. When blast furnace #3 cannot supply enough molten iron to satisfy their demands, blast furnaces #1 and #2 will also supply molten iron to converters #4, #5 and the electric arc furnace.

The performance of the above molten iron allocation scheme relies heavily upon the experiences of the dispatcher. For such a complex allocation problem, it is impossible to obtain an optimal solution by manual operation. In this paper, we formulate the problem as a mathematical programming model and design an efficient algorithm to solve it optimally.

The molten iron allocation problem (MIAP) can be viewed as the following parallel machine scheduling problem with time windows (PMSPTW): There are n jobs and m machines, each of the jobs is associated with a certain processing time, a weight, a release time, and a deadline. Each job needs to be processed on any one of the machines. The goal is to find a non-preemptive schedule that minimizes the total weighted completion time and satisfies the time window constraints which require that the start time of each job is not earlier than its release time and the completion time is not later than its deadline. In the standard notation for scheduling problems, this problem is denoted as P|wj Cj. When the MIAP is viewed as a PMSPTW, each pot of molten iron can be considered as a job and the pretreatment processor at each steel-making furnace can be viewed as a machine; the processing time of a job is the pretreatment time of the pot of molten iron; the release time of a job is the time point at which the pot of molten iron arrives at the steel-making plants (including transport time after it is drained to the pot from a blast furnace); the deadline of a pot of molten iron is the time at which the molten iron has to complete the pretreatment to avoid freezing in the pot.

Bruno et al. [1] showed that the scheduling problem P2||wj Cj is strongly NP-hard. Hence, P|wj Cj can be considered as its special case with zero release times and very loose due dates. Bar-Noy et al. [2] gave constant factor approximation algorithms for parallel machine scheduling problems with release time and deadline constraints to maximize the weights of jobs that meet their deadlines. Jain and Grossmann [3] developed a model combing mixed integer linear programming (MILP) with constraint programming (CP) for the parallel scheduling problem with release time and due date constraints to minimize total processing cost, and proposed a branch-and-bound algorithm to solve it. In the algorithm the relaxed MILP is used to assign jobs to machines and CP is used to find a feasible schedule for the assignment. Chen and Powell [4,5]
proposed column generation-based branch-and-price methods to solve problems of scheduling jobs and job families on parallel machines with objectives of minimizing total weighted completion time and total number of weighted tardy jobs.

Most of the previous work for iron and steel industry (e.g., [6–8]) focuses on scheduling steel-making and hot rolling operations. Little research has addressed molten iron scheduling issues in iron- and steel-making stages. Lübbecke and Zimmermann [9] studied a general engine scheduling problem in iron and steel industry, but the focus was not on the molten iron allocation problem.

In this paper, we first develop an integer programming model for the MIAP, then reformulate it as a set partitioning model by applying the Dantzig–Wolfe decomposition approach, and present an exact branch-and-price algorithm based on column generation to solve it. The rest of the paper is organized as follows. Section 2 presents the characteristics of the molten iron allocation problem and gives the integer programming formulation. The solution methodology combining column generation with branch-and-bound is described in Section 3. Section 4 reports computational results. Finally conclusions are drawn in Section 5.

2. Mathematical formulation of the problem

2.1. Characteristics of the molten iron allocation problem

As discussed in Section 1, the main task of the problem is to determine when and on which pretreatment processor in the steel-making plants each pot of molten iron should be processed. We make the following assumptions before modeling the MIAP:

(1) The draining plan at each blast furnace, the pretreatment time of each pot of molten iron and the time limit for holding the molten iron are known ahead of scheduling.
(2) Constraints on transportation resources are not considered in the MIAP. They will be left to the subsequent torpedo car routing problem.
(3) The pretreatment processors in the steel-making plants are considered identical.
(4) No preemption is allowed.

The characteristics of the molten iron allocation problem can be summarized as follows:

(1) The working procedure, and therefore the time required, for the pretreatment of different pots of molten iron can be different because they contain different chemical elements.
(2) The pretreatment of a pot of molten iron must be completed within a time limit after it arrives at the steel-making plant because if the delay exceeds certain hours the molten iron will freeze, destroying the torpedo car.
(3) Within the time limit, each pot of molten iron should also be processed as quickly as possible because when temperature of the molten iron drops to certain point it needs reheating which will incur significant additional cost.
(4) The objective of molten iron allocation is to find a schedule for pretreatment of all the pots of molten iron in the planning period so as to reduce the heat loss. The heat loss of each pot of molten iron increases with its waiting time (including the constant pretreatment time). To avoid losing a great deal of heat, we should minimize the total weighted waiting time for all pots of molten iron, that is, to minimize \( \sum w_j (C_j - r_j) \), where \( C_j \) is the pretreatment completion time, \( r_j \) is the arrive time, and \( C_j - r_j \) is the waiting time, and \( w_j \) is the weight associated with a pot of molten iron. Because \( r_j \) is a constant known before scheduling, the objective function can be equivalently taken as the total weighted completion time \( \sum w_j C_j \).

2.2. Notation

Before modeling this problem, we assume that the entire planning horizon (e.g., a shift) is divided into small time units such that all the time parameters, such as processing times, release times and deadlines, are of integer time units.
The following symbols are used to define the problem parameters and decision variables.

**Parameters:**

- \( N \): The set of all pots of molten iron, \( N = \{1, 2, \ldots, n\} \), where \( n \) is the total number of pots of molten iron.
- \( M \): The set of all pretreatment processors in the steel-making plants, \( M = \{1, 2, \ldots, m\} \), where \( m \) is the total number of pretreatment processors.
- \( p_j \): The pretreatment time of the \( j \)th pot of molten iron.
- \( r_j \): The release time of the \( j \)th pot of molten iron.
- \( d_j \): The hard deadline of the \( j \)th pot of molten iron, i.e., it must be processed completely before \( d_j \) to avoid freezing.
- \( w_j \): The weight of the \( j \)th pot of molten iron.
- \( P_j \): The set of pots of molten iron that can be performed before the \( j \)th pot of molten iron, \( P_j = \{k \in N | r_k + p_k + p_j \leq d_j\} \).
- \( S_j \): The set of pots of molten iron that can be performed after the \( j \)th pot of molten iron, \( S_j = \{k \in N | r_j + p_j + p_k \leq d_k\} \).

**Decision variables:**

- \( C_j \): the pretreatment completion time for the \( j \)th pot of molten iron.
- \( x_{ij} = \begin{cases} 1 & \text{if the } j \text{th pot of molten iron is processed directly after the } i \text{th pot of molten iron on the same } \\
0 & \text{otherwise.} \end{cases} \)
- \( x_{0j} = \begin{cases} 1 & \text{if the } j \text{th pot of molten iron is the first processed on a pretreatment processor,} \\
0 & \text{otherwise.} \end{cases} \)
- \( x_{jn+1} = \begin{cases} 1 & \text{if the } j \text{th pot of molten iron is the last processed on a pretreatment processor,} \\
0 & \text{otherwise.} \end{cases} \)

2.3. The model

Using the above notation, the MIAP problem can be formulated as the following integer programming model:

\[
\text{Minimize } Z, \text{ with } Z = \sum_{j \in N} w_j C_j \quad (1)
\]

subject to

\[
\sum_{i \in P_j \cup \{0\}} x_{ij} = 1, \quad \forall j \in N, \quad (2)
\]

\[
\sum_{j \in N} x_{0j} \leq m, \quad (3)
\]

\[
\sum_{i \in P_j \cup \{0\}} x_{ij} = \sum_{i \in S_j \cup \{n+1\}} x_{ji}, \quad \forall j \in N, \quad (4)
\]

\[
C_j \geq \sum_{i \in P_j} C_i x_{ij} + p_j, \quad \forall j \in N, \quad (5)
\]

\[
C_j \geq r_j + p_j, \quad \forall j \in N, \quad (6)
\]

\[
C_j \leq d_j, \quad \forall j \in N, \quad (7)
\]

\[
x_{ij} \in \{0, 1\}, \quad \forall i, j \in N. \quad (8)
\]
Objective (1) of the model is to minimize the total weighted completion time for pretreatment of all pots of molten iron. Constraints (2) ensure that each pot of molten iron can only be processed once. Constraint (3) indicates that there are only \( m \) pretreatment processors in steel-making plants for processing molten iron. Constraints (4) serve as the flow conservation constraints similar to those in some net flow problems. Constraints (5) guarantee that the pretreatment of a pot of molten iron can only start after the pot that is scheduled directly before it on the same pretreatment processor completes processing. The time window restrictions of each pot of molten iron are defined by constraints (6) and (7). We can obtain the linear relaxation of the IP model (denoted by LIP) by relaxing the binary constraints (8) to the following:

\[
0 \leq x_{ij} \leq 1, \quad \forall i, j \in N. \tag{9}
\]

### 3. Solution methodology

Four decades ago Dantzig and Wolfe [10] developed a column generation approach for linear programming with block diagonal matrix structure. Column generation algorithm was drastically improved in 1990s with the advances in computer technology, technical method and business tools for solving large-scale linear problems. Column generation has been proven to be one of the most efficient methods for solving very large scale integer problems. It has been successfully used to solve the cutting stock problem [11–13], the machine scheduling problem [4,5], the airline crew scheduling problem [14], generalized assignment problem [15], and the vehicle routing problem [16], etc. Using the ideas from these papers, we have developed a column generation approach embedded in a branch-and-bound algorithm for solving our molten iron allocation problem.

#### 3.1. Column generation

The key idea of column generation is similar to a simplex algorithm but there exists a difference. Instead of pricing out non-basic variables by enumeration, in a column generation approach the most negative reduced cost is found by solving an optimization problem [15]. The column generation approach breaks down the linear programming model to a restricted master problem (a restricted version of the master problem that includes a subset of the columns) and a subproblem. The relationship between the master problem and the subproblem is shown in Fig. 3. The restricted master problem is solved by a simplex algorithm providing shadow prices to be transferred to the subproblem. Based upon the shadow prices the subproblem generates new columns and adds them to the restricted master problem. Then the restricted master problem is re-optimized. This repeats until no new columns can be generated by the subproblem.

The column generation approach is a pricing scheme for solving large-scale linear problems. For a mixed integer program it can only obtain a lower bound. To obtain the exact solution of the mixed integer problem, we need to use a branch-and-bound strategy. Conventional integer programming branching on variables may not be effective because fixing variables can destroy the structure of the pricing problem [17]. Our branching scheme is designed to avoid this problem. This issue will be addressed in detail in Section 3.4.

![Fig. 3. Information flow between the restricted master problem and the subproblem.](image)
3.2. The set partitioning model

In this section, MIAP is reformulated as a set partitioning model by applying the Dantzig–Wolfe decomposition approach. The decomposition splits the IP model into an integer master problem and a subproblem. The master problem retains the objective function (1) and constraints (2) and (3). The subproblem consists of the constraint sets (4)–(8) and an auxiliary objective function which will be detailed later in Section 3.3.

Let \( afii9853 \) be a feasible schedule defined as a sequence of pots of molten iron to be performed within their respective time windows on a certain pretreatment processor in steel-making plants, and let \( afii9821 \) be the set of all feasible schedules. Associated with each \( afii9853 \) \( \in \) \( afii9821 \) is an incidence vector \( afii9838/afii9853 \) whose \( j \)th component \( afii9838/afii9853 \( afii9853 \) equals to 1 if schedule \( afii9853 \) covers the \( j \)th pot of molten iron, and 0 otherwise. The cost of a feasible schedule \( afii9853 \) represented by \( afii9849 \) is the total weighted completion time of the pots of molten iron contained in schedule \( afii9853 \). We introduce a new binary decision variable \( afii9838/afii9853 \) to determine whether a particular \( afii9853 \) \( \in \) \( afii9821 \) is selected or not for the solution of MIAP. With these preparations, the integer master problem can be reformulated as the following set partitioning model:

\[
\text{[SP]} \quad \text{Minimize} \quad Z, \quad \text{with} \quad Z \equiv \sum_{\omega \in \Omega} c_{\omega} \lambda_{\omega} \\
\text{subject to} \quad \sum_{\omega \in \Omega} \lambda_{\omega} \leq m, \quad (11) \\
\sum_{\omega \in \Omega} a_{j\omega} \lambda_{\omega} = 1, \quad \forall j \in N, \quad (12) \\
\lambda_{\omega} \in \{0, 1\}, \quad \forall \omega \in \Omega. \quad (13)
\]

Objective (10) is to minimize the total cost (total weighted completion time) of the selected feasible schedules. Constraints (11) guarantee that no more than \( m \) schedules can be selected for there are only \( m \) pretreatment processors available in the steel-making plants. Constraints (12) require that each pot of molten iron must be covered exactly once by the schedules selected. The linear relaxation of SP denoted by LSP can be obtained by relaxing all \( \lambda_{\omega} \) to continuous variables between 0 and 1.

3.3. Pricing subproblem

Each column in constraints (12) represents a feasible schedule. In the column generation procedure, we should generate such columns with negative reduced cost and add them to the restricted master problem, in other words, the pricing subproblem is to find subsets of pots of molten iron and sequence each pot of molten iron within them respectively to ensure that such feasible schedules have negative reduced cost. Let vector \( \pi \) be an optimal dual solution corresponding to the job assignment constraints in (12) and \( \mu \) be the optimal dual solution corresponding to the convexity constraints in (11). To find a least reduced cost schedule, the objective function of the subproblem can be written as follows:

\[
\text{Minimize} \quad \sum_{j \in J} a_{j\omega} w_j C_j - \sum_{j \in J} a_{j\omega} \pi_j - \mu.
\]

The subproblem can be viewed as a single machine scheduling problem with time windows, denoted by \( 1|r_j, d_j|\sum_{j \in N} a_{j}(w_j C_j - \pi_j) \), where \( \pi_j \) is a fixed value corresponding to the \( j \)th pot of molten iron and \( a_{j\omega} \) is a binary decision variable to determine whether the \( j \)th pot of molten iron should be included in the schedule or not. This problem is still strongly NP-hard because the relatively simpler problems \( 1|r_j|\sum_{j \in N} w_j C_j \) and \( 1|d_j|\sum_{j \in N} w_j C_j \) have been proved to be NP-hard in strong sense [18].

To remedy the difficulty in solving the pricing subproblem, Desrochers et al. [16], Van den Akker et al. [19] and Chen and Powell [4], have used an approach that allows some infeasible columns to be added to the restricted master problem. The idea behind the approach is to solve the pricing subproblem in a relaxed version. In other words, we can adopt the state-space relaxation approach to solve the subproblem.
Let $S \subseteq N$ be an arbitrary subset of all pots of molten iron, and the state $(S, t)$ indicates that all pots of molten iron in $S$ have been processed exactly once on the same pretreatment processor and the processing terminates at time $t$. Let function $f(S, t)$ be the minimum cost of state $(S, t)$. Then, the recursion function for dynamic programming to solve the pricing subproblem can be expressed as follows:

$$f(S, t) = \begin{cases} \min_{i \in S} \{g(S - \{i\}, t - p_i) + w_i t - \pi_i\} & \text{if } t \in [r_i + p_i, d_i], \\ \infty & \text{otherwise.} \end{cases}$$  \hspace{1cm} (15)$$

$$g(S, t) = \min_{t \leq t} f(S, t).$$  \hspace{1cm} (16)

It is can be seen that the number of $S$ sets is exponential which leads to exponential number of states as well. The computational time required to obtained the minimum $f(S, t)$ would be $O(n^2)$. Thus the pricing subproblem cannot be solved efficiently by the above dynamic programming recursion function.

We now relax the original state-space of (15) into a new state-space. In the new state-space, state $(j, t)$ means that the $j$th pot of molten iron is scheduled last and complete at time $t$. Let $f(j, t)$ be the minimum reduced cost of the state. We can obtain the following dynamic programming recursion:

$$f(j, t) = \min_{i \in P_{j \cup \{0\}}} \{g(i, t - p_j) + w_j t - \pi_j\}, \quad \text{for } t \in [r_j + p_j, d_j] \text{ and } j \in N,$$

$$g(j, t) = \min_{t \leq t} f(j, t).$$  \hspace{1cm} (17)$$

The initialization of $f(j, t)$ can be obtained by setting

$$f(j, t) = \begin{cases} 0, & j = 0, t = 0, \\ \infty, & t < r_j + p_j, \text{ or } t > d_j, \end{cases} \quad \text{for } j \in N.$$  \hspace{1cm} (19)

It is not difficult to prove that the number of states in this new state-space is bounded by $nT$ where $T$ is defined as $\max_{j \in N} \{d_j - r_j - p_j + 1\}$ and the computation of $f(j, t)$ requires $O(nT)$ time only. So the complexity of the dynamic programming algorithm is bounded by $O(n^2T^2)$.

We need to point out that each original state $(S, t)$ can be backtracked to a feasible schedule, while the new state $(j, t)$ can be backtracked to a feasible schedule or a pseudo-schedule which allows all pots of molten iron to be scheduled more than once within their respective time windows. For example, let sequence $(j_1, j_2, \ldots, j_h)$ be a schedule, if $j_i \neq j_k$, for all $i, k \in \{1, 2, \ldots, h\}$, we call it a feasible schedule; otherwise, it is called a pseudo-schedule.

Both feasible schedules and pseudo-schedules are generated by the dynamic programming algorithm based on state-space relaxation, and both of them with negative reduced cost are allowed to be added to the restricted master problem. For any column $a_{ij}$ corresponding to a pseudo-schedule in the SP model, every component of $a_{ij}$ is an integer constant, which is a binary constant corresponding to a feasible schedule. It is not difficult to prove that allowing columns corresponding to pseudo-schedules will not affect the optimal solution of the SP formulation due to the integrality of variables $\hat{\lambda}_h$ and constraint (12) guarantees that columns corresponding to pseudo-schedules will not be selected in any integer solution of SP. LSP formulation is likely to be looser and the lower bound of SP obtained by LSP will be weaker when the pseudo-schedule is allowed to be added to the restricted master problem. The introduction of state-space relaxation approach to the subproblem can speed up the solution process, though it loosens the lower bound of the master problem and increases the difficulty of searching in the branch-and-bound tree. It is a compromise between the dynamic programming algorithm for the subproblem and the branch-and-bound algorithm for the master problem. The computational results showed in Section 4.2 indicate that the proposed compromise is potent for our problem.

### 3.4. Branching strategy

The solution from the LP relaxation of the restricted master problem may be not integral. To find an integer optimal solution we need to apply a branch-and-bound procedure. Ryan and Foster [20] suggested a general branching strategy for set partitioning problems based on the fact that any sum of variables covering a pair of constraints $\sum_{o \in \Omega_s} a_{jo} \hat{\lambda}_o = 1$
lies in the interval $[0, 1]$. Branching decisions for our problem are made on the following variables based on the idea of their paper:

$$x_{ij} = \sum_{\omega \in \Omega} \delta_{ij}^{\omega},$$

where $\delta_{ij}^{\omega}$ is the number of times that the $i$th pot of molten iron is processed directly before the $j$th pot of molten iron in schedule $\omega$.

For any node that needs branching in the search tree, the following three steps should be performed. First, the value of $x$ variables can be calculated through expression (20) based on the solution of the current LSP. Second, we need to decide which $x_{rs}$ is selected as a branching variable. Generally, we select the one whose value is closest to 0.5. Third, two new branching nodes are created, the left one with $x_{rs}$ fixed to 0, and the right one with $x_{rs}$ fixed to 1. Meanwhile, we should modify the subproblem structure by setting $P_r = P_r \setminus \{r\}$, $S_r = S_r \setminus \{s\}$ for the left node and $P_r = \{r\}$, $S_r = \{s\}$ for the right node. Such branching decision is able to partition the solution space effectively while not changing the nature of the pricing subproblem. Fig. 4 illustrates the whole branching and pruning process performed on an example search tree.
3.5. Heuristic algorithm for initial feasible solution

To start up the column generation algorithm, an initial feasible solution must be provided. The initial restricted LSP master problem consists of the initial feasible solution. Solving this restricted LSP master problem we can obtain the dual prices. This information can then be used in the pricing problem. We construct two different heuristic algorithms for the initial restricted master problem of LSP. The first heuristics takes the idea from Bar-Noy et al. [2] and will be called the serial-mechanism heuristics. The second proposed by us will be called the parallel-mechanism heuristics.

The details are described below.

We first define the following additional notation before presenting the algorithms:

- $J_n$: the set of pots of molten iron that have not been scheduled.
- $M_n$: the set of pretreatment processors that have not been selected.

Heuristics 1 (serial-mechanism algorithm):

1. Initialization, let $M_n = M, J_n = N, t = \min\{r_k \mid k \in J_n\}$.
2. If $M_n = \emptyset$, stop; otherwise select a pretreatment processor $i$ from $M_n$, and update $M_n = M_n\backslash\{i\}$.
3. Select the $j$th pot of molten iron that can finish earliest among those in $J_n$ which can be scheduled at $t$ or later, in other words, $j = \arg\min_{k \in J_n}\{\tau + p_k \mid \tau \geq t, \tau \geq r_k, \tau + p_k \leq d_k\}$.
4. If no such pot of molten iron exists, go to Step 2.
5. Schedule the $j$th pot of molten iron on pretreatment processor $i$ at time $\tau$; update set $J_n = J_n\backslash\{j\}, t = t + p_j$.
6. If $J_n = \emptyset$, stop; otherwise, go to Step 3.

Heuristics 2 (parallel mechanism algorithm):

1. Initialization, $M_n = M, J_n = N, R = \min\{r_j \mid j \in J_n\}, D = \max\{d_j \mid j \in M_n\}, t = R, A_i = 0$ for all $i \in M_n$.
2. If $t > D$, stop.
3. Select a pretreatment processor $i$ that is free at $t$ or later, $t \geq A_i$.
4. If no such pretreatment processor exists, $t = t + 1$, go to Step 2.
5. Select the $j$th pot of molten iron that can finish earliest among those in $J_n$ which can be schedule at $t$ or later, in the other words, $j = \arg\min_{k \in J_n}\{\tau + p_k \mid \tau \geq t, \tau \geq r_k, \tau + p_k \leq d_k\}$.
6. If no such pot of molten iron exists, $t = t + 1$, go to Step 2.
7. Schedule the $j$th pot of molten iron on pretreatment processor $i$ at time $\tau$; update set $J_n = J_n\backslash\{j\}, A_i = A_i + \tau + p_j$.
8. If $J_n = \emptyset$, stop; otherwise, $t = t + 1$ and go to Step 2.

If a feasible solution cannot be found by heuristics 1, heuristics 2 is employed to do this. If both fail, we relax the deadline constraint, use heuristics 2 to obtain a solution that does not satisfy time window constraint, and then use local search to improve it. Barnhart et al. [17] pointed out that the initial feasible solution can always be found using a two-phase method similar in spirit to the two-phase method incorporated in simplex algorithms: add a set of artificial variables with large costs and with associated columns forming an identity matrix.

3.6. Implementing the branch-and-price algorithm

In this section, we describe the implementation details of the branch-and-price algorithm for our problem. A flow chart of the algorithm is shown in Fig. 5.

The whole algorithm works as follows:

- **Step 1**: Initialization, input problem data, construct data structure, set parameters.
- **Step 2**: Using heuristic algorithms to generate an initial feasible restricted master problem.
- **Step 3**: Initialize the column pool to be empty.
- **Step 4**: Solve the current restricted master problem that provides the dual prices.
- **Step 5**: Delete non-basic columns with high positive reduced costs from the restricted master problem.
- **Step 6**: If the column pool still contains columns with negative reduced costs, select a subset of them and add to the restricted master problem, go to Step 4.
- **Step 7**: Empty the column pool.
Step 8: Solve the pricing subproblem under the dual prices provided from Step 4 to generate one or more columns with negative reduced costs using the dynamic programming approach based on state-space relaxation. If there are columns generated, add them to the column pool and go to Step 6; otherwise, go to Step 9.

Step 9: If the solution of linear relaxation contains fractional values for integer variables, execute branch and bound algorithm.
4. Computational experiment

To test the performance of the algorithm and study the characteristics of the solution, we conducted a computational experiment on a range of test problems. The test problem instances were generated at random but the parameter settings were designed to reflect the practical situations in the iron and steel industry. Our algorithm was coded in Visual C++ and the linear programs embedded in branch-and-bound procedure were solved using IBM OSL. The computational experiment was carried out on a Pentium-V 2.4-GHz PC.

4.1. Generation of the test problems

To generate representative problem instances, we examined the actual production data from Baosteel, China. About 30 pots of molten iron are drained from 4 tapholes at the bottom of each blast furnace in a workday. Three blast furnaces are available for iron-making, so there are about 90 pots of molten iron to be scheduled in each workday and about 30 pots of molten iron to be scheduled in each shift of 8 h. The number of pots of molten iron that can simultaneously receive pretreatment in two steel-making plants is 6 for there are 5 converters and 1 electric arc furnace available for steel-making. Data on time windows are not shown in past records, but they can be extracted from the manual schedule results in the past.

Based on the above raw data, the test problem instances in one shift, two shifts, and one workday, etc. were generated randomly using the approach proposed by Gélinas and Soumis [21]. The details are outlined as follows:

1. The processing times were randomly generated from discrete uniform distribution $U[20, 50]$ or $U[10, 50]$.
2. The weight of all jobs (a job is a pot of molten iron) is fixed to 1.
3. An initial schedule is constructed by first selecting $m$ jobs at random and scheduling them on each machine (a machine is a pretreatment processor) at time 0, then choosing machines and jobs randomly and arranging the selected job $j$ on the selected machine at the earliest possible time $t_j$.
4. The release time and deadline of each job $j$ are obtained by setting $r_j = t_j - U[1, w]$, $d_j = t_j + p_j + U[1, w]$, where random numbers are integer-valued, $w$ is a parameter defining the average width of time window.
5. Adjustments are made to avoid negative release times for the jobs by increasing all jobs’ release times and deadlines by $\min\{\min_{j \in N^t r_j}, 0\}$ units. More precisely, $r_j = r_j + \min\{\min_{j \in N} r_j, 0\}$, $d_j = d_j + \min\{\min_{j \in N} r_j, 0\}$.

Three parameters are chosen to represent the problem structure as described below:

1. The number of jobs $n \in \{20, 30, 40, 50, 60, 80, 100\}$.
2. The number of machines $m \in \{3, 4, 5, 6, 8, 10, 12, 14, 16, 20\}$.
3. The average width of time window $w \in \{60, 80\}$.

Form the combination of parameter levels 70 problem scenarios were selected, and for each scenario, 25 different problem instances were randomly generated. Thus totally 1750 problem instances were used in the experiment.

4.2. Computational results

The computational results for the different average time window widths are reported in Tables 1 and 2, respectively. The headers of the columns in each table are interpreted as follows. Columns (1) and (2) are number of pots of molten iron and number of pretreatment processors, respectively, which are used to represent the problems structure. Columns (3) and (4) present the average and maximum integrality gaps, respectively between the optimal integer solution and the lower bound at the branch-and-bound root node. Column (5) is the number of instances out of 25 for which branching was not required. Columns (6) and (7) are the average and the maximum numbers of search tree nodes, respectively. Columns (8) and (9) present the average and the maximum computation times in seconds, respectively for the problems.
### Table 1
Computational results for problems with processing times drawn from the uniform distribution $U[20, 50]$ and average time windows width equal to 60

<table>
<thead>
<tr>
<th>Problem</th>
<th>Integrality gap</th>
<th>Problems solved at root node</th>
<th>B&amp;B nodes</th>
<th>CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg. (%)</td>
<td>Max. (%)</td>
<td>Avg.</td>
<td>Max.</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>0.00</td>
<td>24/25</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>0.00</td>
<td>23/25</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>0.00</td>
<td>22/25</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>0.00</td>
<td>24/25</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>0.00</td>
<td>25/25</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>0.01</td>
<td>16/25</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>0.01</td>
<td>19/25</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>5</td>
<td>0.01</td>
<td>20/25</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>0.00</td>
<td>20/25</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>0.01</td>
<td>18/25</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>3</td>
<td>0.04</td>
<td>15/25</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>0.11</td>
<td>13/25</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
<td>0.01</td>
<td>11/25</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>6</td>
<td>0.01</td>
<td>15/25</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>8</td>
<td>0.00</td>
<td>16/25</td>
<td>2</td>
</tr>
<tr>
<td>50</td>
<td>3</td>
<td>0.04</td>
<td>14/25</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>0.07</td>
<td>8/25</td>
<td>5</td>
</tr>
<tr>
<td>50</td>
<td>5</td>
<td>0.03</td>
<td>7/25</td>
<td>5</td>
</tr>
<tr>
<td>50</td>
<td>6</td>
<td>0.03</td>
<td>9/25</td>
<td>8</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>0.00</td>
<td>10/25</td>
<td>4</td>
</tr>
<tr>
<td>60</td>
<td>6</td>
<td>0.06</td>
<td>7/25</td>
<td>10</td>
</tr>
<tr>
<td>60</td>
<td>8</td>
<td>0.01</td>
<td>9/25</td>
<td>9</td>
</tr>
<tr>
<td>60</td>
<td>10</td>
<td>0.02</td>
<td>8/25</td>
<td>7</td>
</tr>
<tr>
<td>60</td>
<td>12</td>
<td>0.01</td>
<td>9/25</td>
<td>4</td>
</tr>
<tr>
<td>60</td>
<td>14</td>
<td>0.00</td>
<td>8/25</td>
<td>4</td>
</tr>
<tr>
<td>80</td>
<td>8</td>
<td>0.07</td>
<td>2/25</td>
<td>21</td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td>0.02</td>
<td>1/25</td>
<td>12</td>
</tr>
<tr>
<td>80</td>
<td>12</td>
<td>0.03</td>
<td>2/25</td>
<td>14</td>
</tr>
<tr>
<td>80</td>
<td>16</td>
<td>0.00</td>
<td>6/25</td>
<td>8</td>
</tr>
<tr>
<td>80</td>
<td>20</td>
<td>0.00</td>
<td>7/25</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>8</td>
<td>0.10</td>
<td>4/25</td>
<td>30</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>0.11</td>
<td>0/25</td>
<td>22</td>
</tr>
<tr>
<td>100</td>
<td>12</td>
<td>0.09</td>
<td>1/25</td>
<td>28</td>
</tr>
<tr>
<td>100</td>
<td>16</td>
<td>0.02</td>
<td>1/25</td>
<td>16</td>
</tr>
<tr>
<td>100</td>
<td>20</td>
<td>0.00</td>
<td>4/25</td>
<td>14</td>
</tr>
</tbody>
</table>

From the results presented in Tables 1 and 2, the following observations can be made concerning our branch-and-price algorithm.

1. The average integrality gap of all test problems is less than 0.2%, which indicates that LSP formulation can provide a tight lower bound, though pseudo-schedules were added to the restricted master problem in some cases making the compact LSP formulation looser to some extent. So, we can conclude that it is worthwhile to introduce the state-space relaxation approach to the subproblem for some specific set partitioning formulations.

2. For most test problems, our algorithm can find optimal solutions with reasonable CPU time. It indicates that column generation approach embedded in the branch-and-bound framework has great potential for solving combinatorial optimization problems.

3. When the number of pots of molten iron is fixed, as the number of machines increases, the computation time decreases. This is consistent with the intuition that for a fixed number of jobs, when the number of machines is larger, resource is less demanded and the problem becomes easier to solve.
## Table 2
Computational results for problems with processing times drawn from the uniform distribution $U[10, 50]$ and average time windows width equal to 80

<table>
<thead>
<tr>
<th>Problem</th>
<th>Integrality gap</th>
<th>Problems solved at root node</th>
<th>B&amp;B nodes</th>
<th>CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg. (%)</td>
<td>Max. (%)</td>
<td>Avg.</td>
<td>Max.</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>m</td>
<td>23/25</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>0.00</td>
<td>0.04</td>
<td>21/25</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>0.00</td>
<td>0.00</td>
<td>24/25</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
<td>25/25</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>0.00</td>
<td>0.00</td>
<td>16/25</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>12/25</td>
</tr>
<tr>
<td>30</td>
<td>5</td>
<td>0.00</td>
<td>0.00</td>
<td>9/25</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>0.00</td>
<td>0.00</td>
<td>7/25</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>0.00</td>
<td>0.00</td>
<td>5/25</td>
</tr>
<tr>
<td>40</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
<td>21/25</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>16/25</td>
</tr>
<tr>
<td>40</td>
<td>8</td>
<td>0.00</td>
<td>0.00</td>
<td>13/25</td>
</tr>
<tr>
<td>50</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
<td>10/25</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>7/25</td>
</tr>
<tr>
<td>50</td>
<td>5</td>
<td>0.00</td>
<td>0.00</td>
<td>5/25</td>
</tr>
<tr>
<td>50</td>
<td>6</td>
<td>0.00</td>
<td>0.00</td>
<td>3/25</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>0.00</td>
<td>0.00</td>
<td>2/25</td>
</tr>
<tr>
<td>60</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
<td>11/25</td>
</tr>
<tr>
<td>60</td>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>9/25</td>
</tr>
<tr>
<td>60</td>
<td>8</td>
<td>0.00</td>
<td>0.00</td>
<td>7/25</td>
</tr>
<tr>
<td>60</td>
<td>10</td>
<td>0.00</td>
<td>0.00</td>
<td>5/25</td>
</tr>
<tr>
<td>60</td>
<td>12</td>
<td>0.00</td>
<td>0.00</td>
<td>3/25</td>
</tr>
<tr>
<td>60</td>
<td>14</td>
<td>0.00</td>
<td>0.00</td>
<td>1/25</td>
</tr>
<tr>
<td>60</td>
<td>16</td>
<td>0.00</td>
<td>0.00</td>
<td>0/25</td>
</tr>
<tr>
<td>80</td>
<td>8</td>
<td>0.00</td>
<td>0.00</td>
<td>0/25</td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td>0.00</td>
<td>0.00</td>
<td>0/25</td>
</tr>
<tr>
<td>80</td>
<td>12</td>
<td>0.00</td>
<td>0.00</td>
<td>0/25</td>
</tr>
<tr>
<td>80</td>
<td>14</td>
<td>0.00</td>
<td>0.00</td>
<td>0/25</td>
</tr>
<tr>
<td>80</td>
<td>16</td>
<td>0.00</td>
<td>0.00</td>
<td>0/25</td>
</tr>
<tr>
<td>80</td>
<td>18</td>
<td>0.00</td>
<td>0.00</td>
<td>0/25</td>
</tr>
<tr>
<td>80</td>
<td>20</td>
<td>0.00</td>
<td>0.00</td>
<td>0/25</td>
</tr>
</tbody>
</table>

The sign "—" means that there are some instances in this scenario which cannot be solved to optimality in reasonable time. And we terminated the algorithm when EMS memory is exceeded.

(4) With the increase of the problems scale, the number of hard instances increases as well, which can be seen from the maximum branch node and the maximum computation time.

(5) For problems with the same numbers of jobs and machines, the width of the time windows strongly affects the performance of the algorithm. When the width of time windows is small, the number of feasible schedules and pseudo-schedules is relatively small, so the pricing subproblem is much easier to solve, and the computation time required is shorter.

5. Conclusions

In this paper the molten iron allocation problem, which can be viewed as a parallel machine schedule problem with time windows, was studied. The objective was to find a reasonable assignment of molten iron to reduce the reheating cost while guaranteeing the continuous production of the blast furnace. The problem was formulated as
an integer-programming problem. Applying the Dantzig–Wolfe decomposition approach, the integer programming problem was reformulated as a set partitioning model. An exact branch-and-price algorithm based on column generation was proposed to solve it. Computational results show that our algorithm is efficient and effective to solve medium-sized problems. The method can be modified and extended to other types of scheduling problems that can be reduced to parallel machine scheduling problems $P|\sum w_j C_j$ or $R|\sum w_j C_j$ with some technological constraints.

Acknowledgements

This research is partly supported by National Natural Science Foundation for Distinguished Young Scholars of China (70425003), National Natural Science Foundation of China (Grant no. 60274049) and (Grant no. 70171030), Fok Ying Tung Education Foundation and the Excellent Young Faculty Program and the Ministry of Education, China. The authors would like to thank the anonymous referees for their valuable comments on an earlier draft of the paper.

References