An Iterated Greedy Matheuristic for Scheduling in Steelmaking-Continuous Casting Process

Juntaek Hong, Kwansoo Lee, Kangbok Lee, Kyungduk Moon

An Iterated Greedy Matheuristic for Scheduling in Steelmaking-Continuous Casting Process

Juntaek Hong, Kwansoo Lee, Kangbok Lee ✉, and Kyungduk Moon

Pohang University of Science and Technology, South Korea
kblee@postech.ac.kr

Abstract. The steelmaking-continuous casting (SCC) is a bottleneck process in the steel production. Due to elevated product variety and environmental restrictions on the steelmaking industry, efficient operation of the SCC has become more crucial. This paper considers an SCC scheduling problem to minimize the weighted sum of total waiting time, total earliness, and total tardiness while satisfying the maximum waiting time and the continuous casting constraints. We propose a generic mixed integer linear programming (MILP) model that can express various SCC scheduling requirements. Using the MILP model, we develop an iterated greedy matheuristic inspired by the iterated greedy method. An initial SCC schedule is constructed by solving small MILP models one after another. Then, it is improved by solving a series of small MILP models representing the destruction and construction of the prior schedule. Through a numerical experiment, we show that the proposed algorithm can obtain efficient solutions in a short time and outperforms an NSGA-II algorithm for most test cases of practical size.

Keywords: SCC scheduling · MILP · iterated greedy · matheuristic

1 Introduction

Due to a compelling pressure on environmental sustainability, strict regulations have been applied to steel companies on expanding their production capacity. In addition, customers require a variety of products that are produced with different processes. This calls for better operations and scheduling capabilities to fully utilize the running equipment. Steel production involves three main phases called ironmaking, steelmaking-continuous casting (SCC), and rolling [6]. Among these three phases, the SCC is a bottleneck process and thus its scheduling has a significant impact on productivity.

The SCC process consists of three consecutive stages: steelmaking, refining, and continuous casting. During the SCC process, molten iron is processed as a batch called a *charge*, which has a target chemical composition. The *steelmaking* stage is to produce molten steel from molten iron by removing impurities and adding particular substances. Through *refining* stages, the chemical composition of a charge is further adjusted to a target value. A steel company may have different types of refining stages according to applied technologies such as CS, LF,

RH, and CASOB [3]. These refining stages often have an order that a charge is processed through; however, a charge may skip certain refining stages according to its associated final product. The *continuous casting* stage is to make a solid and flat cuboid called a *slab* from multiple charges. To produce slabs, charges must be continuously poured into a machine one by one without idle time. A sequence of charges to be cast continuously in a casting machine is called a *cast*.

The main challenge in SCC scheduling comes from continuous casting and strict waiting time constraints between stages. By continuous casting restrictions, charges in a cast must arrive at the continuous casting stage in a timely manner. This makes the SCC scheduling difficult since all of the previous stages for those charges need to be finished within a very short time range. Moreover, the temperature of a charge needs to be maintained during waiting time between two processes. If the temperature drops, the charge must be reheated with additional cost and the charge may have serious quality issues. To prevent this, the waiting time between two consecutive stages for each charge has a maximum limit. These two constraints are essential in the SCC scheduling; however, it is difficult to achieve them at the same time. Some recent studies have used artificial bee colony algorithm [7], non-dominated sorting genetic algorithm-II (NSGA-II) [5], and Lagrangian relaxation [1] to solve SCC scheduling problems. We refer to [4] as the summary for previous studies.

In this paper, we propose a generic mixed integer linear programming (MILP) model that can handle most of the practical requirements found in the literature such as stage skipping and unrelated speed of machines. We present our model and its assumptions in Section 2. In Section 3, we develop an algorithm consisting of initial solution heuristic and improving heuristic using our MILP model. Our algorithm employs small MILP subproblems to express hard constraints, which is the key difference from the previous (meta)heuristics. Our numerical experiment in Section 4 shows that the MILP-based algorithm outperforms an NSGA-II for most test instances.

2 Problem description

We consider the steelmaking stage (SM), multiple refining stages in an order (e.g., RF1, ..., RF3), and the continuous casting stage (CC). Each stage has unrelated parallel machines; in other words, the processing time of a charge at a stage depends on the machine to which the charge is assigned. The transportation time between two stages depends on the pair of machines that a charge is processed by. A cast has a sequence of charges that must be continuously processed at the CC stage without idle time at the CC stage. We model this condition as a large penalty for the idle time in the objective. We call this the *cast break penalty*, which must be minimized to be 0 through optimization. At the last stage, the first charge of a cast needs a setup time of a given duration. A charge has its own given route for visiting stages such that (i) the route follows a stage sequence, (ii) SM is the first stage, (iii) CC is the last stage, and (iv) some refining stages can be skipped. The waiting time of a charge between two

consecutive stages in its route has a maximum limit. Each charge has a due date at the CC stage. If processing of a charge at the CC stage is completed earlier or later than its due date, earliness penalty or tardiness penalty is incurred proportional to the time difference. The objective is to minimize the weighted sum of penalties for cast break, the total waiting time of charges, the total earliness, and the total tardiness. We use the following notations to develop our MILP model.

Parameters

Ω	The set of all charges, $\Omega = \{1, 2, ..., n\}$ where n is the number of charges
K	The set of all casts, $K = \{1, 2, ..., m\}$ where m is the number of casts
Ω_k	The sequence of charges in cast k, $\Omega_k := \{\Omega_k[1], \Omega_k[2], ..., \Omega_k[n_k]\}$ where n_k is the number of charges in cast k $(\forall k \in K)$
$\hat{\Omega}_k$	The set of pairs of two consecutive charges in cast k, $\hat{\Omega}_k := \{(\Omega_k[s], \Omega_k[s+1]) : s \in \{1, 2, \ldots, n_k - 1\}\}$ $(\forall k \in K)$
J	The sequence of all stages, $J = \{1, 2, ..., l\}$ where l is the last stage for CC
J_i	The sequence of stages in charge i's route, $J_i := \{J_i[1], J_i[2], ..., J_i[l_i]\}$ $(\forall i \in \Omega)$ where l_i is the number of stages in charge i's route $(\forall i \in \Omega)$; $J_i[1] = 1, J_i[l_i] = l$
$\hat{J}_i$	The set of pairs of two consecutive stages in the route of charge i, $\hat{J}_i := \{(J_i[w], J_i[w+1]) : w \in \{1, 2, \ldots, l_i - 1\}\}$ $(\forall i \in \Omega)$
H_j	The set of machines at stage j $(\forall j \in J)$
p_{ijh}	The processing time of charge i at stage j on machine h $(\forall i \in \Omega, j \in J_i, h \in H_j)$
$t_{h,h'}$	The transportation time from machine h to h' $(\forall h, h' \in \bigcup_{j \in J} H_j)$
r_{ij}	The earliest release time of charge i at stage j given as $r_{i1} := 0$ and $r_{ij'} := r_{ij} + \min_{h \in H_j, h' \in H_{j'}} \{p_{ijh} + t_{h,h'}\}$ for $\forall i \in \Omega, (j, j') \in \hat{J}_i$
S_{kh}	The setup time of cast k on machine h at the last stage $(\forall k \in K, h \in H_l)$
d_i	The due date of charge i at the last stage $(\forall i \in \Omega)$
τ	The maximum waiting time
E_1-E_4	Coefficients of penalty for (cast break / waiting time / earliness / tardiness)
Q	A sufficiently large number

Decision variables (domain)

X_{irj}	1 if charge i precedes charge r on the same machine at stage j, 0 otherwise ($X_{irj} \in \{0, 1\}$ for $\forall i, r \in \Omega$, $i \neq r$, $j \in J_i \cap J_r$)
Y_{ijh}	1 if charge i at stage j is assigned to machine h, 0 otherwise ($Y_{ijh} \in \{0, 1\}$ for $\forall i \in \Omega$, $j \in J_i$, $h \in H_j$)
C_{ij}	The completion time of charge i at stage j ($C_{ij} \geq 0$ for $\forall i \in \Omega$, $j \in J_i$)
u_{ih}	The idle time between two consecutive charges i and i' in a cast on machine h at the last stage ($u_{ih} \geq 0$ for $\forall k \in K, (i, i') \in \hat{\Omega}_k, h \in H_l$)
W_{ij}	The waiting time of charge i between consecutive stages j and j' in its route ($W_{ij} \geq 0$ for $\forall i \in \Omega, (j, j') \in \hat{J}_i$)
α_i	The earliness of charge i ($\alpha_i \geq 0$ for $\forall i \in \Omega$)
β_i	The tardiness of charge i ($\beta_i \geq 0$ for $\forall i \in \Omega$)

The Master MILP

Minimize

$$E_1 \cdot \sum_{k=1}^{m} \sum_{s=1}^{n_k-1} \sum_{h \in H_l} u_{\Omega_k[s],h} + E_2 \cdot \sum_{i=1}^{n} \sum_{w=1}^{l_i-1} W_{i,J_i[w]} + E_3 \cdot \sum_{i=1}^{n} \alpha_i + E_4 \cdot \sum_{i=1}^{n} \beta_i \quad (1)$$

subject to

$$\sum_{h \in H_j} Y_{ijh} = 1 \qquad \forall i \in \Omega, j \in J_i \quad (2)$$

$$X_{irj} + X_{rij} \geq Y_{ijh} + Y_{rjh} - 1 \qquad \forall i, r \in \Omega, i \neq r, j \in J_i \cap J_r, h \in H_j \quad (3)$$

$$X_{irj} + X_{rij} \leq 1 - (Y_{ijh} - Y_{rjh}) \qquad \forall i, r \in \Omega, i \neq r, j \in J_i \cap J_r, h \in H_j \quad (4)$$

$$Y_{ilh} = Y_{rlh} \qquad \forall k \in K, (i, r) \in \hat{\Omega}_k, h \in H_l \quad (5)$$

$$X_{irl} = 1 \qquad \forall k \in K, (i, r) \in \hat{\Omega}_k \quad (6)$$

$$C_{ij} \geq r_{ij} + p_{ijh} \cdot Y_{ijh} \qquad \forall i \in \Omega, j \in J_i, h \in H_j \quad (7)$$

$$C_{rj} - C_{ij} \geq p_{rjh} - Q(2 - Y_{ijh} - Y_{rjh} + X_{rij}) \qquad \forall i, r \in \Omega,\ i \neq r, j \in J_i \cap J_r, h \in H_j \quad (8)$$

$$C_{rl} - C_{il} \geq (p_{rlh} + S_{kl}) - Q(2 - Y_{ilh} - Y_{rlh} + X_{ril}) \qquad \forall k, q \in K,\ k \neq q, h \in H_l, (i, r) = (\Omega_q[n_q], \Omega_k[1]) \quad (9)$$

$$C_{ij'} - (C_{ij} + W_{ij}) \geq (t_{h,h'} + p_{ij'h'}) - Q(2 - Y_{ijh} - Y_{ij'h'}) \qquad \forall i \in \Omega, (j, j') \in \hat{J}_i, h \in H_j, h' \in H_{j'} \quad (10)$$

$$C_{ij'} - (C_{ij} + W_{ij}) \leq (t_{h,h'} + p_{ij'h'}) + Q(2 - Y_{ijh} - Y_{ij'h'}) \qquad \forall i \in \Omega, (j, j') \in \hat{J}_i, h \in H_j, h' \in H_{j'} \quad (11)$$

$$u_{ih} - (C_{rl} - C_{il} - p_{rlh}) \geq -Q(1 - Y_{rlh}) \qquad \forall k \in K, (i, r) \in \hat{\Omega}_k, h \in H_l \quad (12)$$

$$\beta_i - \alpha_i = C_{il} - d_i \qquad \forall i \in \Omega \quad (13)$$

$$W_{ij} \leq \tau \qquad \forall i \in \Omega, j \in J_i \quad (14)$$

The objective function (1) is to minimize the total weighted penalty. Each term denotes the penalty for cast break, the total waiting time, the earliness, and the tardiness, respectively. Constraints (2) ensure that a charge is assigned to exactly one machine at each stage in its route. Constraints (3) and (4) restrict either of two charges precedes the other if and only if they are assigned to the same machine in each stage. Constraints (5) imply that the charges in a cast must be assigned to the same machine at the last stage, and constraints (6) restrict their precedence to conform with the given sequence. Constraints (7) mean that a charge can be processed only after its release time at a stage. Constraints (8) restrict a charge to be processed only after its preceding charges at the same

machine are processed. Constraints (9) mean that the first charge of a cast needs a setup time after completing the last charge of preceding casts. Constraints (10) and (11) imply that the difference of the completion times at two consecutive stages is the sum of waiting time, transportation time, and the processing time at the later stage. Constraints (12) define the idle time between two consecutive charges in a cast at the last stage. Note that the equality holds if the idle time is minimized to be zero. Constraints (13) are from the definition of the earliness and the tardiness of a charge. Constraints (14) provide a limit on the maximum waiting time between two consecutive stages. The domain of each variable is presented together with its definition.

3 Solution methodology

We develop the iterated greedy matheuristic for SCC process scheduling following the framework by [8]. We sequentially add cast after cast to construct an initial schedule considering the earliness and tardiness function of each cast. We propose two heuristics using destruction and construction (DC) of a part of a schedule to improve the objective value. We call them DC-cast and DC-charge according to the destruction method. Construction is done by solving an MILP where most of binary variables remain fixed. After applying DC-cast and DC-charge heuristics, the resulting solution is passed to an MILP solver to improve their objective value until a given time limit is reached.

3.1 Finding an initial schedule

Let MILP(K') be the MILP model in Section 2 with restricted set of casts $K' \subseteq K$ (e.g., MILP(K) denotes the master MILP). We first find the 'desired starting time' of each cast ($:= t_k^*$), which is the starting time of the first charge of cast k in the optimal solution of MILP($\{k\}$) for $k \in K$. We reindex K following the ascending order of t_k^*. An initial solution is generated by sequentially assigning binary variables for all of the charges in cast k in K. Let $\Lambda(k)$ and $\tilde{\Lambda}(k)$ denote the sets of binary variables and their fixed values in MILP($\{1, \ldots, k\}$), respectively. We can find an initial schedule according to the following procedure.

Initial heuristic (:= IH)
1) Solve MILP($\{1\}$) and determine $\tilde{\Lambda}(1)$ from $\Lambda(1)$ part of an optimal solution.
2) For $k = 2$ to m
    Solve MILP($\{1, \ldots, k\}$) subject to $\Lambda(k-1) = \tilde{\Lambda}(k-1)$
    Determine $\tilde{\Lambda}(k)$ from $\Lambda(k)$ part of an optimal solution.

Fig. 1 shows an example with three casts with three charges in each cast. Since an MILP in the loop has a small number of binary variables, an initial schedule can be obtained in a short time.

3.2 Improving a schedule: destruction and construction

Given a feasible schedule, we destruct some charges from the schedule and reinsert them to construct possibly a better schedule, and we call it destruction and

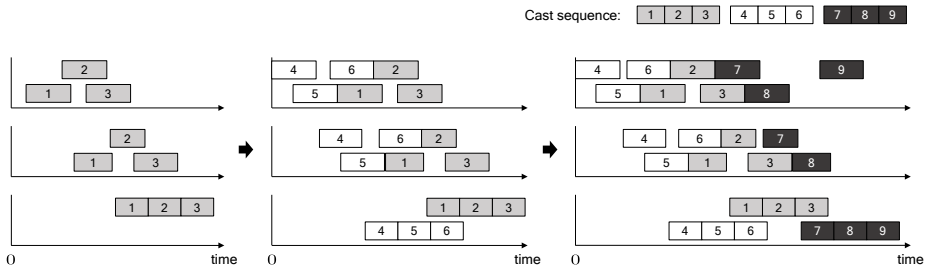

Fig. 1. The initial heuristic

construction (DC) heuristic. In a mathematical model perspective, if all binary variables are fixed, then all continuous variables can be easily (in a very short time) determined by solving the resulting linear programming model. Thus, we assume a given schedule implies that all binary variables have their fixed values. Then, destruction of charges implies that we relax the corresponding binary variables (sequencing variables X and assignment variables Y) and all continuous variables, but keep the remaining binary variables at their values. Since the resulting MILP after destruction has a small number of binary variables, it can be solved in a short time.

DC-cast (:=DA)

Suppose a feasible solution of the MILP model is given. The DC-cast heuristic destructs all of the charges in a cast from the given solution and re-optimize their machine assignments and relative position to the other charges. It iterates for casts in K following the increasing order of starting times of first charges in the given schedule. Fig. 2 shows an example where the cast containing charges 4–6 is selected.

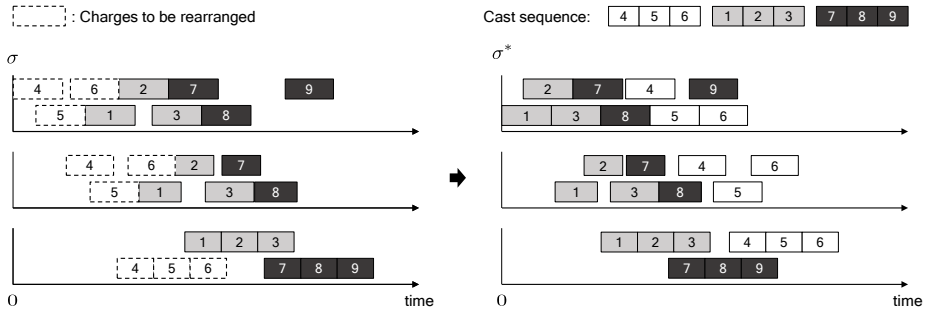

Fig. 2. DC-cast heuristic

DC-charge (:=DH)

The DC-charge heuristic maintains a time window of the given length D for each stage. The algorithm destructs all charges of which the starting times have

an overlap with the time windows, and construct a new schedule by inserting the destructed charges. Then, the time windows are shifted forth by Δ and the procedure is continued until the end of the schedule is reached. Since the starting time of a charge is delayed over stages, we apply the lag of starting times to the windows for two consecutive stages ($:= \delta$). Using a given schedule, δ is computed as the minimum of (i) the average difference of the earliest starting times for the first and the last stages, and (ii) the average difference of latest completion times for the first and the last stages. Fig. 3 shows an example where charges 1, 4, 5, and 6 are selected.

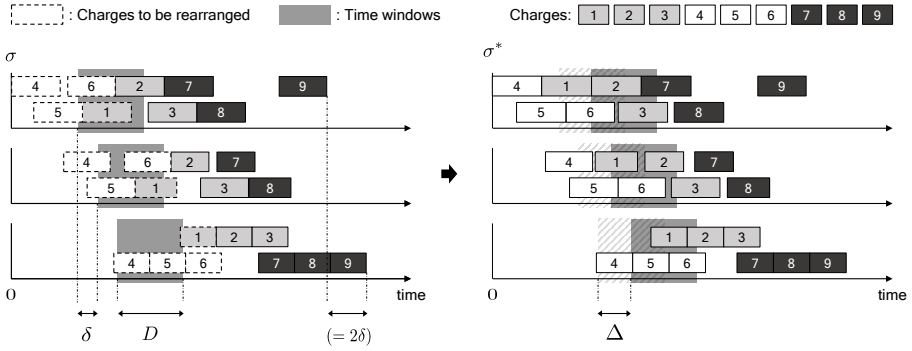

Fig. 3. DC-charge heuristic

3.3 Iterated Greedy Matheuristic

We put the proposed components together in an algorithm which we refer to as the Iterated Greedy Matheuristic (IGM). IGM begins with IH to obtain an initial schedule. Then, we improve the solution by DC heuristic consisting of R^{DA} runs of DA followed by R^{DH} runs of DH. If the objective value is not improved during a run of DA or DH, we stop the successive run and proceed to the next component. The DC heuristic (consisting multiple runs of DA and DH) is repeated R^{DC} times. After obtaining a feasible solution with our heuristics, we feed it into an MILP solver as an incumbent solution to improve it using the master MILP until a given time limit is reached; we call this procedure the MILP Improvement (MI). In short, IGM can be summarized as the following sequence of algorithms:

$$\mathrm{IH} \rightarrow (\mathrm{DA} \times R^{\mathrm{DA}} \text{ times} \rightarrow \mathrm{DH} \times R^{\mathrm{DH}} \text{ times}) \times R^{\mathrm{DC}} \text{ times} \rightarrow \mathrm{MI}.$$

4 Numerical results

To demonstrate the performance of the proposed algorithm, we generated problem instances according to realistic production environment.

– Machine environment: SCC process consists of five stages and twelve machines, where the numbers of machines in SM, RF1, RF2, RF3, and CC stage are 4, 2, 2, 2, and 4, respectively.

- Casts: A cast contains three to nine charges. We create casts of random number of charges until the total number of charges is between 30 and 36. Therefore, each problem instance consists of four to twelve casts.
- Charges: The first and last stages are mandatory for all charges, but the three refining stages in the middle are not; a charge has a probability of two-thirds in skipping each refining stage. A processing time (in minutes) of a charge in each machine is generated from uniform distribution; $U(45, 55)$ for the SM stage, $U(30, 40)$ for the RF1, RF2, and RF3 stages, and $U(35, 45)$ for the CC stage.
- Restrictions: Transportation time between all machines ($t_{h,h'}$) are set to be 10 minutes, and the maximum waiting time (τ) is 30 minutes. The setup times of casts on all machines at the last stage (S_{kh}) are set to be 30 minutes.
- Objective coefficients: Penalty for earliness and tardiness is 1, and penalty for the waiting time is 1.5. Since minimizing the cast break has the top priority among objectives, we set cast break penalty coefficient as 100,000.

We set $(D, \Delta, R^{\mathrm{DC}}, R^{\mathrm{DA}}, R^{\mathrm{DH}}) = (90, 90, 2, 3, 1)$ as the control parameters for IGM. We provide the time limit of 60 seconds to each run of DA and DH. The total time limit for IGM is set as 600 seconds.

Since our problem is more general than problems discussed in the literature, we compared the proposed solution method with an algorithm based on NSGA-II by [2]. We use a pair of lists as a chromosome representation. The first list represents sequence of casts in each CC machines, and the second represents charges ordered by the completion time in the CC stage. We follow the framework by [2] and genetic operations (crossover and mutation) by [5]. Note that, although it takes negligible time for genetic operations, simple dispatching rules often generate infeasible solutions due to hard constraints. Thus, we also devised a decoding method of solving a linear programming problem to get a feasible solution. We consider another competitor which solves the master MILP model in Section 2 (:=MILP). The total time limit for MILP and NSGA-II is set as 1200 seconds, and the population size of NSGA-II is set as 200. All algorithms are implemented by Gurobi version 9.0.3 using Python 3.8.

In order to evaluate the performance of an algorithm, we define the optimality gap as $(Z - LB)/LB$ where Z is the objective function value found by the algorithm and LB is the best known lower bound from IGM and MILP. The result of the experiment on 30 randomly generated problem instances is summarized in Table 1. All best solutions by both methods have no cast break. The average optimality gap is 5.07% for IGM, 25.05% for MILP, and 9.93% for NSGA-II. Although only a half amount of time is given, IGM shows better and stable performance; it obtained the best solutions in each of all problem instances.

In order to see the performance of each component of IGM, we calculated the average optimality gap of the solution for each of the six components of IGM for 30 practical size instances. The average time for each component is calculated as well. We summarise the average values in Table 2 and visualise them in Figure 4. As expected, the improvement percentage decreases over time. From the average

	Objective			LB			Optimality Gap		
No.	IGM	MILP	NSGA-II	IGM	MILP	Best	IGM	MILP	NSGA-II
0	4456.0	4591.0	4543.0	4375.0	4375.0	4375.0	1.85%	4.94%	3.84%
1	5350.0	5954.5	5617.0	5209.0	5218.0	5218.0	2.53%	14.11%	7.65%
2	5227.5	6192.5	5653.0	5080.0	5080.0	5080.0	2.90%	21.90%	11.28%
3	4064.5	4583.0	4301.0	3702.0	3707.0	3707.0	9.64%	23.63%	16.02%
4	3929.5	4387.5	4133.0	3865.0	3851.0	3865.0	1.67%	13.52%	6.93%
5	6395.5	6641.0	6527.0	6091.0	5895.2	6091.0	5.00%	9.03%	7.16%
6	4648.5	5466.0	5078.0	4462.0	4433.0	4462.0	4.18%	22.50%	13.81%
7	5671.0	7750.0	5985.5	5321.0	5333.0	5333.0	6.34%	45.32%	12.24%
8	4703.5	5760.0	4995.5	4436.0	4429.0	4436.0	6.03%	29.85%	12.61%
9	5647.0	7572.0	5850.0	5364.0	5088.0	5364.0	5.28%	41.16%	9.06%
10	6500.5	10553.0	7134.0	5955.0	5778.7	5955.0	9.16%	77.21%	19.80%
11	5669.0	6673.5	6075.5	5349.0	5349.0	5349.0	5.98%	24.76%	13.58%
12	5616.5	6830.0	5868.0	5070.0	4863.4	5070.0	10.78%	34.71%	15.74%
13	5493.5	6597.5	5653.0	5363.0	5353.9	5363.0	2.43%	23.02%	5.41%
14	4472.0	5265.5	4719.5	4329.0	4334.0	4334.0	3.18%	21.49%	8.89%
15	4457.5	4895.0	4833.0	4304.0	4305.0	4305.0	3.54%	13.70%	12.26%
16	4007.5	4496.0	4208.5	3758.0	3765.0	3765.0	6.44%	19.42%	11.78%
17	5597.5	5615.0	5627.0	5509.0	5518.0	5518.0	1.44%	1.76%	1.98%
18	4766.5	6722.0	4946.0	4638.0	4649.0	4649.0	2.53%	44.59%	6.39%
19	4634.0	4965.5	4661.0	4536.0	4536.0	4536.0	2.16%	9.47%	2.76%
20	4668.0	5125.0	4813.0	4466.0	4064.2	4466.0	4.52%	14.76%	7.77%
21	5722.0	6609.5	6132.0	5554.0	5554.0	5554.0	3.02%	19.00%	10.41%
22	4573.5	4976.0	4611.0	4059.0	3929.2	4059.0	12.68%	22.59%	13.60%
23	4406.0	4866.5	4432.0	3799.0	3802.0	3802.0	15.89%	28.00%	16.57%
24	5812.0	8509.0	6230.5	5503.0	5511.0	5511.0	5.46%	54.40%	13.06%
25	4621.0	5143.0	4773.0	4528.0	4534.0	4534.0	1.92%	13.43%	5.27%
26	4577.0	4841.5	4651.0	4513.0	4513.0	4513.0	1.42%	7.28%	3.06%
27	4960.5	6591.5	5230.5	4873.0	4797.1	4873.0	1.80%	35.27%	7.34%
28	4914.0	6310.5	5126.0	4539.0	4502.3	4539.0	8.26%	39.03%	12.93%
29	5462.0	6389.0	5707.0	5254.0	5219.3	5254.0	3.96%	21.60%	8.62%
Average							5.07%	25.05%	9.93%

Table 1. Comparison of three algorithms

performance, the user may choose the total running time of the algorithm. Even when only a short computation time is given, IGM yields better solutions than other algorithms; on average, the initial solution of IGM is already better than the final solution of NSGA-II. Moreover, practitioners have the advantage of spending more time on computing since the difference of 0.1% matters in the cost-intensive steel industry.

5 Conclusion

In this paper, we proposed a generic MILP that can handle various features in the SCC scheduling. We also proposed the iterated greedy matheuristic that can find a feasible solution satisfying hard constraints concerning the maximum

Comp.	Gap (%)	Improv. (%p)	Elapsed time (s)	Comp. time (s)
IH	9.15	-	28.2	28.2
DA	6.94	2.20	75.3	47.0
DH	5.62	1.32	175.7	100.4
DA	5.51	0.11	197.8	22.1
DH	5.25	0.26	277.6	79.8
MI	5.07	0.18	600.5	323.0

Table 2. IGM average performance

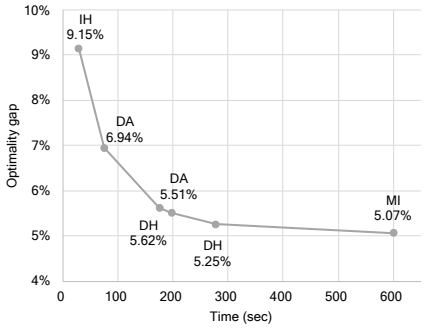

Fig. 4. IGM average performance

waiting time and continuous casting. By the numerical results, we showed that our algorithm is able to obtain better solutions than an NSGA-II algorithm with most test instances in less running time. Since our model can encompass most of the essential features found in the literature, it has much potential to be applied for difficult scheduling problems in practice such as rescheduling and controllable processing time.

References

1. Cui, H., Luo, X., Wang, Y.: Scheduling of steelmaking-continuous casting process using deflected surrogate Lagrangian relaxation approach and DC algorithm. Computers & Industrial Engineering **140**, 106271 (Feb 2020)
2. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (Apr 2002)
3. Dutta, S.K., Chokshi, Y.B.: Secondary steelmaking. In: Dutta, S.K., Chokshi, Y.B. (eds.) Basic Concepts of Iron and Steel Making, chap. 17, pp. 497–536. Springer Nature (2020)
4. García-Menéndez, D., Morán-Palacios, H., Ortega-Fernández, F., Díaz-Piloñeta, M.: Scheduling in continuous steelmaking casting: A systematic review. ISIJ International **60**(6), 1097–1107 (2020)
5. Long, J., Zheng, Z., Gao, X., Pardalos, P.M.: A hybrid multi-objective evolutionary algorithm based on NSGA-II for practical scheduling with release times in steel plants. Journal of the Operational Research Society **67**(9), 1184–1199 (Sep 2016)
6. Missbauer, H., Hauber, W., Stadler, W.: A scheduling system for the steelmaking-continuous casting process. A case study from the steel-making industry. International Journal of Production Research **47**(15), 4147–4172 (Aug 2009)
7. Pan, Q.k.: An effective co-evolutionary artificial bee colony algorithm for steelmaking-continuous casting scheduling. European Journal of Operational Research **250**(3), 702–714 (May 2016)
8. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research **177**(3), 2033–2049 (Mar 2007)