A two-stage local search heuristic for solving the steelmaking continuous casting scheduling problem with dual shared-resource and blocking constraints

Pieter De Moerloose*[1] and Broos Maenhout[1]

[1]Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2, 9000 Gent (Belgium), Pieter.DeMoerloose@gmail.com,Broos.Maenhout@UGent.be

October, 2022

Abstract

The steelmaking continuous casting scheduling problem concerns the construction of production schedules for steelmaking from the basic oxygen furnace to the continuous casters. Start and completion times are determined to complete the operations associated with different steel charges. The real-life problem under study comprises different complicating dual-resource transportation and blocking constraints and embeds routing and sequencing flexibility to process charges of different steel grades. In this paper, we present a mixed-integer linear problem and propose a two-stage local search heuristic adjusting the charge sequence and machine assignments to minimise makespan and waiting times of charges. The algorithm makes use of a novel charge-sequence initialisation method and local search operator to find promising neighbouring schedules. Computational experimentation is performed on real-life instances to validate heuristic design choices and benchmark the proposed method to alternative methodologies. Results demonstrate that the proposed heuristic provides better initial solutions and a faster convergence to high-quality solutions in short run times.

Keywords: Scheduling; Two-Stage Local Search; Flexible Job Shop; Steelmaking Continuous Casting

1 Introduction

The Steelmaking Continuous Casting (SCC) Scheduling Problem plays an important role in the steel industry. It considers at what time, in which sequence, and on which machine charges, i.e. furnace loads of steel, should be processed in the different stages of the steelmaking production process (Atighehchian et al., 2009). The unique feature, discerning this problem from other scheduling problems, is that in the last stage, the casting stage, charges have to be processed in continuous casts, which is a batch of different charges that is operated without waiting times between consecutive charges. The order of charges within a cast and the machine on which the casts are processed are decided upon in advance. An effective schedule can reduce energy consumption and increase productivity. In this paper, we study a variant of the SCC-scheduling problem, encountered in real-life. This real-life problem is more complex compared to problems from literature because of the following features. First, different routing plans for different steel grades are allowed, which turns the underlying problem into a flexible job-shop problem. This is necessary to accommodate for

*Corresponding author - Email: Pieter.DeMoerloose@gmail.com

the large number of steel grades modern steel plants produce. Second, more realistic transport constraints are explicitly implemented by formulating dual-resource constraints. The production floor in steel plants is typically organised in production lines. Transportation of charges between production lines is conducted using cranes, whose availability is limited and are explicitly considered a scarce resource to compose a suitable schedule. Furthermore, steel wagons are used to transport charges between machines lying on the same production lines. These steel wagons are occupied by single charges during the processing of necessary operations, transportation between machines, and waiting times. The scheduling objectives are relative to efficiency and product quality, i.e. we minimise makespan and sum of waiting times. The latter ensures the steel temperature does not drop significantly, which would deteriorate the quality of steel.
We tackle this complex optimisation problem by relying on a two-stage local search heuristic. This hill-climbing heuristic relies on a dual-schedule encoding, composed of two lists representing the sequence of processed charges and machine assignments for the operations of these charges. Schedule encodings are decoded using a recursive schedule generation scheme, which greedily determines start and completion times per charge, and a repair mechanism to ensure the feasibility of the constructed schedule. The schedule generation scheme minimises the makespan or completion time for processing all charges and introduces waiting times between operations to satisfy precedence relations between consecutive operations. These waiting times are improved via the schedule repair mechanism, which relies on mathematical linear programming, considering only a limited subset of the decision variables. In order to find a high-quality schedule in a limited run time, we propose a dedicated initialisation heuristic and two local search neighbourhoods, which improve the initial schedule in two stages. The proposed methodology was tested using real-life data from ArcelorMittal (Ghent, Belgium), one of the largest steelmaking companies in Europe. In the computational experiments, we validate heuristic design choices and benchmark the algorithm to alternative procedures from the literature.

The remainder of the manuscript is organised as follows. An overview of the relevant literature is given in Section 2. A detailed description of the real-life problem and associated mixed-integer linear programming (MILP) formulation are provided in Section 3. Although this model can theoretically be solved using standard software packages, solution approaches based on mathematical programming are not satisfactory and not practical for solving real-life use cases. To this end, we propose an efficient heuristic in Section 4. Section 5 presents computational experiments to demonstrate the heuristic produces high-quality solutions in short run times. Conclusions and directions for future research are given in Section 6.

2 Literature review

As the scheduling problem under study is inherently a flexible job-shop scheduling problem, we discuss in Section 2.1 relevant literature on this scheduling problem and an overview of proposed solution approaches. In addition, we review the studied characteristics and proposed solution methods related to the SCC-scheduling problem in Section 2.2. A conclusion based on the literature is drawn in Section 2.3.

2.1 Flexible Job-Shop Problem

Previous works on the SCC-scheduling problem have modelled the problem as a hybrid flow shop, whereas the problem under study embeds routing and sequencing flexibility, modelling the problem as a flexible job-shop problem (FJSP). The FJSP is an extension of the classic job-shop scheduling problem, in which multiple jobs are processed on several machines.

Related operations are assigned to specific machines and need to be sequenced (Demir and İşleyen, 2013). Because there are many significant real-life applications, the FSJP has been extensively studied in different forms and guises. For the properties of the general FJSP, we refer the reader to the survey papers of Chaudhry and Khan (2016), Xie et al. (2019) and Xiong et al. (2022).

The FSJP has been researched with consideration of different types of objectives. These are categorised by Xiong et al. (2022) into time-based, job-number-based, cost-based, revenue-based and energy and pro-environment-based measures. Most popular have been the time-based objectives (e.g. minimising makespan, sum completion times or job waiting times, (weighted) tardiness, resource idle times) together with minimisation of processing cost (e.g. Zheng and Wang, 2019) and total energy consumption (e.g Zhang et al., 2017). Regarding the schedule requirements, different studies considered some discerning features in addition to the general properties of the FJSP, such as blocking (e.g. Gröflin et al., 2011; Mati and Lahlou C., 2011), no-wait constraints (e.g. Aschauer et al., 2017), controllable processing times (e.g. Lu et al., 2017), dual resource constraints (e.g. Andrade-Pineda et al., 2019), which are relevant attributes related to the problem under study. The blocking of a machine implies that a machine is blocked by a waiting job after the completion of an operation, i.e. a job has to wait on the current machine until a suitable machine for processing the next operation of the job is available and in the meanwhile, no other job can be processed (Gröflin et al., 2011). The research of Hansmann et al. (2014) with application in rail car maintenance extended this concept and is the only other study that considered the blocking of so-called work centres or production lines, i.e. any busy machine in a production line blocks access to all machines of the line. However, in contrast to the problem under study, they do not allow any switching of jobs between production lines. No-wait constraints, which imply that two consecutive operations of a job must be processed without interruption, have been imposed in literature for a variety of reasons, e.g. absence of space for products to wait, reduction of the work-in-process (Allahverdi, 2016). Aschauer et al. (2017, 2020) present the only other studies, apart from this study, that consider not only no-wait constraints but also controllable processing times to prevent the deterioration of product quality caused by interruptions between consecutive operations relative to a job. They assume that the processing times of certain production operations can be adjusted within given bounds without additional costs. In recent years, flexible job-shop problems with dual-resource constraints have gained more attention (cf. the survey of Dhiflaoui et al., 2018), assuming that tasks can only be processed on machines with the required assistance of other resources, which are typically workers. In this context, Agnetis et al. (2014) and Andrade-Pineda et al. (2019) discuss interesting applications related to the problem under study, for which the workers are shared between multiple machines.

Multiple methods have been proposed to solve the problem, such as mathematical programming, dedicated branch-and-bound methods, construction heuristics and meta-heuristics (e.g. Tabu Search, Genetic Algorithm, hybrid techniques). For an exhaustive overview, we refer to the survey papers of Chaudhry and Khan (2016) and Xie et al. (2019). In this review, we focus on Mixed-Integer Linear Programming (MILP) and local search heuristics. These methods typically split the problem into the sequencing of operations and the assignment of operations to machines, which is revealed in the definition of integer variables in the MILP model and the stage-wise design of heuristic methods defining different types of operators to improve schedules.
Formulating and solving the problem via MILP provides an exact approach, albeit slow for larger problem sizes and, hence, not useful for practical purposes. Demir and İşleyen (2013) provide a review of different formulations to solve the FSJP. These mainly differ in the choice

of integer variables for the sequencing of operations, which are either sequence-position variables, precedence variables, or time-indexed variables. Sequence-position variables assume each machine has a number of positions to which jobs can be assigned. Precedence variables are based on the sequence of operations on machines and indicate if an operation is performed before or after another operation on that machine. Time-indexed variables assume a discrete time horizon and indicate whether an operation is started processing on a machine during a particular period. Comparison in computational performance demonstrates the superiority of the use of precedence-based variables instead of considering all possible starting times for every operation Özgüven et al. (2010); Demir and İşleyen (2013); Sierra et al. (2015). The latter observation has not only been verified for the classical FSJP, but also for extended models (e.g. for the dual-resource constrained FSJP (Andrade-Pineda et al., 2019)).
The study by Shi et al. (2018) states that hybrid and multi-stage search structures that apply different heuristic operators in an independent manner, are the most suited heuristic strategies for solving the FJSP. Accordingly, several studies (e.g. Saidi-Mehrabad and Fattahi, 2007; Zhang et al., 2011; Ishigaki and Takaki, 2017; Rooyani and Defersha, 2019) manipulate in an iterative manner the operation sequencing and the machine assignments in separate stages. For example, Fattahi et al. (2007) propose a two-stage optimisation approach for solving the FSJP following the meta-heuristic framework of Tabu Search and Simulated Annealing and a combination of these two methods. Following the structure of the FJSP, the machine assignments are fixed in a first stage, and random exchanges of job positions in the charge sequence are performed. In the second stage, the charge sequence is fixed and the machine assignment of operations is changed. In their experiments, they demonstrate that a two-stage method outperforms a single-stage method that considers changes to machine assignments and charge sequences simultaneously. Zhang et al. (2011) propose a Genetic Algorithm that applies different operators to change the operation sequence and machine assignments individually, relying on a two-vector chromosome notation. Chromosomes are decoded into feasible and active schedules using a dedicated schedule generation scheme. Rooyani and Defersha (2019) present also a genetic algorithm to determine first the order in which operations will be assigned and determine the machine assignment afterwards using a greedy approach that chooses for each operation a machine with the shortest completion time based on machine load and processing time. Aschauer et al. (2017, 2020) solve the no-wait job-shop scheduling problem with controllable processing times by decomposing the problem. They consider first solving the sequencing subproblem followed by the timetabling subproblem. The controllable processing times are handled in the second stage by applying a recursive decoding algorithm to transform the job sequence into a non-delay schedule.

2.2 Steelmaking Continuous Casting Scheduling Problem

A literature review of the SCC-scheduling problem and other relevant scheduling problems within the steel industry is given by Tang et al. (2001). Their review focuses on the description of specific problem types encompassing steel-production methods and proposed solution methodologies to solve related scheduling problems. Since then, a wide range of methods has been applied to different variations of the problem (Long et al., 2018). Table 1 provides a literature synthesis of considered problem characteristics and proposed solution methodologies. All previous studies consider the SCC-scheduling problem as a hybrid flow-shop scheduling problem as they consider no sequencing flexibility, apart from the study of Long et al. (2018) that considers the skipping of particular operations. The only type of scheduling flexibility that has been considered regularly in literature is the adjustment of processing times, most frequently involving operations on the continuous caster (CC) to enable that charges relative to a cast are processed in a continuous manner. Imposed resource

constraints regard typically only the required availability of machines to process charges, whereas other types of resource constraints have not been considered. Hence, problems in literature do not consider any additional resources that may be shared between different machines or production lines. In this perspective, the scheduling of operations on transportation resources is typically not modelled as only the needed transportation times are accounted for to accurately calculate completion and waiting times. Furthermore, most authors consider multiple objectives, depending on the problem definition and implementation of imposed hard or soft constraints. For example, some authors see a cast break as an objective that should be minimised, whereas others consider this to be a hard constraint. Predetermined cast sequences may comprise either the strict planning of start times or the minimisation of earliness and tardiness of completion times relative to previous casts. In general, objective function components for the SCC-scheduling problem are classified into the following categories, i.e. (i) casting interruption cost; (ii) waiting time or steel temperature drop cost; (iii) total completion time; (iv) earliness and/or tardiness; and (v) other objectives such as the minimisation of transportation costs (Atighehchian et al., 2009), poor quality costs (Atighehchian et al., 2009), average sojourn time (Sun and Yu, 2015; Peng et al., 2018) and adjustments of standard processing times (Long et al., 2018).

Table 1: Overview of problem definition and solution methodology of relevant research

Manuscript	**Problem definition**														**Solution method**	
	PT		T		SR		SF			OBJ						
	FPT	APT	TT	TR	No	Yes	NSF	SK	FSF	CIC	WT	TC	ET	Ot	Exact	Heur
Atighehchian et al. (2009)	✓		✓		✓		✓			✓	✓			✓		✓
Bellabdaoui and Teghem (2006)		✓	✓		✓		✓					✓			✓	
Mao et al. (2014)	✓		✓		✓		✓				✓		✓		✓	
Sun and Yu (2015)		✓	✓		✓		✓					✓		✓		✓	
Pan (2016)	✓		✓		✓		✓				✓	✓				✓
Peng et al. (2018)		✓	✓		✓		✓			✓			✓	✓		✓
Tang et al. (2000)	✓		✓		✓		✓			✓	✓		✓		✓	
Tang et al. (2002)	✓		✓		✓		✓			✓	✓		✓		✓	
Xuan and Tang (2007)	✓		✓		✓		✓					✓			✓	
Long et al. (2018)		✓	✓		✓			✓			✓	✓		✓		✓
Xu et al. (2020)	✓		✓		✓		✓				✓		✓		✓	
This paper		✓	✓	✓		✓			✓		✓	✓				✓

Legend

PT	Processing times	**FPT**	Fixed processing times	**APT**	Adjustable processing times
T	Transportation	**TT**	Transportation times	**TR**	Transportation resources
SR	Shared resources	**SF**	Sequencing flexibility		
NSF	No sequencing flexibility	**SK**	Stage skipping	**FSF**	Full sequencing flexibility
OBJ	Objectives	**CIC**	Casting interruption costs	**WT**	Waiting time/temperature drop cost
TC	Total completion time	**ET**	Earliness and/or tardiness	**Ot**	Other objectives
Exact	Exact method	**Heur**	Heuristic method		

A wide range of methods has been proposed in literature for solving the problem. These can be roughly categorised into two categories, i.e. exact and heuristic methods. *Exact methods* typically rely on mathematical programming and Lagrangian relaxation to obtain suitable schedules. In this regard, Tang et al. (2000) and Bellabdaoui and Teghem (2006) propose a MILP model and solve these models using commercial software packages. Tang et al. (2002), Xuan and Tang (2007) and Sun and Yu (2015) apply Lagrangian relaxation and dynamic programming, potentially with some heuristics appended. Mao et al. (2014) present a Lagrangian relaxation approach with an improved subgradient algorithm to improve run times. Apart from exact methods, *(meta-)heuristics* are widely used to solve the

SCC-scheduling problem. The proposed methods concern mainly Genetic Algorithms and Swarm Intelligence. Atighehchian et al. (2009) use a hybrid algorithm relying on Ant Colony Optimisation for machine assignment and sequencing combined with a non-linear optimisation step to determine the continuous variables. Pan et al. (2012) propose an Artificial Bee Colony Optimisation heuristic, which is improved by Pan (2016) relying on two so-called swarms to determine the cast sequences on the one hand and schedule the charges on the other hand. The Artificial Bee Colony Optimisation heuristic is further improved by Peng et al. (2018) via an improved encoding and decoding strategy. Long et al. (2018) use an elitist Genetic Algorithm with a post-processing step to improve the yielded schedules via mathematical programming. Xu et al. (2020) also propose a Genetic Algorithm in combination with a local search method to intensify the search in promising regions.

2.3 Conclusion based on literature

The studies presented thus far in the literature relative to the steelmaking continuous casting problem have not considered full routing and sequencing flexibility of jobs, the presence of resources shared between different machines or operations on scarce transport resources. The basic structure of the encountered real-life problem comprises an FSJP. Unlike the classical FSJP, the problem is characterised by some discerning features, such as the blocking of production lines, dual resource constraints, no-wait constraints, and controllable processing times as some special requirements relative to the steel production process are needed to be met. Although some of these features have been well studied in the literature, no study is present that has considered these characteristics together in a single problem definition and proposed an adequate solution methodology.

3 Problem description and formulation

3.1 Problem context at ArcelorMittal

This section presents a general overview of the steelmaking process relative to the problem under study. The relevant stages are the converter stage, the refining stage, and the continuous casting stage and its main components are illustrated in Figure 1, which presents an abstraction of the real-life process at ArcelorMittal. In these stages, different operations are performed on charges, which are the basic job units in production scheduling within steelmaking. A charge is a furnace load of steel.

The relevant process starts at the *converter stage* that takes charges of pig iron, which is impure liquid iron, from the Basic Oxygen Furnace, removes impurities, and refines it to liquid steel (Verspurten and Henrion, 2019). In the converter stage, an oxygen source is needed to perform some of the operations. At ArcelorMittal, the oxygen resource is shared between the two converter machines. After the converter stage, charges are transported to the *refining stage*, which is recognised to form the bottleneck within the process. Transport is done using transport cranes, which are limited in number and perceived as scarce resources. Hence, in contrast to the literature, we explicitly consider the functioning of these transport cranes, for which importance is stressed by Gao and Pan (2016). In the refining zone, the steel is further refined to the required chemical composition and temperature (Long et al., 2018). In this stage, a charge undergoes a number of different processes depending on the steel grade, which refers to the chemical and physical properties of the steel. These operations do not always follow the same order and are performed on different machines. Consequently, the scheduling of operations allows for some routing and sequencing flexibility. Routing flexibility refers to the choice to schedule operations on different machines, while sequencing

flexibility encompasses the possibility to schedule the operations to process a charge to some extent in a different order. Machines in the refining zone are organised in production lines, which may have different numbers and types of machines. Each of these production lines share one steel wagon to transport the charges between machines of the production line, for which its functioning imposes some constraints on the problem under study. These steel wagons are occupied by a charge during related transports, waiting times and operations on machines. The occupation of a steel wagon blocks other machines on the associated production line from being used for other charges. Transport of charges between production lines is done using the transport cranes.

After the refining stage, the charge is transferred to the *casting stage* using a crane. In this stage, the steel is cast into slabs. This happens on the continuous caster (CC) by means of, as the name indicates, a continuous process after undergoing a cooling process in revolving towers. Multiple charges of similar steel grade, i.e. charges from the same cast, are treated in direct sequence without waiting times in between the charges (Tang et al., 2000). The casting of batches of charges in the continuous casting stage is the discerning feature of SCC scheduling. A batch of charges is referred to as a cast, which is the basic unit in the continuous casting production stage (Pan et al., 2012). The charges within a cast (number and type) are predetermined at ArcelorMittal Ghent and given input to the problem under study. Between two casts, a setup time is needed to prepare the CC. In order to ensure subsequent charges within a cast are processed in a consecutive manner without waiting times, the processing times of the CC can be adjusted to complete a charge. More precisely, casting speeds of charges on the CC can be reduced by 10% at ArcelorMittal in order to ensure direct precedence between charges of a cast. The objectives postulated by ArcelorMittal are relative to minimisation of makespan or duration to process all charges and, especially, waiting times of charges between successive operations. The latter are considered to be more important in steelmaking production (Long et al., 2018).

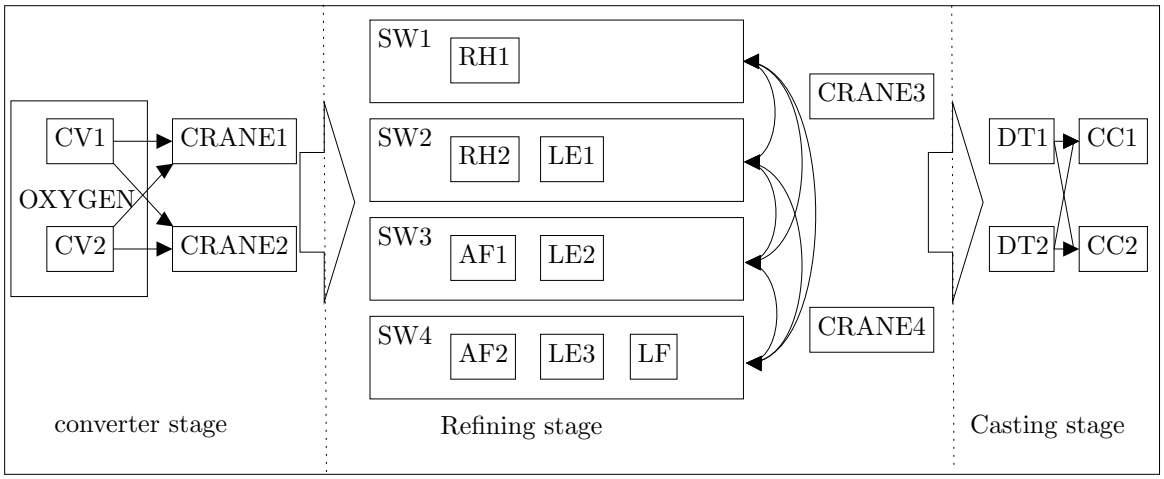

Figure 1: Abstract overview of the steelmaking production process

(**Legend.** CV1-2: converter 1-2; SW1-4: steelwagon 1-4; RH1-2: Ruhrstahl Heraeus degassers 1-2; LE1-3: mettalurgy 1-3; AF1-2: deslagger 1-2; LF: ladle furnace; CRANE1-4: cranes 1-4; DT1-2: revolving tower 1-2; CC1-2: continuous caster 1-2.)

3.2 Steelmaking Continuous Casting Scheduling Problem

The Steelmaking Continuous Casting (SCC) production scheduling problem determines the sequence, time and specific production machines to process the molten steel at various pro-

duction stages from steelmaking to continuous casting. The problem under study focuses on sequencing and scheduling of charges, as the sequence of casts on continuous casting machines is predetermined by a master schedule at ArcelorMittal and is thus input to the problem under study. The basic structure of the scheduling problem is similar to a flexible job shop, for which a job corresponds to a charge, complicated by additional constraints and special requirements relative to the steel production process at ArcelorMittal, described in Section 3.1.

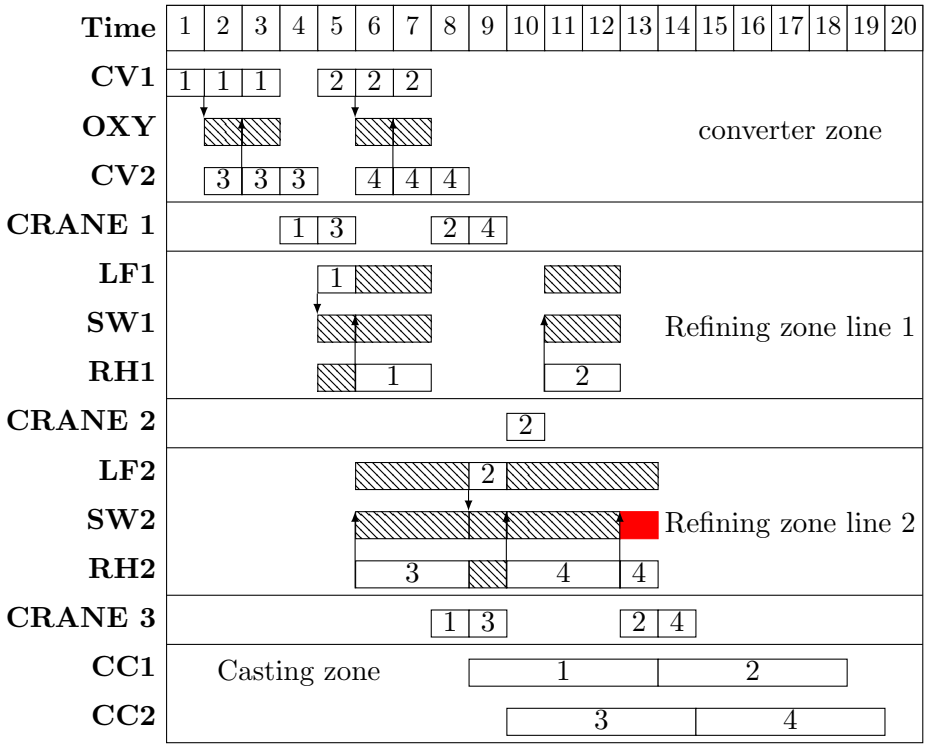

Figure 3: Example schedule from the converter stage to the casting stage

An example schedule is shown in Figure 3 to further illustrate the specific problem properties, based on the production process illustrated in Figure 1. This is a simplified example with a smaller number of machines and operations per charge. We consider four charges (1,2,3,4), which belong to two casts, including charges 1 and 2 and charges 3 and 4, respectively. The casts first enter the converter stage. The first cast is planned on converter 1 (CV1) and the second on converter 2 (CV2). Charge 1 is processed on moments 1 to 3, whereas charge 3 is processed on moments 2 to 4. In the converter stage, an oxygen resource is shared for only some of the operations. Each charge undergoes three operations on a convector. The middle operation always utilises the oxygen resource. Thus, the middle operation of two charges cannot be completed at the same time. Occupation of the oxygen resource (OXY) is shown by the arced bars, whereas the arrows indicate the job that occupies the shared resource. For charges 1 and 3, these operations occur on moments 2 and 3, respectively. Operations of transport cranes are explicitly indicated in Figure 3, as these are considered scarce resources in the problem under study. Crane 1 is used to transport the charges from the converter to the refining zone. The depicted schedule shows that charge 1 (3) is transported to the refining zone on time point 4 (5). Charge 2 and 4 are processed starting from moments 5 and 6 on CV 1 and CV 2, respectively. Two production lines in the refining zone are displayed, which each have a steel wagon (respectively SW1 and SW2), a ladle furnace (LF) and a Ruhrstahl Heraeus degasser (RH). It is shown using the arced bars that operations on

one of these machines also block the other machine on the production line from being used. The latter is due to the use of the associated steel wagon of the production line, which is occupied by a specific charge. In this regard, charge 1 is processed by LF1 on time point 5 and by RH1 on time points 6 and 7. During this period, from time point 5 to 7, no other charge can be processed on this first production line. For example, at time point 5, when charge 1 is processed on LF1, no charge can be processed on RH1. Charge 2 is subsequently processed at time point 9 on LF2 of production line 2, transported by crane 2 to production line 1 on time point 10 and processed on RH1 during time points 11 and 12. The difference in routing plans between steel grades is illustrated by the fact that charges 3 and 4 do not have an operation on the ladle furnace in the refining stage. The fact that a charge also occupies a steel wagon during the waiting time on a production line is shown by the red bar, where charge 4 is waiting on crane 3 for one time unit to be transported to the continuous casting stage. As a result, production line 2 is occupied for an additional time slot (time point 13). In the casting stage, charges 1 and 2, and charges 3 and 4 must follow each other directly due to the required continuity of casting.

3.3 Problem formulation

This section presents the mathematical formulation of the problem under study, for which following notation is used:

Sets

J	Set of charges ($j \in J$)
I	Set of operations ($i \in I$)
I_j	Set of ordered pairs of operations (i, i') with direct precedence relations for charge j, indicating operation i' directly follows operation i
I_k	Set of operations that can be executed on machine k
I_c	Set of ordered pairs of operations (i, i') with direct precedence relations in cast c, indicating casting operation i' directly follows casting operation i
T	Set of transport operations in the refining zone
R	Set of refining operations
O	Set of operations that need the oxygen resource in the converter stage
A	Set of casting operations
D	Set of revolving tower operations
M	Set of machines ($k \in M$, with $k = a$ denoting the artificial machine (cf. infra))
M^{CC}	Set of continuous casting machines
L	Set of production lines ($l \in L$)
M_i	Set of machines on which operation i can be performed
C_k	Set of casts on continuous casting machine k ($c \in C_k$)
B_j	Set of ordered 3-tuples of operations (i, i', i'') with direct precedence relations relative to charge j (with $i, i'' \in R$, $i' \in T$)
D_k	Set of pairs of operations on machine k in which the first operation is the last operation of a cast, and the second operation the first of the subsequent cast
Q_j	Set of ordered pairs of operations (i, i') with direct precedence relations relative to charge j (with $i \in T$, $i' \in R$)

Parameters

p^{min}	Minimum processing time for casting operation
p^{max}	Maximum processing time for casting operation
p_{ik}	Processing time of operation i on machine k
s	Setup time between two consecutive casts
l_k	Production line of machine k
$\mathcal{M}$	Large positive number ($>> 0$)
δ_1	Objective weight coefficient for makespan objective
δ_2	Objective weight coefficient for total waiting time objective

Variables

S_i	Start time of operation i

C_i	Completion time of operation i
C^{max}	Makespan or completion time of all charges
P_i	Processing time of continuous casting operation i
W_i	Waiting time of operation i
W_j	Waiting time of charge j
W	Total waiting time
X_{ik}	1, if operation i is performed on machine k; 0, otherwise
$Y_{ii'}$	1, if operation i is (in-)directly performed before operation i'; 0, otherwise
Z_{li}	1, if steel wagon of production line l is used for operation i; 0, otherwise

Mathematical Formulation

$$\text{Min} \quad \delta_1 \times C^{max} + \delta_2 \times W \tag{1}$$

$$s.t. \sum_{k \epsilon M_i} X_{ik} = 1 \qquad \forall i \in I \tag{2}$$

$$C_i = S_i + \sum_{k \epsilon M_i} p_{ik} X_{ik} + W_i \qquad \forall i \in I \setminus (A \cup D) \tag{3}$$

$$C_i = S_{i'} \qquad i \notin D; \forall (i,i') \in I_j; \forall j \in J \tag{4}$$

$$C_i = S_i + \sum_{k \epsilon M_i} p_{ik} X_{ik} \qquad \forall i \in D \tag{5}$$

$$C_i + W_i = S_{i'} \qquad i \in D; \forall (i,i') \in I_j; \forall j \in J \tag{6}$$

$$Y_{ii'} + Y_{i'i} \geq X_{ik} + X_{i'k} - 1 \qquad \forall i \neq i' \in I_k; \forall k \in M \setminus \{a\} \tag{7}$$

$$C_i \leq S_{i'} + \mathcal{M} * (1 - Y_{ii'}) \qquad \forall i \neq i' \in I \tag{8}$$

$$W_j = \sum_{i \in I_j} W_i \qquad \forall j \in J \tag{9}$$

$$W = \sum_{j \epsilon J} W_j \tag{10}$$

$$C^{max} \geq C_i \qquad \forall i \in I \tag{11}$$

$$C_i + s \leq S_{i'} \qquad \forall (i,i') \in D_k; \forall k \in M^{CC} \tag{12}$$

$$C_i = S_{i'} \qquad \forall (i,i') \in I_c; \forall c \in C_k; \forall k \in M^{CC} \tag{13}$$

$$C_i = S_i + P_i \qquad \forall i \in A \tag{14}$$

Objective (1) minimises a weighted function of the makespan, i.e. the completion time of the last processed charge, and total waiting time. Constraint (2) makes sure that exactly one machine is chosen to process each of the operations relative to a charge. Constraint (3) calculates the completion time of an operation, which is equal to the start time of the operation plus its processing time on the selected machine and waiting time. Constraint (4) ensures that the start time of each operation of a charge j is equal to the completion time of the directly preceding operation of the same charge. Note that these completion times may include waiting times to model the blocking of machines and/or production lines, which is applicable for all types of operations except for the casting operations on the revolving towers and continuous casters (CC). Operations on the CC machines should be processed in a consecutive manner without waiting times, which is facilitated by the possibility to adjust the processing times on these machines (cf. infra). Constraint (5) models completion times of operations on the revolving towers. These calculations do not include waiting times, because the waiting of a charge for the next operation does not prevent these machines from being used for operations relative to other charges. The start time of the next operation for charge j on the revolving tower is calculated via constraint (6). Constraint (7) guarantees that one of the variables $Y_{ii'}$ or $Y_{i'i}$, denoting the (in-)direct sequence between operations, is equal to one in case the same machine is chosen for operation i and i'; i.e. either operation i precedes i' or vice versa. In case the two operations use the same machine, the right-hand side equals one, which forces that one of the precedence variables on the left-hand side is set equal to

one. Constraint (8) prohibits operations from being processed simultaneously on the same machine, imposing that the completion time of a preceding operation i should be smaller than or equal to the start time of any succeeding operation. Variable $Y_{ii'}$, which indicates if an operation is performed (in-)directly before or after another operation, is used to make this constraint relevant or redundant. If this variable is one, the large positive number is multiplied by zero, and the constraint is relevant, forcing the start time of operation i' to be larger than or equal to the completion time of operation i. Otherwise, if the variable is zero, the constraint becomes redundant. Constraint (9) determines the waiting time of each charge by taking the sum of waiting times relative to all operations to process the charge. Constraint (10) calculates the total waiting time by aggregating the waiting times of all charges. Constraint (11) models the makespan or duration to process all charges, which is larger than or equal to the completion time of any operation. Constraint (12) indicates that the start time of a cast's first operation ($S_{i'}$) can only start after the previous cast's completion time (C_i) plus a setup time (s) on the CC machines. Constraint (13) enforces the continuity of casting within a cast, requiring the direct precedence between charges of the same cast on the CC machines without waiting times. Constraint (14) determines the processing times for the casting operations as these are adjustable, represented by the decision variables P_i. Constraints (2) to (14) model the flexible SCC-scheduling problem with adjustable processing times and full sequencing flexibility. In the following, we model constraints relative to resources that are shared between different machines or operations.

$$Y_{ii'} + Y_{i'i} = 1 \qquad \forall i \neq i' \in O \quad (15)$$

Constraint (15) specifies that when two operations require the use of the shared oxygen resource, these operations cannot be performed simultaneously. In other words, either operation i should be processed before operation i' ($Y_{ii'} = 1$) or vice versa ($Y_{i'i} = 1$). Precedence relations are enforced via constraint (8).

$$Y_{ii'} + Y_{i'i} \geq Z_{li} + Z_{li'} - 1 \qquad \forall l \in L; \forall i \neq i' \in (T \cup R) \quad (16)$$

$$Z_{li} \geq X_{ik} \qquad l = l_k; \forall k \in M_i; \forall i \in R \quad (17)$$

$$X_{ik} + X_{i''k''} - 1 \leq X_{i'a} \qquad l_k = l_{k''}; \forall k'' \in M_{i''}; \forall k \in M_i; \forall (i, i', i'') \in B_j; \forall j \in J \quad (18)$$

$$X_{ia} + X_{i'k'} - 1 \leq Z_{li} \qquad l = l_{k'}; \forall k' \in M_{i'}; \forall (i, i') \in Q_j, \ \forall j \in J \quad (19)$$

$$X_{ik} + X_{i''k''} - 1 \leq \sum_{k \neq a \in M_{i'}} X_{i'k} \qquad l_k \neq l_{k''}; \forall k'' \in M_{i''}; \forall k \in M_i; \forall (i, i', i'') \in B_j; \forall j \in J \quad (20)$$

When an operation of a charge is executed in the refining stage, the steel wagon dedicated to the employed production line is occupied and no (regular or transport) operations of the same or another charge can be processed at the same time on this production line. We model this shared resource via constraint (16), which is similar to constraint (7). The following constraints ensure the appropriate steel wagon or transportation resource is selected. Constraint (17) relates to operations in the refining stage, apart from transport operations in this zone, and ensures that the steel wagon is selected relative to the production line of machine k on which operation i is scheduled. When charges are transported from one operation to the next operation between machines on the same production line, the steel wagon is also used. In order to model that a charge occupies the steel wagon for those transport operations, we assume that the transport operation is performed on a so-called artificial machine ($a \in M$). The artificial machine, as its name suggests, does not refer to a physical machine and is left out of constraint (7). Assigning the operation to an artificial machine blocks the production line during the time a charge is transported between subsequent operations on the same production line and prevents the production line from being used to process another charge. This time consists of both the actual time to transport the charge and the waiting time

between successive operations. As the former is negligible, this time basically comprises the waiting time only. Constraint (18) models that this artificial machine is chosen when the previous and next operation's machines are located on the same production line. Constraint (19) stipulates that a steel wagon resource is occupied when relying on an artificial machine to conduct the transport operation to the charge's next operation. The activation of the steel wagon via constraints (16)-(19), together with constraint (8), prevents a production line is occupied by two charges at the same time. However, transport operations in the refining zone do not necessarily use a steel wagon resource. In case a charge is switched between production lines, a crane is used for transport and the steel wagon is no longer occupied. This is represented via constraint (20). This constraint ensures that a crane, and not the artificial machine, is selected in case the operation before and after the transport operation use machines that are on different production lines. Note that the processing time of a charge on the artificial machine is intrinsically equal to zero, in contrast to the effective transportation time required by a crane.

$$
\begin{aligned}
&X_{ik} \in \{0,1\} && \forall i \in I_k,\ \forall k \in M \\
&Y_{ii'} \in \{0,1\} && \forall i \neq i' \in I \\
&Z_{li} \in \{0,1\} && \forall i \in R \cup T,\ \forall l \in L \\
&S_i, C_i, W_i \geq 0 && \forall i \in I \\
&W_j \geq 0 && \forall j \in J \\
&p^{min} \leq P_i \leq p^{max} && \forall i \in A \\
&W, C^{max} \geq 0 && (21)
\end{aligned}
$$

Constraints (21) model the variable domain constraints.

4 Solution methodology

In this section, we describe a hill-climbing heuristic solution procedure to solve SCC instances with dual shared-resource and blocking constraints. This method is an iterative approach that starts from an initial solution, makes incremental changes to the solution encoding and updates the solution only if a better solution is found. Via this method, instances can be solved in a short computational run time, which is required due to the operational character of the problem under study and has been indicated as important by the collaborating company to effectuate its use in real life. The relevance of hill-climbing heuristics has been demonstrated in various application domains (see e.g. Vaughan et al. (2005)). As a drawback, the proposed heuristic does not include mechanisms to escape from local optima (e.g. perturbations, memorisation of previous changes). These require a significantly higher amount of valuable CPU time to again improve the solution point after a perturbation or worsening move and reach an acceptable solution quality. However, the proposed method does not rely on a single operator but applies different operators with problem-specific knowledge embedded to yield high-quality solutions in small CPU times. These operators search the solution space in a diversified manner. A general overview of the procedure is given via the flowchart, depicted in Figure 4. The heuristic operates on a schedule representation that consists of two lists, i.e. a charge-sequence list and a machine-assignment list, which are explained in Section 4.1. The algorithm starts with an initialisation step, which is detailed in Section 4.2. In this step, we build a first schedule taking problem-specific information into account. Instead of randomly selecting individual charges, we select casts produced on CC machines and let the charge sequence of the initial solution follow the sequence of charges on the casters, which is given as input by the cast schedule of ArcelorMittal. The initial schedule is improved following two local neighbourhood searches or so-called improvement

stages, discussed in Section 4.3. In the *first improvement stage*, the machine assignments are kept fixed and better solutions are searched in the neighbourhood of the solution by changing the order charges are processed using a dedicated operator. In the *second improvement stage*, the charge sequence is kept fixed and machine assignments are adapted for singular operations to improve the solution. Each of these neighbourhoods is searched for a number of moves. Each move generates a different schedule by performing a random neighbourhood operator that constructs a new schedule (*new*) in the neighbourhood of the current incumbent schedule (*current*). After each move, the quality of a solution encoding is evaluated in two steps, which is described in Section 4.4. In the first step, the encoding is transformed into a schedule using a decoding mechanism, i.e. a schedule generation scheme, that relies on a recursive algorithm to optimise the makespan or total duration to complete all charges. In the second step, the generated solution is repaired using linear programming to take into account the continuity of casting, setup times, adjustable processing times, and waiting times. The newly generated and repaired schedule is compared with the current schedule based on the objective function value. If the objective value of the current solution is better than the objective value of the incumbent solution, the current solution is retained and the incumbent solution is updated. These two improvement stages are visited for a number of iterations until the maximum number of iterations is reached.

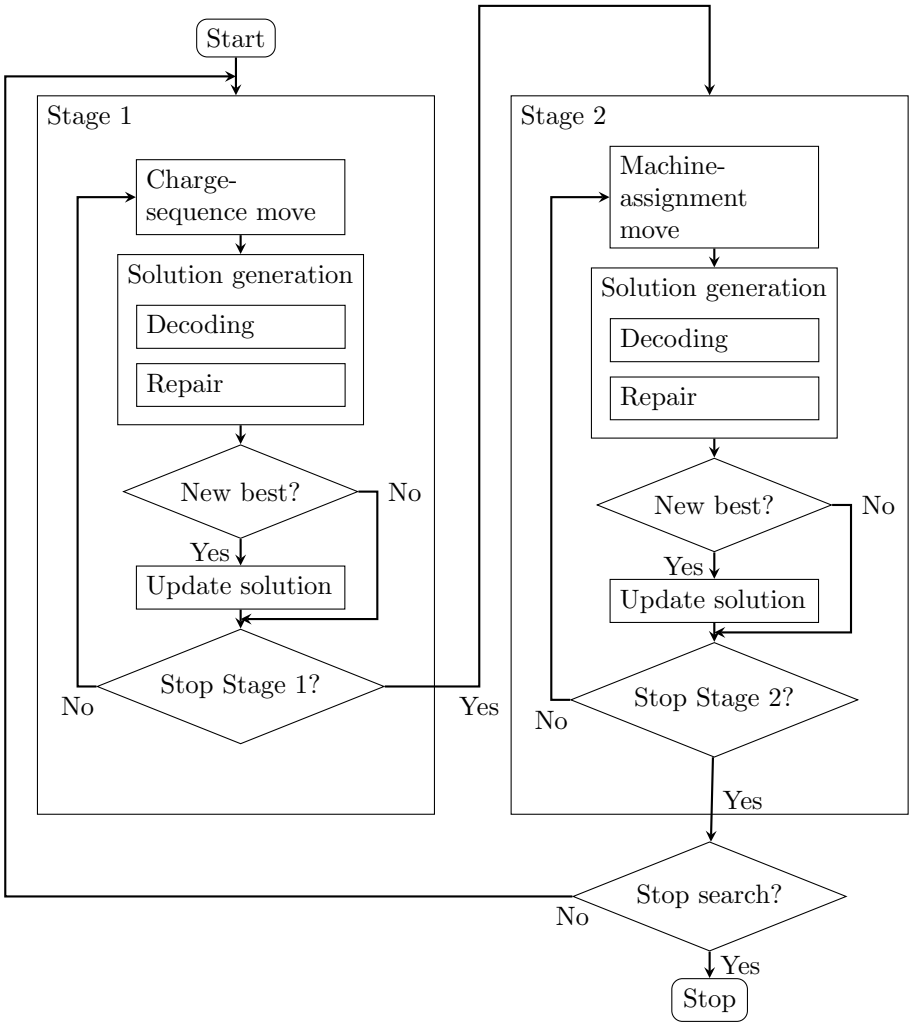

Figure 4: Flowchart of the proposed hill-climbing heuristic

4.1 Encoding

In the proposed heuristic, the different components and heuristic operators are applied on a schedule, which is represented by a schedule representation scheme. In order to encode a schedule, we make use of two vector notations, i.e.

- In order to represent the *sequence of charges*, a Job-Based-Representation-First (JBRF) representation is used, similar to studies of Long et al. (2018) and Xu et al. (2020). This vector encoding represents the order in which charges, indicated by their job number, are processed. The designation 'First' refers to the fact that only the order of charges in the first production stage, i.e. the converter stage, is encoded. The sequence in other stages is decided by the decoder. Figure 5 shows an example encoding of a 5-charge problem designating the order of charges in the first production stage. Following the example encoding, charges are processed as first charge 4, followed by charges 1, 3, 2 and 5. The order of charges guides the decoding mechanism, relying on the FindSchedule algorithm, to determine the full schedule (cf. Section 4.4.1).

Figure 5: Example JBRF encoding for a 5-charge problem

- In order to represent the *machine assignments of necessary operations to complete a job*, the representation displays the tuples (i, m) relative to an operation of a charge, indicating the assignment of operation i to machine m. This representation is similar to the one suggested by Pezzella et al. (2008). This list comprises also transportation operations, except for those in the refining stage. This is caused by the fact that transportation of charges between machines on the same production line uses the associated steel wagon whereas cranes are used for transportation between different production lines, which is dependent on the machines processing the two subsequent production operations. Figure 6 shows an example of a charge that requires 5 operations for completion.

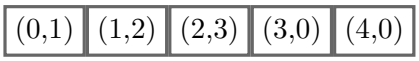

Figure 6: Example machine-assignment encoding for a particular charge

4.2 Initialisation step

4.2.1 Charge sequence

The proposed initialisation heuristic embodies an adaptation of the construction heuristic proposed by Xu et al. (2020) for sequencing charges. The heuristic of Xu et al. (2020) makes use of problem-specific properties of the SCC-scheduling problem, because their initialisation algorithm for the charge sequence is based on the casting sequence in the continuous casting stage. We propose an adaptation to this heuristic. Instead of randomly choosing a charge from each CC machine, we select a cast from each CC machine using a round-robin selection mechanism, i.e. casts are selected from each machine in equal portions and in a circular order, and add the charges in that cast in the same order to the generated sequence of charges. This provides us with a logical order of charges, in the sense that the order of charges in the first stage will be similar to the order of charges on the casters. The specific steps of the initialisation heuristic are shown in Algorithm 1, which takes as inputs the casters and the charge sequence on the CC machines. Figure 7 illustrates the initialisation heuristic for

an example with two CC machines such that we select a list of charges from one of the two casting machines in an alternate manner, i.e. we first select Cast 1 from CC1 followed by Cast 3 from CC2, Cast 2 from CC1 and Cast 4 from CC2. In this way, the generated sequence of charges in the converter stage reflect the order of charges in the continuous casting stage, i.e. first sequence 1-2-3-4-5, followed by sequence 11-12-13-14, sequence 6-7-8 and sequence 16-17-18-19-20.

Algorithm 1: Initialisation of charge sequence (Casters, CC machines)

Step 1: Choose a CC machine using round-robin selection.
Step 2: Select first non-scheduled cast from CC machine.
Step 3: Add charges from cast in predefined order to charge sequence.
Step 4: Return to step 1 if not all casts are sequenced, else finish.
return charge sequence

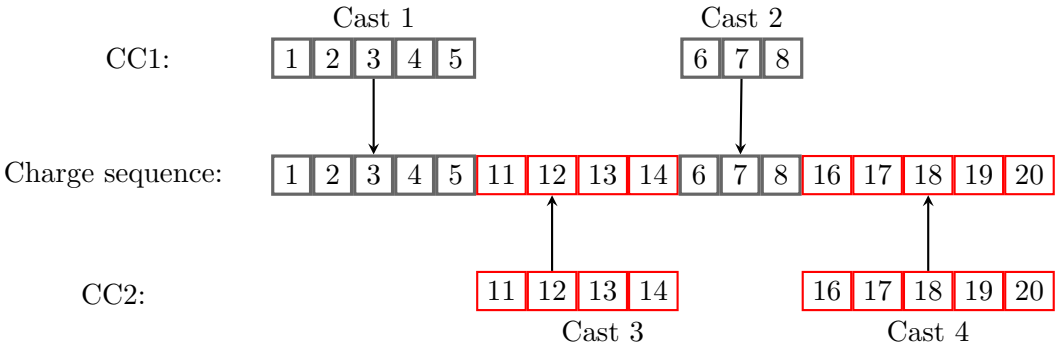

Figure 7: Example initialisation of charge sequence based on four casts on two casting machines

4.2.2 Machine assignment

The initialisation of the machine assignments relies on the random assignment for each operation i to a machine m out of the set of eligible machines M_i. The random assignment is based on random number generation and a discrete probability distribution with equal probabilities for every eligible machine.

4.3 Local neighbourhood search

The initial solution is improved in two different stages, each applying a local search method, defined by a different neighbourhood operator changing either the sequence of charges (Section 4.3.1) or the assignments of operations to machines (Section 4.3.2).

4.3.1 Stage 1: Neighbourhood exploration using the charge-sequence operator

The proposed hill-climbing heuristic relies on the so-called *cast* operator to obtain a neighbouring solution of the considered schedule, by changing the charge sequence. Using the casts as a starting point takes into account the predetermined, and thus immutable, order of charges on a CC in the continuous casting stage and allows for a faster improvement than the mutation operators proposed in the literature, i.e. the *swap* and *shift* operators. The *swap* operator randomly selects two jobs in the sequence and switches their position (Xu et al., 2020). The *shift* operator places a randomly chosen job in another position and shifts

the position of all jobs in between to one position earlier or later, depending whether the selected job is moved forward or backward (Long et al., 2018). The proposed *cast* operator selects a random cast and interchanges encountered charges of the selected cast in the charge sequence and the last encountered charge of the previous cast if specific conditions are met, creating an overlap in the processing of charge sequences of different casts. In this way, the proposed operator primarily focuses on the makespan minimisation, while keeping the waiting times as small as possible.

Algorithm 2 gives an overview of the *cast* operator, which considers the perspective of casts and not of individual charges to change the charge sequence. If we relate the sequence of individual charges to their respective casts, we can clearly see the order of grouped casts resulting from the initialisation method. This will from now on be referred to as the cast sequence. The goal is to let these casts overlap to gain a possible reduction in makespan. This is done by randomly selecting a cast and swapping the first occurrence of this cast with the previous entry relative to the previous cast in this cast sequence. If this is the first time the cast is selected, the new cast sequence can be converted to the charge sequence. Otherwise, we should look further in the cast sequence to perform additional swaps between cast entries. Whenever there are two consecutive entries in the cast sequence after the first, second, ... occurrence of the selected cast and before its last occurrence, which are not equal to the selected cast, an additional swap is done between an entry from the selected cast and the other cast. In this way, an alternating sequence of casts is created and an overlap of charges relative to these casts is generated.

Algorithm 2: Iteration of *cast* operator

Input : Current charge sequence.
Output: New charge sequence.
Step 1. Replace charges in the charge sequence with their respective cast numbers.
Step 2. Choose a random cast which is not the first cast in the initialisation.
Step 3. Find the first occurrence of the selected cast in the cast sequence and switch positions with the previous element in the cast sequence.
Step 4. Iterate starting from the first occurrence of the selected cast for the remaining length of the cast sequence. If there are two consecutive cast numbers on positions $i-1$ and i, which are not the chosen cast, whereas cast number on position $i+1$ is of the selected cast, swap position i with $i+1$ and continue applying step 4 starting from position i.
Step 5. From the new cast sequence, build the new charge sequence by replacing casts by associated charges in the order of the charge sequence.

This is further illustrated with an example. We consider two casts with four charges each, i.e. cast 1 consisting of charges 1 to 4 and cast 2 consisting of charges 5 to 8. The charge sequence that would result from the previously explained initialisation step is shown in Figure 8. An example of two iterations is shown, in which cast 2 is selected as random cast. In the first iteration, the charge sequence is first transformed into a cast sequence. The first occurrence of the selected cast is swapped with the previous sequence entry, i.e. positions 4 and 5 are switched. This new cast sequence is then converted to the new charge sequence. In the second iteration, the same random cast, i.e. cast 2, is selected. The first occurrence of cast 2 is again swapped with the previous entry, switching positions 3 and 4. Since this cast has been selected before, the charge-sequence encoding is further investigated. There are two consecutive positions, positions 4 and 5, in the cast sequence that are different from the selected cast 2. The latter of these positions is swapped with the next entry, switching positions 5 and 6. The result is a new cast sequence that shows a larger cast overlap, leading to an alternating sequence of charges relative to the associated casts. Note that the proposed cast operator relies on a higher built-in intelligence compared to the swap or shift operator

to yield high-quality solutions in a more efficient manner. The cast operator focuses on specific positions in the cast sequence to change the charge sequence, in contrast to the well-known swap operator, which may swap any pair of positions likely with the same selection probability. Moreover, the cast operator may switch multiple pairs of positions in a single operation.

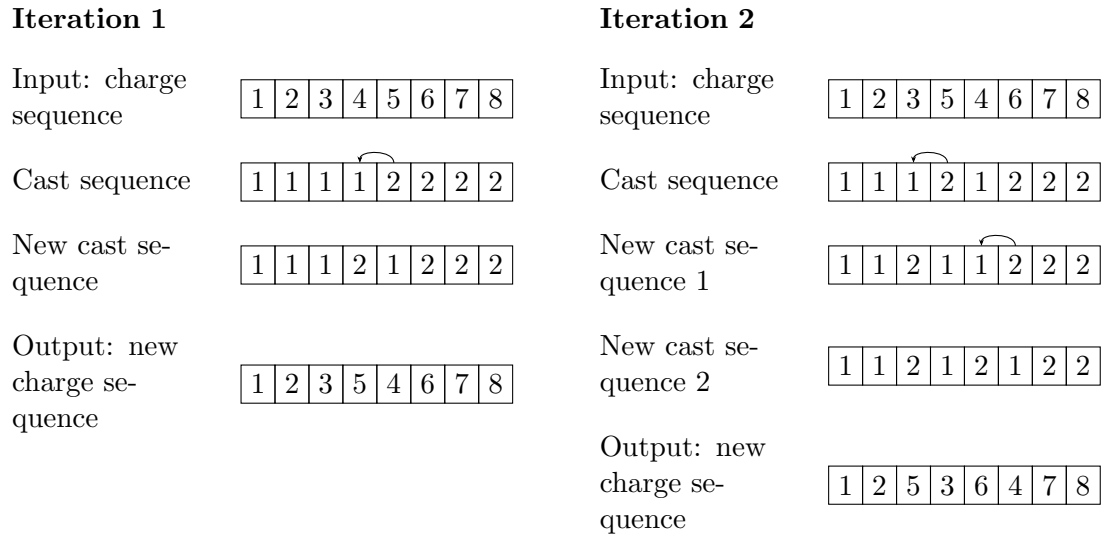

Figure 8: Example of the *cast* operator

4.3.2 Stage 2: Neighbourhood exploration using the machine-assignment operator

The machine-assignment operator randomly changes the machine of an operation in the encoding. One of the operations is chosen in a uniformly random manner, and another machine out of the set M_i, eligible to execute operation i, is chosen.

4.4 Solution generation and repair step

In order to evaluate the associated objective value of an encoding, a solution is generated based on the solution representation in two different steps following a construction-and-repair approach. First, we transform the charge-sequence and machine-assignment encoding into an actual production schedule using a decoding mechanism, which takes only the makespan into account as objective. However, the decoding mechanism results in a sub-optimal or even an infeasible schedule that may violate some of the problem-specific constraints, because the continuity of casting is not taken into account. To that purpose, a repair method is appended that considers adjustable processing times in the casting stage, enforces continuity of casting and required setup times, and minimises waiting times.

4.4.1 Decoding mechanism

Most decoding heuristics in the SCC-scheduling literature are based on the latest completion time on a machine/resource of previous operations, such as the ones proposed in the studies of Long et al. (2018) and Gao and Pan (2016), which implemented a decoding mechanism that schedules production operations one by one as these are identical for all charges. In this way, when planning these operations, they consider the latest completion time of any operation on a machine as the earliest available time for that machine. However, for the

problem under study, for which flexibility is allowed in the number and sequencing of operations to process charges, such a decoding mechanism would negatively impact the makespan, and thus the objective value. Given that we plan charge by charge due to this additional flexibility, using the method described above would not recognise that a machine might have been idle for a long time in between two scheduled operations, during which operations of other charges could have been processed. In response, the proposed heuristic relies on the recursive algorithm by Aschauer et al. (2017) to decode a solution representation for a flexible job-shop problem. In contrast to the decoding mechanisms devised for the SCC-scheduling problem, the decoding mechanism of Aschauer et al. (2017) considers no-wait constraints and controllable processing times, preventing gaps between subsequent operations, and blocking constraints. These blocking constraints model the occupation of resources or machines during waiting times before charges can be transported to the next machine/production line, and/or (potentially longer) times to process charges. However, because the problem considered in Aschauer et al. (2017) does not correspond to the exact problem under study, several adaptations have been conducted to the decoding heuristic, which is described in the following.

The decoding mechanism processes the charges one by one as indicated by the charge sequence and tries to find the earliest available time slots to schedule the subsequent operations of a single charge. The pseudocode for the recursive algorithm FindSchedule is presented in Algorithm 3, which has as input arguments the charge j, the operation number n, the earliest start time S_e and latest possible start time S_l. The selected start time S_{jn} and the processing time P_{jn} are the relevant output of the algorithm. The algorithm recursively calls itself to find a directly connecting time slot with a specific start and completion time for the next operation using the function FindSlot, which is done until the last operation is reached (status *true*). This might not be successful and an error ($err = 1$) might be returned, because of which the start times and/or processing times of previously scheduled operations need to be adjusted. At the end of an iteration of the decoding mechanism, i.e. when all operations relative to a charge are scheduled, the job timetable and resource schedules of involved machines are updated and fixed for future iterations to accurately schedule other charges. In the following, we explain the functions FindSchedule and FindSlot , and the update of the resource schedules in more detail and provide an illustrative example.

The function FindSchedule
The function is initialised as follows. The function is called with a particular charge (in the order of the charge sequence) and its first operation ($n = 0$) as input. Its earliest and latest start times can be set respectively to 0 and $+\infty$ for the first function call of a charge. The charge and its operation number correspond to an operation $i = i_{jn}$. The machine that is assigned to this operation in the encoding is retrieved, i.e. $k = k_i$. Based on the assigned machine, the processing time p_{ik} is retrieved, representing the lower bound (p^{lb}) for the time to process the charge. To calculate the upper bound (p^{ub}) on the processing time, a *margin* is added, representing a reasonable maximum waiting time. The FindSlot function provides the first available time slot with S_1 and S_2 as respective start and end times of the time slot for the selected machine k, considering the earliest start time and the lower bound of the processing time. The processing time of the operation is set to its lower bound ($P_{jn} = p^{lb}$), and the error indicator variable is initialised to a value of 0 ($err = 0$).

Subsequently, after the initialisation step, a while loop is initiated to schedule all operations relative to a charge that will recursively call the function FindSchedule for the next operation, and the current operation's completion time as earliest start time. First, the algorithm checks if the start time of the considered time slot (S_1) is not later than the latest possible

start time (S_l). If $S_l < S_1$, the operation start time S_{jn} is set to S_1 and an error ($err = 1$) is returned, terminating the function because it is impossible to find a feasible schedule. Otherwise, it is verified whether the last operation is reached ($n = N_{max}$). This would indicate that for all the previous operations a suitable time slot has been found, leading to a complete schedule, and the function can terminate. In other cases, the function FindSchedule has to be called for the next operation. This function call for the next operation will return information that is of importance for the current operation, i.e. the start time ($S_{jn_{+1}}$), the processing time ($P_{jn_{+1}}$), and the error code (err_{+1}) of the next operation, indicated by the index '+1'. If the next operation has an error code equal to 0, there are three possible cases. The first one is the ideal case, in which the current operation can be scheduled directly before the next operation, i.e. $S_{jn} = S_{jn_{+1}} - p^{lb}$. The second case adjusts the processing time of the current operation ($P_{jn} = S_{jn_{+1}} - S_l$) to schedule the current and next operations in a consecutive manner, i.e. $S_{jn} = S_l$. In the third case, none of the previous options is possible and an error code of 1 is returned, implying that no proper start time could be assigned due to the violation of a start-time condition and the start time or processing time of the operation before the current operation needs to be adapted. Note that whenever an error code equal to 1 is returned ($err_{+1} = 1$), the *else*-clause relative to the current operation is invoked. Then, the FindSlot function is called until a suitable, later time slot $[S_1, S_2]$ is found, for which some waiting time has been inserted and thus increasing the operation's completion time. This time slot satisfies the required conditions, i.e. the completion time of the current operation can be aligned with the next operation's start time. This new time slot is then used again to determine the start time of the next operations, following the outer while loop. If the algorithm is successful, the charge will have a start time, processing time, and machine assigned for each operation. This procedure is repeated until the whole charge sequence is planned. The resulting schedule ensures all operations of a charge concatenate, and thus a charge utilises a machine or resource for the time it is processed. The decoding algorithm ensures that foremost the makespan is minimised and includes waiting times when necessary to find a feasible schedule. The minimisation of waiting times and other problem-specific constraints are introduced in the repair method in Section 4.4.2.

Note that the machine-assignment encoding does not consider the transportation operations in the refining stage, i.e. transportation of charges between machines on the same production line using the associated steel wagon (modelled via the artificial machine) or between different production lines using cranes. This is caused by the fact that the use of the steel wagon or crane is dependent on the machines processing two subsequent production operations, which cannot be modelled adequately via the machine-assignment list. Accordingly, whenever a suitable time slot for the next operation in the refining zone is searched, the function FindSlot concurrently allocates always the earliest available crane to carry out the transport, which accounts for an upper bound on the actual transportation time between machines. The required resource type and associated processing time for transportation in the refining zone are precisely determined when the schedule information is updated (cf. infra). The reasons for neglecting the artificial machine in this stage are the following. First, the artificial machine is not an actual machine, but rather embodies a technical implementation to model the selection of the steel wagon and enable the blocking of a production line. Hence, there is no artificial machine representing the steel wagon resource per production line nor an actual timetable or resource schedule associated with the artificial machine. Using the function FindSlot relative to this machine, would not be correct. Second, the function FindSlot operates in a greedy manner to determine the earliest time slot and associated machine to operate the next operation. The function would mistakenly always select the artificial machine as processing times on this machine are zero, leading to the inadequate

timing of subsequent operations, or would unnecessarily complicate the function FindSlot increasing the required run times of the algorithm. Third, the introduced unnecessary waiting times are easily rectified via the repair mechanism that must be applied anyway to find feasible schedules for the problem under study.

Algorithm 3: Decoding mechanism FindSchedule(j,n,S_e,S_l)

(**Legend.** k_i: Machine assigned in encoding to operation i; err: Error variable; S_{jn}: Start time assigned to operation n of charge j; i_{jn}: Operation corresponding to operation n of charge j; S_e: Earliest start time; P_{jn}: Processing time of operation n of charge j; p^{lb}: Lower bound processing time; S_l: Latest start time; p^{ub}: upper bound processing time; S_1: Start time slot; $margin$: Allowed extra processing time or waiting time; S_2: End time slot)

Input : j,n,S_e,S_l
Output: S_{jn},P_{jn},err
$i = i_{jn}$
$k = k_i$
$p^{lb} = p_{ik}; p^{ub} = p_{ik} + margin$
$[S_1, S_2] =$ FindSlot(k, S_e, p^{lb})
$P_{jn} = p^{lb}$
$err = 0$
while *true* **do**
  if $S_l \geq S_1$ **then**
    if $n = N_{max}$ **then**
      $S_{jn} = S_1$ **return**
    else
      $S_{jn_{+1}}, P_{jn_{+1}}, err_{+1} =$ FindSchedule$(j, n+1, S_1 + p^{lb}, S_2)$
      if $err_{+1} = 0$ **then**
        if $S_l \geq (S_{jn_{+1}} - p^{lb})$ **then**
          $S_{jn} = S_{jn_{+1}} - p^{lb}$
        else if $S_l \geq (S_{jn_{+1}} - p^{ub})$ **then**
          $S_{jn} = S_l$
          $P_{jn} = S_{jn_{+1}} - S_l$
        else
          $S_{jn} = S_{jn_{+1}} - p^{ub}$
          $err = 1$
        end
        return
      else
        $[S_1, S_2] =$ FindSlot$(k, max(S_{jn_{+1}} - p^{ub}, S_2), p^{lb})$
        while $S_2 < S_{jn_{+1}}$ **do**
          $[S_1, S_2] =$ FindSlot(k, S_2, p^{lb})
        end
      end
    end
  else
    $S_{jn} = S_1$
    $err = 1$
    return
  end
end

The function FindSlot

The function FindSlot finds the earliest time slot to process a particular operation. The arguments of the function comprise a machine k, the earliest start time S_e, and the input processing time p^{lb}. The function determines the earliest time slot with a suitable minimal

length equal to the input processing time on the machine. If a charge is successfully planned using the FindSchedule algorithm, the corresponding time slots cannot be used by subsequent charges anymore. Note that the proposed schedule generation scheme is different compared to the one of Aschauer et al. (2017) due to resource sharing constraints relative to steel wagons, shared between different machines, and the oxygen resource, shared between different operations. The latter should also be taken into account in function FindSlot, as this resource should also be available, similar to a machine, to find a suitable time slot.

Update of schedule information
The function FindSchedule is called following the order of the charge sequence as determined by the local search heuristic. At the end of an iteration of the decoding mechanism, i.e. after all operations relative to a charge have been scheduled and feasible start times and machines are chosen, the availability of involved resources is updated to plan subsequent charges. This boils down to adjusting the timetables of the scheduled machines and the oxygen resource with the start and completion of scheduled operations to denote the resource usage. For production operations in the refining zone, this comprehends not only adjusting the resource schedules of the assigned machines but also the timetables of the other machines on that production line to enable the blocking of the line. This corresponds to the functioning of the steel wagon variables Z_{li} in the MILP model. When planning transport operations in the refining zone, the decoding mechanism always accounts for the use of a crane to carry out the transport between the operations in the refining zone (cf. supra), requiring effective transportation time. However, a crane is not always required. When transporting a charge between operations on machines of the same production line, the production line needs to be blocked, which is done by allocating the artificial machine to the involved operation. For that purpose, when updating the schedule information, we consider the selected machines to process the production operations relative to a charge in a chronological manner and determine if a switch in production lines and associated use of the scheduled crane is required. If this is not the case, the crane is deselected, and the artificial machine is assigned, blocking all machines on the involved production line during the time the charge is transported. The latter implies that the resource schedules of all these machines are adapted and fixed for future iterations. The information concerning the utilisation of machines, the relative position of operations on machines and production lines, and the order in which operations use the shared oxygen resource are input to the repair mechanism. These deduced parameters are further shown in the following section.

Note that, although the processing times relative to the use of an artificial machine are zero in contrast to using a crane and therefore some of the operations can possibly be advanced, the start and completion times of the scheduled operations are retained as long as the algorithm FindSchedule is run to schedule other charges in the charge sequence as the main goal of this algorithm is to determine the charge sequencing, machine assignment, and resource occupation. However, as a result, the schedule yielded by the decoding mechanism may include some unnecessary waiting times due to the needless allocation of a crane to carry out the transport between operations, which overestimates the transport times. These waiting times are minimised together with the makespan via the application of the repair mechanism anyway, after the execution of the decoding mechanism.

Illustrative example
In the following, we provide a simple illustration of the main principles underlying the decoding mechanism based on the example introduced in Section 3.2 and visualised in Figure 3. The decoding mechanism works by iterating over the charge sequence [1,3,2,4]. Assume all operations relative to charge 1 have been scheduled and the associated resource schedules

have been updated. Subsequently, we schedule the operations relative to charge 3. In this illustration, we focus on the operations in the converter stage, consisting of three operations of a single time unit, and following transport operation to the refining zone. The second operation in the converter stage requires the use of the oxygen resource. The function FindSchedule starts with the first operation of charge 3 ($p^{lb} = p_{ik} = 1$, $S_e = 1$, $S_l = +\inf$), which needs to be planned on one of the converter machines. The machine encoding indicates that the operation needs to be processed on CV2 and the function FindSlot returns moment 1 as the first available time slot on CV2. However, to find a feasible time slot for the first operation, the algorithm recursively calls the function FindSchedule. The next operation, i.e. operation 2, though, can only start at moment 3 on CV2 as the oxygen resource is already occupied at moment 2 by the second operation of charge 1 on CV1, which has been detected via the function FindSlot. This would create a waiting time between this operation and the previous operation, which the algorithm does not allow as the maximum waiting ($margin = 0$) is assumed to be exceeded and the processing time of converter operations cannot be extended ($p^{ub} = p_{ik}$). The function FindSchedule returns in that case an error ($err_{+1} = 1$), which implies the first operation is rescheduled via the function FindSlot and delayed to moment 2 to ensure there is no waiting time between the two operations. This newly determined start time for operation 1 is accepted, as operations 2 and 3 can be assigned to consecutive time slots, i.e. moments 3 and 4 respectively. The subsequent operation is the transport to the refining zone that can be planned at moment 5 using CRANE1. In this way, all operations in the converter stage for charge 3 are successfully planned. Based on the availability of all machines for subsequent operations, the function FindSchedule will succeed in finding a suitable time slot for all operations in the refining zone and the casting stage relative to this charge. Afterwards, the production timetable is updated, accounting for the resource usage of assigned machines and shared resources at relevant times, and blocking the machines on the used production lines.

4.4.2 Repair mechanism

Given that the above decoding heuristic only tries to minimise the makespan, and does not consider either the continuity of casting, setup times, adjustable processing times or minimisation of waiting times, an improvement mechanism relying on the MILP model proposed in Section 3.3 tries to repair the schedule generated by the decoding mechanism. A similar approach has been implemented by Long et al. (2018). Note that the quality improvement method proposed by the latter only decreases waiting times and adds the ability to adjust the processing times. For the problem under study, improvement is also necessary to incorporate critical constraints in the casting stage, such as the continuity of casting and setup times in the casting stage. The repair mechanism takes the schedule generated via the decoding mechanism (cf. Section 4.4.1) as input and fixes the binary variables $Y_{ii'}$ ($\forall i \neq i' \in I$), X_{ik} ($\forall i \in I_k, \forall k \in M$) and Z_{li} ($\forall i \in R \cup T, \forall l \in L$) referring to the charge sequences, machine assignments and resource occupation. The fixed decision values are represented by additional parameters e_i, f_k, g_i and h_i defined below and input to the repair mechanism. As a result, the original model (1)-(21), i.e. a MILP problem, is reduced to the simplified model (22)-(35) presented below, i.e. a LP problem, which can be solved via linear programming. In this way, start times S_i ($\forall i \in I$), completion times C_i ($\forall i \in I$), waiting times W_i ($\forall i \in I$) and W_j ($\forall j \in J$), and adjustable processing times P_i ($\forall i \in A$) are specified, together with performance measures C^{max} and W. As the computational effort for executing the repair mechanism is small, no time limit is imposed, and the model is solved to optimality.

Additional parameters

e_i	Machine assigned to operation i
f_k	Operations assigned to machine $k \in M \setminus \{a\}$
g_i	Operation scheduled after operation i on machine k
h_i	Operation scheduled after operation i on the oxygen resource

Mathematical formulation

$$\text{Min} \quad \delta_1 \times C^{max} + \delta_2 \times W \tag{22}$$

s.t.

$$C_i = S_i + p_{ik} + W_i \qquad k = e_i; \forall i \in I \setminus (A \cup D) \tag{23}$$

$$C_i = S_{i'} \qquad i \notin D; \forall (i, i') \in I_j; \forall j \in J \tag{24}$$

$$C_i = S_i + p_{ik} \qquad k = e_i; \forall i \in D \tag{25}$$

$$C_i + W_i = S_{i'} \qquad i \in D; \forall (i, i') \in I_j; \forall j \in J \tag{26}$$

$$C_i \leq S_{i'} \qquad i' = g_i; \forall i \in f_k; \forall k \in M \tag{27}$$

$$W_j = \sum_{i \in I_j} W_i \qquad \forall j \in J \tag{28}$$

$$W = \sum_{j \epsilon J} W_j \tag{29}$$

$$C^{max} \geq C_i \qquad \forall i \in I \tag{30}$$

$$C_i + s \leq S_{i'} \qquad \forall (i, i') \in D_k; \forall k \in M^{CC} \tag{31}$$

$$C_i = S_{i'} \qquad \forall i; i' \in I_c; \forall c \in C_k; \forall k \in M^{CC} \tag{32}$$

$$C_i = S_i + P_i \qquad \forall i \in A \tag{33}$$

$$C_i \leq S_{i'} \qquad i' = h_i; \forall i \neq j \in O \tag{34}$$

$$S_i, C_i, W_i \geq 0 \qquad \forall i \in I$$

$$W_j \geq 0 \qquad \forall j \in J$$

$$p^{min} \leq P_i \leq p^{max} \qquad \forall i \in A$$

$$W, C^{max} \geq 0 \tag{35}$$

Equations (22) to (33) are similar to the model presented in Section 3.3. Contrary to the previously presented model, operations can now occupy more than one machine in constraint (27). More precisely, the input parameter f_k indicates not only the use of a specific machine for a particular operation but also the blocking of other machines. This way, the steel wagons are blocked when an operation is performed on one of the machines of a production line, preventing another charge can be processed on the same line. Constraint (34) avoids that the shared oxygen resource is used simultaneously for different operations, in correspondence to the schedule generated via the decoding heuristic.

5 Computational experiments

In this section, computational experiments are described to validate the proposed methodology. In Section 5.1, we discuss the test design and input values for parameters characterising the solution methodology. Section 5.2 validates design choices of the heuristic. Section 5.3 benchmarks the proposed heuristic to an exact solution methodology based on mathematical programming and the genetic algorithm of Long et al. (2018). Experiments were conducted on a MacBook Pro with a 2 GHz Dual-Core Intel Core i5 and 8GB of memory. The proposed heuristic is programmed using the JDK 8 Java compiler and linked to Gurobi version 9.1.1 as (MI)LP solver.

5.1 Test design

5.1.1 Dataset and problem settings

Experiments are conducted using real-life data from the integrated steel company ArcelorMittal (Ghent, Belgium). The continuous casting facility in Ghent produces yearly approximately six million tons of semi-finished products of steel (known as blooms, billets or slabs), leading to all kinds of high-quality applications for various industries (e.g. automotive, construction, energy, packaging, mining). The company site in Ghent handles every step of the production process, from the supply of raw materials to the coating of steel and the production of laser-welded blanks, and is part of ArcelorMittal, one of the world's largest steel companies. For more information, we refer to `https://belgium.ArcelorMittal.com`. In collaboration with the company, we recorded in the year 2021, 20 data instances relative to the continuous casting scheduling problem. The instance size characteristics are shown in Table 5, giving insight into number of casts ($\sum_k |C_k|$), charges ($|J|$), machines ($|M|$) and operations ($|I|$) to be completed in a limited time span. To give the reader some notion of the complexity of an instance, we illustrate in Appendix A the resulting resource schedule associated with the solution to instance 1, yielded by the proposed procedure. Note that this instance comprises 841 different operations, such that only a subset of the operations could be depicted. Specific instance characteristics and data, averaged over the instances, are further detailed below.

Instance	**# casts**	**# charges**	**# machines**	**# operations**
1	10	53	19	841
2	12	59	19	947
3	12	61	19	974
4	10	57	19	915
5	12	60	19	954
6	10	52	19	832
7	10	60	19	978
8	10	54	19	805
9	10	59	19	921
10	10	58	19	884
11	12	60	19	905
12	12	58	19	860
13	12	61	19	953
14	10	65	19	1006
15	10	58	19	850
16	8	61	19	939
17	10	62	19	904
18	10	63	19	914
19	8	65	19	1005
20	8	62	19	920

Table 5: Overview data instances

Resource characteristics
The necessary operations need to be completed by 19 machines ($|M|$), including transport cranes. An operation can be performed by mostly two to three machines ($|M_i|$), modelling some routing flexibility. This can differ between operations, for example, the CC machine in the casting stage is predetermined. In the converter stage, there are two converter machines that share one oxygen resource for a certain operation, and two metallurgy pan machines.

During operations on these latter machines, samples of the steel are taken and the temperature is checked, among other activities. In the refining zone, production is organised in four production lines ($|L|$) with one steel wagon each. There are 11 machines, comprising eight normal machines (i.e. two Ruhrstahl Heraeus degassers, one ladle furnace, two deslaggers, three metallurgy machines) and three transport machines, including one artificial machine and two transport cranes to switch charges between production lines. In the casting stage, operations are performed on two revolving towers and two continuous casters ($|M^{CC}|$).

Job characteristics
The number of casts ($\sum_k |C_k|$) indicates the number of batches that are processed in a continuous manner on the continuous casters. Casts include charges that are of roughly the same steel grade, each containing, on average, between 5 and 7 charges. For each instance, about 60 charges ($|J|$) are to be scheduled. Note that the required operations to process a charge can be different between charges, consisting of a different set of operations that can be performed on different machines. The number of operations per charge mostly hovers around 15, including transport operations, but this can differ according to the steel grade. The total number of operations ($|I|$) per instance is provided in Table 5 and is a good indication of the complexity of the problem. Note that this number includes transport operations ($|T|$).
The number of operations in every stage of the production process (converter stage, refining stage, casting stage) and related processing times (range and average) of actual production and transport operations are indicated in Appendix B. In the following, we provide some additional information on the different stages. In the converter stage, each charge undergoes three operations on a converter machine, independent of its steel grade. One of these operations requires the use of the shared oxygen resource. The charges are further processed via one operation on one of the metallurgy pan machines. In the refining zone, the number of operations required on each of the machines is different between charges, as this depends on its steel grade. Averaged over the set of data instances, 32%, 30%, 31%, and 7% of the operations are carried out on the Ruhrstahl Heraeus degassers, ladle furnace, deslaggers and metallurgy machines, respectively. In the casting stage, each charge requires a transport operation using a turning tower before two subsequent production operations are performed on the revolving tower and CC, respectively. The casting speeds for processing charges on the continuous casters can be reduced by 10% to ensure precedence between charges of a cast, which implies that the adjustable processing times can range between 35 (p^{min}) and 38.5 (p^{max}) minutes. The time to set up the continuous casters to process a different cast is equal to 40 minutes (s).
The sets modelling the direct precedence relations between operations (I_j, I_c, D_k) are based on real-life data and comprise on average 856, 49, and 8 orders between operations. Sets B_j and Q_j are directly derived from I_j and thus do not comprise additional orders. Between other operations no precedence relations are present, installing to some extent sequencing flexibility between operations relative to a charge.

Objective function structure
The objective function weights have been determined based on discussion with ArcelorMittal. In this way, a larger weight is put on minimising the makespan, i.e. δ_1 is set to 1 and δ_2 to 0.1 in the objective function.

5.1.2 Methodology parameter settings

In this section, we determine the total number of iterations and the proportion of charge-sequence changes (c^s) (stage 1) versus machine-assignment changes (c^m) (stage 2) per iteration of the local search algorithm. In this way, insight is gained into the number of changes to

either the charge-sequence encoding or the machine-assignment encoding per iteration of the two-stage procedure. The different proportions tested are provided in Table 6. In addition, to determine the number of local search iterations, we employed in this parameter-tuning experiment an upper bound of 5000 iterations. Note that, as the heuristic inherently relies on randomness, we conduct 10 different runs with a different seed. For the experiment in this section, we report average results over these runs.

	$\mathbf{c^s}$	$\mathbf{c^m}$
1	5	5
2	5	3
3	5	1
4	10	1
5	3	5
6	1	5
7	1	10

Table 6: Overview of tested combinations (c^s, c^m)

The results are illustrated in Figure 9, which shows the convergence behaviour for a single instance, i.e. instance 1, without loss of generality. The results show that a higher proportion of machine-assignment changes to charge-sequencing changes leads to a faster improvement in objective function values. When $c^s = 1$ and $c^m = 5$, the best convergence is obtained and the objective function value begins to reasonably stagnate around 2000 iterations. Both these settings are used in further experiments.

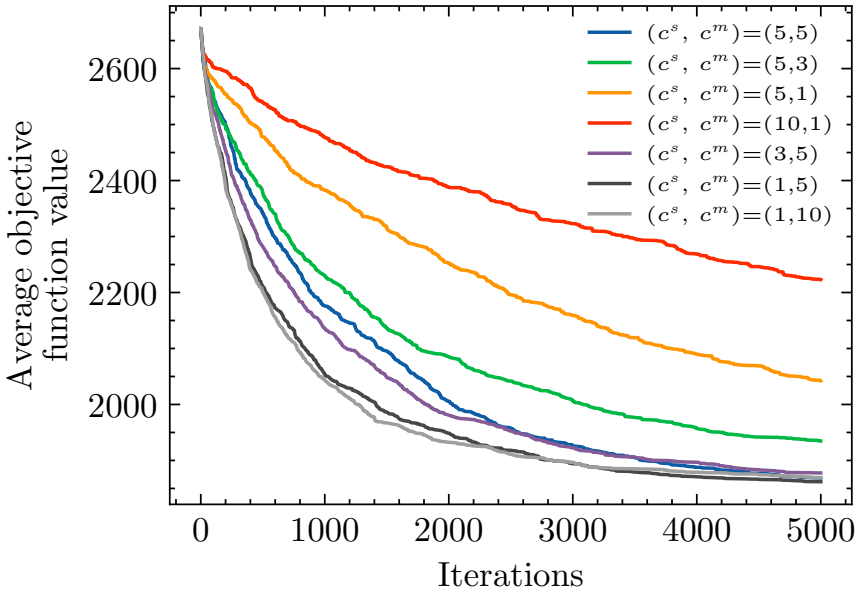

Figure 9: Evolution of objective function values for different combinations (c^s, c^m)

5.2 Validation of heuristic design choices

In order to study the contribution of the proposed operators in different steps of the local search heuristic, we devise different versions of the heuristic with regard to the initialisation method, charge-sequence neighbourhoods and the number of stages. Note that in all experiments, the algorithm is applied in a ceteris paribus setting, which implies that only the settings under investigation are adapted leaving the other settings unchanged to the

ones proposed in the solution methodology to have a fair comparison between the methods. The only exception concerns the initialisation method. The *cast* operator can only function together with the proposed initialisation heuristic. Hence, when comparing different initialisation methods, the best performing combination of the initialisation method and local search operator will be used for the random initialisation and initialisation method of Xu et al. (2020). The results, averaged over the 10 different runs conducted and over the set of 20 test instances are displayed in Table 7. The observed standard deviation is indicated between brackets. Apart from solution quality, we display the required run time in seconds (CPU) and the gap to the best performing version ($\%Gap$). In the following, we explain the conducted experiments and discuss the results per operator.

Design choice	Z	CPU	$\%Gap$
Initialisation method			
Random method	5582.90 (1071.31)	102.10	62.13%
Method of Xu et al. (2020)	2316.70 (207.71)	104.74	8.73%
Proposed method	2114.48 (141.68)	96.88	0.00%
Local search			
Shift method	2188.68 (152.25)	98.10	3.39%
Swap method	2195.82 (152.87)	97.96	3.70%
Cast method	2114.48 (141.68)	96.88	0.00%
Number of stages			
One-stage method	2502.08 (179.27)	97.89	15.49%
Two-stage method	2114.48 (141.68)	96.88	0.00%

Table 7: Validation of heuristic design choices

Initialisation method
The proposed initialisation method is compared to a random initialisation and the initialisation method proposed by Xu et al. (2020), for which the charge sequence is determined by selecting charges from casts in a random manner. Table 7 displays yielded solution quality after running the heuristic for 2000 iterations when employing one of these three initialisation methods. The results show the superior performance of the proposed initialisation method versus the other methods. This performance is thanks to the better schedules that result from the initialisation step; i.e. the quality of the initial schedules equals on average 12503.82, 3367.62 and 2961.98 for the random method, method of Xu et al. (2020) and the proposed initialisation method, respectively. In this way, the proposed method improves initial schedules by 76.31% and 12.05% compared to the random method and method of Xu et al. (2020), respectively. Note that this better performance is observed in a consistent manner, for all individual instances, and is statistically significant (based on the non-parametric Wilcoxon Signed Rank Test). Additional experimentation with smaller and larger cast sizes pointed out that the proposed initialisation method performs better when the instance size increases. This can be explained by the fact that the proposed initialisation method explicitly takes the cast order and the order of charges within casts into account using round-robin tournament selection. Especially for larger instance sizes, the random initialisation can place a job further from its position given by its order on the caster, leading to larger waiting times. For smaller instance sizes, the number of casts is smaller, and the impact on the solution quality is smaller.

Local search operators
We compare the proposed *cast* operator to change the charge sequence with well-known operators from literature, i.e. the *shift* and *swap* operator. Results are displayed in Table 7 and demonstrate that the *shift* and *swap* operators perform worse compared to the *cast* operator, as the quality of the final schedule when employing the *cast* operator outperforms schedules

obtained via the other neighbourhoods by 3.39% and 3.70%, respectively. Moreover, the *shift* and *swap* operators are able to improve the initial schedules only by 26.0% and 25.7% respectively, whereas the *cast* operator improves the initial schedule by 28.5%. Figure 10 shows the realised improvements as a function of the number of local search iterations for a single instance, i.e. instance 1. It can be seen that larger improvements can be realised and that the objective function value improves faster when relying on the *cast* operator compared to the *swap* or *shift* operators.

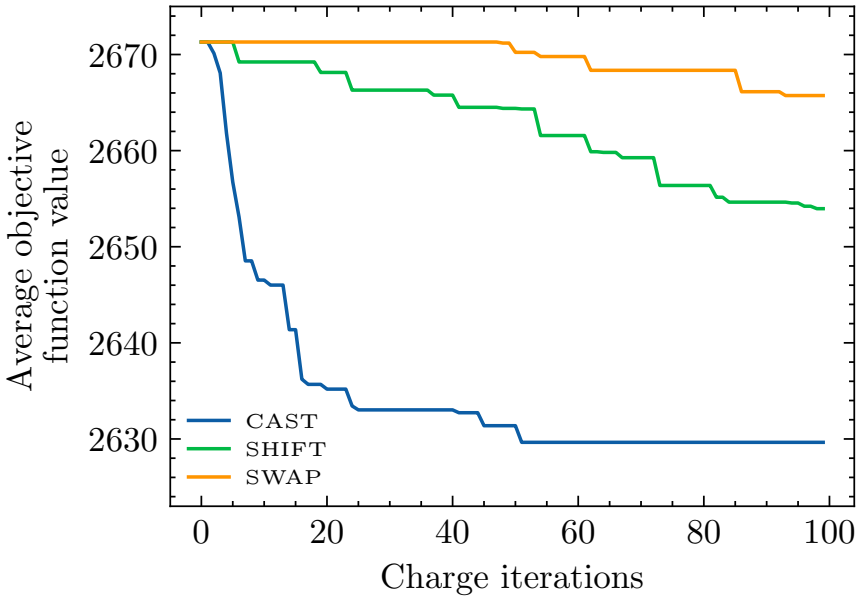

Figure 10: Comparison of convergence behaviour of local search operators on instance 1

Note that additional experimentation with smaller and larger cast sizes pointed out that the proposed *cast* method performs better when cast size increases. The *shift* operator performs better than the *cast* operator when cast sizes are small (< 5), whereas the *cast* operator is superior when dealing with larger, more realistic cast sizes (≥ 5). This can be explained by the fact that the *cast* operator makes use of problem-specific properties and is effective to create and search for better solutions as this operator takes both the order of casts and the order of charges within casts into account. The *swap* operator, ignoring these features, denotes an inferior performance for all problem sizes.

Number of stages

Comparison is made between a one-stage version and the two-stage version of the proposed local search heuristic to validate the two-stage character. In the one-stage version, both the charge sequence and machine assignment are changed together before a solution is decoded, such that the number of charge-sequence changes and machine-assignment changes is identical. The two-stage method considers two separate stages for changing the charge sequence and machine assignments before a solution is decoded, such that the number of changes in each stage can be different. Table 7 indicates that the two-stage version performs significantly better. A further analysis revealed that the two-stage method performs consistently better than the one-stage method for every instance. The better performance of the two-stage method can be attributed to the fact that this method can independently determine the charge sequence and evaluate the outcome without the added complexity of determining the machine assignments at the same time, which is in line with the findings of Ishigaki and Takaki (2017).

5.3 Solution methodology benchmark

Benchmark for the problem under study
In this section, we compare the proposed two-stage local search heuristic with other solution methods to evaluate the computational performance and yielded solution quality of the proposed algorithm. In the following, we analyse the performance of the following methodologies:

- Mixed-integer programming (MIP): This approach solves the defined model (1)-(21) based on mixed-integer programming, using the commercial optimisation solver Gurobi. This method does not take an initial solution explicitly into account but relies on the multiple heuristics induced by Gurobi to obtain an upper bound (early) in the optimisation search process. A time limit is imposed of 3600 seconds.
- GA of Long et al. (2018): We benchmark the proposed method to the method devised for the problem that is most similar in terms of objectives and problem structure. Their method generates the initial population of schedules completely random and makes use of an elitist genetic algorithm in combination with a quality improvement method. The GA makes use of tournament selection, SB2OX as crossover operator, and the *shift* method as a mutation operator. The method is implemented with identical parametrisation of methodology (e.g. stop criterion, number of local search moves) as described in Long et al. (2018). It should be noted that the decoding heuristic and quality improvement method by Long et al. (2018) are not directly applicable to our problem definition. Given that the GA also results in a charge sequence as in our method, we only compare the underlying search mechanism of the proposed GA algorithm to find new solutions with our local search method. Our own decoding heuristic and repair mechanism are implemented to devise production schedules, respecting the specifics of the problem under study. Experiments showed that the algorithm converged after 400 generations. This number was used on all instances and for all seeds.
- Proposed method: This method corresponds to the proposed two-stage local search heuristic (cf. Section 4).

Table 8 shows the performance of the different tested methods, displaying the solution quality (Z), the run time (CPU) and the deviation from the best performing method ($\%Gap$). The displayed results represent averages over the 20 test instances. The results in Table 8 reveal that the proposed two-stage heuristic is able outperform the alternative solution methods. The mixed-integer programming method needs a far larger CPU time and is not able to retrieve any integer solution in a time span of 3600 seconds. The GA of Long et al. (2018) takes significantly longer times to yield significantly worse solutions. The average gap to the solution obtained by the proposed method is 30.77%. This is achieved in CPU times that are on average ten times longer. The better performance of the proposed method can be attributed to the problem-specific initialisation method and *cast* operator. These methods allow a targeted search compared to the GA that utilises the *shift* operator combined with a random initialisation. The bad performance of MIP can be explained by the high number of binary variables for the considered instances and large instance sizes, making such an approach unacceptable for practical use. Note, additional experiments that initialise the alternative solution methods with the proposed initialisation heuristic, revealed that the quality of the solutions realised via these methods improves but these improvements can be attributed only to the better initialisation method as yielded solutions are identical or very close to the initial solutions generated via the proposed initialisation method. Little further improvements have been realised via the optimisation search conducted by these

methods. Consequently, both alternative methods are not able to outperform the proposed algorithm.

Method	Z	CPU	$\%Gap$
Mixed-integer programming (3600s)	-	3600	-
Method of Long et al. (2018)	3054.19 (483.54)	1052.37	30.77%
Proposed method	2114.48 (141.68)	96.88	0.00%

Table 8: Computational benchmark with other methodologies for the problem under study

Benchmark for the problem described in Long et al. (2018)
As demonstrated in the previous paragraph, the proposed heuristic outperforms the GA of Long et al. (2018) for the problem under study. However, in order to make a fair assessment of the performance of both methods, we compare these methods relative to the problem studied in Long et al. (2018). For this problem, we generated five instances of different sizes using the test design described in Long et al. (2018). The generated instance size is represented via S1*S2*S3 (cf. second column of Table 9), where S1 refers to the number of operations, S2 to the number of casts in each CC, and S3 to the number of charges in each cast. Note that to solve this problem, the proposed method does not require a machine assignment in the solution encoding thanks to the simpler structure of the benchmarked problem (e.g. absence of steel wagons, fixed processing times). More precisely, the problem considered by Long et al. (2018) considers a standard SCC-scheduling problem with stage skipping and adjustable processing times. Similar to our problem, the cast schedule is already planned by a master schedule. The objective function considers the makespan, waiting times and the deviation from adjustable processing times. In their comparison with other methods, adjustable processing times are not considered. Therefore, we also did not implement the deviation of processing times in the objective, such that the objective function only consists of the makespan and waiting times, similar to our own problem. For more details concerning the instance generation, the mathematical model, and the solution method, we refer to Long et al. (2018). The results can be found in Table 9, which reports the objective value (Z') yielded for the problem of Long et al. (2018) and the run time (CPU) for every instance. Note that five runs have been conducted for each instance to account for the impact of randomness in the procedures. Table 9 reveals that also for the problem studied by Long et al. (2018), the proposed local search heuristic yields a better solution quality for all instances, except for the smallest instance. This is partly explained by the superior performance of the proposed initialisation method, which performs significantly better for larger instance sizes.

		Method of Long et al. (2018)		**Proposed method**	
	Size	Z'	CPU	Z'	CPU
1	3*2*6	**7431.49 (270.21)**	352.81 (35.58)	8539.21 (0.0)	1.7 (0.53)
2	4*3*6	15358.39 (1067.79)	951.16 (13.3)	**13124.02 (0.0)**	2.46 (0.33)
3	5*3*6	24859.18 (1925.96)	1350.9 (11.12)	**12570.52 (428.67)**	4.56 (1.4)
4	6*4*6	37617.78 (1335.63)	3061.64 (31.23)	**19222.46 (506.99)**	7.35 (1.33)
5	7*4*7	57988.88 (3771.28)	7963.8 (5331.72)	**22590.31 (136.74)**	13.35 (2.35)

Table 9: Benchmark based on problem definition of Long et al. (2018)

6 Conclusions

In this paper, we propose a local search heuristic to solve a real-life steelmaking continuous casting problem. The proposed procedure is able to find high-quality solutions in a reasonable time span in comparison with alternative solution methodologies. The contribution of this

paper is twofold. First, we studied a real-life variant of the scheduling problem, extending the characteristics studied in literature to embed a higher degree of realism. More precisely, we consider different complicating dual-resource transportation and blocking constraints and routing and sequencing flexibility to process charges of different steel grades, turning the problem into a flexible job-shop scheduling problem. Discussions with practitioners revealed that in future research this problem definition can be further extended via the consideration of alternative process plans to process the charges into steel to improve productivity as charges of specific steel grades can often be produced in different ways. Second, we propose a two-stage local search heuristic operating on a dual-vector encoding that is decoded into a production schedule using a recursive algorithm and schedule-repair improvement step. The heuristic is initialised taking problem-specific information relative to the input cast schedule and order of charges within casts into account. The neighbourhood of this initial solution is investigated via a novel cast operator that changes the charge sequence and a machine-assignment operator. Results demonstrate that these heuristic design choices provide better initial solutions and faster convergence to high-quality solutions in acceptable run times for real-life use.

References

Agnetis, A., Murgia, G., and Sbrilli, S. (2014). A job shop scheduling problem with human operators in handicraft production. *International Journal of Production Research*, 52(13):3820–3831.

Allahverdi, A. (2016). A survey of scheduling problems with no-wait in process. *European Journal of Operational Research*, 255(3):665—-686.

Andrade-Pineda, J. L., Canca, D., Gonzalez-R, P. L., and Calle, M. (2019). Scheduling a dual-resource flexible job shop with makespan and due date-related criteria. *Annals of Operations Research*, pages 1–31.

Aschauer, A., Roetzer, F., Steinboeck, A., and Kugi, A. (2017). An efficient algorithm for scheduling a flexible job shop with blocking and no-wait constraints. *IFAC-PapersOnLine*, 50(1):12490–12495.

Aschauer, A., Roetzer, F., Steinboeck, A., and Kugi, A. (2020). Efficient scheduling of a stochastic no-wait job shop with controllable processing times. *Expert Systems with Applications*, 162:113879.

Atighehchian, A., Bijari, M., and Tarkesh, H. (2009). A novel hybrid algorithm for scheduling steel-making continuous casting production. *Computers & Operations Research*, 36(8):2450–2461.

Bellabdaoui, A. and Teghem, J. (2006). A mixed-integer linear programming model for the continuous casting planning. *International Journal of Production Economics*, 104(2):260–270.

Chaudhry, I. A. and Khan, A. A. (2016). A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23(3):551–591.

Demir, Y. and İşleyen, S. K. (2013). Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematical Modelling*, 37(3):977–988.

Dhiflaoui, M., Nouri, H., and Driss, O. (2018). Dual-resource constraints in classical and flexible job shop problems: A state-of-the-art review. *Procedia Computer Science*, 126:1507–1515.

Fattahi, P., Mehrabad, M. S., and Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of intelligent manufacturing*, 18(3):331–342.

Gao, L. and Pan, Q.-K. (2016). A shuffled multi-swarm micro-migrating birds optimizer for a multi-resource-constrained flexible job shop scheduling problem. *Information Sciences*, 372:655–676.

Gröflin, H., Pham, D., and Bürgy, R. (2011). The flexible blocking job shop with transfer and set-up times. *Combinatorial Optimisation*, 22(2):121–144.

Hansmann, R., Rieger, T., and Zimmermann, U. (2014). Flexible job shop scheduling with blockages. *Mathematical Methods of Operations Research*, 79(2):135–161.

Ishigaki, A. and Takaki, S. (2017). Iterated local search algorithm for flexible job shop scheduling. In *2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 947–952. IEEE.

Long, J., Zheng, Z., Gao, X., and Pardalos, P. M. (2018). Scheduling a realistic hybrid flow shop with stage skipping and adjustable processing time in steel plants. *Applied Soft Computing*, 64:536–549.

Lu, C., Li, X., and Gao, L. e. a. (2017). An effective multi-objective discrete virus optimization algorithm for flexible job-shop scheduling problem with controllable processing times. *Computers & Industrial Engineering*, 104:156–174.

Mao, K., Pan, Q.-k., Pang, X., and Chai, T. (2014). A novel lagrangian relaxation approach for a hybrid flowshop scheduling problem in the steelmaking-continuous casting process. *European Journal of Operational Research*, 236(1):51–60.

Mati, Y. and Lahlou C., Dauzere-Peres, S. (2011). Modelling and solving a practical flexible job-shop scheduling problem with blocking constraint. *International Journal of Production Research*, 49(8):2169–2182.

Özgüven, C., Özbakir, L., and Yavuz, Y. (2010). Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 34(6):1539–1548.

Pan, Q.-K. (2016). An effective co-evolutionary artificial bee colony algorithm for steelmaking-continuous casting scheduling. *European Journal of Operational Research*, 250(3):702–714.

Pan, Q.-K., Wang, L., Mao, K., Zhao, J.-H., and Zhang, M. (2012). An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process. *IEEE Transactions on Automation Science and Engineering*, 10(2):307–322.

Peng, K., Pan, Q.-K., Gao, L., Zhang, B., and Pang, X. (2018). An improved artificial bee colony algorithm for real-world hybrid flowshop rescheduling in steelmaking-refining-continuous casting process. *Computers & Industrial Engineering*, 122:235–250.

Pezzella, F., Morganti, G., and Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202–3212.

Rooyani, D. and Defersha, F. (2019). An efficient two-stage genetic algorithm for flexible job-shop scheduling. *IFAC-PapersOnLine*, 52(13):2519–2524.

Saidi-Mehrabad, M. and Fattahi, P. (2007). Flexible job shop scheduling with tabu search algorithms. *The international journal of Advanced Manufacturing technology*, 32(5-6):563–570.

Shi, X., Long, W., Yan-yan Li, Y., and Deng, D. (2018). Different performances of different intelligent algorithms for solving fjsp: A perspective of structure. *Computational Intelligence and Neuroscience*, page 4617816.

Sierra, M. R., Mencìa, C., and Varela, R. (2015). New schedule generation schemes for the job-shop problem with operators. *Journal of Intelligent Manufacturing*, 26:511–525.

Sun, L. and Yu, S. (2015). Scheduling a real-world hybrid flow shop with variable processing times using lagrangian relaxation. *The International Journal of Advanced Manufacturing Technology*, 78(9-12):1961–1970.

Tang, L., Liu, J., Rong, A., and Yang, Z. (2000). A mathematical programming model for scheduling steelmaking-continuous casting production. *European Journal of Operational Research*, 120(2):423–435.

Tang, L., Liu, J., Rong, A., and Yang, Z. (2001). A review of planning and scheduling systems and methods for integrated steel production. *European Journal of Operational Research*, 133(1):1–20.

Tang, L., Luh, P. B., Liu, J., and Fang, L. (2002). Steel-making process scheduling using lagrangian relaxation. *International Journal of Production Research*, 40(1):55–70.

Vaughan, D., Jacobson, S., Hall, S., and McLay, L. (2005). Simultaneous generalized hill-climbing algorithms for addressing sets of discrete optimization problems. *INFORMS Journal on Computing*, 17(4):438–450.

Verspurten, S. and Henrion, P. (2019). Staalfabriek staalbereiding. Internal document ArcelorMittal.

Xie, J., Gao, L., Peng, K., Li, X., and Li, H. (2019). Review on flexible job shop scheduling. *IET Collaborative Intelligent Manufacturing*, 1(3):67–77.

Xiong, H., Shi, S., Ren, D., and Hu, J. (2022). A survey of job shop scheduling problem: The types and models. *Computers & Operations Research*, 142:105731.

Xu, Z., Zheng, Z., and Gao, X. (2020). Energy-efficient steelmaking-continuous casting scheduling problem with temperature constraints and its solution using a multi-objective hybrid genetic algorithm with local search. *Applied Soft Computing*, 95:106554.

Xuan, H. and Tang, L. (2007). Scheduling a hybrid flowshop with batch production at the last stage. *Computers & operations research*, 34(9):2718–2733.

Zhang, G., Gao, L., and Shi, Y. (2011). An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38(4):3563–3573.

Zhang, L., Tang, Q., Wu, Z., and Wang, F. (2017). Mathematical modeling and evolutionary generation of rule sets for energy-efficient flexible job shops. *Energy*, 138:210–227.

Zheng, F. and Wang, Z. (2019). Bi-objective flexible job shop scheduling with operation overlapping costs. *IFAC-PapersOnLine*, 52(13):893–898.

A Illustration of the solution of a real-life instance

In this appendix, we depict in Figure 11 the resulting resource schedule associated with the solution of instance 1, yielded by the proposed algorithm. The schedule depicts the start and completion times of operations on the different resources for a limited time span. The instance considers 19 resources and 841 operations in total. The operations are depicted by blank rectangles. The arced rectangles indicate the resource has been blocked due to a concurrent operation on a different machine on the same production line. For illustrative purposes, only a subset of the operations are depicted.

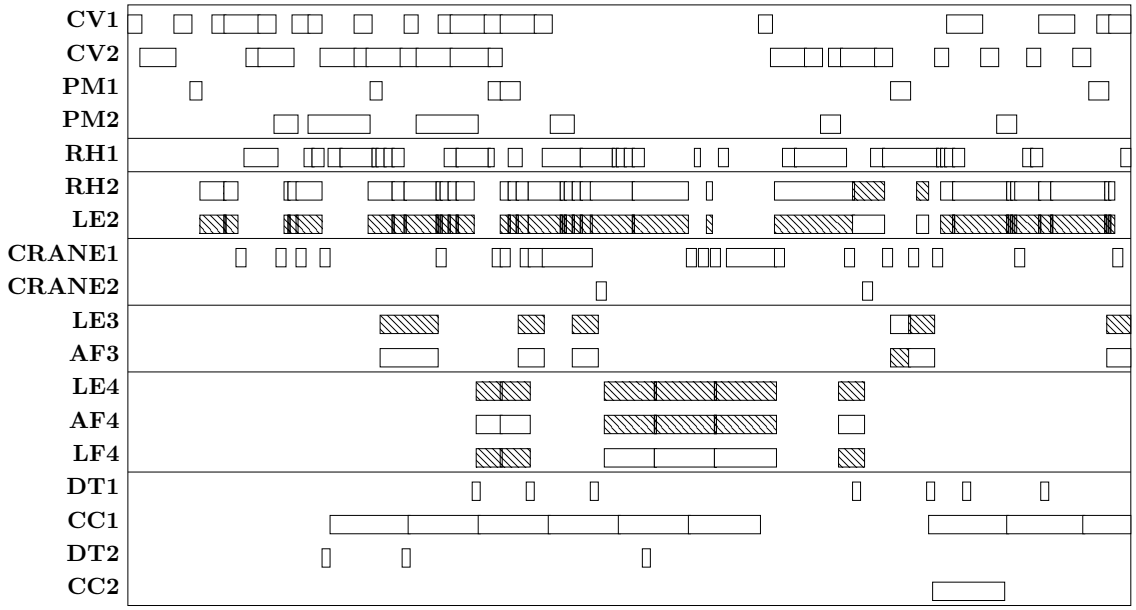

Figure 11: Example resource schedule real-life schedule for instance 1

(**Legend.** CV1-2: converter 1-2; PM1-2: metallurgy pan machines 1-2; RH1-2: Ruhrstahl Heraeus degassers 1-2; LE1-3: mettalurgy machines 1-3; AF1-2: deslagger 1-2; LF4: ladle furnace; ARTI: arificial machine; CRANE1-2: cranes 1-2; DT1-2: revolving tower 1-2; CC1-2: continuous caster 1-2.)

B Detailed information related to the data set instances

Table 10 provides detailed information for each instance related to the operations in every stage of the production process and transport operations.

Instance	Convertor stage			Refining stage			Transport operations			Casting stage		
	#operations	Avg. processing time (min.)	Range processing times (min.)	#operations	Avg. processing time (min.)	Range processing times (min.)	#operations	Avg. processing time (min.)	Range processing times (min.)	#operations	Avg. processing time (min.)	Range processing times (min.)
1	212	9.42	[5.0, 17.0]	288	8.33	[2.0, 30.0]	235	2.67	[0.0, 4.0]	106	13.67	[3.0, 35.0]
2	236	9.70	[5.0, 17.0]	326	8.99	[2.0, 30.0]	267	2.67	[0.0, 4.0]	118	13.67	[3.0, 35.0]
3	240	9.54	[5.0, 17.0]	336	8.44	[1.0, 30.0]	276	2.67	[0.0, 4.0]	122	13.67	[3.0, 35.0]
4	228	9.65	[5.0, 17.0]	315	8.85	[2.0, 30.0]	258	2.67	[0.0, 4.0]	114	13.67	[3.0, 35.0]
5	240	9.67	[5.0, 17.0]	327	8.95	[2.0, 30.0]	267	2.67	[0.0, 4.0]	120	13.67	[3.0, 35.0]
6	208	9.43	[5.0, 17.0]	286	7.96	[2.0, 30.0]	234	2.67	[0.0, 4.0]	104	13.67	[3.0, 35.0]
7	240	9.57	[5.0, 17.0]	339	8.60	[2.0, 30.0]	279	2.67	[0.0, 4.0]	120	13.67	[3.0, 35.0]
8	208	9.72	[5.0, 17.0]	272	8.84	[1.0, 30.0]	220	2.67	[0.0, 4.0]	105	13.80	[3.0, 35.0]
9	236	9.50	[5.0, 17.0]	313	8.16	[2.0, 30.0]	254	2.67	[0.0, 4.0]	118	13.67	[3.0, 35.0]
10	232	9.58	[5.0, 17.0]	297	8.42	[1.0, 30.0]	239	2.67	[0.0, 4.0]	116	13.67	[3.0, 35.0]
11	232	9.49	[5.0, 17.0]	307	8.26	[1.0, 30.0]	249	2.67	[0.0, 4.0]	117	13.79	[3.0, 35.0]
12	224	9.63	[5.0, 17.0]	289	8.99	[1.0, 30.0]	233	2.67	[0.0, 4.0]	114	13.67	[3.0, 35.0]
13	244	9.56	[5.0, 17.0]	324	8.81	[1.0, 30.0]	263	2.67	[0.0, 4.0]	122	13.67	[3.0, 35.0]
14	256	9.48	[5.0, 17.0]	342	8.30	[2.0, 22.0]	278	2.67	[0.0, 4.0]	130	13.67	[3.0, 35.0]
15	220	9.49	[5.0, 17.0]	287	8.43	[2.0, 27.0]	232	2.67	[0.0, 4.0]	111	13.80	[3.0, 35.0]
16	236	9.74	[5.0, 17.0]	321	8.75	[2.0, 27.0]	262	2.67	[0.0, 4.0]	120	13.67	[3.0, 35.0]
17	240	9.56	[5.0, 17.0]	301	8.54	[2.0, 23.0]	241	2.67	[0.0, 4.0]	122	13.67	[3.0, 35.0]
18	244	9.66	[5.0, 17.0]	304	8.87	[2.0, 23.0]	243	2.67	[0.0, 4.0]	123	13.78	[3.0, 35.0]
19	248	9.54	[5.0, 17.0]	346	8.70	[2.0, 25.0]	284	2.67	[0.0, 4.0]	127	13.78	[3.0, 35.0]
20	236	9.71	[5.0, 17.0]	311	8.68	[2.0, 23.0]	252	2.67	[0.0, 4.0]	121	13.78	[3.0, 35.0]

Table 10: Detailed information on considered production and transport operations in test instances