INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 00 (2018) 1–34
DOI: 10.1111/itor.12595

Modeling and solving the steelmaking and casting scheduling problem

Davide Armellini[a], Paolo Borzone[a], Sara Ceschia[b], Luca Di Gaspero[b,*] and Andrea Schaerf[b]

[a]*Danieli Automation S.p.A., via Bonaldo Stringher 4, I-33042 Buttrio (UD), Italy/piazza Borgo Pila 39, I-16129 Genoa, Italy*

[b]*DPIA, University of Udine, via delle Scienze 206, I-33100 Udine, Italy*

E-mail: d.armellini@dca.it [Armellini]; p.borzone@dca.it [Borzone]; sara.ceschia@uniud.it [Ceschia]; luca.digaspero@uniud.it [Di Gaspero]; schaerf@uniud.it [Schaerf]

Received 17 November 2017; received in revised form 29 August 2018; accepted 29 August 2018

Abstract

We propose a general model for the problem of planning and scheduling steelmaking and casting activities obtained by combining common features and constraints of the operations from a real plant and the literature. For tackling the problem, we develop a simulated annealing approach based on a solution space made of job permutations, which uses as submodule a chronological constructive procedure that assigns processing times and resources to jobs. Our technique, properly tuned in a statistically principled way, is able to find good solutions for a large range of different settings and horizons. In addition, it outperforms both a greedy procedure and a constraint-based solver developed for comparison purposes on almost all instances. Finally, we have collected several real-world instances that we make available on the web along with the solution validator and our best results.

Keywords: steelmaking; continuous casting; simulated annealing; hybrid flow shop

1. Introduction

The steelmaking process is one of the most complex industrial production operations due to the presence of many technological (physical, chemical, mechanical, etc.) and business constraints. The operation of a steelmaking plant is quite costly, therefore the throughput maximization by means of an optimized scheduling is of crucial importance to ensure productivity and competitiveness.

The steelmaking process comprises four stages outlined in Fig. 1. At first, the iron scrap is melted in an *electric arc furnace* (EAF), then the liquid metal is poured in a *ladle* that will be used to contain

*Corresponding author.

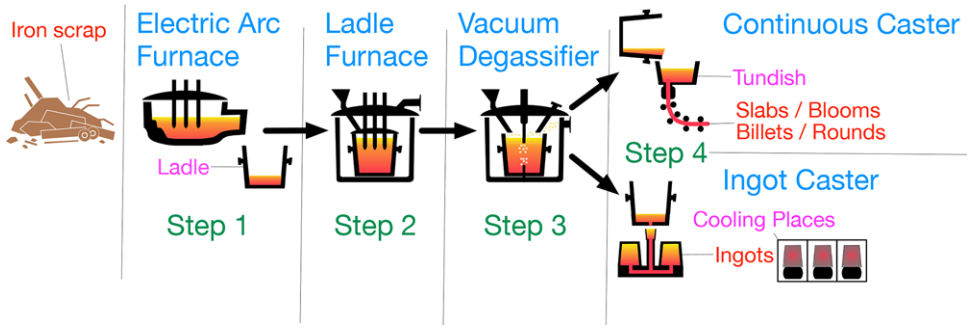


Fig. 1. A schematic illustration of the steelmaking and casting production process. [Colour figure can be viewed at wileyonlinelibrary.com]

the steel in all the following processing steps. The next step is the *ladle furnace* (LF) in which some additives are added for obtaining the desired chemical composition of the product. Afterward, the metal undergoes the *vacuum degasification* (VD) process, which reduces hydrogen and nitrogen gases dissolved in the liquid steel, improving final products quality.

The last production step, called *casting*, consists in the controlled solidification of the liquid steel in order to obtain the desired shape of semifinished product and might differ based on the type of product required. Indeed, depending on the process used for the solidification, usually *continuous casting* (CC) machines (for production of slabs, blooms, and billets) or *ingot casting* (IC) machines (for ingots production) are employed. These semifinished products, then, could be subject to further processing (e.g., hot rolling), which might be planned and scheduled independently from the casting process.

The main physical constraint of the steelmaking process is the practical impossibility to buffer jobs between different processing steps because of the cooling of the liquid metal in case of waiting. As a consequence, the jobs should be scheduled in a *just-in-time* fashion.

A similar version of the problem considered in this paper was presented by Fanti et al. (2016) who proposed an integrated system consisting of a database, an optimization engine, a simulation module, and a user interface for the scheduling problem of a specific plant. The optimization engine models the scheduling as a hybrid flow shop using a mixed integer linear programming (MILP) formulation.

With respect to Fanti et al. (2016), we have made several modifications (see Section 3) to better represent the real-world specifications, including the presence of the *border data*, coming from the previous schedule. In addition, in order to obtain a model that captures the essential features of the problem, we have simplified the problem statement, so as to remove low level details too specific for the situation at hand. On the other side, we have generalized the problem including features specified by Danieli Automation[1] that were not included in the original model of Fanti et al. (2016), but are common across different plants.

For this problem, we propose a metaheuristic method based on simulated annealing (SA) that operates at the level of job sequencing, so that the solution is represented as a permutation of the

[1]Danieli Automation is part of the Danieli Group, one of the world-leading suppliers of equipment and plants to the metal industry.

jobs. The actual schedule is obtained by a deterministic subprocedure that computes start times and resources of all operations, one job at the time, following the order on the current permutation. The neighborhood structure is based on the movement of either a single job or a group of *setup-compatible* ones.

We collected a dataset of real-world instances coming from the same plant of Fanti et al. (2016), but with a different and larger set of operating conditions. Instances and results are available at https://opthub.uniud.it, and hopefully could be used as a *benchmark* for future researches and comparisons, given also the lack of available ones in the literature. To this aim, the website includes an online solution validator that provides against possible misinterpretations of the formulation.

The experimental results show that for many instances, we can find a solution that schedules a number of jobs relatively close to the upper bound, and without violating any important operational conditions. Conversely, both a greedy procedure and a constraint-based solver, developed for comparison, have not being able to reach similar results.

2. Problem formulation

In this section we describe the formulation of the steelmaking and casting scheduling problem (SMCP), which represents the essential part of the real-world problem under consideration. The formal description of the constraint-based model is provided in Appendix.

2.1. Problem structure

The problem structure can be described through the following basic notions.

Machine: There are five different *machine types*, corresponding to the four main stages of the process: EAF for melting, LF for refining, VD for degassing, and CC or IC for continuous or ingots casting. Each machine, except CCs and ICs, belongs to a *line* that identifies the habitual process flow of jobs through the plant. In addition, machines have a *processing time*, a *max stretch time* that is the maximum time a machine can slow down its process before the next stage, and possibly a *production standstill* time window (`MachineStops`) due to maintenance services or temporary machine breakdowns (Tang et al., 2014).

Job: *Jobs* are grouped by *steel grades*, depending on the required chemical composition and the type of process needed (i.e., continuous or IC); moreover a job specifies further processing details such as the *section width* and its possible *incompatibility* with some of the *casters* (e.g., because the job section cannot be processed by the machine). A machine can process only one job at a time, and after the processing is completed the job is moved forward to a different machine of the next stage. The movement between two machines takes up a constant *transportation time*, depending on the distance between the two machines (`Distances`). Finally, because of the independent planning of the postprocessing activities, a job can have an *appointment*, that is a time window for its completion time.

Ladle: The molten steel is poured in a container, called *ladle*, which is moved through the plant till the end of the process. A job can wait in the ladle for the next production step for a maximum fixed amount of time (`MaxWaitingTimeInLadle`). At the end of the process, the ladle has

to be cleaned and then it goes back to the start position; this process requires a constant time (`LadleCleaningAndReturnTime`).

Tundish: The tundish is an intermediate container that is needed at CCs, where the content of the ladle is poured for the CC process (see Fig. 1). When two or more jobs of the same steel grade are sequentially processed the same tundish can be reused. When the steel grade changes, a new *tundish* has to be properly placed on the CC plance, thus leading to a setup time (*fly-tundish* setup). The tundish needs to be previously heated in the designated position, and this operation might take more time than a single casting operation. Therefore, the fly-tundish operation is not always possible, and the CC might need to be rearmed (see the CC setup modes below). The minimum number of consecutive jobs between two fly-tundish operations is a fixed value (`FlyTundishFrequency`).

Pollutant: The pollutant is a chemical element that *pollutes* the ladle and imposes specific sequence constraints for ensuring the production quality. Indeed, the residuals of a pollutant in the ladle make it impossible to reuse straightaway the same ladle for producing jobs of some critical steel grade; the ladle must first undergo a *neutral* production that removes the residuals thus *cleaning* the ladle. Therefore, each steel grade has associated an information about the presence of a given pollutant (`SteelGradePollutionAction`) and prescribes its cleaning requirements for each pollutant (`SteelGradePollutionRequirement`).

CC Setup: For all machines, but the CCs, the *setup times* are virtually absorbed by the *transportation times,* therefore they can be considered as being zero setup. For the CCs, instead, there are a number of setup operations that have an impact on the processing time and which depend on the sequence of the jobs processed on that machine. Namely, we distinguish among the following four setup modes depending on the sequence of jobs. The setup modes are ordered from the least to the most time-consuming.

- **None:** No setup is needed, this happens when the machine is already running and the two consecutive jobs are of the same steel grade and the same section.
- **FlyTundish:** The change of the tundish is performed without major interruptions of the pouring process, resulting in a quite short setup time (`ChangeTundishTime`). This is possible when two consecutive productions are compatible and a specific frequency is respected (allowing for the heating of a new tundish, if available).
- **Rearm:** This kind of setup occurs when the CC machine remains idle for more than a maximum waiting time (`ContinuousCasterMaxWaitingTime`) or it undergoes a production stand-still. From an operational point of view, it consists in the plug of a dummy bar at the bottom of each mold present in the machine, in order to support the initial solidification of the steel. Each CC has its own specific *rearm time*, that is the time necessary to perform a rearm setup.
- **ChangeSection:** A size change is needed between two jobs with different sections. This operation typically requires human intervention; in our model, the time needed is assumed constant for the machine (`ChangeSectionTime`). After the change, the machine must also undergo a rearm, therefore this is the slowest setup operation as the two setup times are added up.

IC Cooling Places: IC machines have no setup, but their products need to be placed in dedicated *cooling places* for a fixed amount of time, in order to be moved later to subsequent steps. Each cooling place is composed of a basement and several ingot molds that have been previously prepared by operators. After the casting, the equipment in the cooling place cannot be moved

for a fixed time before to proceed to subsequent steps. The number of cooling places for each IC machine is fixed, and the cooling time might be much longer than the casting time. For this reason, in some cases it might be not possible to process consecutive IC jobs, due to the absence of available cooling places.

In essence, the scheduling problem is a complex variant of the *hybrid flowshop* problem (for a review, see, e.g., Ruiz and Vázquez-Rodríguez, 2010) with sequence-dependent setup times and heterogeneous processing times.

In a real-world situation, the plant runs for 24 hours a day, thus we have to schedule the production for the next time-horizon (typically a single day, or a weekend) by considering the *border data*, that is, the last jobs scheduled at the end on the previous scheduling stage. In particular, for each machine, we register the time when it is available (`BorderMachineAvailableTime`), the section of last job (`BorderSection`), and its steel grade (`BorderSteelGrade`). In addition, we consider also the availability time of the ladles and their status with respect to each ladle pollutant (`BorderLadleAvailableTime` and `BorderLadlePollutionStatus`).

Figure 2 shows a file that contains a small exemplary instance, written in the file format that adheres to the MiniZinc data file format (Nethercote et al., 2007). Note that jobs and machines are identified by an index, which starts from 0. In addition, in order to keep arrays of a fixed length, the value `-1` is used to represent meaningless data; for example, only EAF, LF, and VD are associated to lines, therefore the line of a CC or an IC machine is meaningless. Further explanations about the data format are provided in Fig. 2 in form of comments.

Given that we have to schedule the production for the next time-horizon, the goal is also to select the jobs from a large pool of orders that have to be processed (considerably larger than those that may be processed during the horizon), assign them to machines and plan the sequencing and timing (start and end time). Therefore, the real-world problem in its essence is a planning and scheduling problem (Li et al., 2012). As a consequence, differently from other models in the literature (including those of Fanti et al., 2016), instead of using the makespan as the objective function for the problem, it is more meaningful to use a *throughput* measure that considers (and maximizes) the number of jobs whose process starts within the given time-horizon.

The upper bound on the schedulable jobs is computed based on the available working time of the furnaces, which are normally one of the bottlenecks of the process. In detail, we multiply the horizon by the number of lines, and we subtract the total standstill time of the furnaces. This value is divided by the processing time of the furnaces (which is normally the same for all of them). From the result, which can be a fractional value, we take the *ceiling*, rather than the *floor*, given that for a job is considered as scheduled if it starts within the horizon. Obviously, if the number of available jobs is less than the upper bound, the latter is set equal to that number.

Figure 3 shows a visual representation of a solution of the instance of Fig. 2 in a Gantt-like diagram. The figure represents the actual processing times and setup modes for the jobs in the sequence represented at the bottom of the figure (Jobs order). The numbered bars represent the actual processing times and the resource usage (ladles at the top of the figure, machines in the lower part) of each job and have a color related to the steel grade. Dark bars represent the border data and/or the temporary unavailability of a resource (see ladles 3 and 4). Besides this information, the figure shows also the CC setup modes: CS for Change Size, R for Rearm, whereas the very short FlyTundish mode is represented by a short unlabeled bar (magenta colored, in the figure on the

```
Machines = 5;
Lines = 1;
Jobs = 10;
SteelGrades = 3;
Ladles = 5;
Polluters = 2;
Horizon = 600;

MachineType = [0, 1, 2, 3, 4]; % EAF = 0, LF = 1, VD = 2, CC = 3, IC = 4 (one machine per type in this file)
MachineLine = [0, 0, 0, -1, -1]; % CC and IC have no line (value −1)
IngotCoolingPlaces = [-1, -1, -1, -1, 3]; % only IC machines have cooling places
ProcessingTime = [54, 40, 55, 70, 75];
MaxStretchTime = [5, 60, 15, 15, 15]; % maximum extension on the processing time
RearmTime = [-1, -1, -1, 90, -1]; % only CC need to rearm and change section
ChangeSectionTime = [-1, -1, -1, 60, -1];

LadlePouringTime = 5;
LadleCleaningAndReturnTime = 30;
ContinuousCasterMaxWaitingTime = 15;
MaxWaitingTimeInLadle = 15;
ChangeTundishTime = 15;
FlyTundishFrequency = 2;
Distances = [|0, 1, 2 % set of triples: [machine1, machine2, distance (in minutes)]
             |1, 2, 2 % only meaningful distances are stored
             |2, 3, 8
             |2, 4, 11|];
JobType = [0, 0, 1, 0, 0, 1, 0, 0, 0, 1];  % 0 = CC, 1 = IC
CoolingTimes = [-1, -1, 200, -1, -1, 300, -1, -1, -1, 200]; % meaningful only for IC jobs
JobAppointments = [|0, 200, 300     % set of triples: [job, start−time, end−time]
                   |4, 0, 480|];
JobSection = [200, 600, -1, 400, 200, -1, 200, 200, 400, -1];
SteelGrade = [0, 0, 2, 1, 1, 2, 1, 1, 1, 2];
SteelGradePollutionAction = [|1, 0, 0       % 1 = pollutes ladle, 0 = no effect (cleans)
                             |0, 0, 1|];
SteelGradePollutionRequirement = [|1, 0, 1 % 0 = requires clean, 1 = no request
                                  |0, 1, 1|];
MachineStops = [|0, 430, 480|]; % set of triples: [machine, start−time, end−time]
JobCasterIncompatibility = [];  % set of pairs [job, caster]
FlyTundishIncompatibility = [|0, 2|]; % set of pairs [steel−grade−before, steel−grade−after]

BorderMachineAvailableTime = [0, 0, 0, -100, 0];% for CC, stores negative values (to start setup)
BorderSection = [-1, -1, -1, 500, -1];  % meaningful only for CC machines
BorderSteelGrade = [-1, -1, -1, 0, -1];
BorderLadleAvailableTime = [0, 0, 0, 16, 70];
BorderLadle
PollutionStatus = [|0, 0, 0, 0, 0
                   |0, 0, 0, 0, 0|];
```

Fig. 2. Example of input file in MiniZinc format. [Colour figure can be viewed at wileyonlinelibrary.com]

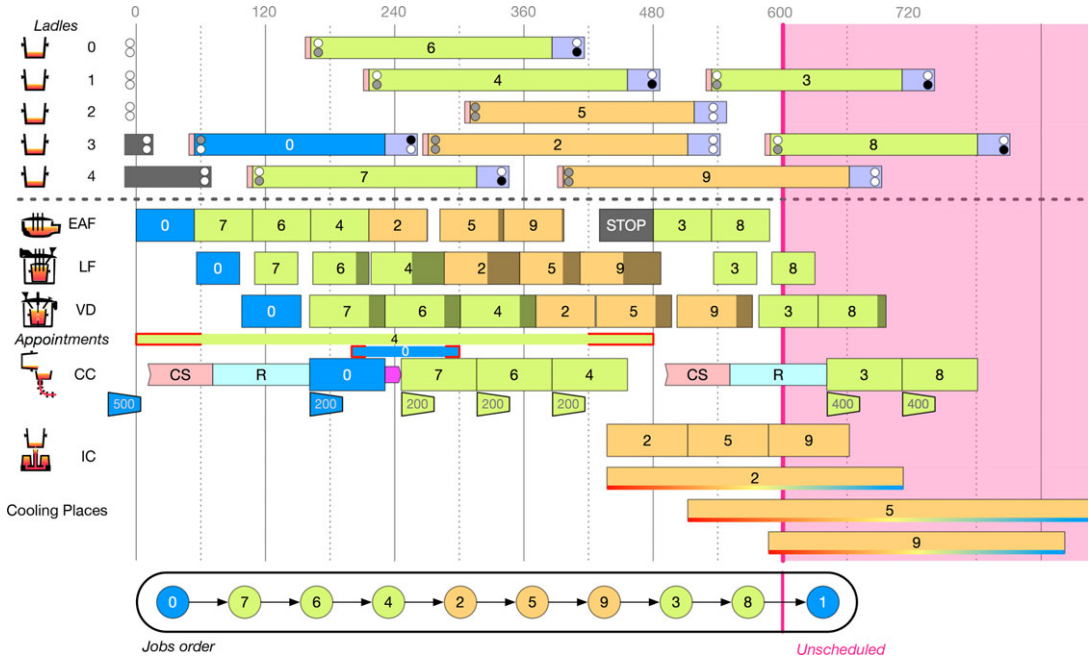


Fig. 3. Visualization of a solution of the example in Fig. 2. [Colour figure can be viewed at wileyonlinelibrary.com]

website). In order to allow for double checking the setup modes, the casting section is reported just below each casting process bar.

The information about pollution is represented by the two small circles, once for each pollutant, in the ladles bar. The leftmost ones show the requirements of the processed job (white for clean and gray for do not care), whereas the rightmost report the status of the ladle at the end of the production (white for clean and black for dirty). The color correspondence between subsequent jobs allows to check the fulfillment of the ladle cleaning constraints (see Section 2.2).

In the machines scheduling part, the darker portion of each job bar represents the *stretch* time, that is, the amount of time the job will be idle in a given machine because of the no-buffering requirements. Just above the casters, the intervals representing the appointments are reported (see jobs 0 and 4), which allow to check whether these constraints are fulfilled.

From the figure it is possible to note that job number 1 has not been scheduled, and this corresponds to a penalty since the theoretical upper bound on the number of schedulable jobs within the time horizon is 10. The reason why this job cannot fit in the current schedule is that it has a section value of 600 mm, which would require a change section operation, such that the job cannot be started within the given horizon without unacceptable waiting times.

2.2. *Constraints and objectives*

We first introduce the constraints that must be always satisfied (*hard constraints*) and then we move to the constraints (*soft constraints* or *objectives*) that contribute to the objective function of the problem to minimize.

The hard constraints are the following:

LadleCleaning: A ladle that is dirty with respect to a given pollutant is used immediately by a job that would require a clean ladle.

LateLadle: A ladle does not return on time to be used for the next job.

JobWaitingTime: A job waits in the ladle for a time longer than the maximum allowed.

The soft constraints are the following.

Appointments: A job misses an appointment. This can happen in two possible ways:

- the end time of the job is outside the appointment window; in this case, we add a penalty proportional to its earliness/tardiness expressed in minutes;
- the job is unscheduled; in this case, we add a fixed time penalty equal to the horizon in minutes, not depending on the position and the width of the window.

Unscheduled: The number of unscheduled jobs is a penalty. In order to have a penalty measure independent on the number of available jobs, we penalize the difference between the upper bound of schedulable jobs and the actual number of scheduled ones.

FurnaceOvertime: All jobs that start before the horizon line are considered as scheduled, and do not contribute to the Unscheduled objective. However, the number of minutes that the last scheduled jobs uses after the horizon is also considered as a penalty.

Note that minimizing the number of unscheduled jobs is equivalent to maximize the total throughput. In addition, the FurnaceOvertime is a lower level objective with respect to Unscheduled. Indeed, a solution that has more scheduled jobs is preferable to one with less, but between two solutions with the same number of jobs we give preference to the one that overruns the horizon by a smaller time, so as to leave more time for the next scheduling period.

In practice, the weights of the different components can be selected by the final user depending on the specific situation at hand. However, in order to create a common ground for comparison, in this formulation we set them to a fixed set of values. That is, each unscheduled job is valued 100, every minute of earliness/tardiness to an appointment costs five units, and every minute of overtime costs one unit.

3. Related work

The literature on production planning and scheduling in the steel industry is extensive; for a comprehensive and comparative analysis of early works the reader can refer to Tang et al. (2001). Since then, different versions of the SMCP problem have been considered in numerous and relevant contributions. In the following sections we review them, grouped by solution approach.

3.1. Mathematical programming methods

Traditional approaches to this problem consists mainly on mathematical programming. Tang et al. (2000) first formulated the SMCP as a non-linear programming model; the authors then converted the model into a linear one and solved it using a commercial software package. The same research

group proposed an integer programming (IP) formulation for the SMCP that takes into account job grouping, precedence constraints, setup and removal times on the machines, and job waiting costs. The solution methodology combines Lagrangian relaxation, dynamic programming, and heuristics (Tang et al., 2002).

Mathematical programming techniques were proposed by Harjunkoski and Grossmann (2001) to solve large-scale SMCPs through a decomposition scheme. Pacciarelli and Pranzo (2004) formulated the SMCP as an alternative graph and then solved it by a heuristic that uses a beam search procedure. The performance of their algorithm is tested on a single real-world instance, which is reported in the Appendix, and on a set of artificial randomly generated ones. Bellabdaoui and Teghem (2006) proposed an MILP model for a real-world SMCP arising in a plant in Belgium. It considers several real-world constraints about job grouping, precedence constraints, management of idle time, and processing time. A different linear programming model was proposed by Park and Yang (2009) who studied the steel casting process in a small/medium-sized foundry.

Li et al. (2012) developed an MILP model for the SMCP and a rolling-horizon approach that decomposes the entire MILP problem into two subproblems: the first one that solves the planning problem by defining the current horizon and the demand, and the second tackles the scheduling problem by defining the sequencing and timing of ladles and jobs at machines.

As already proposed by Tang et al. (2002), Mao et al. (2014) modeled the problem as an MILP, then introduced a Lagrangian relaxation for the capacity constraints and developed different methods based on subgradient optimization to explore the boundedness and convergence of the relaxed problem. A similar approach was followed by Pang et al. (2017) who proposed a Lagrangian relaxation for the machine capacity constraints and job precedence constraints, and developed an approximate subgradient algorithm to solve the relaxed problem.

Recently, as already mentioned, Fanti et al. (2016) proposed an MILP model to solve an SMCP formulation that is close to the one considered in this paper; the main differences of our problem with respect to their formulation are the following. First, Fanti et al. (2016) consider the makespan as their objective and schedule a selected subset of the available jobs that fit the daily horizon. Conversely, we deal with a considerably larger set of jobs to be selected and use a throughput measure that better fits with the way the scheduling is handled in practice. Moreover, we enrich their model by adding the possibility of standstills of single machines (for limited times) and, in order to increase the adherence to the practice, we consider also the border data coming from the schedule execution of the previous scheduling period. Finally, for the purpose of generality of the problem so that it can be applied in many operating conditions, we choose to slightly simplify the management of IC machines, the fly-tundish procedure and the cleaning process of the ladles, as described in detail in Section 2.

3.2. Constraint programming methods

A few attempts to model and solve the SMCP by constraint programming (CP) have been proposed in the literature. The first work modeling SMCP as a constraint satisfaction problem is ascribed to Li et al. (2005) who proposed a CP approach to solve the problem and experimented different search strategies. Gay et al. (2014) presented a CP model that uses standard disjunctive and cumulative resource constraints for scheduling. To solve real instances of large size (with more than 500 jobs),

the authors developed and compared different search heuristics: a greedy procedure to quickly obtain a feasible solution, and a large neighborhood search procedure to improve the incumbent one. Finally, Yadollahpour and Arbab Shirani (2016) devised a two phase approach: in the first phase, a CP procedure is used to group jobs into *casts* and assign them to casters in a continuous-time domain; in the second phase, the sequences obtained are reoptimized by a heuristic algorithm taking into account new additional assignment rules. The efficiency of the proposed method is evaluated against a greedy procedure on two pilot data tests.

3.3. Heuristic, metaheuristics, and hybrid methods

Metaheuristic approaches are widely used to solve the SMCP, given that they are domain-independent, but also hybrid techniques that combine exact method with heuristic algorithms are implemented to speed up the solution process and solve efficiently large-sized instances.

Lopez et al. (1998) presented a mathematical formulation for the hot mill production scheduling process, as a generalization of the traveling salesman problem. To solve the real-world problem of a Canadian steel company, they developed a metaheuristic method based on tabu search (TS). The TS procedure is applied at different phases of the solution approach: after an initial solution has been found by a greedy algorithm, a short TS is run to obtain a reasonable schedule by exchanging a single scheduled job with an unscheduled job; then a *cannibalization* technique is applied by performing swaps of groups of jobs and cross exchanges; finally, a long TS stage is used to improve the solution by individual swaps. Huegler and Vasko (2007) implemented a domain-specific heuristic that quickly generated near-optimal solutions, which are then used as initial schedule by an evolutionary programming approach. In Tang and Luo (2007), the authors formulated the SMCP as quadratic IP model with the objective of minimizing the dissimilarity between charges in the same cast, the number of casts used, and the unscheduled jobs. The problem is solved through an iterated local search (ILS) algorithm that starts from an initial solution generated by a constructive heuristic and then applies a large neighborhood that transfers a number of jobs among casts in a cyclic manner. The ILS is evaluated on five instances coming from a Chinese steel company and compared with a linear solver for the equivalent IP model. Pan et al. (2013) formulated a real-world SMCP as a hybrid flow shop problem using an MILP model with the objective function of minimizing the earliness/tardiness penalty and the average completion times of jobs; the problem is then solved by an artificial bee colony algorithm, which is compared with a heuristic method and two improvement procedures. Tang et al. (2014) devised a differential evolutionary algorithm to solve SMCP in a dynamic context such that the problem is to adjust and reoptimize the current schedule when unexpected real-time events occur, while keeping production continuity. A different approach has been proposed by Wang et al. (2015) who adopted a cross entropy method.

Based on the work of Bellabdaoui and Teghem (2006), Touil et al. (2016) developed a SA algorithm restricted by a tabu list. Differently from our formulation, they have to schedule many fixed sequences of jobs, each one dedicated to a particular caster, with the objective of minimizing the total completion time. The SA algorithm starts with a greedy initial solution built by assigning first jobs with higher processing times, then the neighborhoods basically swap two blocks of jobs of a caster or exchange two sequences; the solution obtained is checked against the tabu list.

Cowling and Rezig (2000) modeled the problem for the integration of steel CC and hot strip mills as a tripartite graph and solve it by a hybrid approach that is a combination of mathematical programming and heuristic techniques. Atighehchian et al. (2009) presented a hybrid two-phase algorithm to solve the SMCP: in the first phase, the job sequence in each casting machine is selected using ant colony optimization, while in the second phase a nonlinear optimization method determines the job scheduling, that is, the timing of the jobs on their assigned machines. In addition, the efficiency of the solution approach is compared to a heuristic algorithm and a genetic algorithm.

Several works present complex integrated systems that could include the optimization engine, user interface, automatic controller, and simulation module. Santos et al. (2002) developed a control system for the CC process based on a genetic algorithm. The controller interacts with a heat transfer model that analyzes the solidification progress and determines the process parameters. Cowling (2003) proposed a scheduling decision support system for steel hot rolling mills, complete user interface, and tools to simplify analysis of the data. The algorithmic core of this application is a commercial semiautomatic sequencing system that combines different *route-improvement* heuristics and TS. Finally, Missbauer et al. (2009) implemented a hierarchical scheduling system for the SMCP that operates at different planning levels at increasing level of detail. In the first two levels, the allocation and sequencing of jobs is obtained through heuristic methods, while in the last level start and finish dates of jobs are formulated as a LP model.

4. Solution methods

In this section, we first introduce our main proposal, which is a metaheuristic approach based on local search, and more specifically on SA.

Subsequently, we discuss the methods that we have developed in order to have the first assessment of the quality of our search method, based on the comparison with these alternatives. The first is a constraint-based solver obtained by the direct application of the formal CP model reported in Appendix. The second solver is a multistart greedy (MS-Greedy) method obtained by scheduling the jobs one at the time.

4.1. Simulated annealing

In the next subsections, we introduce the key components of our procedure.

4.1.1. Search space and initial solution

The *search space* is an indirect representation of the schedule, consisting of a sequence of all the available jobs (i.e., a permutation). The state also includes the assignment of each job to the machine for each specific step. Figure 4 shows the state corresponding to the solution presented in Fig. 3. Essentially, it is a sequence of jobs together with the specification of the machine that has to perform each processing step. Groups (see Section 4.1.2) are highlighted in dashed ovals.

The mapping between the job sequence and actual execution times of each job is obtained by applying a chronological constructive subprocedure, which assigns also ladles and cooling places for IC jobs. In detail, jobs are considered one by one according to the solution sequence. Each

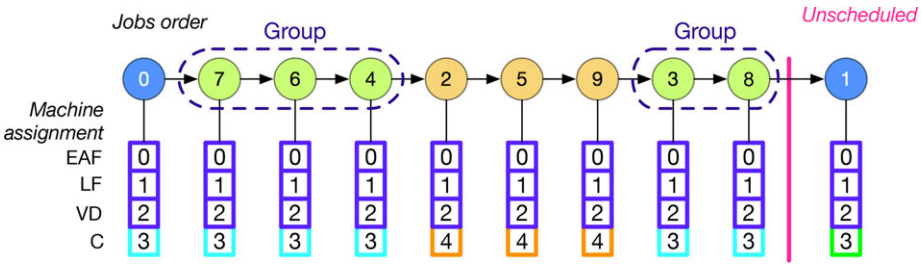


Fig. 4. Visualization of a state. [Colour figure can be viewed at wileyonlinelibrary.com]

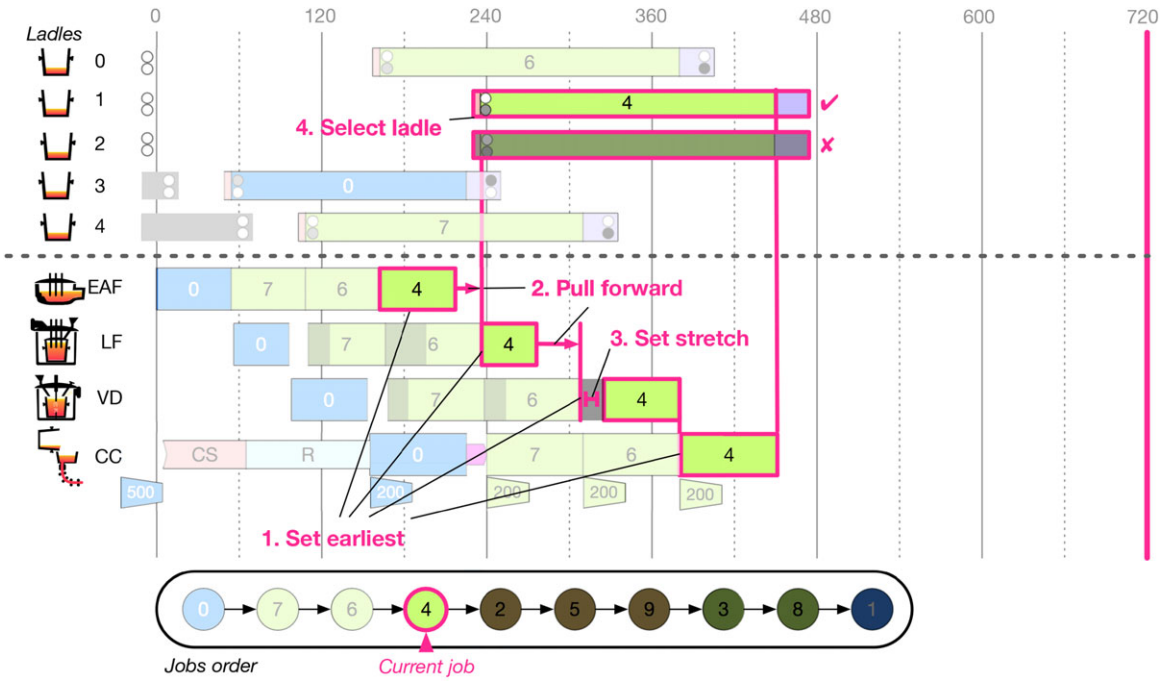


Fig. 5. A step of the constructive subprocedure. [Colour figure can be viewed at wileyonlinelibrary.com]

job is initially processed forward and each step is scheduled at the earliest time. For IC jobs, the subprocedure selects the first available cooling place, possibly delaying the start time, if necessary, and, for CC jobs, it selects the best possible setup mode. Based on this initial scheduling, the end time of the job is computed and fixed. Subsequently, the job is processed again backward and each step is stretched in case of excessive waiting time. If the maximum stretching is not sufficient to avoid waiting time violations, the start time of the step is postponed. If stretching is not possible due to a machine standstill, the job is left unmoved and a violation of the JobWaitingTime constraint is stored.

Figure 5 shows the initial schedule (step 1) of job no. 4 and the steps toward its final one, which reduce waiting times to the acceptable level by postponing (step 2) and stretching (step 3) the first three processing phases.

Finally (step 4), a ladle that is available and possibly does not violate the LADLECLEANING constraint is selected. When more than one ladle is available, the most recently used ladles are chosen. Instead, a dirty ladle is chosen when only ladles that violate the LADLECLEANING constraints are available, penalizing the solution accordingly. In case that no ladle is available at the given time, the earliest one is selected, obtaining a violation of the LATELADLE constraint. Violations of all hard constraints (JOBWAITINGTIME, LADLECLEANING, LATELADLE) are summed up and multiplied by a weight larger than all possible soft penalties, so as to always prefer feasible states to high-quality ones.

In order to save computing time, the subprocedure is not called anymore when the first job that is not schedulable within the given horizon has been identified. Such a job and all the following ones are considered as unscheduled. We call the scheduled part of the sequence, its *head*, and the unscheduled part, the *tail*. In the example, the tail of the state is composed of only job no. 1.

For the initial solution strategy $\mathcal{I}$, we tested two alternatives. The first one is obtained generating a random permutation of jobs (using the Fisher–Yates shuffle), and assigning to each step of each job a random line (and the corresponding machines) and a random compatible caster of the correct type. The second one is a single run of the greedy method described in Section 4.3. These two methods, called $\mathcal{I}_r$ and $\mathcal{I}_g$, are compared experimentally in a statistically principled ground as discussed in Section 5.

4.1.2. Neighborhood relations

The *neighborhood* relation is defined as the movement of a job to a new position in the sequence. All jobs in the state within the current position of the candidate job and its destination are shifted by one (either forward or backward). In addition, the move includes a new machine for each step, which might also be the same of the old one.

This general neighborhood is simplified according to the observation that in the real-world situations we have come across, the plant is always organized is such a way that it is more efficient to process a job using the machines of the same line, so that there is never an advantage in moving jobs between intermediate machines belonging to different lines. For this reason, in this work we consider changes of assignments of a job only to the full line and a caster (which does not belong to the line), and not to each single machine of the line, consequently limiting the size of the neighborhood considerably.

A move is thus identified by a quadruple of integers $\langle p_{job}, p_{new}, l, c\rangle$, representing the position of the job to be moved p_{job}, its destination p_{new}, the new line l, and the new caster c, respectively.

The neighborhood is further refined by pruning moves that are clearly ineffective, such as moves that take a job in the tail of the state and reinsert it in a different position still in the tail. We call the resulting neighborhood $\mathcal{N}_j$ (j for job), as it moves a single job.

A second neighborhood relation, called $\mathcal{N}_g$, considers the possibility that groups of jobs are moved together to a new position. This is applied to jobs on CC casters that are processed with no setup in the current state. In detail, when in a given state, jobs in positions $p_{job}+1, \ldots, p_{job}+i$ are all *setup-compatible* with the one in p_{job}, they are grouped and moved together with it. Setup-compatible means that they have the same steel grade, the same section, and are processed on the same machines (see, e.g., jobs 4, 6, 7 in Fig. 4).

Groups of jobs are not fixed, but are created dynamically. When a job (or a group) is moved next to a setup-compatible one, they become a group and thus could be moved together in subsequent iterations. On the other side, groups may also be broken, in the following two situations.

1. As explained above, a group is moved together when the first one is selected as candidate to be moved. On the other hand, when a job in the middle of a group is selected, only the jobs *after* it are involved, but not the jobs *before* it. As a consequence, when we select a move such that the job in position p_{job} is setup-compatible with the one in position $p_{job} - 1$, then if the move is accepted the group they form is split.
2. Similarly, when there is a group in the destination position of a move, the group could be split. That is, when jobs in position p_{new} and $p_{new} + 1$ are setup-compatible, the group is split by the insertion of the job (or the group) coming from position p_{job}. This happens when $p_{job} < p_{new}$ (i.e., the job is move backward), otherwise it regards jobs in position p_{new} and $p_{new} - 1$.

Preventing group-breaking could be helpful to make a more effective search, as groups normally lead to more compact schedules. Therefore, we experimented also the possibility that moves that result in one or the other of the above group-breaking moves are prohibited. Therefore, we have four different variants of the $\mathcal{N}_g$ neighborhood, based on which group-breaking situation they prohibit. We call them $\mathcal{N}_g$ (no prohibitions), $\mathcal{N}_{g_1}$ (prohibit moves in situation 1), $\mathcal{N}_{g_2}$ (prohibit moves in situation 2), $\mathcal{N}_{g_{12}}$ (prohibit all group-breaking moves).

The effectiveness of the five different neighborhood relations, $\mathcal{N}_j$, $\mathcal{N}_g$, $\mathcal{N}_{g_1}$, $\mathcal{N}_{g_2}$, and $\mathcal{N}_{g_{12}}$, are compared experimentally in Section 5.

4.1.3. Metaheuristics

Our SA procedure at each iteration draws a random move from the neighborhood and runs the subprocedure in order to compute the actual schedule and hence the costs. As customary for SA, the move is accepted if it is improving or sideways, whereas it is accepted according to a time-decreasing exponential distribution in case it is worsening. Specifically, a worsening move is accepted with probability $e^{-\Delta/T}$, where Δ is the difference of total cost induced by the move, and T is the *temperature*. The temperature starts at an initial value T_0, and it is decreased during the search by multiplying it by a *cooling factor* α ($0 < \alpha < 1$), after a fixed number of samples n_s has been drawn. The temperature thus evolves according to the typical geometric cooling scheme of SA.

In order to speed up the early stages of the SA procedure, we use the *cutoff* mechanism. To this aim we add a new parameter n_a, representing the maximum number of accepted moves at each temperature. That is, the temperature is decreased by multiplying it by α when the first of the following two conditions occurs: (a) the number of sampled moves reaches n_s and (b) the number of accepted moves reaches n_a. This allows us to save computational time in early stages of the search when the search dynamics is rather chaotic, which could be exploited later when the search behavior is steadier.

We decided to use as stop criterion the total number of iterations I_{max}. This choice has the advantage with respect to others (e.g., a threshold temperature) that, given the same instance, the running time is the same for all configurations of the parameters. In addition, it is not dependent

on the experimental environment and it is deterministic (given the random seed), so that each run can be reproduced exactly.

4.2. Constraint-based solver

We implement the CP model of the problem, whose details are reported in Appendix, in a constraint-based solver obtained using the IBM CPOptimizer (CPO) library. For the sake of presenting some details of the solver implementation, we just outline here the decision variables that are relevant to the search strategies implemented and some distinct modeling and search alternatives.

As for the decision variables, the model comprises a set of interval variables (i.e., CPO scheduling primitives) $\texttt{job}_j$ that will represent the schedule of a job $j \in J$ and a set of machine sequences variables $\texttt{m\_seq}_m$ that will be instantiated with the specific sequence of jobs processed by each machine $m \in M$.

The way the CPO search engine instantiates these variables depends on the branching strategy adopted. In particular, we develop four heuristic strategies, inspired by the problem structure.

- `None`: no branching heuristic is specified, therefore the CPO engine uses its own automatic selection;
- `BranchOnAppointments`: first try to set the details of the $\texttt{job}_j$ variables for those jobs which have an appointment;
- `BranchOnFurnaceSequence`: first try to set the sequence of jobs at `EAF` machines;
- `BranchOnCasterSequence`: first try to set the sequence of jobs at `CC` or `ING` machines.

Furthermore, we consider a modeling alternative regarding the objective function, which can be one of the following.

- `Aggregated`: The three cost function components are aggregated in a weighted sum.
- `Lexicographic`: The cost function considers its components in lexicographic ordering. In particular, this is the order of importance of the different cost elements APPOINTMENTS $\succ_{lex}$ UNSCHEDULED $\succ_{lex}$ FURNACEOVERTIME.

Finally, we also expose the propagation level of the different constraints, that is, how much effort has to be invested by the CPO engine to perform constraint-propagation. The propagation level values are the following: `Low`, `Basic`, `Medium`, `Extended`.

4.3. Greedy procedure

The greedy procedure works on the sequence of jobs, such as the local search one. At each greedy step i, it selects the *best* job to be scheduled in position i, and the best set of machines to assign it. When the job and its machines are selected, it is inserted in the schedule and the procedure goes on to step $i + 1$, with no backtracking.

At step i, for each candidate job and each selection of machines, the procedure computes the execution times and selects the ladle, using the same construction subprocedure developed for local

Table 1
Layouts and operating conditions used for generating the benchmark

Layout/operating conditions	Values
Layout (lines-CCs-ICs)	{(1-1-1), (2-3-2), (3-4-3)}
Whole horizon appointments	{0, 0.05}
Three-hour appointments	{0.1, 0.2}
Machine stops at the furnace	{0, 1}
Machine stops at another machine	{0, 1}
Horizon (in minutes)	{1440, 4320}
Ladles	5 × #lines

search. The best configuration is selected based on the violations of constraints and, in suborder, to the completion time of the job. Special treatment is given to jobs with appointments. If the candidate job finishes before the appointment interval, this is given the same penalty of a hard violation. In the case that the job finishes within the appointment interval, this is given a negative penalty (e.g., a reward). Finally, if the job would finish after the interval, a smaller penalty is given, as in this case the appointment cannot be fulfilled anymore (due to the backtrack-free nature of the greedy procedure).

When for the first time no job can be scheduled within the horizon, the rest of the jobs (i.e., the tail) is sequenced at random. Any time two or more alternatives have the same cost, a uniform random selection is performed to break the tie. This feature makes the procedure nondeterministic, and given that it is very fast, it is used in a multistart fashion (MS-Greedy). That is, we run it a fixed number of times and report the overall best solution.

5. Experimental analysis

All code is written in standard C++11 and compiled using GNU g++ (v. 5.4.0). All experiments ran on an Ubuntu Linux 16.04 machine with 32GB of RAM memory and 16 Intel® Xeon E5-2660 (2.20 GHz) cores. A single core has been dedicated to each experiment.

5.1. Instances

The instances employed in this work come from real-world job orders of a mid-sized steelmaking plant located in northeastern Italy. The specific plant layout and casting features have been properly modified to obtain instances with different layouts and a larger set of realistic operating conditions.

The original plant consists of two lines followed by three CC machines, and two IC machines as casters (denoted by 2-3-2). On this basis, we have created other two plant layouts, one with just a single production line, one CC and one IC (1-1-1), and another with three lines, four CCs and three ICs (3-4-3). Also the following operating conditions can vary: the percentage of *whole horizon appointments*, the percentage of *three-hour window appointments*, the number and type of *machine stops*, the scheduling *horizon*. The specific values are reported in Table 1.

We have collected 59 real-world production requests coming from the original plant at different times. From each of these instances, we have created several instances by generating all possible

Table 2
Specific features of the validation instances

Feature	Values
Lines	1–2
Jobs	66–318
Horizon	1440–4320
Machines	5–16
Steel grades	19–97
Pollutants	2
Ladles	5–15
Number of appointments	2–43
Total time of machine stops (in minutes)	0–420

Table 3
The simulated annealing best parameter configuration found by the F-Race procedure

Name	Description	Value
$\mathcal{N}$	Neighborhood relation	$\mathcal{N}_{g_1}$
T_0	Start temperature	600.0
α	Cooling rate	0.99
n_s	Moves sampled at each temperature	20,540
n_a	Moves accepted at each temperature	1848
$\mathcal{I}$	Initial solution method	$\mathcal{I}_g$

combinations of the layouts and operating conditions. We obtained a set of about 6000 instances, which we use as *training set*, in order to test our solver on a wider set of combinations between plant layouts and different practical situations. From this large set we have extracted 40 instances that we use as *validation set*, upon which we report our results. The ranges of the values of the features of the validation instances are shown in Table 2.

5.2. Experimental design and tuning

The proposed SA method has several parameters, which are the typical ones of SA plus the selection of one of the neighborhood relations among the five identified in Section 4.1.2. The tuning phase has been performed on the training instances using the dedicated tool JSON2RUN (Urli, 2013), which samples the space of the continuous parameters using the *Hammersley point set* (Hammersley and Handscomb, 1964) and runs the F-Race procedure (Birattari et al., 2010) for comparing the different configurations. We set the confidence level of the Friedman rank-sum test embedded in the F-Race procedure to 98% (i.e., p-value ≤ 0.02). The total number of iterations has been set to $I_{max} = 10^7$, corresponding to an average running time of 347.5 seconds on the above-described computer.

The parameter values of the winning configuration are shown in Table 3. The best neighborhood turned out to be $\mathcal{N}_{g_1}$, which is the one that never moves a "partial" group, but possibly splits groups by inserting jobs inside them. The explanation for this result is that groups are indeed useful, and some sort of limitation on group splitting, especially in later phases of the search, seems to be

Table 4
The CPO best parameter configurations

Modeling/solving parameter	CPO	Greedy+CPO
Objective function	`Aggregated`	`Aggregated`
Propagation	`Basic`	`Basic`
Branching strategy	`BranchOnAppointments`	`BranchOnCasterSequence`

beneficial. On the other hand, a complete absence of group-splitting moves may lead to premature group configurations that are impossible to be rearranged more suitably later on.

With regard to the generation of the initial solution, the method that uses the greedy procedure $\mathcal{I}_g$ has obtained results statistically better that the random one $\mathcal{I}_r$. The importance of having a "good" initial solution was already pointed out by the literature, and it is related to the high constrained nature of the problem.

As for the constraint-based solver, this method has also been tuned on the same training instances as the SA method. Moreover, also in this case we try to boost the solver by providing a warm-start solution with the greedy procedure. In this case, just a single solution has been generated as the starting point of the CPO solver. The constraint-based solver has been stopped after a fixed time limit (same as SA) taking the best solution found that far. Since the CP engine is deterministic, only a single run of the different configurations has been executed on each instance.

The outcome of the tuning, summarized in Table 4, is that there are two slightly different configurations of the solver exposed parameters for the plain CPO method (i.e., starting from scratch) and the greedy-boosted method. In both cases, the use of the lexicographic objective function is not beneficial and the aggregated one works better. Moreover, the most effective propagation strategy is the `Basic` one. However, the plain CPO method enjoys the `BranchOnAppointments` strategy, whereas the greedy-boosted method works better using the `BranchOnCasterSequence` strategy. This can be explained by the fact that, starting from an empty solution, there is a need to fix first those jobs with appointments and then improve the solution with other jobs. Conversely, a greedy-based starting point probably has the appointments already set (possibly in a wrong place), therefore shuffling the caster sequence is more beneficial.

Regarding the MS-Greedy solver, no tuning has been necessary. It has been granted approximately the same running time of the other two methods, by fixing the number of restarts (30,000).

5.3. Experimental results

First, we present the results on the validation instances of 30 runs of the best configuration of SA. Table 5 shows the main instance features (lines, jobs available and upper bound, horizon, machines), average and deviation of the total cost, and value of best solution, reporting the total cost and the value of each single soft component in terms of number of minutes for unmet appointments, number of unscheduled jobs, and minutes of overtime at furnace. The corresponding solutions are available at https://opthub.uniud.it, along with the validation tools.

We observe that, unsurprisingly, the cost increases with the size of the instance. In particular, there is a big increase moving from the horizon of 1440 minutes (one day) to the one of 4320 minutes (three days). Less significant and less homogeneous is the increase of cost related to the number

Table 5
Experimental results of SA on the validation instances

	Instance features							Best solution			
		Jobs				Total cost		Total	App	Unsch	FurnOver
Instance	Lines	Available	UB	Horizon	Machines	Average	Deviation	cost	minutes	number	minutes
sp-4714	2	145	53	1440	11	600.2	58.4	492	0	4	92
sp-4722	1	66	27	1440	5	1690.0	0.0	1690	90	12	40
sp-4725	2	196	161	4320	11	3146.2	199.1	2883	2	28	73
sp-4727	2	192	161	4320	11	3366.8	231.8	3050	0	30	50
sp-4942	3	215	82	1440	16	914.9	92.9	731	0	6	131
sp-4943	2	215	53	1440	11	408.4	81.6	277	0	2	77
sp-4945	1	110	81	4320	5	2612.5	160.6	2294	58	20	4
sp-4946	2	220	162	4320	11	3061.4	252.5	2673	0	26	73
sp-4947	3	305	239	4320	16	4948.0	245.2	4621	3	45	106
sp-4949	1	100	81	4320	5	3893.7	140.5	3413	60	31	13
sp-4950	2	177	51	1440	11	405.7	48.1	270	0	2	70
sp-4951	2	187	54	1440	11	495.7	57.8	367	0	3	67
sp-4952	3	164	81	1440	16	1203.3	101.4	1040	44	7	120
sp-4954	1	99	81	4320	5	3077.8	103.3	2796	30	26	46
sp-4956	2	223	50	1440	11	264.2	36.0	182	2	1	72
sp-4957	2	225	55	1440	11	442.6	64.1	355	0	3	55
sp-4958	2	256	161	4320	11	2371.3	146.2	2069	1	20	64
sp-4960	3	297	237	4320	16	4320.2	561.1	3028	0	30	28
sp-4962	2	182	54	1440	11	797.6	85.5	598	47	3	63
sp-4963	2	199	158	4320	11	3664.0	450.0	3031	55	27	56
sp-4964	3	142	77	1440	16	1156.8	188.9	890	0	8	90
sp-4966	3	148	81	1440	16	1402.1	187.9	1111	0	10	111
sp-4967	3	314	241	4320	16	6125.6	1193.6	4314	5	42	89
sp-4968	2	152	50	1440	11	932.6	75.3	793	65	4	68
sp-4972	3	306	241	4320	16	10,015.9	2243.9	7378	43	71	63
sp-4975	1	83	27	1440	5	331.4	15.7	326	0	3	26
sp-4976	1	95	81	4320	5	3383.7	118.2	3007	160	22	7
sp-4979	3	146	77	1440	16	1017.3	77.8	850	50	5	100
sp-4985	3	296	239	4320	16	4751.8	261.5	4383	0	43	83
sp-4987	2	196	161	4320	11	3727.8	143.6	3469	112	28	109
sp-4989	3	318	242	4320	16	5806.1	249.0	5351	96	48	71
sp-4990	3	296	76	1440	16	1498.6	387.8	893	38	6	103
sp-4992	2	200	157	4320	11	4578.1	508.0	3688	50	34	38
sp-4993	1	100	81	4320	5	8000.3	919.7	6500	380	46	0
sp-4994	3	178	81	1440	16	1234.8	107.5	1012	0	9	112
sp-4995	2	200	161	4320	11	4976.5	297.8	4592	275	32	17
sp-4997	3	170	82	1440	16	1598.6	175.2	1376	0	13	76
sp-4998	2	171	55	1440	11	799.5	70.1	658	0	6	58
sp-4999	3	167	82	1440	16	1765.1	143.5	1502	5	14	77
sp-5000	3	178	78	1440	16	869.2	75.7	678	0	6	78

Table 6
Comparative results on the validation instances. Values in bold highlight the best cost value obtained by the different algorithms for each instance

Instance	SA (average)	MS-Greedy	CPO	Greedy+CPO
sp-4714	**600.2**	1179	25,650	769
sp-4722	**1690.0**	8600	2150	8300
sp-4725	**3146.2**	4959	222,915	6500
sp-4727	**3366.8**	7523	510,585	77,740
sp-4942	**914.9**	1858	57,575	10,400
sp-4943	**408.4**	1455	39,107	2737
sp-4945	**2612.5**	3512	28,900	3200
sp-4946	**3061.4**	6960	394,965	50,900
sp-4947	**4948.0**	7360	375,865	9447
sp-4949	**3893.7**	6605	15,020	13,150
sp-4950	**405.7**	743	31,295	1204
sp-4951	**495.7**	1287	54,520	1327
sp-4952	**1203.3**	1791	6700	5872
sp-4954	**3077.8**	13,894	22,785	10,610
sp-4956	**264.2**	760	33,600	608
sp-4957	**442.6**	1163	68,000	8827
sp-4958	**2371.3**	3879	231,855	5300
sp-4960	4320.2	**2015**	237,100	5128
sp-4962	**797.6**	2747	53,835	16,800
sp-4963	**3664.0**	22,040	443,450	67,618
sp-4964	**1156.8**	4533	35,358	11,912
sp-4966	**1402.1**	2594	92,200	9827
sp-4967	**6125.6**	11,600	474,965	95,997
sp-4968	**932.6**	4094	32,805	30,100
sp-4972	**10,015.9**	21,276	795,045	79,587
sp-4975	331.4	628	1500	**243**
sp-4976	**3383.7**	5770	10,860	24,500
sp-4979	**1017.3**	2003	49,300	9000
sp-4985	**4751.8**	6299	447,905	8349
sp-4987	**3727.8**	7837	340,100	27,141
sp-4989	**5806.1**	8570	401,805	33,721
sp-4990	**1498.6**	1991	13,535	8539
sp-4992	**4578.1**	14,586	586,880	107,375
sp-4993	**8000.3**	38,655	97,765	64,775
sp-4994	**1234.8**	1461	42,125	3925
sp-4995	**4976.5**	*10,055	279,440	29,110
sp-4997	**1598.6**	1969	14,770	2732
sp-4998	**799.5**	1258	4525	1743
sp-4999	**1765.1**	2653	36,600	3310
sp-5000	869.2	**494**	4100	1529

of lines (from 1 to 2, to 3). This is explained by the fact that, for some instances, the total cost is dominated by the unscheduled component, which is independent of the number of available lines.

Table 6 compares the average results obtained by SA against the MS-Greedy procedure described in Section 4.3 and the constraint-based model (CPO) presented in Section 4.2 and Appendix.

We can note that SA clearly outperforms CPO, due to the difficulty of exact methods to solve complex problems on large instances. The performance of CPO is considerably improved by the introduction of the initial greedy solution (Greedy+CPO), confirming the behavior already observed for SA. This hybrid method is able to find the best result for one small instance with few appointments (`so-4975`). It is worth mentioning that in this method, CPO was always able to improve the initial solution of the greedy procedure.

In most cases the average results obtained by SA are better than those of the MS-Greedy, due to its limited ability to manage appointments and ladles. However for two instances (`sp-4960`, `sp-5000`), MS-Greedy finds solutions better than the results of SA. This can be explained by the unusual structure of these instances, characterized by few appointments in general, and all of them are whole horizon ones, thus are easily satisfied. In one case marked with "*," the MS-Greedy is not able to find a feasible solution, given that the corresponding solution counts two LadleCleaning violations.

6. Discussion, conclusions, and future work

We have defined a "standardized" version of the planning and scheduling problem in the steelmaking and casting industry, proposed by Danieli Automation. We have designed a metaheuristic approach based on local search, and in particular on SA, that uses a constructive subprocedure to compute times and resources assigned to each job. The neighborhood relation is based on insertion moves, which are combined with a dynamic management of groups of jobs.

For assessing the quality of the solutions, we have developed two alternative methods, whose results turned out not to be comparable, thus suggesting that the problem and its instances are far from being trivial. In any case, for the sake of measurability, all instances and solutions are available online, along with a web-based solution validator. We hope that our dataset could become a growing benchmark that could stimulate future research and comparisons on this problem. Indeed, to the best of our knowledge, no extensive dataset is available for problems within this area.

For the future, we plan to explore new solution techniques, both metaheuristic and exact, in order to try to gain more insights on which technique works best, and in which specific type of instances. To the same aim, we plan to explore new neighborhood relations that could outperform the proposed ones in some or all instances.

References

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

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

Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T., 2010. F-race and iterated F-race: an overview. In Bartz-Beielstein, Th., Chiarandini, M., Paquete, L., Preuss, M. (eds) *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, Berlin, pp. 311–336.

Cowling, P., 2003. A flexible decision support system for steel hot rolling mill scheduling. *Computers & Industrial Engineering* 45, 2, 307–321.

Cowling, P., Rezig, W., 2000. Integration of continuous caster and hot strip mill planning for steel production. *Journal of Scheduling* 3, 4, 185–208.

Fanti, M.P., Rotunno, G., Stecco, G., Ukovich, W., Mininel, S., 2016. An integrated system for production scheduling in steelmaking and casting plants. *IEEE Transactions on Automation Science and Engineering* 13, 2, 1112–1128.

Gay, S., Schaus, P., De Smedt, V., 2014. Continuous casting scheduling with constraint programming. *International Conference on Principles and Practice of Constraint Programming*, Vol. 8656. Springer, Berlin, pp. 831–845.

Hammersley, J.M., Handscomb, D.C., 1964. *Monte Carlo Methods*. Chapman and Hall, London.

Harjunkoski, I., Grossmann, I.E., 2001. A decomposition approach for the scheduling of a steel plant production. *Computers & Chemical Engineering* 25, 11–12, 1647–1660.

Huegler, P.A., Vasko, F.J., 2007. Metaheuristics for meltshop scheduling in the steel industry. *Journal of the Operational Research Society* 58, 6, 791–796.

Laborie, P., Rogerie, J., Shaw, P., Vilím, P., 2018. IBM ILOG CP optimizer for scheduling. *Constraints* 23, 2, 210–250.

Li, J., Xiao, X., Tang, Q., Floudas, C.A., 2012. Production scheduling of a large-scale steelmaking continuous casting process via unit-specific event-based continuous-time models: short-term and medium-term scheduling. *Industrial and Engineering Chemistry Research* 51, 21, 7300–7319.

Li, T., Li, Y., Zhang, Q., Gao, X., Dai, S., 2005. Constraint programming approach to steelmaking process scheduling. *Communication of the IIMA* 5, 3, 17.

Lopez, L., Carter, M.W., Gendreau, M., 1998. The hot strip mill production scheduling problem a tabu search approach. *European Journal of Operational Research* 106, 97, 317–335.

Mao, K., Pan, Q.K., Pang, X., 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.

Missbauer, H., Hauber, W., Stadler, W., 2009. A scheduling system for the steelmaking-continuous casting process. A case study from the steel-making industry. *International Journal of Production Research* 47, 15, 4147–4172.

Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G., 2007. MiniZinc: towards a standard CP modelling language. *Proceedings of 13th International Conference on Principles and Practice of Constraint Programming (CP 2007)*. Springer, Berlin, pp. 529–543.

Pacciarelli, D., Pranzo, M., 2004. Production scheduling in a steelmaking-continuous casting plant. *Computers & Chemical Engineering* 28, 12, 2823–2835.

Pan, Q.K., Wang, L., Mao, K., Zhao, J.H., Zhang, M., 2013. 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.

Pang, X., Gao, L., Pan, Q., Tian, W., Yu, S., 2017. A novel Lagrangian relaxation level approach for scheduling steelmaking-refining-continuous casting production. *Journal of Central South University* 24, 2, 467–477.

Park, Y.K., Yang, J.M., 2009. Optimization of mixed casting processes considering discrete ingot sizes. *Journal of Mechanical Science and Technology* 23, 7, 1899–1910.

Ruiz, R., Vázquez-Rodríguez, J.A., 2010. The hybrid flow shop scheduling problem. *European Journal of Operational Research* 205, 1, 1–18.

Santos, C.A., Spim, J.A., Ierardi, M.C., Garcia, A., 2002. The use of artificial intelligence technique for the optimisation of process parameters used in the continuous casting of steel. *Applied Mathematical Modelling* 26, 11, 1077–1092.

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

Tang, L., Liu, J., Rong, A., 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., Fang, L., 2002. Steel-making process scheduling using Lagrangian relaxation. *International Journal of Production Research* 40, 1, 55–70.

Tang, L., Luo, J., 2007. A new ILS algorithm for cast planning problem in steel industry. *ISIJ International* 47, 3, 443–452.

Tang, L., Zhao, Y., Liu, J., 2014. An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production. *IEEE Transactions on Evolutionary Computation* 18, 2, 209–225.

Touil, A., Echchtabi, A., Bellabdaoui, A., Charkaoui, A., 2016. A hybrid metaheuristic method to optimize the order of the sequences in continuous-casting. *International Journal of Industrial Engineering Computations* 7, 3, 385–398.

Urli, T., 2013. json2run: a tool for experiment design & analysis. *CoRR* abs/1305.1112.

Wang, G.R., Li, Q.Q., Wang, L.H., 2015. An improved cross entropy algorithm for steelmaking-continuous casting production scheduling with complicated technological routes. *Journal of Central South University* 22, 8, 2998–3007.
Yadollahpour, M.R., Arbab Shirani, B., 2016. A comprehensive solution for continuous casting production planning and scheduling. *International Journal of Advanced Manufacturing Technology* 82, 1–4, 211–226.

Appendix: Constraint programming model

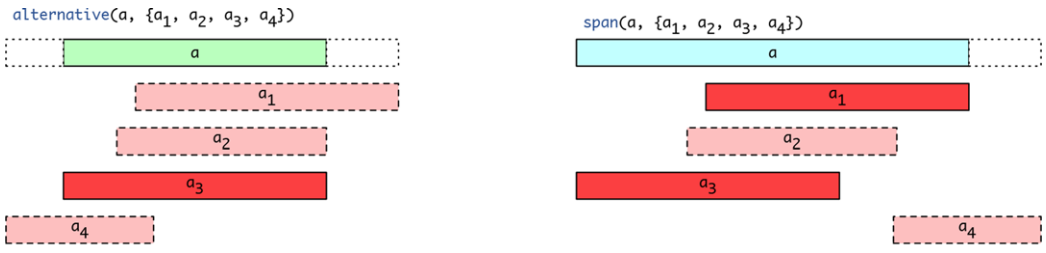
We provide a constraint programming (CP) model of the problem using the scheduling primitives available in IBM ILOG CP Optimizer (CPO) that allows us to capture the structure of complex scheduling problems in a compact model. In the following, we briefly sketch the CPO capabilities we used for our modeling and we refer to Laborie et al. (2018) for a comprehensive overview of the scheduling primitives and constraints.

In particular, CPO allows to employ *(optional) interval variables*, a scheduling construct whose domain is a subset of $\{\perp\} \cup \{[s, e) | s, e \in \mathbb{Z}, s \leq e\}$ with the special value $\perp$ that indicates the *absence* of the interval in the schedule. Besides its presence, represented as a Boolean variable, each interval variable I has associated three integer variables, namely its start and end point $\texttt{start}(I) = s$ and $\texttt{end}(I) = e$ and its length $\texttt{size}(I) = e - s$.

Interval variables can be combined in a hierarchical way, that is, they can represent coarse grain activities that are composed out of a set of fine grain activities. Fine grain activities can be *alternative* or *spanning* ones (see Fig. A1). In the case of the alternative activity (Fig. A1a), just one of the fine grain activities is set to be present and its interval coincides with the coarse grain activity, whereas in the spanning activity (Fig. A1b), the composed activity interval encompasses all the intervals of its fine grain activities. In both cases, the composed coarse grain activity is present in the solution only if at least one of its constituents is present.

Another relevant CPO modeling construct is the *sequence variable* concept (see Fig. A2), which is defined on a set of interval variables and represents a total ordering of the interval variables of the set. In the ordering only the variables that are present are considered.

The intervals in the sequence can be *tagged* with a type. In that case, it is possible to impose a minimum *type-dependent* transition time between two consecutive interval variables in the sequence



(a) Alternative composite activity (b) Spanning composite activity

Fig. A1. Composite activities in CP Optimizer: a darker color and a solid border denotes the activities that are *present* in the final schedule. In an *alternative* composition (a) only one of the composing activities is present and the interval of the composite one coincides with the present activity, whereas in a *spanning* composition (b) the composite interval *spans* over all the composing activities that are present in the solution.
[Colour figure can be viewed at wileyonlinelibrary.com]

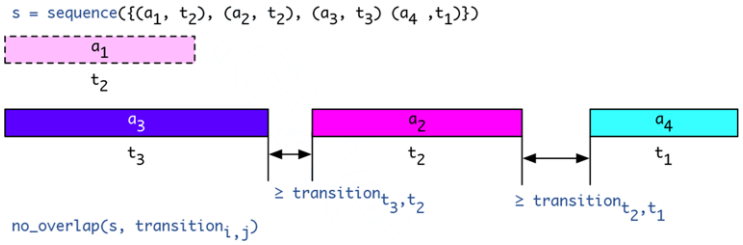


Fig. A2. Sequence variables in CP Optimizer: present variables will be totally ordered. Moreover, a *type-dependent* `no_overlap` constraint with a transition matrix guarantees that no pair of interval variables will overlap and a minimum transition time between the two is ensured.
[Colour figure can be viewed at wileyonlinelibrary.com]

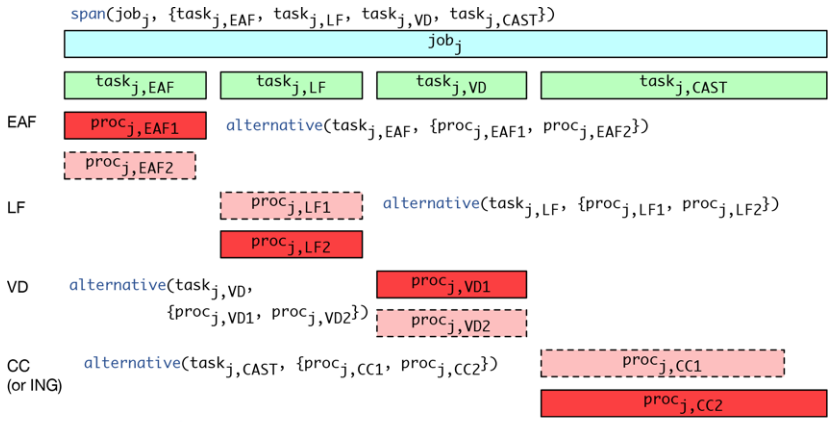


Fig. A3. The CP modeling of a single job.
[Colour figure can be viewed at wileyonlinelibrary.com]

through the `no_overlap` constraint. Moreover, given an interval variable and the sequence that it belongs to, it is possible to write expressions and constraints that refer to the relevant information (e.g., `start`, `end`, and `type`) of the previous and the next interval variable in the sequence.

In order to define our model, we will use the following notation. Let $J = \{0, \ldots, n-1\}$ be the set of n jobs, $M = \{0, \ldots, m-1\}$ the set of m machines, $L = \{0, \ldots, l-1\}$ the set of l ladles, the processing time, and the maximum stretching time on machine m are denoted by π_m and σ_m, respectively. The type of machine m is referred as τ_m, and the scheduling horizon is denoted by $h \in \mathbb{Z}^+$. The steel grade and the section of a job j are denoted by γ_j and ω_j, respectively.

A.1. Job schedules, tasks, and processes

A schematic representation of how the jobs are modeled in our CP formulation is presented in Fig. A3. The schedule of each job $j \in J$ is represented by an optional[2] interval variable $\texttt{job}_j$ that

[2]Since some of the jobs could be left unscheduled, in general, unless otherwise specified, all interval variables considered are *optional*.

spans over a set of *task* interval variables $\texttt{task}_{j,\texttt{EAF}}$, $\texttt{task}_{j,\texttt{LF}}, \ldots, \texttt{task}_{j,\texttt{CAST}}$, which represent the task of processing the job at a specific production step. In the following, we refer to S as the ordered set of processing steps, that is, $S = \langle \texttt{EAF}, \texttt{LF}, \texttt{VD}, \texttt{CAST} \rangle$:

$$\texttt{span}(\texttt{job}_j, \{\texttt{task}_{j,s} | s \in S\}) \qquad \forall j \in J \tag{A1}$$

The starting time of the $\texttt{job}_j$ variables must be within the time horizon, therefore,

$$\texttt{start}(\texttt{job}_j) \in [0, h] \qquad \forall j \in J. \tag{A2}$$

In turn, each of the tasks is then mapped into a *process*, which is an alternative execution on a determined physical machine for that specific production step. For example, in case two furnaces $\texttt{EAF}_1$ and $\texttt{EAF}_2$ are available, the task $\texttt{task}_{j,\texttt{EAF}}$ is set to be an alternative between $\texttt{proc}_{j,\texttt{EAF}_1}$ and $\texttt{proc}_{j,\texttt{EAF}_2}$ machine processes. The mapping for the CAST task on the physical CC or ING machine for its processing is determined by the type of job (either continuous or ingot casting). Moreover, since some casting machines might be incompatible with the job, only those that are compatible could be considered as process variables. For the purpose of abstracting these different cases, for each job j at a given processing step s we define the set of physical machines are compatible with the job at that step and we discard from the alternatives the $\texttt{proc}_{j,m}$ interval variables for those machines m that are incompatible with job j (A3). Also, the discarded machine processes are set to be absent from the solution (A4).

$$\texttt{alternative}(\texttt{task}_{j,s}, \{\texttt{proc}_{j,m} | m \in \mu_{j,s}\}) \qquad \forall j \in J, \forall s \in S \tag{A3}$$

$$\texttt{presence\_of}(\texttt{proc}_{j,m}) = \bot \qquad \forall j \in J, \forall s \in S \qquad \forall m \notin \mu_j s. \tag{A4}$$

For these concrete process variables, the size attribute can be constrained to lay in a range including the actual processing time π_m on a specific machine m possibly plus the stretching time σ_m allowed.

$$\texttt{size}(\texttt{proc}_{j,m}) \in [\pi_m, \pi_m + \sigma_m] \qquad \forall j \in J, \forall s \in S, \forall m \in \mu_{j,s}. \tag{A5}$$

Clearly, the tasks must be executed in the right processing order, therefore the following constraints must hold:

$$\texttt{task}_{j,\texttt{EAF}} \prec \texttt{task}_{j,\texttt{LF}} \prec \texttt{task}_{j,\texttt{VD}} \prec \texttt{task}_{j,\texttt{CAST}} \qquad \forall j \in J, \tag{A6}$$

where $\prec$ denotes the *precedence* and *no-overlap* constraint between two interval variables, that is, $I_1 \prec I_2$ means that I_1 must end before the start of I_2.

Since the $\texttt{task}_{j,s}$ variables are optional, their simultaneous presence or absence must be ensured.

$$\texttt{presence\_of}(\texttt{task}_{j,s_1}) \iff \texttt{presence\_of}(\texttt{task}_{j,s_2}) \qquad \forall s_1, s_2 \in S, s_1 < s_2. \tag{A7}$$

A.2. Job movements and waiting times

In order to model the movements of the job between two consecutive production steps, we use sequence variables to specify transition times between interval variables of different types.

Specifically, for each job j, its process variables $\texttt{proc}_{j,m}$ are composed in a sequence variable $\texttt{proc}_j$ and each of them is tagged with m, the index of the machine on which the process will eventually take place. This way, each interval variable in the sequence refers to the specific machine on which the process will take place.

$$\texttt{proc}_j = \texttt{sequence}(\{(\texttt{proc}_{j,m}, m) | m \in M\}) \qquad \forall j \in J. \tag{A8}$$

Given that the sequence is aware of the machine on which the process take place, we will model the transitions between two different steps by means of a distance matrix that contains the time needed to transit from one machine to another one:

$$\Delta_{m_1,m_2} = \begin{cases} +\infty & \text{if } m_1 \text{ is not reachable from } m_2 \\ \texttt{Distance}[m_1, m_2] & \text{otherwise} \end{cases} \qquad \forall m_1, m_2 \in M, m_1 \neq m_2. \tag{A9}$$

As already mentioned, the CPO `no_overlap` constraint on a sequence variable ensures that, for each pair of consecutive processes scheduled (i.e., actually present) in the sequence, the transition time between them is no less than the value specified in the distance matrix.

$$\texttt{no\_overlap}(\texttt{proc}_j, \Delta) \qquad \forall j \in J. \tag{A10}$$

It is worth noting that composing all the processes into the `proc` sequence variable does not require them to be present in the sequences (and, indeed, this will be not the case since the processes belonging to the same processing step are mutually exclusive because of the `alternative` constraints A3).

Also the upper bound of the actual transitions between two consecutive production steps has to be bounded, since the molten steel cannot indefinitely wait between steps.

$$\texttt{start}(\texttt{task}_{j,s_2}) - \texttt{end}(\texttt{task}_{j,s_1}) \leq \texttt{MaxWaitingTimeInLadle} \qquad \forall j \in J, \forall s_1, s_2 \in S, s_1 < s_2. \tag{A11}$$

A.3. Machine sequences and availabilities

All machines, but continuous casters, have no setup, therefore the only main constraint for their use is that the sequence of the tasks assigned to them do not overlap. Since we have to distinguish between `CC` machines and all the other ones, we define the set of continuous casters $C = \{m \in M | \tau_m = \texttt{CC}\}$, to be used in the following.

$$\texttt{no\_overlap}(\{\texttt{proc}_{j,m} | j \in J\}) \qquad \forall m \in M \setminus C. \tag{A12}$$

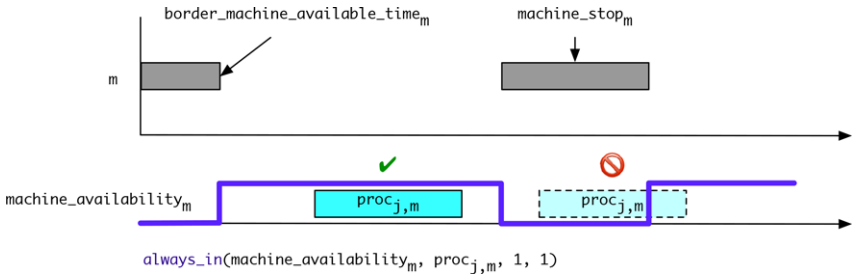


Fig. A4. Machine availability is modeled through cumulative function that has value 1 when the machine is available, and 0 otherwise. Each process $\texttt{proc}_{j,m}$ can be scheduled only where $\texttt{machine\_availability}_m$ is consistently equal to 1. [Colour figure can be viewed at wileyonlinelibrary.com]

Machines might be unavailable because they still have to finish the schedule of the previous period (i.e., border data) or because some stops are planned (e.g., for maintenance). This is modeled through a cumulative function that has value 1 when the machine is available (see Fig. A4), and 0 otherwise. In the following constraints, $\texttt{pulse}(I, x)$ denotes a pulse function, that is, a function of time whose value is x during interval I and 0 elsewhere, whereas $\texttt{step\_at}(t, x)$ is the step function having value x from time t on.

$$\begin{aligned}&\texttt{machine\_availability}_m(t)\\&\quad = \texttt{step\_at}(\texttt{BorderMachineAvailableTime}[m], 1)\\&\qquad + \sum_{I \in \texttt{machine\_stops}[m]} \texttt{pulse}(I, -1) \qquad \forall m \in M.\end{aligned} \tag{A13}$$

The correct use of machines is achieved by constraining each $\texttt{proc}_{j,m}$ variable to lay within regions of the cumulative function, where its value is always 1.

$$\texttt{always\_in}(\texttt{machine\_availability}_m, \texttt{proc}_{j,m}, 1, 1) \qquad \forall j \in J, \forall m \in M. \tag{A14}$$

A.4. Continuous caster sequences and setups

Modeling casters processing sequences is the most problematic characteristic of the problem because the setup mode (and consequently the time) depends on two features of the job, namely the *steel grade* and the *section*. Moreover, there are also other business constraints, due to the possible presence of waiting times between consecutive casting processes and a limitation on the frequency of the quicker `FlyTundish` setup mode.

Since we will map the processing sequences to sequence variables, the idea is to model the minimum setup times through transition matrices as for the job movements between processing steps (Section A2). However, for this purpose we have to identify each steel grade/section combination as a single scalar value (i.e., an integer index in a set T). We will do that using an encoding function $\eta : \texttt{steel\_grades} \times \texttt{job\_sections} \rightarrow T$.

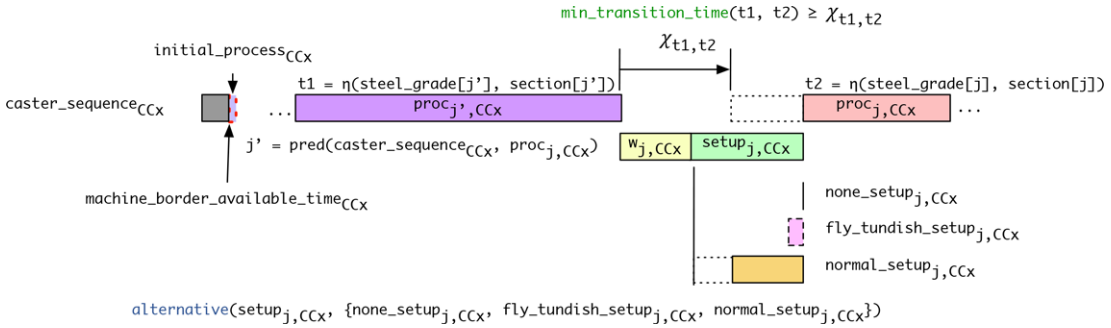


Fig. A5. Continuous caster sequencing and setup modes.
[Colour figure can be viewed at wileyonlinelibrary.com]

Moreover, we have to take into account also the border steel grade and section information (i.e., the final setup of the machine in the previous scheduling period) and the time when the machine is available. In order to model that, we introduce a *dummy* initial process $\texttt{initial\_proc}_m$ that starts and ends when the machine is initially available and its steel grade and section features will be those recorded in the border data.

$$\texttt{end}(\texttt{initial\_proc}_m) = \texttt{BorderMachineAvailableTime}[m] \qquad \forall m \in C. \tag{A15}$$

Overall, the schedule of each continuous caster $m \in C$ is comprised in a sequence variable $\texttt{cc\_seq}_m$ that consists of the dummy initial process $\texttt{initial\_proc}_m$ and the processes $\texttt{proc}_{j,m}$ together with the encoding of their steel grades and sections for each job $j \in J'_m$. From now on, J'_m denotes the subset of jobs that can be scheduled on the machine $m \in M$. A schematic illustration of the caster sequence is presented in Fig. A5.

$$\begin{aligned}\texttt{cc\_seq}_m = \texttt{sequence}(&\{(\texttt{initial\_proc}_m, \eta(\texttt{initial\_proc}_m))\} \\ &\cup \{(\texttt{proc}_{j,m}, \eta(\texttt{proc}_{j,m}))|j \in J'_m\}) \qquad \forall m \in C.\end{aligned} \tag{A16}$$

The first activity of the sequence must be the dummy initial process, so that all the other processes will temporally follow.

$$\texttt{first}(\texttt{cc\_seq}_m) = \texttt{initial\_proc}_m \qquad \forall m \in C. \tag{A17}$$

According to the process type encodings, it is possible to define a transition matrix χ^m that will ensure that the activities do not overlap and a minimum transition time between each pair of activities in the sequence is granted. This minimum time depends on the quickest setup mode

allowable because of their types (i.e., None, FlyTundish, Rearm, ChangeSection in increasing order of time).

$$\chi^{m}_{\eta(g_1,w_1),\eta(g_2,w_2)} = \begin{cases} 0 & \text{if } g_1 = g_2 \wedge w_1 = w_2 \\ \texttt{ChangeTundishTime} & \text{if } \neg \texttt{FlyTundishIncompatibility}[g_1, g_2] \\ & \quad \wedge w_1 = w_2 \\ \texttt{RearmTime}[m] & \text{if } \texttt{FlyTundishIncompatibility}[g_1, g_2] \\ & \quad \wedge w_1 = w_2 \\ \texttt{RearmTime}[m] & \\ \quad + \texttt{ChangeSectionTime}[m] & \text{otherwise} \end{cases} \quad \forall g_1, g_2 \in \texttt{steel\_grades}, \; \forall w_1, w_2 \in \texttt{job\_sections} \tag{A18}$$

$$\texttt{no\_overlap}(\texttt{cc\_seq}_m, \chi^m) \qquad \forall m \in C. \tag{A19}$$

Note that the former constraint only ensures that consecutive casting processes are scheduled apart at least of the minimum time needed for their sequence-dependent setup mode. Unfortunately, for the two quicker setup modes (i.e., None and FlyTundish) there is also an upper bound ContinuousCasterMaxWaitingTime on the waiting time between the end of the previous process and the start of the setup mode of the current one. In particular, if a None or a FlyTundish setup is delayed too much, it must be converted into a Rearm one.

In order to model this business constraint, the $\texttt{setup}_{j,m}$ interval variables are introduced, whose purpose is to record (and constrain) the actual setup time that is taking place between the companion $\texttt{proc}_{j,m}$ process and its predecessor process in the sequence $\texttt{cc\_seq}_m$. Moreover, we introduce another set of interval variables $\texttt{wait}_{j,m}$ whose purpose is to account for the waiting time before the setup start.

$$\texttt{start}(\texttt{proc}_{j,m}) = \texttt{end}(\texttt{setup}_{j,m}) \qquad \forall j \in J'_m, \forall m \in C. \tag{A20}$$

Obviously, these interval variables will be present only when the specific $\texttt{proc}_{j,m}$ process they refer to is present.

$$\begin{aligned} \texttt{presence\_of}(\texttt{setup}_{j,m}) &\iff \texttt{presence\_of}(\texttt{proc}_{j,m}) \\ \texttt{presence\_of}(\texttt{wait}_{j,m}) &\iff \texttt{presence\_of}(\texttt{proc}_{j,m}) \qquad \forall j \in J'_m, \forall m \in C. \end{aligned} \tag{A21}$$

Setup and wait variables must be synchronized with the processes they refer to. In the following constraints, $\texttt{pred}(\texttt{proc}_{j,m}, \texttt{cc\_seq}_m)$ refers to the process j' that precedes $\texttt{proc}_{j,m}$ in the sequence $\texttt{cc\_seq}_m$.

$$\begin{aligned} \texttt{start}(\texttt{wait}_{j,m}) &= \texttt{end}(\texttt{pred}(\texttt{proc}_{j,m}, \texttt{cc\_seq}_m)) \\ \texttt{start}(\texttt{setup}_{j,m}) &= \texttt{end}(\texttt{wait}_{j,m}) \\ \texttt{start}(\texttt{proc}_{j,m}) &= \texttt{end}(\texttt{setup}_{j,m}) \qquad \forall j \in J'_m, \forall m \in C. \end{aligned} \tag{A22}$$

The abstract interval variables $\texttt{setup}_{j,m}$ will be composed of different concrete setup activities according to the different setup modes. Namely, the alternatives for these variables are set as follows:

$$\begin{aligned}\texttt{alternative}(\texttt{setup}_{j,m},\ &\{\texttt{none\_setup}_{j,m},\\ &\texttt{fly\_tundish\_setup}_{j,m},\\ &\texttt{complete\_setup}_{j,m}\}) \qquad \forall j \in J'_m, \forall m \in C. \end{aligned} \tag{A23}$$

In particular, the following optional interval variables are introduced:

- $\texttt{none\_setup}_{j,m}$ have *fixed size* 0, accounting for no setup;
- $\texttt{fly\_tundish\_setup}_{j,m}$ have *fixed size* `ChangeTundishTime`, accounting for a fly-tundish;
- $\texttt{complete\_setup}_{j,m}$ are *stretchable*, whose minimum size is the duration of a `Rearm` setup, accounting also for the `ChangeSize` setup mode.

The $\texttt{none\_setup}_{j,m}$ and $\texttt{fly\_tundish\_setup}_{j,m}$ variables model the fact that these types of setup are possible only when the two consecutive processes on which they are interleaved are scheduled in the sequence exactly within that specific time (0 and `ChangeTundishTime` minutes, respectively). If this constraint is not met, then a `Rearm` must occur and consequently a longer $\texttt{complete\_setup}_{j,m}$ will take place. Note that $\texttt{complete\_setup}_{j,m}$ also implicitly includes the `ChangeSection` setup mode, since when this setup mode occurs the two consecutive processes in the sequence are already spread apart at least of $\texttt{rearm\_time}[m] + \texttt{change\_section\_time}[m]$ because of the `no_overlap` constraints A19.

The following constraint ensures that the business rule defined before is satisfied. That is, if the delay measured by the waiting time is above the maximum waiting time allowed for a continuous caster, than a rearm setup is forced.

$$\begin{aligned}\texttt{size}(\texttt{wait}_{j,m}) \geq \texttt{ContinuousCasterMaxWaitingTime}\\ \Rightarrow \texttt{presence\_of}(\texttt{complete\_setup}_{j,m}) = \texttt{true} \quad \forall j \in J'_m, \forall m \in C. \end{aligned} \tag{A24}$$

Finally, the $\texttt{fly\_tundish\_setup}_{j,m}$ variables are used also to model a requirement on the frequency of this setup mode. Specifically, a maximum number `fly_tundish_frequency` of setups of type `FlyTundish` could occur consecutively. We model this requirement by building a sequence of $\texttt{fly\_tundish\_setup}_{j,m}$ variables and constraining it to have a transition between two consecutive setups at least at a time distance of that number multiplied by the processing time π_m. This way, the `fly_tundish_frequency` rule will be enforced.

$$\texttt{fly\_tundish\_sequence}_m = \texttt{sequence}(\{(\texttt{fly\_tundish\_setup}_{j,m}, 0) | j \in J'_m\}) \quad \forall m \in C \tag{A25}$$

$$\texttt{no\_overlap}(\texttt{fly\_tundish\_sequence}_m, [\texttt{FlyTundishFrequency} \cdot \pi_m]) \quad \forall m \in C. \tag{A26}$$

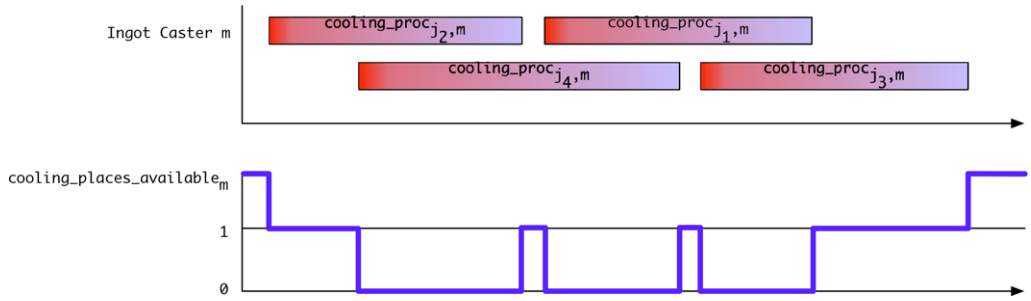


Fig. A6. The cumulative function modeling the usage of the cooling places. At the beginning there are `IngotCoolingPlaces[m]` available resources and each $\texttt{cooling\_proc}_{j,m}$ process is decreasing this availability. [Colour figure can be viewed at wileyonlinelibrary.com]

A.5. Ingot cooling

Each Ingot Caster, whose set of machines is denoted by $I = \{m | m \in M \wedge \tau_m = \texttt{ING}\}$, has a fixed number of cooling places, which are additional limited resources that are employed for further time after the casting has finished. Therefore, a set of $\texttt{cooling\_proc}_{j,m}$ interval variables are introduced for ingot casting jobs.

The occupation of the cooling place starts at the beginning of the ingot casting process and it lasts for `CoolingTimes[j]` minutes after its end.

$$\texttt{start}(\texttt{cooling\_proc}_{j,m}) = \texttt{start}(\texttt{proc}_{j,m}) \qquad \forall j \in J'_m, \forall m \in I \tag{A27}$$

$$\texttt{end}(\texttt{cooling\_proc}_{j,)} = \texttt{end}(\texttt{proc}_{j,m}) + \texttt{CoolingTimes}[j] \qquad \forall j \in J'_m, \forall m \in I. \tag{A28}$$

The usage of the cooling places are modeled through cumulative functions, so that no more than `IngotCoolingPlaces[m]` are simultaneously used by ongoing cooling processes (see Fig. A6).

$$\texttt{step\_at}(0, \texttt{IngotCoolingPlaces}[m]) - \sum_{j \in J'_m} \texttt{pulse}(\texttt{cooling\_proc}_{j,m}, 1) \geq 0 \qquad \forall m \in I. \tag{A29}$$

A.6. Ladles

Besides machines and cooling places, the other resources needed to process the jobs are the *ladles*. The modeling of the ladle schedule is similar to the one used for the jobs (see Fig. A7). That is, for each job j, we introduce an abstract $\texttt{ladle}_j$ interval variable that spans over the production tasks that have to be accomplished while the job is in the ladle. Differently from the job activity, the ladle is not needed during the whole `EAF` task, but only shortly before its end when the molten material is *poured* into the ladle. For this reason, an additional $\texttt{pouring}_j$ task of size `LadlePouringTime`

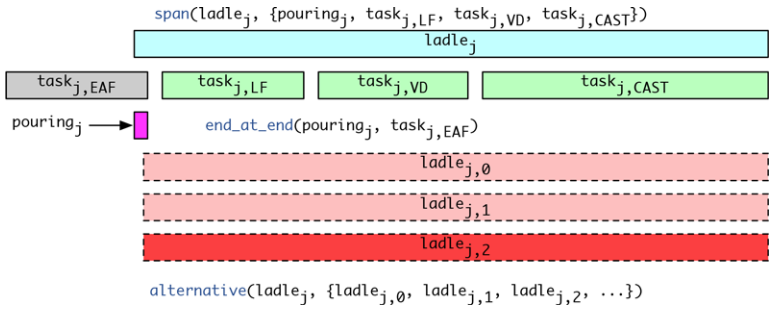


Fig. A7. The CP modeling of the ladle usage.
[Colour figure can be viewed at wileyonlinelibrary.com]

is introduced, whose end is synchronized with the end of the EAF task.

$$\texttt{size}(\texttt{pouring}_j) = \texttt{LadlePouringTime} \qquad \texttt{end\_at\_end}(\texttt{task}_{j,\texttt{EAF}}, \texttt{pouring}_j) \qquad \forall j \in J \tag{A30}$$

$$\texttt{span}(\texttt{ladle}_j, \{\texttt{pouring}_j \texttt{task}_{j,\texttt{LF}}, \texttt{task}_{j,\texttt{VD}}, \texttt{task}_{j,\texttt{CAST}}\}). \tag{A31}$$

Similar to the jobs, each abstract interval variable $\texttt{ladle}_j$ is then mapped on a set of alternative concrete ladle processes $\texttt{ladle\_proc}_{j,l}$, each on a given physical ladle l.

$$\texttt{alternative}(\texttt{ladle}_j, \{\texttt{ladle\_proc}_{j,l} | l \in L\}) \qquad \forall j \in J. \tag{A32}$$

The simultaneous presence of the $\texttt{job}_j$ and the $\texttt{pouring}_j$ and $\texttt{ladle}_j$ schedules must be ensured:

$$\texttt{presence\_of}(\texttt{job}_j) \iff \texttt{presence\_of}(\texttt{pouring}_j) \qquad \forall j \in J \tag{A33}$$

$$\texttt{presence\_of}(\texttt{job}_j) \iff \texttt{presence\_of}(\texttt{ladle}_j) \qquad \forall j \in J. \tag{A34}$$

Now we consider the schedule of each ladle, which is comprised in a sequence variable $\texttt{ladle\_proc}_l$ so as to ensure that single usages do not overlap and enough time for cleaning and return is granted. This is achieved using a single type for each task and setting a degenerate transition matrix consisting only of the value LadleCleaningAndReturnTime.

$$\texttt{ladle\_proc}_l = \texttt{sequence}(\{(\texttt{ladle\_proc}_{j,l}, 0) | j \in J\}) \quad \forall l \in L \tag{A35}$$

$$\texttt{no\_overlap}(\texttt{ladle\_proc}_l, [\texttt{LadleCleaningAndReturnTime}]). \tag{A36}$$

Ladle cleaning constraints are expressed through a set of sequence variables $\texttt{ladle\_pollution\_seq}_{l,r}$, $l \in L$, so to encode in their type the pollution action of the steel grades with respect to the pollutant $r \in \texttt{pollutants}$. This way, it will be possible to access the previous ladle process variable in the sequence and forbid the prohibited sequences. Similar to the caster modeling, the ladle is not generally available at time zero because it can be used for finishing

the schedule of the previous period and its initial pollution status has to be recorded. Therefore a dummy initial process $\texttt{initial\_ladle\_proc}_{l,r}$ in the sequence is used for both purposes.

$$\begin{aligned}\texttt{start}(\texttt{initial\_ladle\_proc}_{l,r}) &= \texttt{BorderLadleAvailableTime}[l] \\ \texttt{size}(\texttt{initial\_ladle\_process}_{l,r}) &= 0 \\ &\forall l \in L, \forall r \in \texttt{pollutants} \quad \text{(A37)}\end{aligned}$$

$$\begin{aligned}\texttt{ladle\_pollution\_seq}_{l,r} = \texttt{sequence}(&\{(\texttt{initial\_ladle\_proc}_{l,r}, \\ &\texttt{BorderLadlePollutionStatus}[l,r])\} \\ &\cup\{(\texttt{ladle\_proc}_{j,l}, \\ &\texttt{SteelGradePollutionAction}[\gamma_j, r])\}) \\ &\forall l \in L, \forall r \in \texttt{pollutants}. \quad \text{(A38)}\end{aligned}$$

The dummy initial process must be set as the first interval variable of the sequences and the activities in the sequence cannot overlap:

$$\begin{aligned}\texttt{first}(\texttt{ladle\_pollution\_seq}_{l,r}) &= \texttt{initial\_ladle\_proc}_{l,r} \quad \text{(A39)} \\ \texttt{no\_overlap}(\texttt{ladle\_pollution\_seq}_{l,r}) &\quad \forall l \in L, \forall r \in \texttt{pollutants}.\end{aligned}$$

After this setup, the ladle processes that will be present in each pollution sequence are tagged with the corresponding pollution action. Now, the semantics of the pollution actions/requirements values is so that the no-pollution constraints are satisfied if for two consecutive ladle usages, the value of the action is less than or equal to the value of the pollution requirement. Therefore, due to the availability of the pollution action information through the type of the interval variable in the sequence, we can model these constraints as follows:

$$\begin{aligned}&\texttt{pred\_type}(\texttt{ladle}_j, \texttt{ladle\_pollution\_seq}_{l,r}) \\ &\quad \leq \texttt{SteelGradePollutionRequirement}[\gamma_j, r] \quad \forall j \in J, \forall l \in L, \forall r \in \texttt{pollutants}.\end{aligned} \quad \text{(A40)}$$

These constraints are implicitly satisfied when the ladle process $\texttt{ladle}_j$ is absent from the solution.

A.7. Objective function

The objective function consists of the aggregation of three components, namely the number of unscheduled jobs, the unmet appointments and the overtime at the furnace.

The unscheduled jobs component is the difference between the upper bound `max_schedulable_jobs` and the actual number of scheduled jobs:

$$\texttt{unscheduled} = \texttt{max\_schedulable\_jobs} - \sum_{j \in J} \texttt{presence\_of}(\texttt{job}_j). \quad \text{(A41)}$$

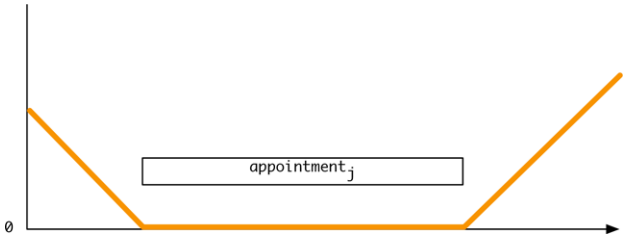


Fig. A8. Earliness/tardiness components of the cost for unmet appointments.
[Colour figure can be viewed at wileyonlinelibrary.com]

As for the unmet appointments, we adopt an earliness/tardiness approach. In particular, for each job j having an appointment J^* we define a piecewise linear function $\texttt{unmet\_cost}_j$, such as the one depicted in Fig. A8. In particular, the slope of the slanted segments will be ± 1, measuring one point of penalty for each early/tardy minute.

$$\begin{aligned}\texttt{unmet\_cost}_j = {} & \texttt{piecewise}(0, \texttt{appointment\_start}[j], -1) \\ & + \texttt{piecewise}(\texttt{appointment\_end}[j], h, +1) \qquad \forall j \in J^*. \end{aligned} \tag{A42}$$

The cost component is the aggregation of the $\texttt{unmet\_cost}_j$ evaluated at the end point of each job $j \in J^*$ that is present in the solution. The absence of the job from the solution is penalized with a default cost corresponding to the value of the time horizon h.

$$\texttt{unmet\_appointments} = \sum_{j \in J^*} \texttt{unmet\_cost}_j(\texttt{end}(\texttt{job}_j)). \tag{A43}$$

Finally, the overtime at the furnace is measured by recording in a set of sequence variables the processes at the furnaces (E will denote the set of EAF machines).

$$\texttt{eaf\_seq}_m = \texttt{sequence}(\{\texttt{proc}_{j,m} | j \in J\}) \qquad \forall m \in E. \tag{A44}$$

Therefore, we can access the last process in the sequence and sum up how much it goes overtime.

$$\texttt{furnace\_overtime} = \sum_{m \in E} \max\{0, \texttt{end}(\texttt{last}(\texttt{eaf\_seq}_m)) - h\}. \tag{A45}$$