Continuous Casting Scheduling with Constraint Programming

Steven Gay[1], Pierre Schaus[1], and Vivian De Smedt[2]

[1] Université Catholique de Louvain, Belgium
[2] PSI Metals, Belgium

Abstract. Although the Steel Mill Slab problem (prob 38 of CSPLib) has already been studied by the CP community, this approach is unfortunately not used anymore by steel producers since last century. Continuous casting is preferred instead, allowing higher throughput and better steel quality. This paper presents a CP model related to scheduling of operations for steel making with continuous casting. Activities considered range from the extraction of iron in the furnace to its casting in continuous casters. We describe the problem, detail a CP scheduling model that is finally used to solve real-life instances of some of the PSI Metals' customers.

Keywords: Continuous Casting, Steel Production, Scheduling, Constraint Programming

1 Introduction

Steel Production problems have already been tackled with CP. In particular the Steel Mill Slab problem (prob 38 of CSPLib) has been studied in [4, 5, 14, 7]. Although interesting from a theoretical point of view, this problem is of limited practical interest since this technique has been replaced since 1950's by continuous casting.

A schematic representation of continuous casting is given on Fig. 1. A ladle is poured into the tundish, a reservoir of hot metal, to feed the casting machine. The strand (solidifying metal, output of the casting machine) passes through straightening rolls before being cut into predetermined lengths by mechanical shears. We refer to [16] for more information related to steel production in general.

The continuous casting depicted on Fig. 1 is the last of a three step process, the first two steps being the carbon removal and the steel refining. The goal of the continuous casting scheduling problem (CCSP) is to determine the timing of operations on molten steel at the three steps of production and the device/facility where they happen, when some flexibility is allowed.

There are two reasons for using computer optimization for scheduling operations related to continuous casting. The first one is the cost of the machines involved in the process. They are so costly that they should effectively be used

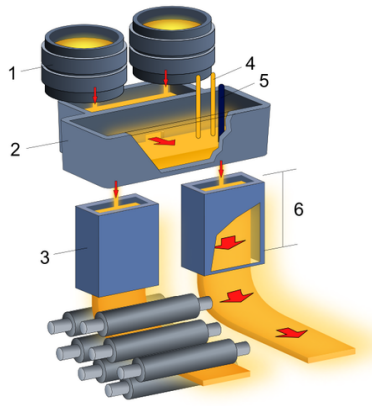

Fig. 1. 1: Ladle. 2: Tundish. 3: Mold. 4: Plasma torch. 5: Stopper. 6: Straight zone (Image from [19]).

24 hours per day. The cost of a modern steel making facility is on the order of 10 billion euros. A single continuous caster costs on the order of half a billion euros. Since the lifetime of a caster is around 30 years of lifetime, which is around 45 thousand euros per day. Thus, one of the main objective is to ensure continuity of the production process at the caster. The second reason is satisfiability. Many instances are very constrained, some constraints compete and make it difficult to solve the problem manually or using heuristics. The problem requires typically to plan operations up to 36 hours which means about 500 activities to schedule.

PSI Metals develops software for decision making in steel production. This work aims at developping flexible and maintainable tools to deal with continuous casting in the software suite of PSI Metals.

Outline. Section 2 describes the CCSP problem in detail. Section 3 motivates the usage of CP for solving the problem. Section 4 details the CP model, before experimenting the model in Section 5 on real-life instances.

2 Description of the problem

The problem is a scheduling problem presenting some similarities with the job-shop problem in the sense that some activities must be processed sequentially by different *machines*. Each job is a *heat*, which is a pocket of molten metal, that undergoes several transformations ; each transformation may be modeled as an activity with start date, duration and end. However, application-specific requirements makes this problem different from a pure job-shop problem.

Continuous Casting as a Scheduling problem. A *heat job* is composed of activities that must take place in a fixed order, as represented in Fig. 2.

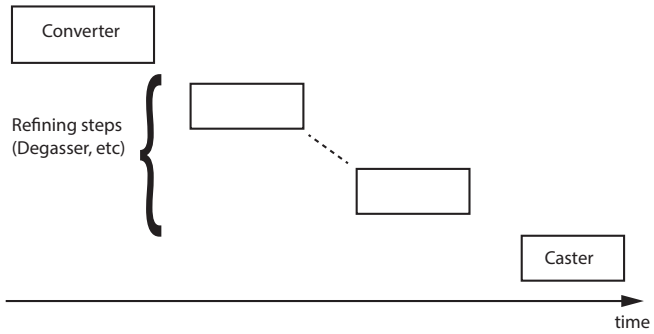

Fig. 2. A heat job: the converter activity must happen first, then the refining activities (degasser, stew, transports, etc), and finally the caster activity.

The first activity of a heat is its creation in a *converter*, either a blast oxygen furnace that processes iron ore or an electric furnace that melts recycled scrap. At this end of this activity, metal is tapped in liquid state into a ladle. The metal contents of a ladle is a *heat*, it embodies a discrete unit of molten metal. Note that this discretization into heat/ladles comes from a physical reality.

Then the heat is treated in various facilities, to undergo treatments that will modify the metal's final properties. These treatments and the transports between the facilities can be described as activities of the heat, with start date, duration, end date, and associated resource. For the plant considered in our experiments, there are two intermediate treatments, alloying (addition of external components) and vacuum degassing (removal of excess oxygen blown by the conversion process).

Finally a heat is tapped into a continuous caster, to be transformed into solid metal. This is the last activity of a heat and the end of the problem under consideration.

Application-Specific requirements. Although the problem presents some similarities with the job-shop problem, there are some constraints specific to the CCSP. The first is the temperature requirement, depicted in Fig. 3. From its generation in the converter facility to its final transformation in the caster, a heat *has* to remain in liquid state. Even though there are no release times nor deadlines specified, there is a maximum time a heat can spend in the relatively cold atmosphere before solidifying. To avoid this situation, a maximum duration is specified between the exit of the converter and the entry in the caster. Note that it is not possible to raise the initial temperature at the converter artificially to ensure arrival at the caster in liquid state, since the metal could boil away and the containing equipment could be damaged. Moreover, heating metal has an energy cost.

The second is the *continuity* requirement, also depicted in Fig. 3. Continuous casting is a technique where to make n shapes that are effectively truncated

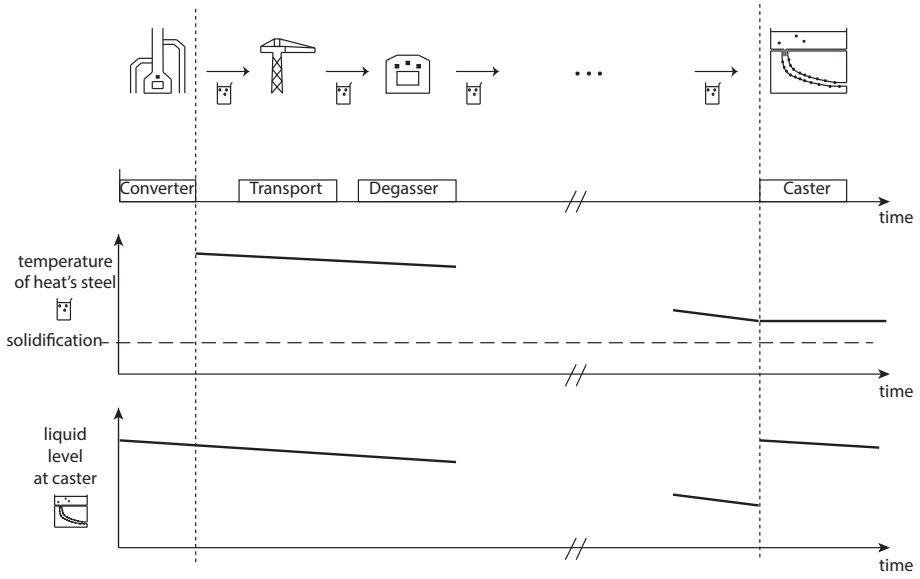

Fig. 3. As a heat undergoes treatments in different facilities, it cools down and the level of the molten metal in the caster waiting for it decreases.

cylinders (such as I-beams, sheets ...), a single cylinder is cast in a continuous process in a first stage, and cut into the required lengths in the following stage, on-the-fly. The advantages are better metal quality and cheaper processing costs, but the drawback is that during the first stage, the caster can not be stopped. This translates into additional constraints for the scheduling: a *program* is a set of heats that will be used for the continuous casting of one object. The heats of a program have thus to be treated consecutively at the caster (without any gap between corresponding activities). This is shown in the bottom part of Fig. 3 Adjacent heats may arrive on-time, early, but not late. Typically, the allocation of programs to the casters and the order of programs at each caster is received from a previous computation in the planning and must remain unchanged.

Example 1. A heat has to undergo several transformations before being cast in a solid shape, as shown in Fig. 3. The top part of this diagram shows a heat going through different facilities along the timeline, from conversion to casting. The middle part shows the temperature of the metal inside the heat, which starts decreasing as soon as the heat leaves the converter. Then, the temperature starts decreasing, and the heat must undergo all its treatments and arrive at the caster before it turns solid. Temperature is only relevant between these two events. The bottom part shows the level of the liquid metal at the caster as it is waiting for the heat to arrive: if the caster is in the middle of a program, then the heat must arrive before the liquid level reaches 0.

The last typical variation on job-shop is the running time production, where some activities have already been scheduled. Keeping in mind that steel making is a 24 hours per day operation, in practice, there are always fixed activities in the schedule: when adding activities required by a command from a customer, the schedule is not remade from scratch ; instead, some activities that may be happening while the algorithm is re-computing a new schedule are fixed.

Example 2. This example is depicted on Fig. 4. There are two programs represented with gray and white activities. There are two casters, the gray program must be scheduled on caster 1 and the white program on caster 2. The converter and the degasser can each process two heats in parallel. Each activity has duration 1 to simplify the example. There is a maximum delay of 2 from exit of converter to entry into caster to avoid solidifying metal. Notice how heat number 3 of the gray program is scheduled at the last possible moment to avoid solidifying. On the right, an unexpected event occurred: heat 2 of the white program took more time than expected at caster. Unfortunately, heat 3 of the white program was started at the converter and heat 1 is already being treated at caster 1, so no matter the schedule, heat 3 of the gray program can never arrive on time at the caster.

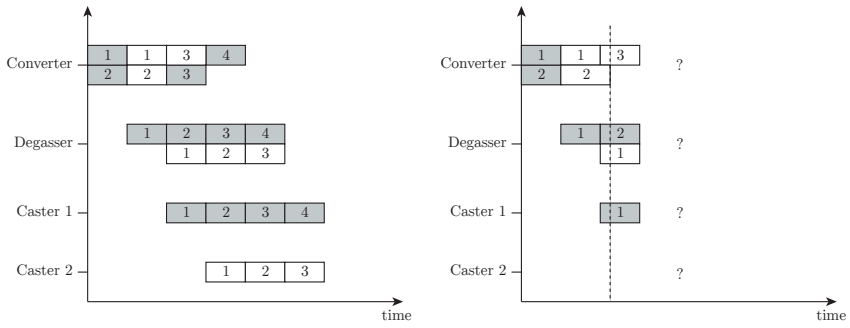

Fig. 4. Representation of Example 2.

Instance-Specific requirements There may be additional constraints depending on the plant and its casting facilities (converter, degasser, etc).

Although facilities may be cumulative (able to treat several heats in parallel) with fixed capacity, some might require that sub-parts of the activities do not overlap. In particular, referring to Fig. 5, the converter activity of a heat is made of three internal continuous parts: Loading, Converting and Tapping. The converter has two chambers and can be seen as a resource with capacity 2.

However, there is only one oxygen blower, so that the actual conversion part can not overlap the conversion part of another. While the loading and tapping parts may still overlap with the same conditions as a resource of capacity 2, the non overlapping constraint induced by the oxygen blower prevents the two activities of chamber 2 to be scheduled consecutively: some gap must be introduced to prevent conversion parts from overlapping.

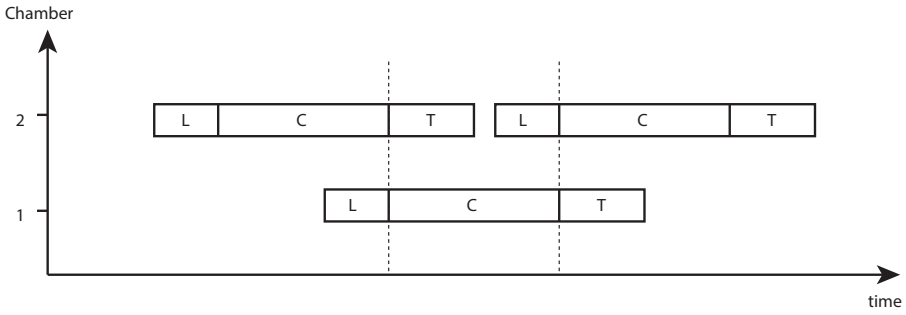

Fig. 5. Illustration of the disjunctive aspect for sub-part of the converter activities.

Generally a program can be treated only in a specific caster, but some activities might have alternatives, i.e. they can be treated indifferently in one facility or another (say they might use vacuum degasser number 1 or 2 indifferently). However, another activity may require vacuum degasser number 1 only, distinguishing the two facilities and preventing us from using a single cumulative constraint for both degassers.

Some facilities might be unavailable for a fixed interval of time for maintenance ; in flexible cases, the maintenance activity may be schedulable. Furthermore, casters require a setup time in-between programs.

An incident may force an activity to take longer than scheduled, or may break down some facility. In such an event, activities that have not yet happened have to be re-scheduled just for feasibility, since temperature and continuous casting constraints may become violated, see Example 2. In the event where a schedule is infeasible due to a continuity violation at a caster, a program may be split in two parts, even though it will probably interfere with the subsequent operations on the slab (typically the subsequent cutting will not be able to stop at a whole number of smaller cylinders). If the temperature constraint of a heat will be violated, the liquid metal can be re-heated in some facilities or even recycled in the converter. In these exceptional cases, the schedule has to be heavily penalized ; we do not take the possibility of splitting programs into account here.

3 Using CP for caster scheduling

In order to satisfy clients at reasonable development costs, PSI needs a framework that enables the coding of algorithms that yield good feasible solutions at industrial scales while requiring little tuning to minimize the development time. The framework should also be expressive enough to allow direct encoding of side constraints. We believe CP offers these advantages thanks to its high level declarative modeling. Furthermore, CP has proved competitive for scheduling applications, mainly due to effective filtering algorithms for global scheduling constraints [18, 10] ; Large Neighborhood Search (LNS) [15] makes it scalable [8, 12, 3, 11]. Consequently, we believe CP is a good candidate for solving the CCSP.

3.1 Two sources of difficulty

Two other problem-specific arguments support the choice for CP on this problem:

1. The continuous casting problem is very constrained, making it sometimes difficult to find a feasible solution.
2. Bottleneck resources are instance dependent, making it difficult to build generic heuristics.

These two points are discussed next.

A very constrained problem. There is a basic strategy to ensure continuity of heats of a same program at the caster: make heats ahead of time, then deliver stored heats at the caster. The major obstacle to this strategy is the temperature constraint, that prevents us from creating heats too long in advance. Recall that the temperature constraint is ensured by allowing a maximum delay from the exit of the converter to the entry at the caster, representing a maximum initial temperature. This tension between the desire to produce in advance to ensure continuity and the just-in time aspect induced by the temperature constraint may cause some instances to be unsatisfiable or at best difficult to solve.

Instance-dependent bottlenecks. As in most scheduling problems, some resources behave more or less like a bottleneck in the schedule. An interesting aspect of the continuous casting scheduling problem is that the bottleneck resource(s) are very instance-dependent. Let us approximate the metal as a continuous flow going through facilities instead of being discretized in heats. Then one can think of facilities as having a continuous throughput, say in tonne/minute; each step of the transformation process is taken care of by one or several facilities, amounting to a total throughput of the step. The step with the smallest throughput is a bottleneck that limits the maximum throughput of the whole factory. From experience, this approximation still holds when the liquid metal is discretized as heats (remind that this is a physical constraint due to the metal being contained, moved and treated in ladles). Identifying the bottleneck facility/resource

is a valuable indicator in order to construct good feasible solutions. Indeed, those resources will be used at maximal throughput in a good solution, and a partial instantiation that satisfies the resource constraint of the bottleneck facility can most likely be extended into a complete feasible schedule. The key factors impacting bottleneck resources are the wide range of parameters possible in each step of the process. For instance, a customer may want a cylinder of small diameter, and thus the throughput at the corresponding caster will be decreased enough to move the bottleneck from another step to the casting step. If the customer wants a higher quality metal, a step corresponding to some secondary treatment will take longer and have a smaller throughput, making it the new bottleneck.

3.2 Alternative/Existing approaches

Although we were not able to find exactly the same problem description in the literature, the problem described in [1] presents many similarities with our CCSP: although the central problem is similar, the objective function and the modeling of facilities are different.

The authors decompose the resolution in two phases. First an Ant Colony Optimization (ACO) metaheuristic is used to sequence the jobs using MATLAB. Then a second phase assigns starting time with CONOPT non-linear optimization solver. In [17], the authors describe a decomposition approach for solving a similar problem combining Lagrangian relaxation, dynamic programming and heuristics.

Although similar local search, decomposition and heuristics have been engineered in the past for other problems, PSI finds it requires much manpower for tuning heuristics and designing moves. Recall that the CCSP problem is strongly constrained, it is thus difficult to design efficient feasible moves manually. LS was also evaluated as a difficult approach to maintain on this CCSP because requirements change according to both the specificities of each production plant and the typical set of programs to schedule.

MILP requires time discretization, and the logical constraints used lead to rather weak linear relaxations. The current solution developed by PSI uses MIP but it required a lot of simplifications (such as time bucketing) and tuning, just to come up with solvable instances. Moreover, these simplifications generally lead to sub-optimal final solutions, when compared to solutions found by CP.

4 Modeling using CP

Our model for the CCSP uses standard resource constraints for scheduling with additional side constraints where needed mainly to impose offsets, set-up times and precedence constraints between activities.

4.1 Model overview

The activities to schedule are structured hierarchically in heats, programs and casters: an activity belongs to a heat, a heat to a program, and a program to a caster.

- $\forall c$ caster, the activities of programs $p_1, \ldots, p_{n_c}$ at the caster must be scheduled in a predefined order, separated by a setup time of the caster.
- $\forall p$ program, the activities of heats $h_1, \ldots, h_{n_p}$ of p happening at the caster must be scheduled in listed order, and be contiguous.
- $\forall h$ heat, activities $a_1, \ldots, a_{n_h}$ of the heat must be scheduled in listed order.

Every activity a has a fixed duration d_a, a set of resources R_a it can be scheduled on, and a delay t_a that modelizes the temperature constraint. Every resource has an associated capacity, ranging from 0 to $+\infty$. The demand of our activities is always 1.

Example 3. Fig. 6 represents a schedule respecting all given constraints. Every resource has a capacity limit, e.g. here 1 for the casters, 2 for the converter, $+\infty$ for the transports, etc. Activities of the same program have the same color, we single out the white program in this example. Caster order constrains the white program to be scheduled before the blue one. Program order forces the activities of the white program at the caster to be consecutive. Heat order constrains the order of activities of the same heat, here the converter activity of white-1 must happen before its stew activity, which must happen before its first transport, etc. Temperature constrains the gap between the converter and caster activities to be smaller than some value, given as an input.

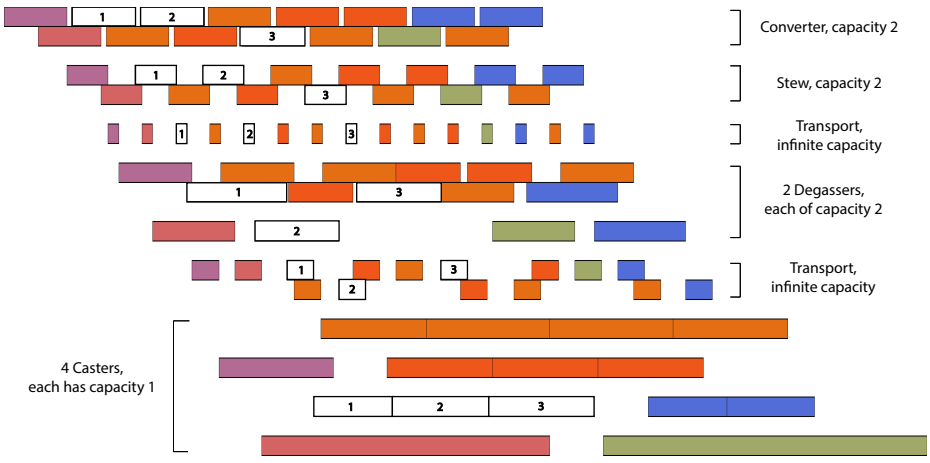

Fig. 6. A small example of continuous casting schedule.

4.2 CP model

The pure model uses conventional scheduling constraints for disjunctive and cumulative resources (see [2] for more information on scheduling constraints). The temperature constraint is modeled with a fixed time delay depending on the heat.

Variables. We write the decision variables in boldface. They are, for each activity a, start $\mathbf{s}_a$, duration $\mathbf{d}_a$, end $\mathbf{e}_a$, and resource $\mathbf{r}_a$. The set of all activities equipped with their decision variables is written $\mathbf{A}$. Durations are fixed, the initial domain for starts and ends is $\mathbb{N}$, and resources are initialized according to user specifications, for instance, it is expected, for a resource variable of activity a at caster c, that the initial domain of $\mathbf{r}_a$ is $\{c\}$.

Constraints. There are two types of constraints in this problem, resource limitations and time dependencies:

- Resource constraints. For every resource r, depending on its capacity $C(r)$, we add a constraint :
  - $C(r) = 0$: the resource is not available, add the constraints $\forall a \in A, \mathbf{r}_a \neq r$
  - $C(r) = 1$: $\mathbf{disjunctive}(\mathbf{A})$
  - $C(r) > 1$: $\mathbf{cumulative}(\mathbf{A}, C(r))$
  - $C(r) = +\infty$: the resource does not constrain the problem, no constraint added
- Time Dependencies. We write $h(c)$ the activity of heat h at caster c, and $\texttt{setup}_c$ the setup time that must separate programs at caster c.
  - Caster order. $\forall c$ caster, $\forall p_i, p_{i+1}$ consecutive programs of c with $p_i = h_1 \dots h_n$ and $p_{i+1} = h'_1 \dots h'_{n'}$, $\mathbf{e}_{h_n(c)} + \texttt{setup}_c \leq \mathbf{s}_{h'_1(c)}$.
  - Program order. $\forall p$ program, $\forall h_j, h_{j+1}$ consecutive heats of p, $\mathbf{e}_{h_j(c)} = \mathbf{s}_{h_{j+1}(c)}$.
  - Heat order. $\forall h$ heat, $\forall a_k, a_{k+1}$ consecutive activities of h, $\mathbf{e}_{a_k} \leq \mathbf{s}_{a_{k+1}}$.
  - Temperature. $\forall h = a_1 \dots a_n$, $\mathbf{e}_{a_1} + delay_h \geq \mathbf{s}_{a_n}$

Objective. The goal of the optimization here is to minimize the sum of the completion times at casters, so if $\texttt{last}(c)$ is the last activity at the caster c, the objective value to minimize is $\sum_c \mathbf{e}_{\texttt{last}(c)}$. This forces the resources to be free earlier, to deal with new customer demands that are unknown at scheduling time: a makespan objective would only be relevant if all jobs were known in advance.

Search Heuristic. The problem sizes are so large (more than 500 activities) that there is little hope to complete the search on real instances. Designing custom heuristic and search strategies is thus crucial for CCSP. An important aspect for PSI customers is the anytime behavior of the application. The operator should have a solution available quickly, and the application can not afford to return no solution. To this end, we designed a search heuristic behaving similarly to a

greedy manual schedule construction minimizing the risk of any backtrack before reaching the first feasible solution. Recall that one important constraint is that activities of a same program at a caster should be contiguous. We order the heats (jobs) in a static order to ensure the feasibility of the continuity property. This order on the jobs is obtained as follows (see Fig. 7):

1. Caster activities are placed without any gap between them (unfortunately this schedule for caster activities is surely not feasible because of capacity limitations at the converter, degasser, etc).
2. Programs are numbered according to their earliest starting time (see numbers 1 to 7 on top of the programs on Fig. 7)).
3. Heats are numbered increasingly inside a program by numbering first programs with lowest starting time (see numbers 1 to 18 in each heat on Fig. 7)).

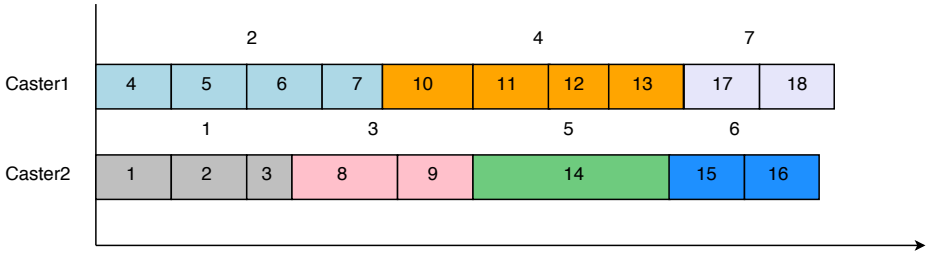

Fig. 7. Job ordering strategy obtained by first ordering programs, then heats inside each programs. Heats in a same program have the same color.

Then for a given heat (job) activities are scheduled facility by facility, starting at the converter and finishing with the caster activity, in their program order. Because the domains are huge (duration are specified in seconds) a binary split search is used instead of a labeling search. This static search is randomized by applying some random modifications on the activity durations when creating the program ordering.

Although the previous strategy is very good to obtain quickly feasible solutions, it is too conservative to obtains high quality values for the sum of completion times. Indeed, this strategy may introduce large gaps between the programs at the casters.

We use a Large Neighborhood Search (LNS) to improve the incumbent solution. Some structure of the current best solution is kept by restricting a randomized partial order schedule between the programs at the caster (inspired from [6]). At each LNS restart we randomly choose between two searches:

- The safe search with static (randomized) ordering, or
- A schedule or postpone search (also called SetTimes) described in [9] on all the activities of the problem.

SetTimes is a search that tries to assign early activities at their earliest starting time (EST), postponing on backtrack the scheduling of activities that failed until their EST changes: it only considers them again when their EST change from propagation. This search exploits dominance properties that do not hold for our problem. Indeed, it may be interesting in our case to postpone activities on the converter to satisfy the itemperature constraints. Thus SetTimes is not necessarily complete on our problem. However, this search proves very useful in practice to minimize the sum of completion times when it finds a feasible solution.

5 Experiments

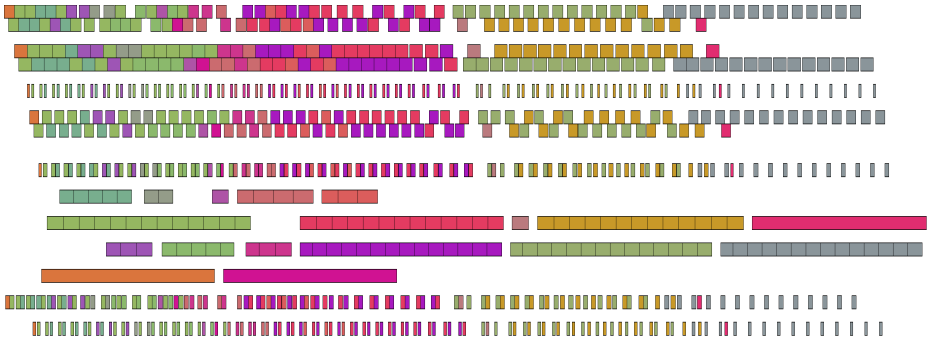

Fig. 8. A real-life schedule.

Table 1 shows the results obtained for 48 CCSP instances. Those instances were generated as variants of 2 real-life instances coming from a customer of PSI. The horizon varies from 12 to 48 hours of operations to schedule, where 36 hours is a good enough time horizon for the industrial setting.

The number of activities given in the table does not take into account subpart activities, in this particular instance 3 resources need subparts, our modeling codes them as additional activities ; the number of activities the solver actually has to handle is roughly the number in the table times 1.5.

Durations of activities were artificially changed to make the bottleneck resource change, allowing us to test the robustness of the search. In table 1, "step" is the bottleneck resource (Converter or STEW), and "extension" is a fixed number of seconds added to the duration of each activity on this resource, making the step slower and more bottleneck-like.

Example 4. Fig. 5 shows an example of a real-life schedule produced by our model. The two bottom lines are subpart activities (as explained on Fig. 5). This schedule is well optimized since activities are very well packed on the bottleneck

					Safe	SetTimes	LNS
instance	limit	activities	step	extension	objective	objective	objective
Instance1	12	234	Converter	600	274620	**250740**	**250740**
				900	308280	270060	**265200**
				1200	342720	**298140**	**298140**
			STEW	600	253980	222600	**221640**
				900	255180	223800	**221640**
				1200	282300		**221640**
	24	436	Converter	600	493560	440760	**437760**
				900	570720	**474480**	**474480**
				1200	630420	515940	**513900**
			STEW	600	448440	**378060**	382441
				900	449640	**379260**	383344
				1200	482100		**424500**
	36	598	Converter	600	588660	540720	**536641**
				900	718560	**610440**	**610440**
				1200	760920	658860	**652920**
			STEW	600	539820	**469440**	475800
				900	541020	**470640**	477181
				1200	**573480**		574140
	48	736	Converter	600	658260	610680	**610441**
				900	776940	**676380**	**676380**
				1200	861900	759660	**757981**
			STEW	600	614580	**553020**	559080
				900	615780	**554160**	559980
				1200	649200		**589140**
Instance2	12	240	Converter	600	294258	**268228**	268583
				900	318596	284783	**279656**
				1200	358828	296848	**290128**
			STEW	600	266398	**243448**	**243448**
				900	271672	**251478**	**251478**
				1200	282533	**264348**	**264348**
	24	420	Converter	600	427218	**421996**	**421996**
				900	547138		**462369**
				1200	609786	**482013**	492857
			STEW	600	388098	**344727**	347307
				900	386397	**352757**	**352757**
				1200	414408		**374121**
	36	576	Converter	600	514408	509266	**475486**
				900	641972		**579460**
				1200	740546	**687380**	691115
			STEW	600	475288	**431917**	436177
				900	473587	**439947**	**439947**
				1200	501598		**461311**
	48	666	Converter	600	595276	590134	**557004**
				900	723262		**580207**
				1200	863598	**782701**	873943
			STEW	600	556156	**512876**	516145
				900	554455	**520815**	**520815**
				1200	582466		**542179**

Table 1. Results on 48 CCSP instances testing 3 different search strategies with a timeout of 30 seconds.

resources. For the first half of the schedule, the second facility is the bottleneck, then the casters become the bottleneck.

A time-out of 30 seconds is given to the algorithm[3] and we compare the final objective function (sum of completion times) using different searches:

- A "Safe" search as the one described above diving quickly toward feasible solution.
- A SetTimes search on the whole set of activities.
- A LNS search using a partial order schedule relaxation and alternating between the two previous searches on each LNS restart.

The LNS search obtains the best objective most of the time on 34/48 instances. The SetTimes search obtains the best solutions on 25/48 instances but is not able to find any feasible solution for 10/48 instances. As expected the "safe" search is always able to find a feasible solution, but the quality of the objective is generally worse compared to the two previous searches. When the LNS search is not the optimal one, it is reasonably close to the SetTimes results and it has the main advantage to always produce a feasible solution. This search is thus the most robust and should be preferred for the final deployment of the application.

6 Conclusion and future work

We have described the continuous casting scheduling problem (CCSP) for steel production and introduced a complete CP model and search strategy using LNS for solving this problem with good any-time behavior. Although the application is still in the prototyping phase, the high quality results obtained using CP and the ease of maintaining and modifying the CP model should allow to deploy it in the near future at some of PSI's customers.

As a future work, we would like to experiment the Variable Objective Large Neighborhood Search (VO-LNS) introduced in [13]. We believe VO-LNS might be a good strategy to focus on a different restricted number of casters at each LNS restart. We may also try to optimize activities according to a rolling horizon scheduling strategy. This should allow us to consider a limited number of activities, fix some of the earliest scheduled ones before sliding the horizon and considering a few more activities. Finally we believe dominances and/or redundant constraints may be inferred to improve the filtering on this problem.

References

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

[3] 30s was chosen as a realistic wait an operator can allow, 300s would be unrealistic.

2. Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer, 2001.
3. Tom Carchrae and J Christopher Beck. Principles for the design of large neighborhood search. *Journal of Mathematical Modelling and Algorithms*, 8(3):245–270, 2009.
4. Alan M Frisch, Ian Miguel, and Toby Walsh. Modelling a steel mill slab design problem. In *Proceedings of the IJCAI-01 workshop on modelling and solving problems with constraints*. Citeseer, 2001.
5. Antoine Gargani and Philippe Refalo. An efficient model and strategy for the steel mill slab design problem. In *Principles and Practice of Constraint Programming–CP 2007*, pages 77–89. Springer, 2007.
6. Daniel Godard, Philippe Laborie, and Wim Nuijten. Randomized large neighborhood search for cumulative scheduling. In *ICAPS*, volume 5, pages 81–89, 2005.
7. Stefan Heinz, Thomas Schlechte, Rüdiger Stephan, and Michael Winkler. Solving steel mill slab design problems. *Constraints*, 17(1):39–50, 2012.
8. Philippe Laborie and Daniel Godard. Self-adapting large neighborhood search: Application to single-mode scheduling problems. *Proceedings MISTA-07, Paris*, pages 276–284, 2007.
9. Claude Le Pape, Philippe Couronné, Didier Vergamini, and Vincent Gosselin. Time-versus-capacity compromises in project scheduling. In *Proceedings of the Thirteenth Workshop of the UK Planning Special Interest Group*, 1994.
10. Arnaud Letort, Mats Carlsson, and Nicolas Beldiceanu. A synchronized sweep algorithm for the k-dimensional cumulative constraint. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 144–159. Springer, 2013.
11. Jean-Noël Monette, Yves Deville, and Pascal Van Hentenryck. Just-in-time scheduling with constraint programming. In *ICAPS*, 2009.
12. Dario Pacino and Pascal Van Hentenryck. Large neighborhood search and adaptive randomized decompositions for flexible jobshop scheduling. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 1997–2002. AAAI Press, 2011.
13. Pierre Schaus. Variable objective large neighborhood search: A practical approach to solve over-constrained problems. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)-2013*, 2013.
14. Pierre Schaus, Pascal Van Hentenryck, Jean-Noël Monette, Carleton Coffrin, Laurent Michel, and Yves Deville. Solving steel mill slab problems with constraint-based techniques: Cp, lns, and cbls. *Constraints*, 16(2):125–147, 2011.
15. Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming—CP98*, pages 417–431. Springer, 1998.
16. steel.org. The online resource for steel. Accessed: 2014-04-20.
17. Lixin Tang, Peter B Luh, Jiyin Liu, and Lei Fang. Steel-making process scheduling using lagrangian relaxation. *International Journal of Production Research*, 40(1):55–70, 2002.
18. Petr Vilım. *Global constraints in scheduling*. PhD thesis, PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science and Mathematical Logic, KTIML MFF, Universita Karlova, Malostranské námestı 2/25, 118 00 Praha 1, Czech Republic, 2007.
19. Wikipedia. Continuous casting — Wikipedia, the free encyclopedia, 2013. Accessed: 2014-04-20.