Journal of Artificial Intelligence Research 66 (2019) 1–32 Submitted 01/2019; published 09/2019

Solving the Torpedo Scheduling Problem

Martin Josef Geiger M.J.GEIGER@HSU-HH.DE
Helmut-Schmidt-University
University of the Federal Armed Forces Hamburg
Holstenhofweg 85, 22043 Hamburg, Germany

Lucas Kletzander LKLETZAN@DBAI.TUWIEN.AC.AT
Nysret Musliu MUSLIU@DBAI.TUWIEN.AC.AT
Christian Doppler Laboratory for Artificial Intelligence
and Optimization for Planning and Scheduling
DBAI, TU Wien, Karlsplatz 13, 1040 Vienna, Austria

Abstract

The article presents a solution approach for the Torpedo Scheduling Problem, an operational planning problem found in steel production. The problem consists of the integrated scheduling and routing of torpedo cars, i. e. steel transporting vehicles, from a blast furnace to steel converters. In the continuous metallurgic transformation of iron into steel, the discrete transportation step of molten iron must be planned with considerable care in order to ensure a continuous material flow.

The problem is solved by a Simulated Annealing algorithm, coupled with an approach of reducing the set of feasible material assignments. The latter is based on logical reductions and lower bound calculations on the number of torpedo cars.

Experimental investigations are performed on a larger number of problem instances, which stem from the 2016 implementation challenge of the Association of Constraint Programming (ACP). Our approach was ranked first (joint first place) in the 2016 ACP challenge and found optimal solutions for all used instances in this challenge.

1. Introduction

This paper investigates a planning and scheduling problem in steel production that was proposed in the 2016 ACP (Association for Constraint Programming) challenge (Schaus et al., 2016). The aim is to provide optimal solutions for the Torpedo Scheduling Problem where the transport of hot metal across various zones needs to be optimized. The solutions should fulfill various hard constraints regarding material flow, time deadlines, capacity and duration when moving through the zones of the steel production plant. The optimization criteria include the minimization of the number of vehicles (torpedoes) and the time spent for a chemical process called 'desulfurization'.

To the best of our knowledge the Torpedo Scheduling Problem proposed in the 2016 ACP challenge (Schaus et al., 2016) has only recently been considered in the literature. Our solution approaches that were ranked first and third in this challenge were published in conference proceedings (Geiger, 2017b; Kletzander & Musliu, 2017). After the completion of the challenge, a solution approach based on Logic-based Benders Decomposition using Mixed-Integer and Constraint Programming was proposed (Goldwaser & Schutt, 2017, 2018). The authors report optimal solutions for instances used in the ACP challenge and also apply their method for other larger instances.

The current paper is an extension of our previous conference papers (Geiger, 2017b; Kletzander & Musliu, 2017). In particular, the current work improves our solution approach that was ranked first (joint first place) in the ACP challenge competition by combining it with a reduction approach that prunes the search space. Furthermore, the experimental section is significantly extended and methods are evaluated on a larger set of instances that were generated with the problem generator, proposed by the challenge organizers.

Overall, the main contributions of this article comprise:

- The proposal of a new approach to reduce the set of possible material assignments prior to solving the problem by a search technique. This approach is based on temporal and logical reductions, as well as lower bounds regarding the number of torpedoes in use. This approach enables the reduction of the solution space and thus leads to a more effective use of the proposed search techniques.
- The search for the optimal solutions by a proposed multi-stage Simulated Annealing algorithm adapted for the provided lexicographic evaluation function. The first round of Simulated Annealing focuses on the primary goal of optimizing the number of torpedoes while the second round deals with the secondary goal of reducing the desulfurization time. The algorithm is designed to efficiently apply a large number of moves in a short time by emphasis on efficient move calculations. Various parameters for the algorithm were determined by empiric evaluation.
- The combination of both approaches and the evaluation of our methods on the ACP challenge instances and additional instances generated by a randomized generator. Our method was able to generate optimal solutions for all instances used in the challenge, and we show that the reduction procedure has a significant impact on the efficiency of the search procedure for the problem at hand.

2. Related Work

The Torpedo Scheduling Problem investigated in this paper was solved by teams that participated in the 2016 ACP challenge (Schaus et al., 2016). The most successful approaches proposed in the competition included a local search based approach based on Simulated Annealing (Kletzander & Musliu, 2017), a mathematical programming approach based on state-expanded networks (Römer, 2018) and a Branch-and-Bound technique (Geiger, 2017b). All these methods were used on the limited set of avaliable instances from the competition.

Recently, Goldwaser and Schutt (2017, 2018) proposed a new exact method based on Logic-Based Benders Decomposition. Their solution method consists of Benders decomposition that includes solving of a master problem and scheduling problems. In the master problem the lexicographic objective is optimized, and the oxygen converter events and unmatched blast furnace events are assigned to torpedo runs and emergency pit trips, respectively. A MIP formulation is used for the master problem. The rest of the problem is then decomposed into sub-problems, which are solved by a Constraint Programming Solver.

In this phase the desulfurization time is minimized. Based on the solutions obtained for sub-problems, the method concludes that the optimal solution has been reached, or computes infeasibility cuts (if some sub-problems are infeasible) that are then used to re-optimize the MIP, or adds optimality cuts (if some sub-problems require extra desulfurization time) to force the objective to consider extra desulfurization time. The proposed method was evaluated on the competition instances and larger generated instances. This was the first published method that could prove the optimality of the ACP 2016 Challenge instances. Our method used in the competition was able to find optimal solutions, but there was no guarantee that these solutions are optimal. According to Römer (2018), he also used an exact method in the competition, but the method has not been published, yet.

2.1 Related Problems in Steel Production

In the wider area of steel production, various related problems have been reported in the literature. The molten iron allocation problem, modeled as a parallel machine scheduling problem (Tang, Wang, & Liu, 2007) and the molten iron scheduling problem, modeled as a flow shop problem (Huang, Chai, Luo, Zheng, & Wang, 2011; Li, Pan, & Duan, 2016), deal with assignments of torpedoes to machines. Various torpedo scheduling problems are defined for planning the transport of hot metal with the focus on vehicle routing across a network of rails, and the objective is to minimize transportation times (Kikuchi, Konishi, & Imai, 2008; Deng, Inoue, & Kawakami, 2011; Liu & Wang, 2015). Further production stages of steel making are also considered, e. g. steelmaking – continuous casting (Tang, Zhao, & Liu, 2014) which comes after the stage considered in this paper.

2.2 Simulated Annealing

The technique to simulate the physical process of annealing was introduced by Kirkpatrick, Gelatt, and Vecchi (1983) and is a widely used technique in many applications (Dowsland & Thompson, 2012; Pham & Karaboga, 2012). Applications of multi-stage algorithms can be found, e. g., in the domain of vehicle routing problems with time windows. These applications range back to Homberger and Gehring (1999) and also include application of Simulated Annealing (Bent & Van Hentenryck, 2004), however, only for one stage. Several of these problems share a primary goal to minimize a number of vehicles, but pursue very different secondary objectives.

3. Problem Definition

In this section, we first present the industrial context of the problem, followed by the constraints and formal representation we use in this article.

3.1 Context and Movements of Torpedoes

The Torpedo Scheduling Problem considers the transportation of liquid iron from a blast furnace to a set of converters, in which the conversion of iron into steel (of a desired quality) is carried out. Figure 1 depicts this phase within the wider context of steel production: Hot molten iron is transported by means of torpedo cars, running on tracks. Torpedo scheduling therefore presents the link between the iron making and the secondary refinement of iron within the steel making.

It is noteworthy and possible to see that both the preceding, upstream production phase, i. e. the reduction of iron ore to molten iron within the blast furnace, as well as the succeeding casting of steel are continuous processes that cannot be interrupted. Conversely, the transportation and the conversion phases are batch production steps.

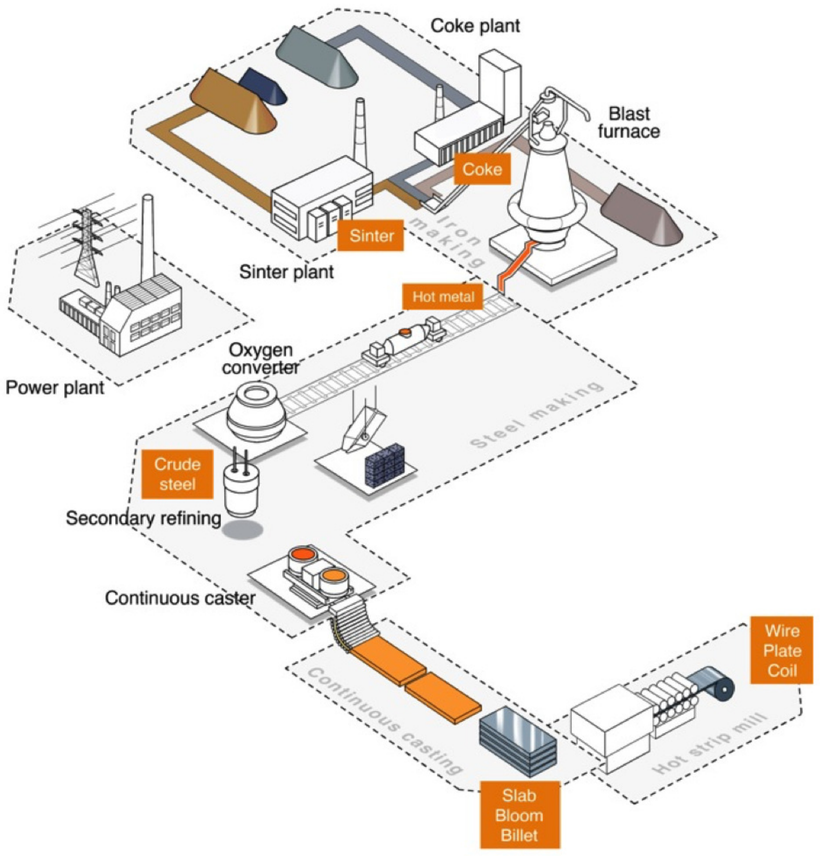


Figure 1: Illustration of steel production plant (Schaus et al., 2016)

Figure 2 depicts the usage of torpedo cars in more detail. Coming from an empty buffer, which in principle represents a waiting zone, they approach the blast furnace, shown on the right, where liquid iron is poured into. The next step consists of a buffer zone. Here, the torpedoes may be re-ordered, or simply parked for a limited time before moving further. A desulfurization phase is reached afterwards, in which the sulfur level of the liquid iron may be reduced, if necessary. Finally, the converters are reached, and the molten iron is transferred into these production facilities. The corresponding empty torpedo then moves back to the empty buffer.

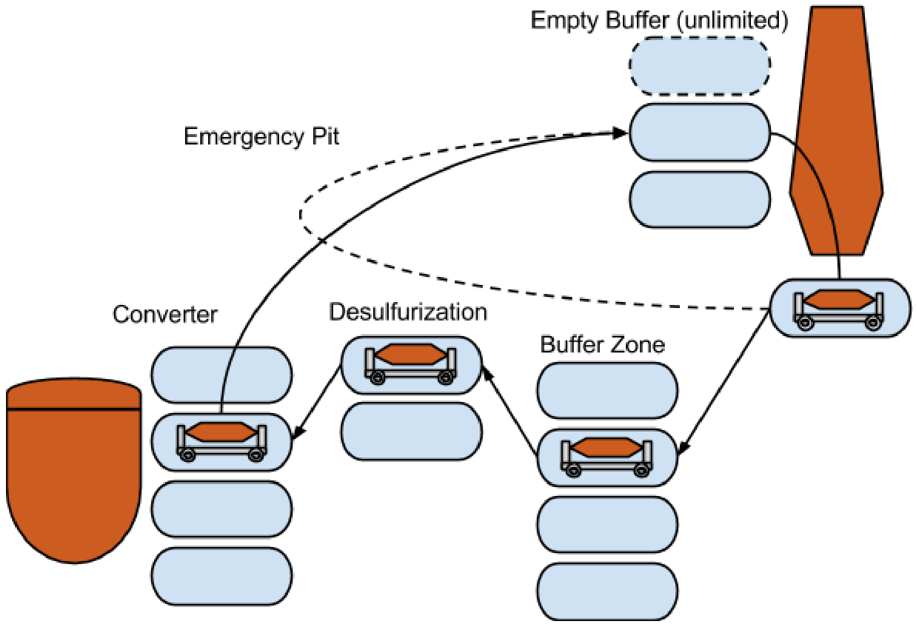


Figure 2: Torpedo rotation (Schaus et al., 2016)

Besides this general rotation, an emergency process may be triggered, depicted in Figure 2 by a dashed arc. In this process, excessive material is discarded into an emergency pit. This happens in phases in which the blast furnace produces more material than can be accepted by the downstream converters, thus balancing the material flow.
In the real-world application of the Torpedo Scheduling Problem, we can expect that such emergency cycles are scarce, as the capacities of the production stages of a steel plant are typically well-balanced. Besides, the material moved into the emergency pit may be re-used later, so this process presents a short-term solution only, and does not correspond to a waste of expensive resources.

More formally, the movements of torpedoes through the system can be described by means of an activity-on-node network as depicted in Figure 3. Three processes p become apparent: Process 1 on the left applies to any torpedo moving through the facility and comprises the activities from the empty buffer to the blast furnace. Then, the torpedoes either move on to the converters (process 2), or discard the material in the emergency pit (process 3). In each case, they return to the waiting area for empty cars after completing their activities.

Let $\mathcal{P}$ be the set of processes, and $\mathcal{K}$ be the set of activities. For every $p \in \mathcal{P}$ and $k \in \mathcal{K}$ we denote the starting time of each activity k of process p with s_{pk}, its end with e_{pk}, and its duration with d_{pk}. Note that $e_{pk} = s_{pk} + d_{pk}$, $\forall p, k$. Moreover, each activity k starts immediately after the completion of its direct predecessor k' of process p': $s_{pk} = e_{p'k'}$.

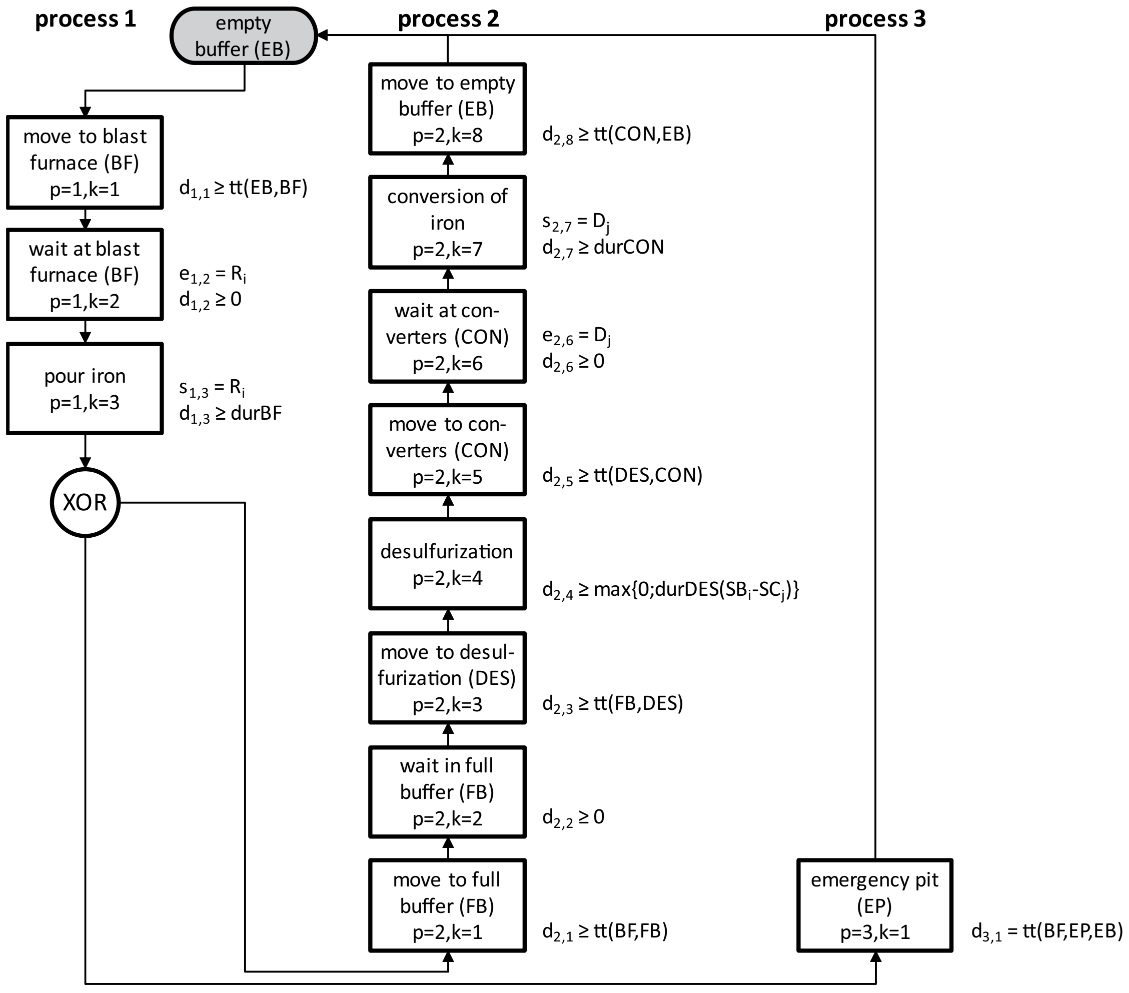

process 1
empty buffer (EB)
process 2
process 3
move to blast furnace (BF) p=1,k=1
$d_{1,1} \geq tt(EB,BF)$
wait at blast furnace (BF) p=1,k=2
$e_{1,2} = R_i$
$d_{1,2} \geq 0$
pour iron p=1,k=3
$s_{1,3} = R_i$
$d_{1,3} \geq durBF$
XOR
move to empty buffer (EB) p=2,k=8
$d_{2,8} \geq tt(CON,EB)$
conversion of iron p=2,k=7
$s_{2,7} = D_j$
$d_{2,7} \geq durCON$
wait at converters (CON) p=2,k=6
$e_{2,6} = D_j$
$d_{2,6} \geq 0$
move to converters (CON) p=2,k=5
$d_{2,5} \geq tt(DES,CON)$
desulfurization p=2,k=4
$d_{2,4} \geq \max\{0;durDES(SB_i-SC_j)\}$
move to desulfurization (DES) p=2,k=3
$d_{2,3} \geq tt(FB,DES)$
wait in full buffer (FB) p=2,k=2
$d_{2,2} \geq 0$
move to full buffer (FB) p=2,k=1
$d_{2,1} \geq tt(BF,FB)$
emergency pit (EP) p=3,k=1
$d_{3,1} = tt(BF,EP,EB)$

Figure 3: Activity-on-node formulation of the torpedo rotation

3.2 Time Constraints

The continuous nature of the preceding and succeeding production steps within steel production results in tight time constraints for the scheduling of the torpedoes. The material at the blast furnace must be picked up at given times. We denote these material release dates with R_i, with i as an index of the operations at the blast furnace, $i = 1, \ldots, n$, where n is the number of release dates. Conversely, the material must reach the converters at pre-defined due dates, denoted as D_j, with j representing the operations at the converters, $j = 1, \ldots, m$, where m is the number of due dates at the converters.

As introduced above, Figure 3 illustrates the two possible schedules for torpedo cycles. In the above mentioned emergency cycle (process 3), the iron is picked up at the blast furnace at $s_{1,3} = R_i$. Note that the torpedo may arrive prior to the actual material transfer, and spend an additional time $d_{1,2}$ waiting at the blast furnace. Overall, the durations are bounded by minimum times / problem parameters.

The more common movements of torpedoes to the converters are shown in Figure 3 by process 2. Here, the start of pouring the iron into the torpedo is again set to $s_{1,3} = R_i$, and the transfer of the iron into the converter is defined by $s_{2,7} = D_j$. Again, the durations of the activities must assume minimum values, which stem from travel times ('tt') between places in the production plant or given minimum processing times for pouring the liquid iron.
The minimum duration spent in the desulfurization phase, $d_{2,4}$, depends on SB_i and SC_j, which denote the sulfur level of the iron coming from the blast furnace (SB_i), and going to the converter (SC_j). In the available data, possible values are from 5 (high sulfur level) to 1 (low level), and a reduction of one step results in a minimum desulfurization time of *durDES*. It follows that the duration of the desulfurization step, denoted as $d_{2,4}$ must be: $d_{2,4} \geq \max\{0; durDES \cdot (SB_i - SC_j)\}$.

For a torpedo picking up iron at the blast furnace at R_i and delivering it to the converters at D_j, it follows that the total duration of all activities in between cannot exceed the length of the timespan $D_j - R_i$. Formally, this can be expressed by computing a lower bound LB on all activities between the two dates R_i and D_j, which is derived by taking, for each activity, its minimum duration. In the particular example of our application this means that the following equation must hold:

$$D_j - R_i \geq LB\left(d_{1,3} + \sum_{k=1}^{6} d_{2,k}\right) \tag{1}$$

3.3 Capacity Constraints

The resources that can hold torpedoes are limited. This means that, at any given time, a maximum number of torpedoes must not be exceeded for each resource, which presents, due to the size of the torpedoes and the limited space within the production facility, a practical yet obvious side constraint. For the problem at hand, we denote the maximum number of torpedoes (i. e., the number of available slots) for the stages with *sFB*, *sDES*, and *sCON*, respectively. Moreover, at any given time, at most a single torpedo might travel between two stages. The movements to the emergency pit are assumed to be unlimited.

Appropriate mathematical models for such aspects are found in the area of resource constrained project scheduling, and we here refer to such problems (Geiger, 2017a).

3.4 Feasible Material Flow

Part of the scheduling is to ensure an uninterrupted material flow, which means that the demand at the converters must be met at all times.

Let us denote the assignment of material coming from the blast furnace at R_i to the converters at D_j with x_{ij}. The variables x_{ij} assume a value of 1 if the material flow assignment $i \rightarrow j$ is made, and 0 otherwise. As described above, material that does not reach the converters is moved into the emergency pit. This is mathematically captured by a variable y_i that assumes, if this is the case, a value of 1, and 0 otherwise. Then, the following expressions must hold.

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j \tag{2}$$

$$\sum_{j=1}^{m} x_{ij} + y_i = 1 \quad \forall i \tag{3}$$

$$x_{ij} \in \{0;1\} \quad \forall i,j \tag{4}$$

$$y_i \in \{0;1\} \quad \forall i \tag{5}$$

Expression (2) ensures that all material requirements at the converters are met, and Expression (3) ensures that the material deliveries from the blast furnace are at most used once.

It follows that a feasible solution cannot exist for data sets with $n < m$. For the activity-on-node network introduced in Figure 3 this means that there will be exactly m processes 2 (material deliveries to the converters), and $n - m$ processes 3 (discarding of material in the emergency pit).

3.5 Optimality Criteria

The above described problem comes with two optimality criteria. First of all, the maximum number of torpedoes in use is to be minimized. This clearly presents a strategic / tactical planning problem and is justified by the fact that investments in torpedo cars are typically cost intensive. As a secondary objective, the time spent in the desulfurization phase is to be minimized. The two criteria are lexicographically ordered, with the minimization of the maximum number of torpedoes preceding the minimization of the total desulfurization time.

Minimizing the maximum number of torpedoes in use means rotating them through the system as fast as possible. This insight is based on the observation that once a torpedo returns from carrying out its activities, it can be re-used and sent on the next cycle. Technically, the empty buffer can be modeled as a stack of torpedoes from which we always take the topmost torpedo to serve the next job at the blast furnace.
Hence, returning quickly is the key to the minimization of the number of torpedoes used over time. In between, as little time as possible should be spent, resulting in assignments $i \to j$ with possibly 'close' due dates R_i and D_j, i. e., with a small gap in between the release of the material R_i and the consumption D_j, and ultimately resulting in short cycles. Intuitively, this strategy of rotating torpedoes makes sense, as the overall time spent in the system is foremost determined by these two dates.
Moreover, it makes sense to arrive as late as possible at the blast furnace, avoiding waiting times ($d_{1,2}$) before the liquid iron is poured into the torpedo.

Note that this intuition of fast rotating torpedoes through the system is reflected by the subsequent introduction of lower bounds, see the following section.

4. Lower Bounds and Reductions of Material Assignments

Prior to solving the actual scheduling problem we investigate ways of reducing the search space to facilitate an easier solution of the problem. The underlying idea is to prune the set of feasible assignments as much as possible, resulting in a reduction of the solution space.

4.1 Modeling Approach

The further investigations are based on modeling the problem as a tripartite graph with three parallel time lines, depicted in Figure 4. In this formulation, the due dates at the blast furnace and at the converters are nodes at given times R_i and D_j, shown in the middle and bottom time axes.

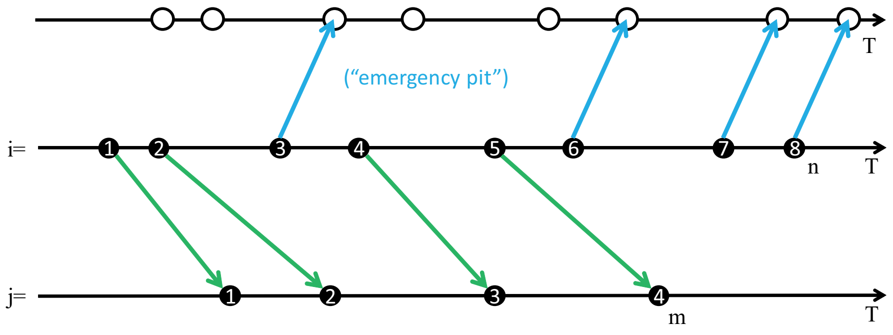


Figure 4: Tripartite graph formulation

As a necessary condition for a feasible solution, each D_j is linked to exactly one preceding R_i, resulting in an arc in the graph formulation. The length of the arc over the time axis, plus the time needed for traveling to and waiting at the blast furnace $(d_{1,1} + d_{1,2})$ and the time spent at and after the converters $(d_{2,7}+d_{2,8})$ represents the cycle time of a torpedo through the system. With LB denoting a lower bound on the minimum duration of an activity, Expression (6) gives the minimum time interval for the material flow to the converters.

$$[R_i - LB(d_{1,1} + d_{1,2}), D_j + LB(d_{2,7} + d_{2,8})] \quad \forall i, j \mid x_{ij} = 1 \tag{6}$$

Material flows from the blast furnace that do not serve the converters are moved into the emergency pit, followed by traveling back to the empty buffer. As this is assumed to be an unlimited resource, the time for this activity is a constant $(durBF + tt(BF, EP, EB))$. In the tripartite graph formulation, those processes are represented by arcs pointing to nodes on a third axis, given on the top of Figure 4. More formally, the minimum time interval of all torpedoes moving into the emergency pit is given in Expression (7).

$$[R_i - LB(d_{1,1} + d_{1,2}), R_i + durBF + tt(BF, EP, EB)] \quad \forall i \mid y_i = 1 \tag{7}$$

It can be seen that a feasible solution of the mathematical model corresponds to an introduction of n distinct arcs, m of which are linked to the nodes in the bottom axis, and $n - m$ linked to nodes on the top axis. The number of torpedoes used in the solution corresponds to the maximum number of parallel (simultaneous) processes over the entire planning horizon T. It follows that the effort to compute the number of torpedoes in use is only dependent on the number of arcs n.

4.2 Temporal Reductions

Per problem definition, material cannot travel backwards in time. Therefore, and for each i, j:

$$\text{If } R_i + LB\left(d_{1,3} + \textstyle\sum_{k=1}^{6} d_{2,k}\right) > D_j \text{ , then } x_{ij} = 0, \text{ and } x_{ij} \in \{0,1\} \text{ otherwise.}$$

Note that this follows directly from Expression (1). Again, the minimum durations are taken for the computation of the lower bound LB.

Based on this straight-forward computation, potential assignments can be excluded. By applying the reasoning presented above for each i, j, we derive, for each j, a set of potential deliveries i, and conversely, for each delivery i, a set of potential converter consumptions j. For the example in Figure 4, the values might assume:

$$\begin{array}{lcl}
j = 1 & : & i \in \{1,2\} \\
j = 2 & : & i \in \{1,2\} \\
j = 3 & : & i \in \{1,2,3,4\} \\
j = 4 & : & i \in \{1,2,3,4,5,6\} \\
i = 1 & : & j \in \{1,2,3,4\} \\
i = 2 & : & j \in \{1,2,3,4\} \\
i = 3 & : & j \in \{3,4\} \\
i = 4 & : & j \in \{3,4\} \\
i = 5 & : & j \in \{4\} \\
i = 6 & : & j \in \{4\} \\
i = 7 & : & j \in \{\} \\
i = 8 & : & j \in \{\}
\end{array}$$

4.3 Logical Reductions

Additional logical considerations can further prune the set of possible assignments / schedules. First of all, material deliveries that cannot reach a converter can be automatically assigned to the emergency pit. In the example above, this applies to $i \in \{7,8\}$.

Then, we may compute subsets of j which share the same subset of possible values of i. As each converter demand requires a distinct material delivery, this constitutes an *alldifferent* constraint and Hall sets (Hall, 1935) can be applied. If both subsets are of the same cardinality, it follows that all values of i can be discarded from the potential assignments outside the computed subset on j. In the example above, this is found for $j \in \{1,2\}$. Both material demands j can be satisfied by either $i = 1$ or $i = 2$. Therefore, $i = 1$ and $i = 2$ cannot be used to satisfy any other demand than $j = 1$ or $j = 2$:

$$\begin{array}{lcl}
j = 1 & : & i \in \{1, 2\} \\
j = 2 & : & i \in \{1, 2\} \\
j = 3 & : & i \in \{\cancel{1}, \cancel{2}, 3, 4\} \\
j = 4 & : & i \in \{\cancel{1}, \cancel{2}, 3, 4, 5, 6\} \\
i = 1 & : & j \in \{1, 2, \cancel{3}, \cancel{4}\} \\
i = 2 & : & j \in \{1, 2, \cancel{3}, \cancel{4}\} \\
i = 3 & : & j \in \{3, 4\} \\
i = 4 & : & j \in \{3, 4\} \\
i = 5 & : & j \in \{4\} \\
i = 6 & : & j \in \{4\}
\end{array}$$

Such logical checks may recursively be applied until no further domain reduction is possible. Further, instances might be proven infeasible by the Hall sets if demand cannot be satisfied, indicated by the subset on i having lower cardinality than the subset on j.

4.4 Lower Bounds

The subsequent calculations are based on a special property in the data sets, i. e., that emergency cycles are shorter than the lower bound on 'regular' cycles:

$$d_{3,1} < LB\left(\sum_{k=1}^{6} d_{2,k}\right) \tag{8}$$

The left-hand side of the Expression (8) stems from the fact that the emergency pit is assumed to be an unlimited resource. Therefore, the actual processing times can assume their minimum values. The right-hand side is obtained by summing up the minimum processing times and minimum required travel times between resources. For the available data sets, the latter yields larger values than what we find on the left-hand side. On a more practical note, having to deal with data sets with such values makes sense, as the actual production facilities in steel production are rather big plants with long transportation tracks.

It follows that replacing regular cycles with emergency cycles never overestimates the minimum usage of torpedoes at any given time. More formally speaking, the time interval given in Expression (7) never exceeds the one of (6).

Type ECO – emergency cycles only. Following the above mentioned property (see Expression (8)), a naïve lower bound on the usage of torpedoes can be obtained by discarding all material deliveries, replacing them with n emergency cycles. With respect to the mathematical model, this implies that we relax all the constraints given in Expression (2). As a result, all y_i-variables assume a value of 1, and all x_{ij} are 0. The formal computations are depicted in the following Algorithm 1.

ALGORITHM 1: Computation of bound ECO

```
set $y_i = 1 \quad \forall i$;
set $x_{ij} = 0 \quad \forall i, j$;
```

Figure 5 illustrates the result of this first approach. Clearly, this type of a lower bound does not yield the best possible lower bound, as any other cycle than an emergency cycle will be of a longer duration. We merely mention this approach in order to be complete.

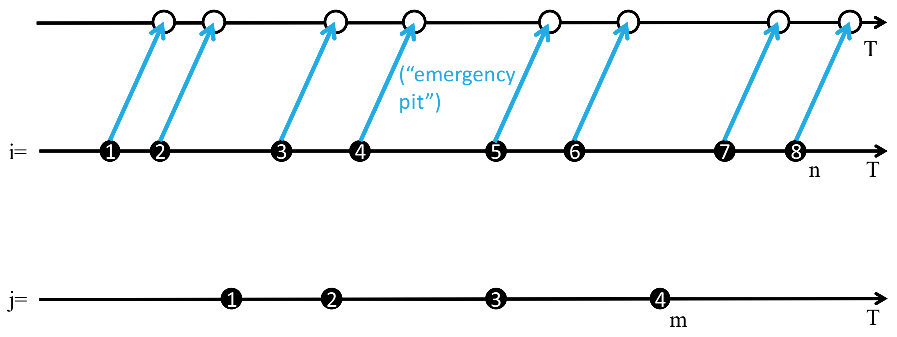


Figure 5: Emergency cycles only

Type CCF – converter coverage forward. In comparison to the bound of type ECO, a better lower estimate is obtained by connecting each material demand to its closest possible delivery. In the illustration of Figure 6, and in the pseudo-code provided in Algorithm 2, this is done in increasing order of j, not permitting multiple assignments of values of i. In this sense, some, but not all side constraints of Expression (2) are reconsidered. At the same time, multiple assignments, as forbidden by Expression (3), are respected. As discussed above, assigning material from the blast furnace to the closest possible consumptions at the converter does not over-estimate the duration of torpedoes in the production system. Recalling the time interval as given in Expression (6), it follows that this is a feasible way of computing a lower bound.

ALGORITHM 2: Computation of bound CCF

```
initialize $y_i = 0 \quad \forall i$;
initialize $x_{ij} = 0 \quad \forall i, j$;
for all material requests at the converters $j, j = 1, \ldots, m$ do
    obtain largest index $i$ of material coming from the blast furnace that can be assigned to $j$;
    if $\sum_{k=1}^{m} x_{ik} = 0$ then
        set $x_{ij} = 1$;
    end
end
for all material coming from the blast furnace $i, i = 1, \ldots, n$ do
    if $\sum_{j=1}^{m} x_{ij} = 0$ then
        set $y_i = 1$;
    end
end
```

In brief, this typically leads to some demands j that cannot be connected to their closest delivery i, see $j = 2$. Unassigned material deliveries from the blast furnace are handled in emergency cycles (which do not overestimate the torpedo usage).

Type CCB – converter coverage backward. Similar to the type CCF, the material coverage at the converters can be done in decreasing order of j, with the result illustrated in Figure 7. Although this is a slight variation of the bound type CCF, it is, depending on the instance data, different enough to be tested.

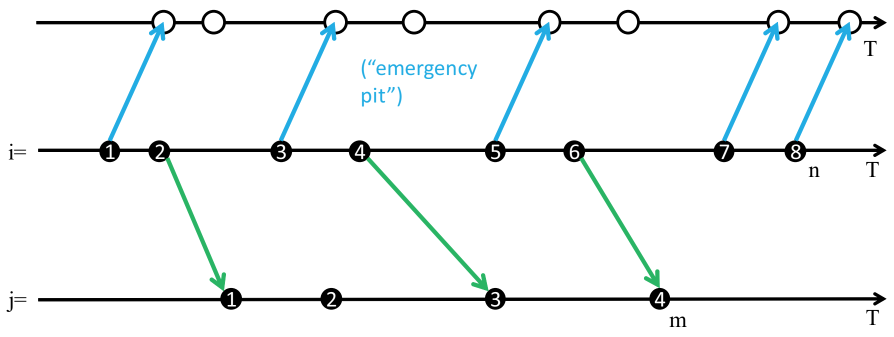


Figure 6: Bound type CCF – converter coverage forward in time

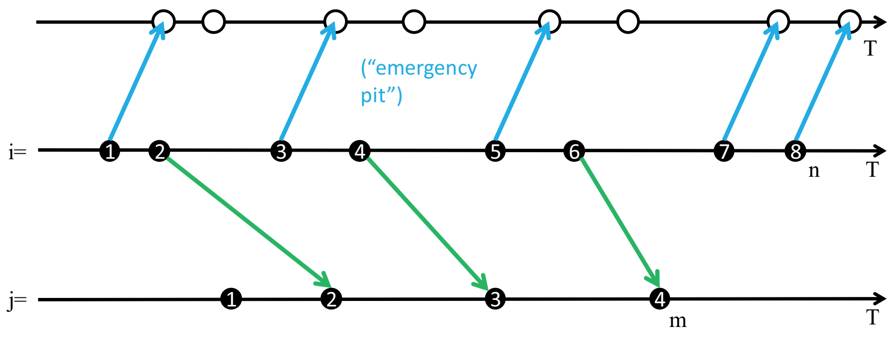


Figure 7: Bound type CCB – converter coverage backwards in time

Note that the pseudo-code for this procedure is almost identical to the one given in Algorithm 2, with the order of j reversed from m to 1 instead of $1, \dots, m$.

Type CCMA – converter coverage with multiple assignments. Yet another lower bound can be found by discarding all emergency cycles and assuming deliveries for each material supply at the blast furnace. Algorithm 3 describes the logic of this procedure, and Figure 8 gives the result for the above introduced example. Clearly, we here relax side constraint (3), but re-introduce (2): The material demand at the converter is fully satisfied, however, at the price of possibly assigning material from the blast furnace multiple times.

ALGORITHM 3: Computation of bound CCMA

```
initialize $y_i = 0 \quad \forall i$;
initialize $x_{ij} = 0 \quad \forall i, j$;
for all material requests at the converters $j, j = 1, \dots, m$ do
    obtain largest index $i$ of material coming from the blast furnace that can be assigned to $j$;
    set $x_{ij} = 1$;
end
```

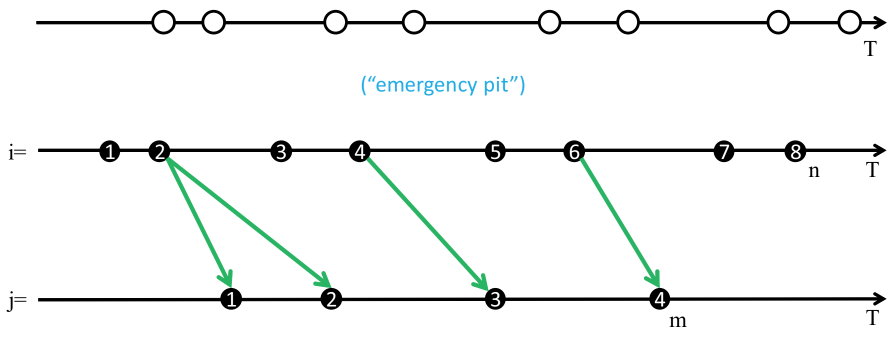


Figure 8: Bound type CCMA – converter coverage with multiple assignments

4.5 Reductions

Each of the above introduced procedures yields, for each time-step, a lower estimate on the number of torpedoes in use. The maximum over all time-steps therefore presents a lower bound on the minimum number of torpedoes that are required in a feasible solution. In the example above, we can see that any solution will employ at least two torpedoes.

On a more technical note, we can see that the computation of the lower bounds introduced above requires the sorting of release dates and due dates. Therefore, the complexity is bounded by the one of sorting, which is $n \log n$ and $m \log m$, respectively. This underlines the applicability of the ideas, even for larger data sets.

In the following, let us assume we are able to identify a feasible solution making use of the same number of torpedoes as given by the lower bound computations, i. e., an optimal solution for the minimization of the maximum number of torpedoes. Then (and only then), additional reductions of material assignments are possible.

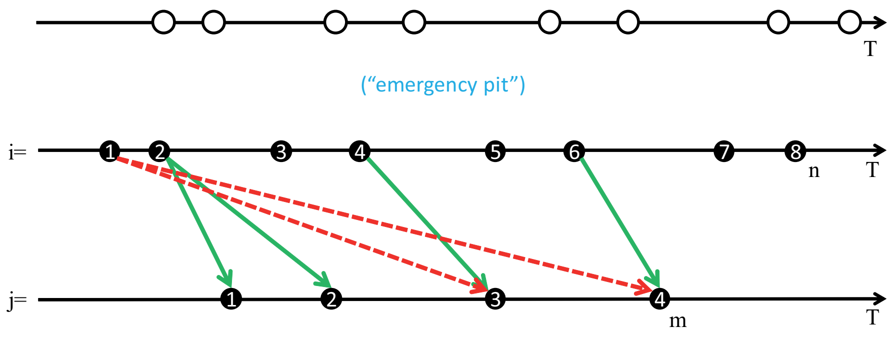


Figure 9: Reductions of assignments

Figure 9 illustrates this principle. For the purpose of our example, we assume that a feasible solution with using two torpedoes exists. In this case, and using the computations of bound type CCMA, assignments of $i = 1$ to $j \in \{3, 4\}$ would result in a solution with a larger number of torpedoes. As a corollary, such connections can be discarded from the search space.

5. A Simulated Annealing Approach for the Problem at Hand

To solve the given problem we introduce a new two-stage Simulated Annealing based algorithm. Our algorithm is described below. We first formulate solution candidates and a set of equations to efficiently track data determining the solution quality. We then provide the overview of the algorithm and describe the motivation for the design of individual parts and their contributions for improving the solution quality. Emphasis is put on the design of the objective functions to use the power of the two-stage approach.

5.1 Representing a Solution Candidate

We model a possible solution with the concept of a torpedo cycle that is tied to exactly one blast furnace due date and either one converter due date or the emergency pit. The total amount of cycles is equal to the number of blast furnace due dates n. The amount of runs targeting the converter is equal to the number of converter due dates m and the rest is targeting the emergency pit.

As a solution candidate we use an array of such torpedo cycles that contains the correct number of cycles targeting the converter and emergency pit. The algorithm is designed to respect the order of zones, prevent violations of zone durations and transition times and to always meet the blast furnace due dates. Other violations are possible in the algorithm, in particular $s_{2,7} = D_j$ and $d_{2,4} \geq \max\{0; durDES \cdot (SB_i - SC_j)\}$ can be violated as well as the capacity constraints. Instead of preventing them they are monitored and penalized by the objective functions used in the Simulated Annealing algorithm.

5.1.1 Monitoring Constraint Violations

Constraint violations are tracked by maintaining an array of 10 values called *badness* each representing a certain kind of violation that can be individually weighted in the objective function.

The first set of constraint violations regarding the converter demands can be described as follows given the notation that R_C is the set of torpedo cycles targeting the converter:

$$\sum_{r \in R_C} \max\{s^r_{2,7} - D^r_j; 0\} \tag{9}$$

$$\sum_{r \in R_C} \max\{SB^r_i - \left\lfloor \frac{d^r_{2,4}}{durDES} \right\rfloor - SC^r_j; 0\} \tag{10}$$

Equation (9) sums up the delays at the converter across all runs r that miss their deadline, in which case the start time at the converter $s^r_{2,7}$ will be after the time of the assigned converter deadline.

Equation (10) first calculates the final sulfurization level for torpedo cycle r using the level at the blast furnace SB^r_i and the amount of time spent in the desulfurization zone $d^r_{2,4}$. By subtracting the maximum allowed level at the converter SC^r_j it calculates the difference between the maximum allowed level and the actual level and penalizes values above 0 that indicate sulfurization level misses.

For each of the remaining capacity constraints the algorithm maintains an array with the size T equal to the amount of time units in the whole planning period. One such array is maintained for each capacitated zone (cBF, cFB, cD and cC) and for the corresponding transitions ($cBFtoFB$, $cFBtoD$, $cDtoC$ and $cCtoE$). Each element of such an array counts the amount of torpedo cycles using the respective zone or transition at the given point in time.

To track the violations for each of these arrays X the corresponding badness value is calculated by

$$\sum_{0 \leq t < T} \max\{X[t] - maxOccupation_X; 0\} \tag{11}$$

where $maxOccupation_X$ is either one of the given capacities sFB, $sDES$, $sCON$ for the corresponding arrays or 1 in all other cases.

5.1.2 Monitoring Optimization Goals

To keep track of the number of torpedoes another array spanning the entire planning period is used. The array *occupation* counts for each point in time the number of torpedo cycles that are currently active. Therefore the maximum value of this array represents the current value of the main evaluation goal.

However, for this array tracking the maximum of all values is not good enough for use in the objective functions. Therefore, the array *occupationCount* counts the number of elements of *occupation* having a certain value by

$$occupationCount[i] = \text{count}\{t : occupation[t] = i\}\ , \tag{12}$$

e. g. *occupationCount*[1] counts the amount of time points where exactly one torpedo cycle is active. This concept allows to individually weigh specific levels of occupation.

Finally the desulfurization is tracked by *timeDesulf* holding the total amount of desulfurization time for all torpedo runs. Further the array *desulfMismatch* keeps track of the difference between the sulfurization levels of the metal provided at the blast furnace compared to the required levels at the converter. It calculates

$$desulfMismatch[i] = \text{count}\{r : SB_i^r - SC_j^r = i\} \tag{13}$$

for $0 < i < 5$, e. g., *desulfMismatch*[3] counts the amount of torpedo runs that need at least 3 desulfurization steps in order to be feasible. This again allows individual weights for specific difference values.

5.2 The Simulated Annealing Algorithm

The Simulated Annealing process is done in two rounds using almost the same parameters, but very different objective functions. The general design of each round uses the usual process of Simulated Annealing as shown in Algorithm 4.

ALGORITHM 4: Simulated Annealing

```
generateInitialSolution();
for round = 0...1 do
    value = evaluateSolution();
    t = value / t_fraction;
    for outer = 0...outerIterations do
        for inner = 0...innerIterations() do
            chooseMove();
            newValue = evaluateSolution();
            if shouldAccept() then
                acceptMove();
            else
                abortMove();
            end
        end
        t *= t_factor;
    end
end
```

5.2.1 Heuristic Generation of the Initial Solution Candidate

The concept of *generateInitialSolution*() as shown in Algorithm 5 is to take the ordered lists of blast furnace and converter due dates and pair them according to this ordering. In *assignNextConverter*(i, c), assigning blast furnace due date i to the next unassigned converter due date in c, the torpedo cycles are initialized to spend the minimum amount of time in the desulfurization zone that allows them to meet the sulfurization level requirement of the assigned converter. They arrive at the converter just at the time of the due date or too late in case this pairing is actually infeasible and spend any required waiting time in the full buffer.

ALGORITHM 5: *generateInitialSolution*()

```
Data: Ordered lists of blast furnace due dates bf and converter due dates c
blocked = -1;
for $i$ = 0...$|bf| - 1$ do
    if $i \neq$ blocked $\wedge$ countUnassignedConverters() $\neq |bf| - i$ $\wedge$ $\neg$converterMiss(i+1,c) then
        assignEmergencyPit(i);
    else
        assignNextConverter(i,c);
        if converterMiss(i,c) then
            blocked = previousEmergencyPitAssignment(i);
            if blocked $\geq 0$ then
                i = blocked-1;
                unassignLaterConverters(i);
            end
        end
    end
end
```

As long as emergency pit runs are available (condition *countUnassignedConverters*() $\neq$ $|bf| - i$), they are set whenever it is possible to put the current blast furnace output to the emergency pit and still transport the next blast furnace output to the current converter due date in time (condition $\neg converterMiss(i+1, c)$). As emergency pit cycles tend to get scheduled a bit too early by this approach and cause converter due date misses a few runs later, the algorithm includes a repair procedure on such due date misses to remove the previous emergency pit run and schedule it later by using *blocked*.

At first, time and space proportional to the total length of the planning period are required as all the capacity tracking arrays need to be initialized. The effort of constructing the initial solution is proportional to the number of blast furnace due dates (actually not linear due to the repair procedure, but in practice still very close) times the duration of a cycle as each cycle needs to be added to the capacity tracking system.

Note that in most cases the initial solution will not be feasible as some degree of constraint violation in regard to missing due dates and capacity constraints is to be expected. However, it is designed to have a structure that produces a small amount of constraint violations while still keeping the execution very fast.

5.2.2 Parameter Tuning

The parameters for the algorithm were carefully chosen by experimental evaluation of various combinations of parameters to increase the performance. In the following the best values used in the final computation for the competition are presented. Reasons for the provided choice are given as well as problems encountered with different values. Unless otherwise stated, changes in most parameters only resulted in small changes in the results.

Some parameters were chosen differently depending on whether the algorithm was executed with or without the reduction procedures, mostly in the cooling scheme. The used values are presented in the evaluation section in such cases.

5.2.3 Iteration Parameters

For each round the algorithm uses a fixed number of outer iterations *outer* that depends on inclusion of the reduction procedures. The selected value should always ensure that the algorithm converges to a stable result.

The temperature was set to start with the initial value of the objective function divided by *t_fraction*. In each outer iteration the temperature is decreased by multiplying with the factor *t_factor*. This choice, especially combined with the starting temperature and the number of inner iterations was one of the more critical choices for the quality of the results and therefore subject of detailed empirical evaluation. The initial solution is constructed in a way to already have a structure limiting the amount of constraint violations. While a certain increase in such violations is expected in the early stage of the Simulated Annealing process to prevent getting stuck too close to the initial solution, keeping the temperature high for too long destroys the structure of the initial solution requiring extensive amounts of repair at lower temperatures that make the overall result worse. On the other hand, dropping the temperature too fast leads to getting stuck in local optima.

The number of inner iterations *innerIterations*() depends on the size of the instance, more precisely it is the number of blast furnace deadlines. This choice was made as the number of possible moves also depends on this value. The second round experiments showed an increase by a factor of 4 to be beneficial.

5.2.4 Efficient Moves

We proposed four moves that are chosen randomly with certain probabilities in *chooseMove*(). The first move is a switch between the assigned converter deadlines for two torpedo cycles. It is chosen with a probability of 0.4 in the first round and 0.6 in the second round. The rather high probability is due to the fact that this is the move with the highest impact on the structure of the solution. The selection of the two cycles is not randomized, but actually a sensitive choice. The reason is that choosing two cycles at very different time points in the whole planning period will likely not be a good move as large deadline misses can be expected. On the other hand, only switching closely adjacent runs will likely end up in local optima too soon. Therefore, once the first cycle is selected, the second is chosen within the 10 closest blast furnance deadlines before or after the first cycle, as this showed to provide good results.

Additionally cycles are locally improved after such a move to reduce the amount of converter due date misses and sulfurization level misses. First the time spent at the desulfurization station is set to exactly the amount of time needed to pass the required maximum sulfurization level at the converter. Second if the converter due date is missed, the time at the full buffer is reduced just enough to get the due date, or to 0 if the deadline miss is larger than the full buffer time.

The other moves consist of changing the time spent at the full buffer (probability 0.4 in the first round and 0.2 in the second), the time spend at the desulfurization zone (probability 0.1), or the time spend at the converter (probability 0.1). As these moves are intended to change the internal structure of a cycle towards a good feasible solution, new values for the respective times are chosen randomly within limits preventing a converter due date miss if possible.

The key concept in tracking the capacity and goal data is to allow very fast calculation of changes triggered by moves in the algorithm and therefore to be able to try a lot of moves in a short amount of time. For the moves it is necessary to first compute the effects of the move and then either accept or reject it. Therefore, every move is reflected by first creating copies of the torpedo cycles that are affected by the move. Then all tracking data is updated by removing the original runs and adding the changed copies. In case the move is rejected, the copies are removed again and the originals added back to the tracking data. In case it is accepted, the copies replace the originals in the array of torpedo runs.

The important aspect is that all badness and goal tracking data can be updated incrementally. The sums or counts in (9), (10) and (13) allow easy removal and addition of torpedo cycles. For (11) and (12) only the parts of the arrays X and *occupation* affected by the currently changed torpedo runs need to be updated, the effects on the sum and the count can easily be computed incrementally again.

Using this principle it is possible to update all tracked data in time that just depends on the duration of the respective torpedo cycles that are removed or added. This duration is usually very small compared to the whole time span of the planning period and is independent of the number of torpedo cycles.

Moves are accepted by *shouldAccept*() if their evaluation yields a better or equal result according to *evaluateSolution*() or else with a probability of $\exp\left(\frac{value-newValue}{t}\right)$.

5.2.5 Selection Bias

As the main objective in the first round is to reduce the maximum number of torpedoes used, optimization in areas with already low numbers of torpedoes might not be relevant for the result at all while a single point with a high number determines the value of the result. Therefore, the selection for a move is biased to prefer runs in areas with a high number of torpedoes in use. On the other hand, for the desulfurization time the total sum is relevant, therefore every optimization matters and no selection bias is used in the second round.

5.3 Objective Functions

To use the power of the two-stage approach we proposed specific objective functions for *evaluateSolution*() for each round tailored to the respective goals:

$$\sum_{0\le i<10} w_1[i]\cdot \mathit{badness}[i] + \sum_{i\ge 0} \mathit{occupationCount}[i]\cdot i^k + \mathit{timeDesulf} \tag{14}$$

$$\sum_{0\le i<10} w_2[i]\cdot \mathit{badness}[i] + \sum_{i>\mathit{fixed}} 1000000\cdot \mathit{occupationCount}[i]\cdot i^k + \sum_{0\le i<5} 10000\cdot \mathit{desulfMismatch}[i]\cdot i + \mathit{timeDesulf} \tag{15}$$

Equation (14) denotes the objective function for the first round and (15) the objective function for the second round. Again, the weights were chosen by experimental evaluation.

5.3.1 Constraint Violations

First, both objective functions take into account the constraint violations maintained by *badness*, however, with different weight vectors w_1 and w_2.

Both objective functions use a weight of 100000 for the total converter due date miss time as missing such deadlines potentially indicates structural problems of the solution and therefore is considered a priority for optimization.

For the optimization of the sulfurization level misses the first round again uses a weight of 100000 as it focuses on finding a feasible solution with the least possible amount of torpedoes. The high value ensures the focus on feasibility. For the second round, however, the weight is only 1000 as there is a special part of the objective function that also deals with the sulfurization level misses in more detail.

Capacity constraint violations are all penalized by a weight of 10000 in the first round. Again the values were chosen rather high to focus on feasibility. For the second run weights for capacity misses at specific zones are only weighted by 10, for transitions by 1000. Transitions need to be weighted higher as their maximum occupation of 1 leaves less margin in general, but the weights for the zones were chosen very low to prevent the algorithm to get stuck in local optima. This actually introduces a small probability for constraint violations still present in the final result, however, higher values focused the process more on these constraints in the first place and only afterwards optimizing the desulfurization times within the limits already set by the constraints while the low values allow a kind of parallel optimization of both desulfurization times and capacity violations at a similar pace.

5.3.2 Number of Torpedoes

The next part of the objective functions uses the *occupationCount* array. For the first round each element *occupationCount*$[i]$ is weighted by i^k. This polynomial weighting strategy ensures a strong optimization towards a small number of currently active torpedo cycles at any point in time with a special emphasis on eliminating areas using a high number of torpedoes. This showed to be a successful approach to optimize the first objective. While $k = 2$ is sufficient when the reduction procedures are included, the value needs to be even higher ($k = 4$) without them in order to achieve the best possible results regarding the number of torpedoes.

For the second round the goal of this part of the objective function is completely changed. This part is the reason for choosing to use two separate rounds of optimization in the first place. The key point is the structure of the solution created in the first round. The number of currently active torpedoes is kept as low as possible across the whole time span in order for the optimization to work. However, as only the maximum number of torpedoes counts for the value of the solution and this maximum value might only be reached at a small part of the whole process, this kind of optimization restricts the possibilities to optimize the desulfurization time more than necessary. Section 6.3 will highlight this in the results.

Therefore, the second round memorizes the amount of torpedoes reached in the first round (*fixed*) and sets the weight for every i up to this value to 0. For all i above this value previous weights are increased by a factor of 1000000. This allows free use of any number of torpedoes up to the set limit allowing much more flexibility for the reduction of desulfurization times by utilization of torpedoes that would otherwise be on standby. On the other hand the excessive weights for going beyond this limit ensure that the optimization result from the first round is kept throughout the second round.

5.3.3 Desulfurization

In the first round the objective function adds the optimization of desulfurization times as a low priority goal to the process.

In the second round the desulfurization times are included in more detail to encourage better matching of blast furnace and converter deadlines with respect to their sulfurization levels. To incorporate the difference in initial sulfurization levels compared to the actual converter level demands the array *desulfMismatch* is used. A level miss is weighted by $10000 \cdot i$ where i counts the number of missed levels, therefore linearly penalizing the distance to the required level. Here a range of other methods were tried as well, in particular using polynomial strategies like for the number of torpedoes or also penalizing levels that are lower than required in order to reduce potentially wasteful situations where torpedo cycles with low level are used for converter demands with high maximum level. However, none of these strategies gave better results than the one described above.

Finally, the total desulfurization time is added to the objective function as well. Using a weight of 1, this (actually the overall optimization goal of the second round) is a rather low priority target in the optimization process. This is because putting more emphasis on parts like the sulfurization level mismatch works towards producing an optimal structure of the solution earlier. This is important especially regarding the assignment of converter due dates to the torpedo cycles. As such switches can easily produce at least temporary constraint violations it is beneficial to work on an optimized assignment while the temperature is still high and then focus on optimizations within cycles by shifting times to reduce desulfurization times locally at a later point in the process.

5.3.4 Reduction Procedures

The algorithm as described above can be used without applying the reduction procedures. However, applying these procedures can eliminate a large part of potential blast furnace to converter assignments beforehand and therefore allow a major reduction of the search space. This in turn can be used to greatly reduce the time for application of the Simulated Annealing algorithm while maintaining the same or even slightly better results.

The application is implemented by storing a boolean matrix of possible blast furnace to converter assignments and checking every potential assignment during application of the moves in Simulated Annealing whether it is allowed. Only when the resulting assignments of a move are in the allowed search space the evaluation of the move is triggered.

6. Experimental Investigation

The algorithm was executed on an Intel Core i7-6700K with 4 x 4.0 GHz using a single core. First we present the data sets and evaluate the reduction procedures. The next part evaluates the need for the two-stage algorithm by exploring the structure of a solution in different stages of the algorithm. Then the algorithm is applied to the full data set both with and without utilizing the reduction process.

6.1 Data Sets

An original set of six instances was provided for the ACP 2016 Challenge (Schaus et al., 2016). Those instances were generated by an instance generator that was published along with the configuration files used for the generator. To evaluate our approach on a larger set of instances, we applied the following process to generate more instances.

The features of an instance are given by the configuration file, setting numbers of slots as well as ranges for the durations and by providing the number of blast furnace and converter due dates directly to the generator. Table 1 shows the used configuration files.

Table 1: Instance configuration files

File	*sFB*	*sDES*	*sCON*	Small TT	Medium TT	Large TT	Duration length
01	6	2	3	1-3	10	17	13-27
02	6	2	2	1-3	10	22	11-23
03	6	2	2	1-3	9	20	10-27
04	4	1	2	1-3	6	16	14-17
05	4	1	2	1-3	9	13	11-27
06	4	1	2	1-3	10	27	11-25
high	15	5	5	1-3	6-10	13-27	10-27
long	6	2	2	3-9	18-30	29-81	30-81
narrow	2	2	2	1-3	6-10	13-27	10-27
short	6	2	2	1	2-3	4-9	3-9

Table 2 shows the number of blast furnace and converter due dates for the competition instances.

Table 2: Number of blast furnace and converter due dates

	instance01	instance02	instance03	instance04	instance05	instance06
Blast furnace due dates	850	1500	2200	1000	1800	2500
Converter due dates	800	1400	2100	1000	1780	2350

Now for each existing instance *instance0x* three new instances with the same configuration file $0x$ and the same number of blast furnace and converter due dates are generated. Further each of the configuration files *high*, using 2 to 3 times more slots, *long*, using triple durations, *narrow*, using only a third of the number of slots for the full buffer, and *short*, using durations at roughly a third of the usual values, is combined with each of the sets of due dates to investigate the effect of those features on the solutions.

An additional set of instances is created to investigate the importance of the ratio between the number of blast furnace and converter due dates on the difficulty of the problem. Instance 4 from the original instances has equal numbers of these due dates and turned out to be the easiest original instance to solve. Therefore, using the same configuration file 04 and the same number of 1000 blast furnace due dates, instances with the number of converter due dates set to 999, 995, 990, 975, 950, 900, 750 and 500 are generated as *instance04_ratioY*.

6.2 Effect of the Reduction Procedures

A detailed analysis of the effect of reducing the possible material assignments has been conducted. First, we apply the logical reductions as described in Section 4.3. Then, all assignments based on the lower bound computations (types CCF, CCB, CCMA) are removed. Here, no particular order has to be respected as all types of lower bounds are independent from each other. Once these assignments have been removed (Log-1, Type CCF, Type CCB, Type CCMA), the logical checks can yet once more be run, and additional material assignments might be removed. It terminates after the second application of the logical checks. At this point, no further reductions of variable values are possible. Formally, the procedure is given in Algorithm 6.

ALGORITHM 6: Application of the reduction procedures

```
apply logical reductions based on Hall sets;
if reductions based on the lower bounds (type CCF, CCB, CCMA) then
    re-apply logical reductions;
end
```

Table 3 reports the findings of these experiments: We first give for each instance the number of possible assignments $i \to j$ in column 2. The number of removed assignments by applying the logical tests is shown in the third column, 'Log-1'. Then, columns 4–6 give the number of removed assignments for the different types of lower bound calculations, Type CCF, CCB, and CCMA.

The effect of this second run of the logical checks is reported in column 'Log-2'. Finally, the remaining number of possible $i \to j$-assignments is given in column 8, and the reduction

Table 3: Reductions of possible material assignments

Instance	poss.ass.	Log-1	Type CCF	Type CCB	Type CCMA	Log-2	net	red.
instance01	339506	4779	0	0	122740	0	213127	37.2%
instance01_1	341486	25839	285185	285185	285453	15227	36581	89.3%
instance01_2	341560	15010	169312	169312	170976	829	162649	52.4%
instance01_3	339448	0	282257	282257	284578	2542	44627	86.9%
instance01_high	340193	3190	298114	301921	304251	5991	27718	91.9%
instance01_long	341717	9522	0	0	61200	0	271955	20.4%
instance01_narrow	342332	7155	273433	273433	264132	10396	55151	83.9%
instance01_short	339728	9522	107007	107007	46620	0	195476	42.5%
instance02	1049611	55180	221725	221725	226625	8547	808259	23.0%
instance02_1	1045626	30547	962367	966245	972065	10850	50547	95.2%
instance02_2	1050299	19495	964782	969723	967166	6865	65127	93.8%
instance02_3	1054856	27790	0	224907	228594	0	823052	22.0%
instance02_high	1054967	26406	954919	984951	1001123	7968	41569	96.1%
instance02_long	1045039	11164	0	0	689550	10239	341782	67.3%
instance02_narrow	1053142	25029	440580	440580	442260	4961	588452	44.1%
instance02_short	1053648	26409	483171	483171	484692	0	552180	47.6%
instance03	2316980	0	2166964	2186186	2208911	24205	75436	96.7%
instance03_1	2307282	16764	2199520	2210854	2226926	18314	59146	97.4%
instance03_2	2314182	52175	2218779	2218779	2215054	18471	58946	97.5%
instance03_3	2324378	2099	0	0	1069027	32420	1222855	47.4%
instance03_high	2312085	130277	0	529590	1376628	49686	840301	63.7%
instance03_long	2310500	10485	2157887	2172908	2173538	24586	105365	95.4%
instance03_narrow	2313388	35547	728945	728945	728828	9587	1542934	33.3%
instance03_short	2307957	16764	0	0	716984	0	1587737	31.2%
instance04	500518	499462	444322	444322	450682	0	1056	99.8%
instance04_1	500499	499467	436326	436866	443898	0	1032	99.8%
instance04_2	500496	499474	427153	427153	447344	0	1022	99.8%
instance04_3	500509	499473	437172	437172	440155	0	1036	99.8%
instance04_high	500512	499390	438154	438154	455028	0	1122	99.8%
instance04_long	500497	499481	89600	89600	89900	0	1016	99.8%
instance04_narrow	500500	499479	382606	382984	404708	0	1021	99.8%
instance04_ratio1	500254	212347	462951	464643	473310	21106	1031	99.8%
instance04_ratio2	498760	40887	451685	451685	458403	26938	7661	98.5%
instance04_ratio3	496456	25389	428430	428430	442963	33492	19494	96.1%
instance04_ratio4	489920	4860	432088	432088	446602	18194	22865	95.3%
instance04_ratio5	473646	2844	414319	414858	405249	5119	49789	89.5%
instance04_ratio6	456267	8945	403400	403400	410621	4553	39036	91.4%
instance04_ratio7	381719	3735	191100	191100	0	0	188634	50.6%
instance04_ratio8	242379	1493	230938	231142	206717	93	10117	95.8%
instance04_short	500502	499478	294569	294569	295853	0	1024	99.8%
instance05	1606620	58177	0	0	0	0	1548443	3.6%
instance05_1	1605055	28344	1500791	1503682	1534642	42626	27045	98.3%
instance05_2	1604175	79065	31986	31986	32040	0	1493880	6.9%
instance05_3	1603430	5334	1531455	1531455	1537263	45922	17324	98.9%
instance05_high	1603041	70377	791586	791586	257637	7679	659833	58.8%
instance05_long	1603350	217931	1498030	1498030	1516777	36671	41253	97.4%
instance05_narrow	1604282	5334	1465417	1476834	1498512	44507	60035	96.3%
instance05_short	1601786	142557	656676	656676	695566	42788	823069	48.6%
instance06	2937339	28122	1450224	1450224	1454112	0	1470657	49.9%
instance06_1	2943650	74672	2795463	2795841	2802034	19431	101386	96.6%
instance06_2	2933658	25784	2822590	2822590	2826441	28731	68046	97.7%
instance06_3	2925694	202883	2740734	2740734	2765510	27989	123617	95.8%
instance06_high	2938013	116220	1454007	1812163	2027787	17109	877998	70.1%
instance06_long	2930533	16422	0	0	1200394	2686	1715749	41.5%
instance06_narrow	2943567	37464	502196	502196	503088	222	2406361	18.3%
instance06_short	2939371	74671	0	0	754820	6425	2177439	25.9%

is summarized in column 9 ('red.', in percent). Note that the different tests typically do not remove distinct assignments, but that there is a considerable overlap between the different approaches.

Our analysis has shown that the reductions of the lower bounds Type CCF and Type CCB are overlapping to a large degree. In some cases, they are even identical. This is to be expected, as the logic for constructing the lower bounds is very similar, differing only in the sequence in which the assignments are made. Still, there are some differences, and some material assignments are excluded by one type but not the other. Ultimately, both should be applied as they both contribute.

Cases in which the lower bound computed by our procedures is not found as an upper bound are scarce. In our experiments, this was only the case for instance05. Consequently, the reductions by the lower bound calculations are, here, zero.

We are also able to see that there does not seem a clear pattern of when the concepts are particularly useful and when not, apart from the case of instance04. Here, close to $mn - m$ assignments are removed. Clearly, this relates to the structure of the instances, as $m = n$. Effectively, the tightness of the constraints does not allow for many different material assignments, and this is reflected in the large number of reductions.

Overall, the investigation reveals that the ideas put forward in this article are highly useful for reducing the set of possible material assignments. On average, and after applying all reductions, 72.8% of all potential assignments are discarded, which is quite impressive. Bearing in mind that the computations proposed by us can be done as a preprocessing step, keeping the outcomes for further optimization runs, we believe them to be a valuable contribution for the problem at hand.

6.3 Structure of a Solution

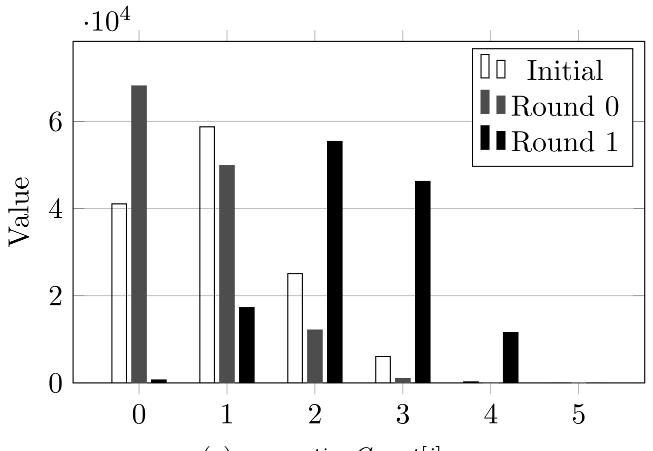
To see the importance of using two rounds of Simulated Annealing, data collected from one particular computation of *instance01* is presented after the creation of the initial solution and after each round of Simulated Annealing. The resulting distribution is similar in all instances, therefore it is only described for *instance01* to highlight the way the algorithm transforms the solution.

Table 4 shows the elements of *occupationCount* for indices 0 to 5 and *timeDesulf* for *instance01*. All values *occupationCount*[*i*] with $i > 5$ are 0.

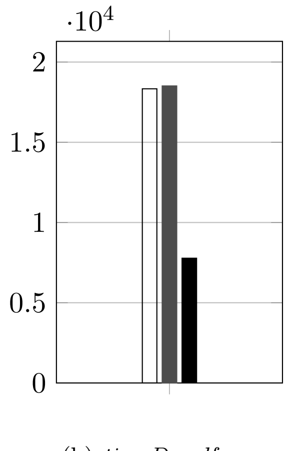
Table 4: Objective values in various stages of the algorithm (*instance01*)

Value	[0]	[1]	[2]	[3]	[4]	[5]	*timeDesulf*
Initial	41077	58754	25050	6091	239	11	18333
Round 0	68141	49830	12136	1053	62	0	18512
Round 1	681	17303	55387	46254	11597	0	7776

The initial solution generated by the heuristic typically uses only a few torpedoes more than the final result, in this case 5 torpedoes are used. However, as to be expected, this solution is infeasible, and capacity constraints are slightly violated at the transitions *FBtoD*, *DtoC* and *CtoE* as well as at the desulfurization zone. Figure 10 shows the distribution of



(a) *occupationCount*[*i*]



(b) *timeDesulf*

Figure 10: Objective values in various stages of the algorithm (*instance01*)

the occupation values. The most frequent occupation at this stage is to have 0 or 1 torpedo active.

After the first round of Simulated Annealing the number of torpedoes was lowered to 4, however, the desulfurization time slightly increased despite the fact that desulfurization is added as a low level optimization goal to this stage. This solution is already feasible and fixes the number of torpedoes used in the final solution. As the figure shows, the distribution for *occupationCount* is shifted as far as possible to the lower indices. The highest number of torpedoes is only used at a very small number of time points. In fact, for almost half the time no torpedoes are active at all.

In the second round of Simulated Annealing this distribution completely changes as the algorithm now permits free use of any number of torpedoes up to 4 in order to optimize the desulfurization time as much as possible. The results show that there is almost no time left without any torpedo on the move while the most frequent occupation shifts to 2 and 3. This allows much more flexibility for optimizing the desulfurization time and results in less than half the time compared to the first round.

6.4 Results

For these results the algorithm was evaluated 20 times on each instance. The evaluation without reduction procedures in Table 5 uses *outer* = 10000, *t_fraction* = 10 and *t_factor = 0.998*, while the evaluation with reduction procedures in Table 6 uses the much faster cooling with *outer* = 1000, *t_fraction* = 1000 and *t_factor* = 0.99.

The column Ab. 1 shows the number of runs where the best known number of torpedoes could not be reached in stage 1. Ab. 2 shows the number of runs where the final result had to be discarded due to constraint violations. These cases typically included a minor capacity violation that could not be resolved.

The column Torp. shows the result for the main objective, the maximum number of torpedoes. The next columns show the best respectively average number of time steps spent in disulfurization as well as the standard deviation for this value. Note that average

Table 5: Results of Simulated Annealing without reductions

Instance	Ab. 1	Ab. 2	Torp.	Best desulf	Avg. desulf	Std. dev.	Time
instance01	2	1	4	7722	7769.88	32.78	257.38
instance01_1	0	0	3	7463	7480.85	12.58	164.40
instance01_2	0	0	4	4743	4752.35	10.02	187.34
instance01_3	0	0	3	5151	5162.05	9.73	165.66
instance01_high	0	0	4	5344	5360.00	13.39	138.89
instance01_long	0	0	4	14184	14281.35	97.22	569.71
instance01_narrow	1	0	3	4914	4924.32	10.00	154.86
instance01_short	0	0	4	1314	1323.60	6.95	112.47
instance02	0	0	4	5302	5318.30	10.90	458.66
instance02_1	1	0	3	14278	14302.32	20.06	387.29
instance02_2	0	0	3	12188	12216.60	33.44	360.96
instance02_3	0	0	4	**8020**	8076.05	25.30	415.99
instance02_high	0	0	4	14823	14845.95	23.03	233.16
instance02_long	0	0	4	34438	34505.60	52.03	946.51
instance02_narrow	0	0	4	9051	9091.10	20.87	411.02
instance02_short	0	0	4	2646	2659.50	8.87	220.47
instance03	11	0	3	27150	27186.11	20.79	620.29
instance03_1	7	0	3	20940	20961.54	14.60	533.40
instance03_2	2	0	3	20800	20820.00	20.00	596.48
instance03_3	0	0	4	14560	14608.00	25.61	688.61
instance03_high	0	0	5	15000	15040.80	26.40	354.67
instance03_long	3	0	3	57230	57296.24	63.75	1084.58
instance03_narrow	0	0	4	14288	14320.50	20.87	579.19
instance03_short	0	0	5	**2860**	2877.85	9.41	329.64
instance04	0	1	3	10676	10676.00	0.00	181.57
instance04_1	0	0	3	12128	12128.00	0.00	174.59
instance04_2	0	0	3	11824	11824.00	0.00	177.67
instance04_3	0	0	3	11856	11856.00	0.00	226.42
instance04_high	0	0	4	12407	12407.00	0.00	167.72
instance04_long	0	0	4	49352	49352.00	0.00	498.38
instance04_narrow	0	0	3	16588	16588.00	0.00	250.48
instance04_ratio1	2	0	3	12144	12144.00	0.00	190.54
instance04_ratio2	7	0	3	11120	11122.46	5.77	186.48
instance04_ratio3	3	0	3	9168	9186.82	13.70	187.73
instance04_ratio4	4	0	3	8176	8192.00	11.31	188.22
instance04_ratio5	1	0	3	6512	6529.68	14.59	195.38
instance04_ratio6	1	0	3	6672	6692.21	12.55	179.92
instance04_ratio7	0	0	4	1824	1836.80	11.97	175.98
instance04_ratio8	1	0	3	1328	1328.84	3.57	100.85
instance04_short	0	0	4	3710	3710.00	0.00	107.34
instance05	4	1	4	16362	16401.67	33.94	520.27
instance05_1	4	0	3	15119	15126.31	7.92	326.68
instance05_2	0	0	4	7826	7840.95	14.98	388.73
instance05_3	7	0	3	13052	13067.00	11.22	421.95
instance05_high	0	0	5	17064	17105.85	32.48	384.10
instance05_long	2	0	3	47520	47566.50	17.79	785.83
instance05_narrow	4	0	3	23730	23733.94	8.20	423.11
instance05_short	1	0	4	5248	5258.11	10.00	269.69
instance06	2	0	4	7755	7788.00	22.30	605.97
instance06_1	5	0	3	26460	26508.60	32.99	699.28
instance06_2	3	0	3	27891	27949.76	36.15	772.33
instance06_3	4	0	3	27513	27582.19	40.46	814.06
instance06_high	1	0	5	14278	14352.11	38.58	460.70
instance06_long	0	0	4	49984	50202.20	120.56	1801.16
instance06_narrow	0	0	4	10387	10418.25	18.13	566.34
instance06_short	0	0	4	6320	6342.00	9.76	414.50

Table 6: Results of Simulated Annealing with reductions

Instance	Ab. 1	Ab. 2	Torp.	Best des.	Avg. des.	Std. dev.	Time	Avg. diff	Time diff
instance01	0	0	4	**7695**	7744.95	32.26	77.33	-0.32%	-69.96%
instance01_1	0	0	3	7463	7473.20	9.91	46.40	-0.10%	-71.78%
instance01_2	0	0	4	4743	4746.40	6.80	53.39	-0.13%	-71.50%
instance01_3	0	0	3	5151	5162.05	12.35	43.30	0.00%	-73.86%
instance01_high	0	0	4	5344	5353.60	12.80	31.85	-0.12%	-77.07%
instance01_long	0	0	4	14184	14331.60	114.92	197.44	0.35%	-65.34%
instance01_narrow	0	0	3	4914	4917.50	7.51	43.71	-0.14%	-71.78%
instance01_short	0	0	4	1314	1419.70	97.81	30.83	7.26%	-72.59%
instance02	0	0	4	5302	5314.35	14.94	104.13	-0.07%	-77.30%
instance02_1	0	0	3	14278	14289.00	13.02	108.61	-0.09%	-71.96%
instance02_2	0	0	3	**12166**	12183.60	16.46	105.79	-0.27%	-70.69%
instance02_3	0	0	4	8027	8064.05	27.66	131.77	-0.15%	-68.32%
instance02_high	0	0	4	14823	14845.95	28.67	67.11	0.00%	-71.22%
instance02_long	0	0	4	**34371**	34558.60	91.38	354.21	0.15%	-62.58%
instance02_narrow	0	0	4	9051	9088.10	22.37	147.47	-0.03%	-64.12%
instance02_short	0	0	4	2646	2661.20	39.07	55.82	0.06%	-74.68%
instance03	0	0	3	27150	27160.00	12.25	205.50	-0.10%	-66.87%
instance03_1	0	0	3	20940	20955.00	16.58	166.41	-0.03%	-68.80%
instance03_2	0	0	3	**20780**	20802.00	14.00	167.38	-0.09%	-71.94%
instance03_3	0	0	4	14560	14592.00	18.33	206.45	-0.11%	-70.02%
instance03_high	0	0	5	**14976**	15015.60	31.54	115.01	-0.17%	-67.57%
instance03_long	0	0	3	57230	57274.25	55.58	397.46	-0.04%	-63.35%
instance03_narrow	0	0	4	**14269**	14369.05	118.47	220.20	0.34%	-61.98%
instance03_short	0	0	5	2870	2944.15	78.66	128.46	2.30%	-61.03%
instance04	0	2	3	10676	10676.00	0.00	36.15	0.00%	-80.09%
instance04_1	0	0	3	12128	12128.00	0.00	33.05	0.00%	-81.07%
instance04_2	0	0	3	11824	11824.00	0.00	32.32	0.00%	-81.81%
instance04_3	0	0	3	11856	11856.00	0.00	33.84	0.00%	-85.06%
instance04_high	0	0	4	12407	12407.00	0.00	34.70	0.00%	-79.31%
instance04_long	0	0	4	49352	49352.00	0.00	42.86	0.00%	-91.40%
instance04_narrow	0	0	3	16588	16588.00	0.00	34.18	0.00%	-86.35%
instance04_ratio1	0	0	3	12144	12144.00	0.00	32.27	0.00%	-83.07%
instance04_ratio2	0	0	3	11120	11120.00	0.00	52.31	-0.02%	-71.95%
instance04_ratio3	0	0	3	9168	9184.80	7.96	60.54	-0.02%	-67.75%
instance04_ratio4	0	5	3	8176	8179.20	8.67	57.78	-0.16%	-69.30%
instance04_ratio5	0	0	3	6512	6520.80	12.87	55.43	-0.14%	-71.63%
instance04_ratio6	0	0	3	6672	6681.65	10.65	49.09	-0.16%	-72.72%
instance04_ratio7	0	0	4	1824	1834.40	10.46	46.21	-0.13%	-73.74%
instance04_ratio8	1	0	3	1328	1329.68	4.91	21.07	0.06%	-79.11%
instance04_short	0	0	4	3710	3710.05	0.22	32.88	0.00%	-69.37%
instance05	3	0	4	**16308**	16360.41	29.97	178.73	-0.25%	-65.65%
instance05_1	0	0	3	15119	15120.30	3.90	109.03	-0.04%	-66.63%
instance05_2	0	0	4	7826	7837.70	12.26	129.69	-0.04%	-66.64%
instance05_3	0	0	3	13052	13053.95	4.64	113.62	-0.10%	-73.07%
instance05_high	0	0	5	**17037**	17082.90	21.09	105.01	-0.13%	-72.66%
instance05_long	0	0	3	47520	47526.60	15.71	277.08	-0.08%	-64.74%
instance05_narrow	0	0	3	23730	23730.00	0.00	145.83	-0.02%	-65.53%
instance05_short	0	0	4	5248	5252.00	5.37	91.76	-0.12%	-65.98%
instance06	0	5	4	7755	7779.93	14.74	199.93	-0.10%	-67.01%
instance06_1	0	0	3	**26433**	26455.95	24.56	231.14	-0.20%	-66.95%
instance06_2	0	0	3	27891	27916.65	30.16	228.74	-0.12%	-70.38%
instance06_3	0	0	3	**27486**	27530.55	33.37	231.97	-0.19%	-71.50%
instance06_high	0	0	5	14278	14312.10	22.52	123.24	-0.28%	-73.25%
instance06_long	0	0	4	**49920**	50218.35	103.74	659.01	0.03%	-63.41%
instance06_narrow	0	0	4	10387	10409.15	15.41	193.21	-0.09%	-65.88%
instance06_short	0	0	4	**6312**	6327.25	9.89	126.25	-0.23%	-69.54%

and standard deviation are calculated on feasible runs reaching the best known number of torpedoes only. The time is given in seconds and represents the average running time of those runs.

Even without application of the reduction procedures this algorithm was able to find the best solutions for the six original instances in the ACP 2016 Challenge, using 50 runs per instance. For the extended evaluation 20 runs where performed per instance. As we mentioned in the related work the first published method that proved the optimality of ACP challenge instances was proposed by Goldwaser and Schutt (2017, 2018). Comparing to this approach, our method was also able to find the optimal solutions for the ACP challenge instances. Results on these instances show that although our method is based on local search and in general can not guarantee the optimality, it is very robust.

The results show that in general the algorithm produces very stable solutions that do not differ much. This is important for practical use as the calculation could be done with only few runs in a short time while still staying within a short distance to the best solutions the algorithm might find given more runs.

6.4.1 Effect of the Reduction Procedures

Several effects can be noticed when applying the reduction procedures. In total 56 instances were evaluated, leading to 1120 runs for each of the result tables. As presented in Table 5, out of those 83, which amounts to 7.4%, did not reach the best known number of torpedoes. In Table 6, this amout reduces to 4 runs, or 0.4%, of all runs, which is a major improvement. Note that the amount of infeasible instances raises from 3 (0.3%) to 12 (1.1%), but still the overall sum of aborts shows a major drop from 86 (7.7%) to 16 (1.4%). The best known number of torpedoes does not change for any of the instances. However, the rate for reaching this best value is significantly improved.

The tables show that the best desulfurization time out of 20 runs is improved for 12 out of 56 instances (avg. −0.2%), while only 2 instances reach a slightly worse result (avg. +0.2%). The corresponding best values are printed in bold in the tables. For the other 42 instances the same best values where reached, however, the number of times the best value is reached out of 20 runs is improved for 30 instances, while it decreased for only 6 instances, in total raising the number from 388 to 548 out of 1120 runs.

The average desulfurization time is improved for 37 out of 56 instances, only 9 show a worse average. 32 instances show smaller standard deviation, while 16 show larger standard deviation. Most of these differences are rather small, still a clear trend towards slightly better results in Table 6 compared to Table 5 can be seen. A noteable negative effect can be seen on some of the *short* instances (mainly *instance01_short* and *instance02_short*), where the best result is very similar, but average and standard deviation get worse.

A large improvement can be seen across all instances regarding the runtime, where in the average a reduction by 71% can be achieved with the reduction procedures. Note that the runtime results from the settings for the cooling scheme, not directly from the reductions. However, simply reducing the runtime in the same way without applying the reduction has significant negative influence on the results. A comparision shows that for 4 instances the best known number of torpedoes can not be reached any more (number of torpedoes +33.3% each), the total number of aborts (not reaching the best known number of torpedoes or including constraint violations) goes up to 153 (13.7%) and out of the 52 instances with the same number of torpedoes 16 have a worse best desulfurization time (2 better) and 43 have a worse average desulfurization time (3 better). In the average the desulfurization time would raise by 0.3% with worst instances getting up to 7% higher desulfurization times, while improvements are not better than 0.2% at best.

In contrast, using the same cooling scheme as used without reductions, but including the reductions, 14 solutions have improved best desulfurization time (0 worse, up to 0.5% better) and 47 have improved average desulfurization time (1 worse, average 0.2% better).

6.4.2 Instance Analysis

Regarding the differences between the individual instances, a few conclusions can be drawn. The individual instances generated with the same configurations often show similar runtimes and highlight the conflict between the two given objectives that requires the two-stage algorithm. Typically instances with a higher number of torpedoes in turn show a lower desulfurization time and vice versa.

As expected due to the design of the algorithm, the *long* versions have significantly higher runtime, the *short* versions have significantly lower runtime within their category. *high* instances often resulted in more torpedoes in the solution, *narrow* did not seem to make much difference. Regarding the *ratio* instances the first two (1 and 5 emergency pit cycles) have a standard deviation of 0 in Table 6, indicating that the algorithm most likely always reaches the best result, while for more emergency pit cycles this is not the case any more.

7. Conclusions

This paper presented a new approach to solve a scheduling problem in steel production plants. Our overall algorithm includes a new approach for pruning the search space based on reducing the set of possible material assignments, and a search algorithm following the concept of Simulated Annealing. The pruning approach is based on logical reductions and lower bound calculations on the number of torpedo cars. The Simulated Annealing algorithm utilizes several proposed efficient moves and optimizes the result for two lexicographically ordered optimality criteria. In order to achieve this, each part of the solution algorithm is tailored towards the optimal progress for its corresponding goal. Hence, a two-stage Simulated Annealing is employed.

Our approach has been experimentally evaluated on instances of the ACP challenge. We conclude that by using our reduction procedure the efficiency can significantly be improved across all instances. On average, a runtime reduction of 71% can be achieved. Furthermore, the quality of the solutions for several instances could be improved by applying the reduction procedure. Our Simulated Annealing algorithm was able to obtain best existing results in the ACP challenge and it consistently obtains very good results for new and larger generated instances regarding the main optimization criteria, which is the number of used torpedoes.

The results show that the approach is a valid and competitive method to solve the given problem. The reduction procedure can also be used as prepossessing step for any other search technique. Besides, the ideas of Simulated Annealing are generic and can also be adapted to various other problems utilizing lexicographic evaluation functions.

From our perspective, future work could include the adaption of the approach to other related problems in this domain. Furthermore, it would be interesting to evaluate the impact of the reduction procedure on the efficiency of other search techniques for the Torpedo Scheduling Problem.

Acknowledgments

This work was supported by the Austrian Science Fund (FWF): P24814-N23. The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

References

Bent, R., & Van Hentenryck, P. (2004). A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows. *Transportation Science*, *38*(4), 515–530.

Deng, M., Inoue, A., & Kawakami, S. (2011). Optimal path planning for material and products transfer in steel works using ACO. In *The 2011 International Conference on Advanced Mechatronic Systems*, pp. 47–50. IEEE.

Dowsland, K. A., & Thompson, J. M. (2012). Simulated Annealing. In Rozenberg, G., Bck, T., & Kok, J. N. (Eds.), *Handbook of Natural Computing*, pp. 1623–1655. Springer Berlin Heidelberg, Berlin, Heidelberg.

Geiger, M. J. (2017a). A multi-threaded local search algorithm and computer implementation for the multi-mode, resource-constrained multi-project scheduling problem. *European Journal of Operational Research*, *256*(3), 729–741.

Geiger, M. J. (2017b). Optimale Torpedo-Einsatzplanung – Analyse und Lösung eines Ablaufplanungsproblems der Stahlindustrie. In Spengler, T., Fichtner, W., Rommelfanger, H., Geiger, M. J., & Metzger, O. (Eds.), *Entscheidungsunterstützung in Theorie und Praxis*, pp. 63–86. Springer.

Goldwaser, A., & Schutt, A. (2017). Optimal torpedo scheduling. In Beck, J. C. (Ed.), *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, Vol. 10416 of *Lecture Notes in Computer Science*, pp. 338–353. Springer.

Goldwaser, A., & Schutt, A. (2018). Optimal torpedo scheduling. *J. Artif. Intell. Res.*, *63*, 955–986.

Hall, P. (1935). On representatives of subsets. *Journal of the London Mathematical Society*, *1*(1), 26–30.

Homberger, J., & Gehring, H. (1999). Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR: Information Systems and Operational Research*, *37*(3), 297–318.

Huang, H., Chai, T., Luo, X., Zheng, B., & Wang, H. (2011). Two-Stage Method and Application for Molten Iron Scheduling Problem between Iron-Making Plants and Steel-Making Plants. *IFAC Proceedings Volumes*, *44*(1), 9476–9481.

Kikuchi, J., Konishi, M., & Imai, J. (2008). Transfer Planning of Molten Metals in Steel Worksby Decentralized Agent. *Memoirs of the Faculty of Engineering, Okayama University*, *42*(1), 60–70.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, *220*(4598), 671–680.

Kletzander, L., & Musliu, N. (2017). A multi-stage simulated annealing algorithm for the torpedo scheduling problem. In Salvagnin, D., & Lombardi, M. (Eds.), *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, Vol. 10335 of *Lecture Notes in Computer Science*, pp. 344–358. Springer.

Li, J.-q., Pan, Q.-k., & Duan, P.-y. (2016). An Improved Artificial Bee Colony Algorithm for Solving Hybrid Flexible Flowshop With Dynamic Operation Skipping. *IEEE Transactions on Cybernetics*, *46*(6), 1311–1324.

Liu, Y., & Wang, G. (2015). The Mix Integer Programming Model for Torpedo Car Scheduling in Iron and Steel Industry. In *International Conference on Computer Information Systems and Industrial Applications*, pp. 731–734. Atlantis Press.

Pham, D., & Karaboga, D. (2012). *Intelligent optimisation techniques: genetic algorithms, tabu search, simulated annealing and neural networks.* Springer Science & Business Media.

Römer, M. (2018) Private e-mail communication.

Schaus, P., Dejemeppe, C., Mouthuy, S., Mouthuy, F.-X., Allouche, D., Zytnicki, M., Pralet, C., & Barnier, N. (2016). The torpedo scheduling problem: Description. `http://cp2016.a4cp.org/program/acp-challenge/problem.html`. Accessed: 2017-02-02.

Tang, L., Wang, G., & Liu, J. (2007). A branch-and-price algorithm to solve the molten iron allocation problem in iron and steel industry. *Computers & Operations Research*, *34*(10), 3001–3015.

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.