Journal of Artificial Intelligence Research 63 (2018) 1–32 Submitted 03/18; published 11/18

# Optimal Torpedo Scheduling

**Adrian Goldwaser** ADRIAN.GOLDWASER@GMAIL.COM
*Data61, CSIRO, & The University of New South Wales*
*Sydney, NSW, Australia*

**Andreas Schutt** ANDREAS.SCHUTT@DATA61.CSIRO.AU
*Data61, CSIRO, & The University of Melbourne*
*Melbourne, VIC, Australia*

## Abstract

We consider the torpedo scheduling problem in steel production, which is concerned with the transport of hot metal from a blast furnace to an oxygen converter. A schedule must satisfy, amongst other considerations, resource capacity constraints along the path and the locations traversed as well as the sulfur level of the hot metal. The goal is first to minimize the number of torpedo cars used during the planning horizon and second to minimize the time spent desulfurizing the hot metal. We propose an exact solution method based on Logic-based Benders Decomposition using Mixed-Integer and Constraint Programming, which optimally solves and proves, for the first time, the optimality of all instances from the ACP Challenge 2016 within 10 minutes. In addition, we adapted our method to handle large-scale instances and instances with a more general rail network. This adaptation optimally solved all challenge instances within one minute and was able to solve instances of up to 100,000 hot metal pickups.

## 1. Introduction

Steel production is a complex process of sequential stages from raw materials to a final product in the form of, *e.g.*, wire plate coils. In the first stage, the iron making, raw iron ore powders are combined with other mineral powders in a sinter and coke is produced for fuel. The coke is then used to power a blast furnace to melt the output of the sinter, creating hot metal. In the second stage, the steel making, the hot metal is loaded in torpedo cars, or *torpedoes*, and transported to an oxygen converter, at which the content of sulfur, phoshor, carbon, and silicon is reduced (Kumakura, 2013). In modern plants, a desulfurization step of the hot metal is performed in the torpedoes before the oxygen converter (Kumakura, 2013). Once at the oxygen converter, the hot metal is converted into crude steel which is then further refined in order to finalise the characteristic and quality of the produced steel. The last two stages involve a continuous caster to create steel slabs which are then passed to a hot strip mill where it is thinned and coiled. Figure 1 provides a schematic view of a these sections in a steel production plant. This work focuses on the rotation of the torpedoes between the blast furnace and oxygen converter in the steel making stage, shown in Figure 2. At the steel making area, there are a number of blast furnaces producing hot metal of different qualities. At certain times or *events*, the hot metal in the blast furnace has to be loaded into a torpedo. Then the torpedo moves on a rail network to different locations for improving the quality of the hot metal if needed. After that, the hot metal is

transported to the oxygen converter and poured into it at a pre-defined event time. Now, the empty torpedo is available for the next pick up of hot metal.

We study the torpedo scheduling problem that was proposed by Schaus et al. (2016) for the ACP Challenge 2016. This problem focuses on the assignment of blast furnace events to oxygen converter events and the scheduling problem of transporting the hot metal through different locations while satisfying all scheduling constraints and the quality constraint, sulfurization level, on the hot metal. Figure 3 shows the rail network considered. There are five different locations: blast furnace (bf), full buffer (fb), desulfurization station (ds), oxygen converter (oc), and empty buffer (eb). The full and empty buffers are waiting areas for full and empty torpedoes, whereas at the desulfurization station the sulfur level of the hot metal can be reduced by chemical processes. Each location has a torpedo capacity, which is shown above the node. Each link or edge has a minimal transition time and a torpedo capacity, which is shown next to the edge in the same order. The dashed edge from bf to eb represents the emergency pit, in which hot metal can be dumped if required. The objective is a lexicographical one, first to minimize the number of torpedoes and second to minimize the total time spent at the desulfurization station by torpedoes.

**Example 1.1.** *Figure 4 shows a small problem with five blast furnace* $1, 2, \ldots, 5$ *and four converter events* $6, 7, 8, 9$*. Each converter event is specified by its due date and (maximal) sulfur level given above or below its node. We assume that the loading time at the blast furnace, the unloading time at the oxygen converter, and the time to desulfurize the hot metal by one level are* 5 *time units. The transition times between different locations and the torpedo capacities are shown in Figure 3,* e.g.*, the blast furnace* bf *has torpedo capacity* 1 *and the emergency trip (dashed line) a transition time of* 20 *and no capacity limit. A solution is depicted by the arrows between the events, which shows the usage of three torpedoes. The first torpedo serves the events 1, 6, 4, 8, the second one only 2, and the third one 3, 7, 5, 9 in this order. The scheduling of this solution is depicted in Figure 5, showing the movement of each torpedo. For fulfilling the demands for oxygen converter events 6, 7, and 8, a total of 20 time units have to be spent for desulfurization — 5 time units per unit decrease with 4 unit decreases required.* □

To the best of our knowledge, the torpedo scheduling problem we study was first proposed in the 2016 ACP Challenge. From the ten teams who took part, we are only aware of the publication of the winning team (Kletzander & Musliu, 2017) and the third placed team (Geiger, 2017b). Kletzander and Musliu (2017) propose a two-stage simulated annealing approach. The first stage minimizes the number of torpedoes by tracking the maximal number of torpedoes simultaneously used at any one time, whereas the second stage minimizes the desulfurization time. They relax some constraints, but add penalty terms to their objective. One iteration of the method takes between four and ten minutes time for the ACP challenge instances. They run 50 iterations to get their best results, which is a runtime of more than 3 hours. Geiger (2017b) proposes a Branch-and-Bound method, which branches over the assignments of converter events to blast furnace events in a depth first manner with chronological backtracking. In each node, a resource-constrained scheduling problem is solved by a serial generation scheme with variable neighborhood search (Geiger, 2017a). In order to reduce the search tree size, Geiger removes infeasible assignments in a preprocessing step and after a solution is found. Both methods (Geiger, 2017b; Kletzander

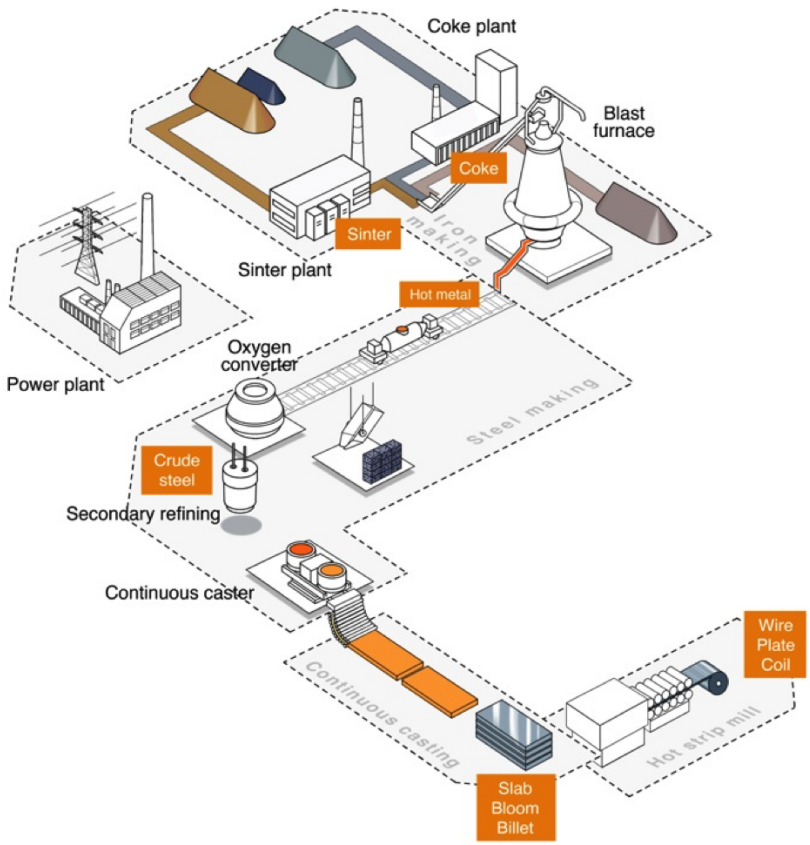


Figure 1: Overview of a steel production plant taken from (Schaus et al., 2016).

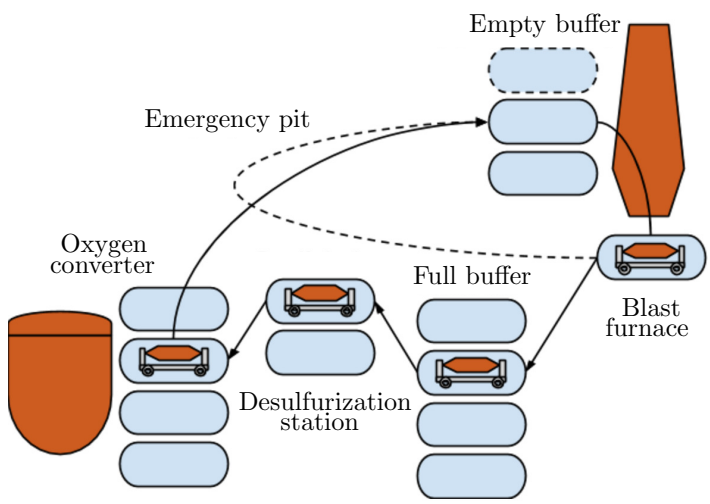


Figure 2: Close up view of blast furnace to oxygen converter section of a steel production plant modified from (Schaus et al., 2016).

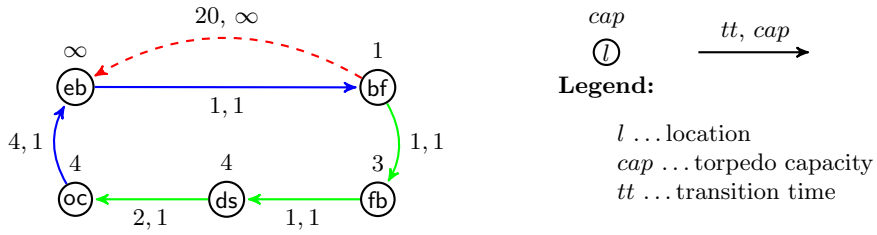


Figure 3: The graph for Example 1.1.

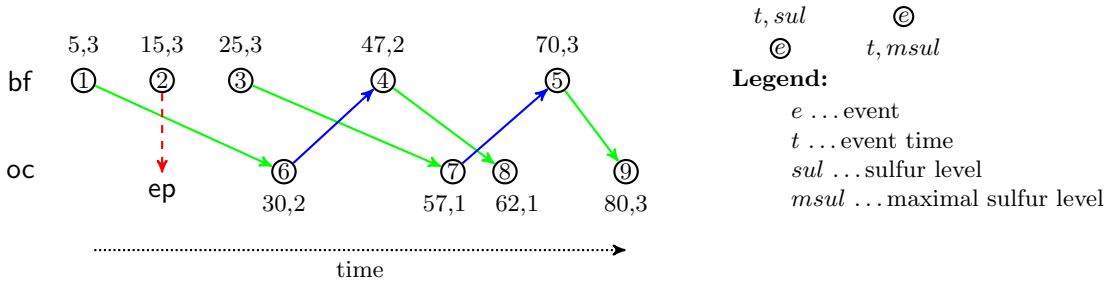


Figure 4: A small example of a torpedo scheduling problem.

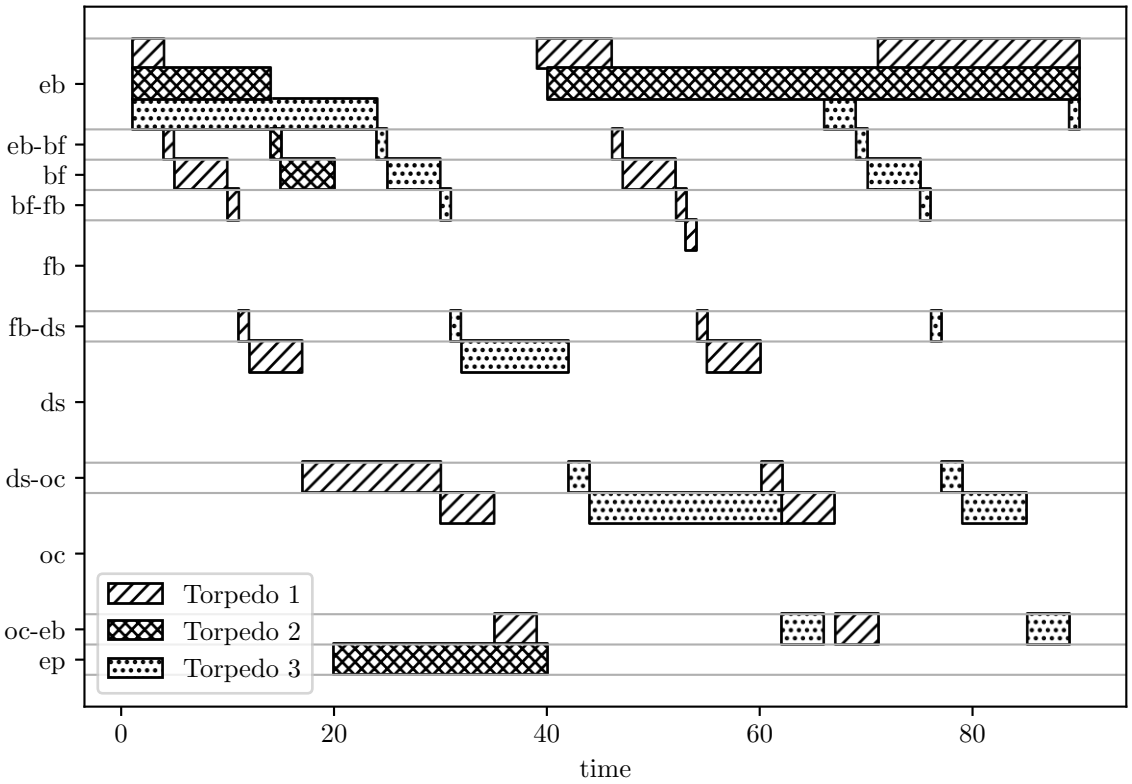


Figure 5: Scheduling of the solution

& Musliu, 2017) are incomplete and thus cannot prove optimality of an instance, unless the lower bound of the objective is the optimal value.

Different aspects on the torpedo scheduling problem have been studied in the literature: the routing of torpedoes through the rail network while minimizing the transportation time of the hot metal (Deng, Inoue, & Kawakami, 2011; Kikuchi, Konishi, & Imai, 2008; Liu & Wang, 2015), the molten iron allocation problem (Tang, Wang, & Liu, 2007), the molten scheduling problem (Huang, Chai, Luo, Zheng, & Wang, 2011; Li, Pan, & Duan, 2016), and the locomotive scheduling problem (Wang & Tang, 2007). All those works use different solution methods such as local search, mixed integer programming, and column generation, but none use Logic-based Benders Decomposition (Hooker & Ottosson, 2003) and Constraint Programming (CP) as we do in the present paper.

We propose a Logic-Based Benders Decomposition (Hooker & Ottosson, 2003) method, in which the assignment problem and the lexicographical objective is handled in the master problem. The remaining scheduling is partitioned by one of our two algorithms, if possible, and solved minimizing the desulfurization time. The master problem is solved by a Mixed Integer Programming (MIP) solver whereas the scheduling problems by a CP solver with Nogood Learning applying Lazy Clause Generation (LCG) (Ohrimenko, Stuckey, & Codish, 2009). In preprocessing, we simplify the problem by removing symmetries. To the best of our knowledge, our method is the first published complete method for the torpedo scheduling and proves the optimality of found solutions of the simulated annealing approach (Kletzander & Musliu, 2017) for all ACP challenge instances, in an even shorter runtime. We modified our method for handling large scale problems, but with the price of losing optimality. The modified method improved the runtime by orders of magnitude and was able to solve instances with 100,000 events in 70 minutes.

Moreover we use the same idea of handling large scale problems but instead restricting the time between pick up and drop off of the hot metal in order to keep the hot metal at a high enough temperature. We also generalise our approach to work on a general rail network having the same structure by allowing arbitrary capacities at the locations and the paths between two locations. The generalisation allows us to test our method on denser torpedo scheduling problems, which exhibit different solving characteristics.

The paper is organised as follows: Section 2 covers the formal definition of the torpedo scheduling problem. Section 3 presents the preprocessing steps used in the solution. Section 4 explains the solution method, including both the MIP and CP models and the other algorithms used. Section 5 shows the results of running different versions of the solution on a range of benchmarks.

## 2. Torpedo Scheduling

In this section, we formally introduce the torpedo scheduling problem considered in this work. For convenience and later reference, Table 1 provides a brief overview of notations defined in this and the next section.

The torpedo scheduling problem consists of a set of *blast furnace events* $N = \{1, 2, \ldots, n\}$, a set of *(oxygen) converter events* $M = \{n+1, n+2, \ldots, n+m\}$, and a set of *locations* $L = \{\mathsf{bf}, \mathsf{fb}, \mathsf{ds}, \mathsf{oc}, \mathsf{eb}\}$ in the production plant. In addition, the *torpedo graph* $G = (L, P)$ is a directed graph which specifies the two possible traversals of the torpedoes through the

| Notation | Description |
|---|---|
| $N$ | The set of $n$ events at the blast furnace, which are 1, 2, ..., $n$. |
| $M$ | The set of $m$ events at the oxygen converter, which are $n+1$, $n+2$, ..., $n+m$ |
| $L$ | The set of locations in the production plant, which are $\mathsf{bf}$ (blast furnace), $\mathsf{fb}$ (free buffer), $\mathsf{ds}$ (desulfurization station), $\mathsf{oc}$ (oxygen converter), and $\mathsf{eb}$ (empty buffer). |
| $(l, l',)$ | A (directed) rail link between two locations $l$ and $l'$ in $L$. |
| $P$ | The set of rail links in the production plant, which are $(\mathsf{eb}, \mathsf{bf})$, $(\mathsf{bf}, \mathsf{eb})$, $(\mathsf{bf}, \mathsf{fb})$, $(\mathsf{fb}, \mathsf{ds})$, $(\mathsf{ds}, \mathsf{oc})$, and $(\mathsf{oc}, \mathsf{eb})$. |
| $G$ | A directed graph with the nodes $L$ and the edges $P$. |
| $tt_{(l,l')}$ | The minimal transition time from $l \in L$ to $l' \in L$ with $(l, l') \in P$. |
| $dd_i^{\mathsf{bf}}$ | The due date of the event $i \in N$ at the blast furnace. |
| $dd_j^{\mathsf{oc}}$ | The due date of the event $j \in M$ at the oxygen converter. |
| $sul_i^{\mathsf{bf}}$ | The sulfur level of the hot metal for events $i \in N$. |
| $sul_j^{\mathsf{oc}}$ | The maximal sulfur level of the hot metal for events $j \in M$. |
| $dur^{\mathsf{bf}}$ | The duration of loading hot metal into a torpedo at $\mathsf{bf}$. |
| $dur^{\mathsf{oc}}$ | The duration of unloading hot metal from a torpedo at $\mathsf{oc}$. |
| $dur^{\mathsf{ds}}$ | The duration of reducing the sulfur level of hot metal by one unit at $\mathsf{ds}$. |
| $cap_l$ | The torpedo capacity of the location $l \in L$. |
| $cap_{(l,l')}$ | The torpedo capacity of the rail link $(l, l') \in P$. |
| $\mathbf{arr}_l^i$ | The variable storing the arrival time at $l \in L$ for the event $i \in N$. |
| $\mathbf{dep}_l^i$ | The variable storing the departure time at $l \in L$ for the event $i \in N$. |
| $\mathbf{ep}_i$ | The variable representing whether event $i \in N$ is an emergency pit trip. |
| $\mathbf{oc}_i$ | The variable representing which oxygen converter event in $M$ is matched to the event $i \in N$ if $i$ is not a emergency pit trip. If $i$ is an emergency pit trip, then $\mathbf{oc}_i = 0$. |
| $S$ | A solution vector of $n$ torpedo runs. |
| $X$ | The set of possible matchings $(i, j) \in N \times M$ from blast furnace events $i$ to oxygen converter events $j$. |
| $\mathtt{bm}$ | A backward matching that assigns a blast furnace event $i \in N$ or none ($\infty$) to an oxygen converter event $j \in M$, *i.e.*, $\mathtt{bm} : M \to N \cup \{\infty\}$. |
| $V$ | The set of all blast furnace events matched by $\mathtt{bm}$, *i.e.*, $\mathtt{bm}(M) \setminus \{\infty\}$. |
| $U$ | The set of all blast furnace events unmatched by $\mathtt{bm}$, *i.e.*, $N \setminus \mathtt{bm}(M)$. |
| $R$ | The set of possible matchings $(i, i')$ of blast furnace events $i \in N$ to unmatched blast furnace events $i' \in U$. |

Table 1: Brief overview of notations introduced in Sections 2 and 3.

plant. The *oxygen converter trip* delivers the hot metal to the converter and visits the locations in the order eb, bf, fb, ds, oc, then eb, whereas the *emergency pit trip* dumps the hot metal at the emergency pit and visits the locations in the order eb, bf, then eb. Thus, $P = \{(\mathsf{eb}, \mathsf{bf}), (\mathsf{bf}, \mathsf{eb}), (\mathsf{bf}, \mathsf{fb}), (\mathsf{fb}, \mathsf{ds}), (\mathsf{ds}, \mathsf{oc}), (\mathsf{oc}, \mathsf{eb})\}$.

Each location $l \in L$ has a *torpedo capacity*, $cap_l$, where the empty buffer has no limit, *i.e.*, $cap_{\mathsf{ep}} = \infty$. We extend this notation for edges $p \in P$, which is $cap_p$, where trips via the emergency pit are also unlimited, *i.e.*, $cap_{(\mathsf{bf},\mathsf{eb})} = \infty$. A torpedo traversing the edge $p$ requires a minimal *transition time* of $tt_p$. Note that for the ACP Challenge 2016 (Schaus et al., 2016) the following capacities were given as shown in Figure 3: $cap_{\mathsf{bf}} = 1$, and $cap_p = 1$ for all $p \in P \setminus \{(\mathsf{bf}, \mathsf{eb})\}$.

Each blast furnace event $i \in N$ is characterized by a *due date*, $dd_i^{\mathsf{bf}} \in \mathbb{N}^0$, at which hot metal is picked up by exactly one empty torpedo, and a *sulfur level*, $sul_i^{\mathsf{bf}} \in \{1, 2, \ldots, 5\}$, of the hot metal. Each oxygen converter event $j \in M$ has a *due date*, $dd_j^{\mathsf{oc}} \in \mathbb{N}^0$, at which hot metal from exactly one full torpedo is poured into the converter, and a maximal *sulfur level*, $sul_j^{\mathsf{oc}} \in \{1, 2, \ldots, 5\}$, of the hot metal. Loading of a torpedo takes $dur^{\mathsf{bf}} \in \mathbb{N}$ time units at the blast furnace, while unloading takes $dur^{\mathsf{oc}} \in \mathbb{N}$ time units at the oxygen converter. Reducing the sulfur level of hot metal by one unit requires $dur^{\mathsf{ds}} \in \mathbb{N}$ time units at the desulfurization station.

Following (Kletzander & Musliu, 2017), we describe a solution as $n$ torpedo runs, in which the $i$-th run picks up the hot metal of the $i$-th event at the blast furnace bf, *i.e.*, $i \in N$. For simplicity, we abuse the notation of the events in $N$ and refer to blast furnace event or torpedo run by $i \in N$. A *torpedo run* $i$ is either a converter or emergency pit trip. In the former case, it is specified by variable departure times $\mathbf{dep}_i^l$, variable arrival times $\mathbf{arr}_i^l$ for locations in $\{\mathsf{eb}, \mathsf{bf}, \mathsf{fb}, \mathsf{ds}, \mathsf{oc}\}$, and the variable converter event $\mathbf{oc}_i \in M$, that it serves. For the latter case, it is specified by the variable departure and variable arrival times for only the locations eb and bf. We denote by $\mathbf{ep}_i$ whether it is a converter trip $\mathbf{ep}_i = 0$ or an emergency pit trip $\mathbf{ep}_i = 1$. In the case where $\mathbf{ep}_i = 1$, then $\mathbf{oc}_i = 0$.

**Definition 2.1** (Torpedo Scheduling Problem)**.** *A* torpedo scheduling problem *consists of a triplet* $(N, M, G = (L, P))$*. A solution* $S = (1, 2, \ldots, n)$ *is a vector of* $n$ *torpedo runs, in which the* $i$*-th run picks up the hot metal of the* $i$*-th blast furnace event, matches the blast furnace event to an oxygen converter event or an emergency pit trip, and assigns all corresponding arrival and departure times,* i.e.*, each run,* $i$*, consists of* $\mathbf{arr}_i^l$ *and* $\mathbf{dep}_i^l$ *for all locations* $l$*, as well as* $\mathbf{oc}_i$ *and* $\mathbf{ep}_i$*. A solution satisfies the capacity constraints on each location (1) and on each edge (2),*

$$\sum\nolimits_{i \in N : \mathbf{arr}_i^l \leq t < \mathbf{dep}_i^l} 1 \leq cap_l \qquad \forall l \in L, \forall t \in \mathbb{N}^0 \tag{1}$$

$$\sum\nolimits_{i \in N : \mathbf{dep}_i^l \leq t < \mathbf{arr}_i^k} 1 \leq cap_{(l,k)} \qquad \forall (l,k) \in P, \forall t \in \mathbb{N}^0 \tag{2}$$

*the minimal transition times for oxygen converter (3) and emergency pit trips (4),*

$$\mathbf{arr}_i^k - \mathbf{dep}_i^l \geq tt_{(l,k)} \qquad \forall i \in N : \mathbf{ep}_i = 0, \forall (l,k) \in P \setminus \{(\mathsf{bf}, \mathsf{eb})\} \tag{3}$$

$$\mathbf{arr}_i^k - \mathbf{dep}_i^l \geq tt_{(l,k)} \qquad \forall i \in N : \mathbf{ep}_i = 1, \forall (l,k) \in \{(\mathsf{eb}, \mathsf{bf}), (\mathsf{bf}, \mathsf{eb})\} \tag{4}$$

*the loading constraints at the blast furnace (5), the unloading constraints (6), and the maximal sulfurization level (7) at the oxygen converter.*

$$\mathbf{arr}_i^{bf} \leq dd_i^{bf} \quad \wedge \quad dd_i^{bf} + dur^{bf} \leq \mathbf{dep}_i^{bf} \qquad \forall i \in N \tag{5}$$

$$\mathbf{arr}_i^{oc} \leq dd_{\mathbf{oc}_i}^{oc} \quad \wedge \quad dd_{\mathbf{oc}_i}^{oc} + dur^{oc} \leq \mathbf{dep}_i^{oc} \qquad \forall i \in N : \mathbf{ep}_i = 0 \tag{6}$$

$$sul_i^{bf} - \left\lfloor \frac{\mathbf{dep}_i^{ds} - \mathbf{arr}_i^{ds}}{dur^{ds}} \right\rfloor \leq sul_{\mathbf{oc}_i}^{oc} \qquad \forall i \in N : \mathbf{ep}_i = 0 \tag{7}$$

*All torpedoes, which are identical, are located at* eb *at time* 0*. Here, we are interested in a solution that minimizes two objective functions in lexicographic order. The primary objective (8) is to minimize the number of torpedoes used, which can be stated as minimizing the maximal number of "active" torpedo runs at any time (Kletzander & Musliu, 2017; Geiger, 2017b). The secondary objective (9) is to minimize the total time spent at the desulfurization station.*

$$\min \max_{t \in \mathbb{N}^0} |\{i \in N \mid \mathbf{dep}_i^{eb} \leq t \wedge t < \mathbf{arr}_i^{eb}\}| \tag{8}$$

$$\min \sum\nolimits_{i \in N : \mathbf{ep}_i = 0} \mathbf{dep}_i^{ds} - \mathbf{arr}_i^{ds} \tag{9}$$

Note that the solutions do not provide an assignment of torpedo runs to individual torpedoes, but such an assignment can be computed in polynomial time with respect to the number of torpedoes. Algorithm 1 generates such an assignment with the worst case complexity $\mathcal{O}(n \log n)$. It iterates over the departure times and arrival times of the torpedo runs at the empty buffer in chronological order while recording the available torpedoes at the empty buffer using a stack (lines 6–14). If the departure time of a torpedo run $d$ is earlier then arrival time of a returning torpedo of the torpedo run $a$ then a torpedo $c$ is popped from the stack *avail*, assigned to $d$, and the next departure time is considered in the next loop iteration. Otherwise the torpedo used for $a$ is pushed to the stack *avail* and the next arrival time is considered in the next loop iteration. Note that there can be many different assignments for one solution.

Moreover, as already observed by Kletzander and Musliu (2017) and Geiger (2017b) the possible oxygen event matches for a blast furnace event can be reduced by simply calculating the minimal travel time including a minimal time for desulfurization from the blast furnace to the oxygen converter. We denote $X = \{(i,j) \in N \times M \mid dd_i^{\mathsf{bf}} + tt_{(\mathsf{bf},\mathsf{fb})} + tt_{(\mathsf{fb},\mathsf{ds})} + tt_{(\mathsf{ds},\mathsf{oc})} + dur^{\mathsf{ds}} \cdot \max(0, sul_i^{\mathsf{bf}} - sul_j^{\mathsf{oc}}) \leq dd_j^{\mathsf{oc}}\}$ the set of possible matchings of blast furnace to oxygen converter events. In addition, they also observed that there is no reason to delay a departure of a torpedo from the blast furnace in the case of an emergency trip due to the uncapacitated path $(\mathsf{bf}, \mathsf{eb})$ and empty buffer. Thus, we can fix $\mathbf{dep}_i^{\mathsf{bf}} = dd_i^{\mathsf{bf}} + dur^{\mathsf{bf}}$ and $\mathbf{arr}_i^{\mathsf{eb}} = \mathbf{dep}_i^{\mathsf{bf}} + tt_{(\mathsf{bf},\mathsf{eb})}$ if the torpedo run $i$ goes to the emergency pit.

**Example 2.1.** *Given the example from Example 1.1. Then,* $X = \{(1,6),(1,7),(1,8),(1,9),(2,7),(2,8),(2,9),(3,7),(3,8),(3,9),(4,8),(4,9),(5,9)\}$ *and the departure times at* bf *respectively are 10, 20, 30, 52, and 75 for events 1, 2, 3, 4, and 5 if they go to the emergency pit. Note that only* bf *event 1 can deliver hot metal for event 6, we leave such simple reductions to the solver.*

**Algorithm 1:** Assignment of torpedo runs to torpedoes.

**Input** : $S$ a solution of a torpedo scheduling problem with $n$ torpedo runs.
**Input** : $C$ the number of torpedoes required for solution $S$.
**Output** : $assgmt$ an array of assignments of torpedo runs to torpedo cars.

```
for i := 1 to n do X[i] := i; Y[i] := i;
sort X in ascending order of their departure times dep_i^eb in S;
sort Y in ascending order of their arrival times arr_i^eb in S;
for c := 1 to C do avail.push(c);
ii := 1; jj := 1;
while ii ≤ n do
    d := X[ii]; a := Y[jj];
    if dep_d^eb < arr_a^eb then
        c := avail.pop();
        assgmt[d] := c;
        jj := jj + 1;
    else
        avail.push(assgmt[a]);
        ii := ii + 1;
return assgmt;
```

## 3. Preprocessing

The original problem definition requires solving many parts simultaneously. We make some observations which allow us to decompose and solve some of these components optimally to simplify the problem. The problem can be decomposed into three components: the assignment of blast furnace events to oxygen converter events, the scheduling of the torpedoes and the assignment of torpedoes to torpedo runs. In the preprocessing, we take this third section and note that due to all torpedoes being identical, we can solve part of this — specifically the optimal reuse of torpedoes after they exit the oxygen converter. This analysis rests on the unlimited capacity of the empty buffer and allows the removal of much of the cyclic nature of the problem.

The rest of this section explains these preprocessing steps in detail and sets up the structure of the resulting problem necessary for our solution approach.

### 3.1 Departure Times from the Oxygen Converter

The empty buffer has unlimited capacity, which means that it is never suboptimal to get an empty torpedo there earlier rather than later as it can be reused earlier, it frees space at the oxygen converter earlier, and clears the path from the oxygen converter to the empty buffer earlier. Thus, an empty torpedo should leave the oxygen converter as early as possible, which is the latest time of the completion unloading the torpedo and the arrival time of one of the previous torpedoes at the empty buffer from the oxygen converter.

Since the due dates for the oxygen converter events are known a priori, the departure times from the oxygen converter and the arrival times to the empty buffer can be computed in linear time with the respect to the number of those events, if the events are given in chronological order, as shown in Algorithm 2. The idea of Algorithm 2 is to compute the departure times in chronological order of the events while recording the earliest available time on $(\mathsf{oc}, \mathsf{eb})$ for each track. In line 1, for each converter event $j$ the departure time

**Algorithm 2:** Computation of departure times from the oxygen converter.

**Input** : $M$ an array of $m$ oxygen converter events sorted in ascending order of their due dates.
**Output** : $depOC$ an array of departure times at oxygen converter for each converter event.

1 **for** $j := 1$ **to** $m$ **do** $depOC[j] := dd_j^{\mathsf{oc}} + dur^{\mathsf{oc}}$;
2 **for** $track := 1$ **to** $cap_{(\mathsf{oc},\mathsf{eb})}$ **do** $availAt[track] := -\infty$;
3 **for** $jj := 1$ **to** $m$ **do**
4 $\quad j := M[jj]$;
5 $\quad track := 1 + (jj \mod cap_{(\mathsf{oc},\mathsf{eb})})$;
6 $\quad$ **if** $availAt[track] > depOC[j]$ **then** $depOC[j] := availAt[track]$;
7 $\quad availAt[track] := depOC[j] + tt_{(\mathsf{oc},\mathsf{eb})}$;
8 **return** $depOC$;

$depOC[j]$ of the torpedo is initialised by the end of the unloading, *i.e.*, $dd_j^{\mathsf{oc}} + dur^{\mathsf{oc}}$. In line 2, the earliest available time $availAt[track]$ is initialised for each track $track$. The for-loop (lines 3–7) iterates in chronological order over the converter events. It determines which track to use (line 5) and checks if the track is available after the end of loading of the current event $j$ (line 6). If so then the departure time $depOC[j]$ is updated to the available time on the track (line 7), otherwise it remains unchanged. Finally, the available time is updated to the arrival time $depOC[j] + tt_{(\mathsf{oc},\mathsf{eb})}$ of the torpedo at the empty buffer when departing at $depOC[j]$ from the converter (line 7). Note that the order of torpedoes serving oxygen converter events remains unchanged by the algorithm.

**Proposition 3.1.** *Algorithm 2 computes the earliest departure times for each oxygen converter event without changing the order of their corresponding earliest arrival times at the empty buffer and without creating an overload on the path between both locations,* i.e., $(\mathsf{oc}, \mathsf{eb})$.

*Proof.* Let $1, 2, \ldots, m$ be the order of converter events sorted in ascending order of their due dates, *i.e.*, for all $1 \leq j_1 < j_2 \leq m$ hold $dd_{j_1}^{\mathsf{oc}} \leq dd_{j_2}^{\mathsf{oc}}$. The departure times are initialised by their end time of their loading regardless whether it will cause an overload at $(\mathsf{oc}, \mathsf{eb})$ (line 1), whereas the entries in the array $availAt$ by $-\infty$. We prove the claim by induction. The departure times for the first $cap_{(\mathsf{oc},\mathsf{eb})}$ converter events $j$ remain unchanged, because the condition on line 6 does not hold. Since the duration of unloading and the transition time on $(\mathsf{oc}, \mathsf{eb})$ are the same and constant, the order of the events according to the departure times from the converter and arrival times at the empty buffer remain unchanged, *i.e.*, for all $1 \leq j_1 < j_2 \leq cap_{(\mathsf{oc},\mathsf{eb})}$ holds $depOC[j_1] \leq depOC[j_2]$ and $depOC[j_1] + tt_{(\mathsf{oc},\mathsf{eb})} \leq depOC[j_2] + tt_{(\mathsf{oc},\mathsf{eb})}$. Thus, $availAt[1 + (j_1 \mod cap_{(\mathsf{oc},\mathsf{eb})})] \leq availAt[1 + (j_2 \mod cap_{(\mathsf{oc},\mathsf{eb})})]$ holds, too, because they are respectively assigned to the arrival times at the empty buffer $availAt[1 + (j_1 \mod cap_{(\mathsf{oc},\mathsf{eb})})] = depOC[j_1] + tt_{(\mathsf{oc},\mathsf{eb})}$ and $availAt[1 + (j_2 \mod cap_{(\mathsf{oc},\mathsf{eb})})] = depOC[j_2] + tt_{(\mathsf{oc},\mathsf{eb})}$ (line 7). Obviously, there is no overload at $(\mathsf{oc}, \mathsf{eb})$ for the first $cap_{(\mathsf{oc},\mathsf{eb})}$ events. Let $j$ be any event after the first $cap_{(\mathsf{oc},\mathsf{eb})}$ events, *i.e.*, $j > cap_{(\mathsf{oc},\mathsf{eb})}$. Let $j_1, j_2, ..., j_{cap_{(\mathsf{oc},\mathsf{eb})}}$ be the $cap_{(\mathsf{oc},\mathsf{eb})}$ events before $j$ in this order. We know that their order of arrival times is the same as their unloading times and the entries in $availAt$ reflecting their arrival times. We also know that $j \mod cap_{(\mathsf{oc},\mathsf{eb})} = j_1 \mod cap_{(\mathsf{oc},\mathsf{eb})}$. If $availAt[1 + (j \mod cap_{(\mathsf{oc},\mathsf{eb})})] \leq depOC$ then the departure time of $j$ remains the completion of the unloading, which is the earliest possible departure time. In the other case, the departure time $depOC[j]$ is updated to $availAt[1+(j$

mod $cap_{(\mathsf{oc},\mathsf{eb})})]$, which is the earliest available time of available track, because $availAt[1+(j$ mod $cap_{(\mathsf{oc},\mathsf{eb})})] \leq availAt[1+(j' \mod cap_{(\mathsf{oc},\mathsf{eb})})]$ for all $j_1 < j' \leq j_{cap_{(\mathsf{oc},\mathsf{eb})}}$. Consequently, the orders of the events do not change. In both, one track at $(\mathsf{oc},\mathsf{eb})$ needs to be reserved during the time interval $[depOC[j], depOC[j] + tt_{(\mathsf{oc},\mathsf{eb})})$, but at most $cap_{(\mathsf{oc},\mathsf{eb})} - 1$ time intervals of previous events are able to intersect with the time interval by construction. The track is reversed by the algorithm by updating $availAt[1+(j \mod cap_{(\mathsf{oc},\mathsf{eb})})]$ to $depOC[j] + tt_{(\mathsf{oc},\mathsf{eb})}$. Thus, there is no resource overload. □

**Theorem 3.1.** *Let $(N, M, G)$ be a feasible torpedo scheduling problem. Then there exists an optimal solution $S$ using the departure times at the oxygen converter computed by Algorithm 2.*

*Proof.* Without loss of generality, assume that all oxygen converter due dates are unique. Let $S$ be an optimal solution. Let $depOC'[1], depOC'[2], \ldots$ and $depOC[1], depOC[2], \ldots$ be the sequence of departure times at the oxygen converter in solution $S$ and from Algorithm 2, respectively. Due to the construction of Algorithm 2, it holds $depOC'[1] \geq depOC[1]$. Since the empty buffer is uncapacitated, the torpedoes identical, and Proposition 3.1, we can replace the departure times for each oxygen converter event in $S$ by the computed ones of Algorithm 2. □

**Example 3.1.** *Given the example from Example 1.1 from page 2. Then Algorithm 2 respectively computes departure times 35, 62, 67, and 85 for the oxygen converter events 6, 7, 8, and 9.* □

Algorithm 2 computes the earliest departure times at the converter and the arrival times at the empty buffer while guaranteeing a non-overload on the path between both locations. However, the earliest departure times may lead to a resource overload at the converter, because one spot for a torpedo has to be reserved at the converter during the time interval $[dd_j^{\mathsf{oc}}, depOC[j])$ for each converter event $j$. A check can be simply performed with a standard algorithm as shown in Algorithm 3. It takes the parameters *start*, *end*, and *cap* as input, which would respectively be in our case the unloading times $dd_j^{\mathsf{oc}}$ of all events $j$, the departure times $depOC[.]$ of all events $j$, and the converter capacity $cap_{\mathsf{oc}}$. A few initialisations are performed such as sorting the events regarding their start (unloading) times and their end (departure) times in the first four lines. The while-loop (lines 5–10) iterates over the start and end times in chronological order, breaks ties by processing end times before start times, and tracks the current number of torpedoes by *height*. If a start time is processed (line 7) then the height is incremented. If an end time is processed then it is checked for a resource overload and in a case of it the algorithm fails (line 9). In the other case, the height is updated by a decrement (line 10). If no resource overload is found then the algorithm succeeds (line 11).

**Proposition 3.2.** *Let $(N, M, G)$ be a torpedo scheduling problem. If Algorithm 3 detects a resource overload when provided with the due dates of the converter events, the earliest departure times computed by Algorithm 2, and the converter capacity $cap_{\mathsf{oc}}$ then the torpedo scheduling problem is infeasible.*

*Proof.* Assume the problem is feasible, but Algorithm 3 detects a resource overload when processing the end time $t$. Let $\Omega$ be the set of converter events that contributed to the height

**Algorithm 3:** Resource overload check.

```
Input  : start an array of start times of k events.
Input  : end an array of end times of k events where start[i] < end[i] for all 1 ≤ i ≤ k.
Input  : cap a positive integer representing a resource capacity.
Output : True if there is no resource overload; otherwise False.
for i := 1 to k do X[i] := i; Y[i] := i;
sort X in ascending order to start;
sort Y in ascending order to end;
ii := 1; jj := 1; height := 0;
while ii ≤ k or jj ≤ k do
    if ii ≤ k and start[X[ii]] < end[Y[ii]] then
        height := height + 1; ii := ii + 1;
    else
        if height > cap then return False;
        height := height − 1; jj := jj + 1;
return True;
```

before the time $t$, *i.e.*, events for which their start time has been processed but not yet their end time. By construction of the algorithm, $start[j] < t$ and $t \leq end[j]$ hold for all $j \in \Omega$ where $start[j] = dd_j^{\mathsf{oc}}$ and $end[j] = depOC[j]$ computed by Algorithm 2 with the property $dd_j^{\mathsf{oc}} + dur^{\mathsf{oc}} \leq depOC[j]$. Thus, there is a resource overload at the converter, *i.e.*, $\sum_{j\in\Omega} 1 > cap_{\mathsf{oc}}$, during the time interval $[\max_{j\in\Omega} dd_j^{\mathsf{oc}}, t = \min_{j\in\Omega} depOC[j])$. Since the problem is feasible, we know that there is no resource overload at the converter when torpedoes immediately depart from the converter once they are unloaded, *i.e.*, $dd_j^{\mathsf{oc}} + dur^{\mathsf{oc}}$ for all $j \in \Omega$. Therefore, Algorithm 2 must have delayed the departure of at least one torpedo for a converter event $j$, such that $dd_j^{\mathsf{oc}} + dur^{\mathsf{oc}} < end[j]$ holds. Due to Proposition 3.1, we know that $end[j]$ are the earliest departure time from the converter without changing the order of the events for unloading and departure. Consequently, the order in a solution without a resource overload must be different, in which a delayed departure event in Algorithm 2 departs earlier. However, the only way to move such a event $j$ forward is to delay an event $j'$ in the order before $j$, but this means that we have to reserve one torpedo for $j'$ for at least the same time interval that we freed the space from $j$. Thus, we still have a resource overload and the solution $S$ is incorrect. □

### 3.2 Arrival Times at the Blast Furnace

A similar observation to the departure times at the oxygen converter can be seen for the arrival times at the blast furnace. Since the empty buffer is uncapacitated and the hot metal cannot be picked up before its due date, it is never suboptimal to get an empty torpedo there later than rather earlier.

Algorithm 4 is symmetric to Algorithm 2 for the arrival times at the blast furnace. It computes the times in reverse-chronological order of the blast furnace events. As for Algorithm 2, the computed arrival times $arrBF$ may lead to a resource overload at the blast furnace. We can use Algorithm 3 for checking of an overload by respectively using $arrBF[i]$, and $dd_i^{\mathsf{bf}} + dur^{\mathsf{bf}}$ of all blast furnace events $i$ for *start*, and *end* as well as $cap_{\mathsf{bf}}$ for *cap*.

With similar arguments as in the oxygen converter case, the following two claims hold.

**Algorithm 4:** Computation of arrival times at the blast furnace.

**Input** : $N$ an array of $n$ blast furnace events sorted in chronological order.
**Output** : $arrBF$ an array of arrival times at the blast furnace for each blast furnace event.

```
for i := 1 to n do arrBF[i] := dd_i^bf;
for track := 1 to cap_(eb,bf) do availAt[track] := ∞;
for ii := n down to 1 do
    i := N[ii];
    track := 1 + (n mod cap_(eb,bf));
    if availAt[track] < arrBFi then arrBF[i] := availAt[track];
    availAt[track] := arrBF[i] − tt_(eb,bf);
return arrBF;
```

**Proposition 3.3.** *Algorithm 4 computes the latest arrival time for each blast furnace event and their latest departure time from the empty buffer without creating an overload on the path between both locations,* i.e., $(\mathsf{eb}, \mathsf{bf})$.

*Proof.* Proof is similar to the proof of Proposition 3.1. □

**Theorem 3.2.** *Let $(N, M, G)$ be a feasible torpedo scheduling problem. Then there exists an optimal solution $S$ using the arrival times at the blast furnace computed by Algorithm 4.*

*Proof.* Proof is similar to the proof of Theorem. 3.1. □

Note that for torpedo runs using the emergency pit, we can now fix its remaining departure and arrival times. Thus, we only have to decide which run is an emergency pit trip.

**Example 3.2.** *Consider Example 1.1 from page 2. Then Algorithm 4 respectively computes arrival times 4, 14, 24, 46, and 69 for the events 1, 2, 3, 4, and 5.* □

Since the blast furnace and oxygen converter events are independent of each other, it follows that an optimal solution exists, which has the same arrival and departure times for the corresponding events as computed in Algorithm 2 and 4. Thus, fixing the corresponding variables to those times removes symmetries from the problem.

**Theorem 3.3.** *Let $(N, M, G)$ be a feasible torpedo scheduling problem. Then there exists an optimal solution $S$ using the arrival times at the blast furnace computed by Algorithm 4 and the departure times at the oxygen converter computed by Algorithm 2.*

### 3.3 Backward Matching

We introduce the concept of *backward matches*, *i.e.*, matches from oxygen converter events to blast furnace events. The meaning of such a match is that a torpedo fulfilling the demand for the oxygen converter event $j \in M$ is used to serve the request for the blast furnace event $i \in N$. In other words, the torpedo used for $j$ is *reused* for $i$.

Since the torpedoes are identical and each blast furnace event requires exactly one torpedo, it does not matter which empty torpedo serves the event if more than one can be at the blast furnace in time. Algorithm 5 computes a backward matching in linear time with respect to the number of blast furnace events, when these events and the oxygen converter

**Algorithm 5:** Computation of a backward matching.

**Input** : $N$ an array of $n$ blast furnace events sorted in chronological order.
**Input** : $M$ an array of $m$ oxygen converter events sorted in chronological order.
**Output** : $\texttt{bm}$ a matching from oxygen converter to blast furnace events.

$arrBF :=$ Algorithm 4($N$);
$depOC :=$ Algorithm 2($M$);
**for** $o = 1$ **to** $m$ **do** $\texttt{bm}[o] := \infty$;
$bb := 1$; $oo := 1$;
**while** $bb \leq n$ **and** $oo \leq m$ **do**
  $b := N[bb]$; $o := M[oo]$;
  **if** $depOC[o] + tt_{(oc,eb)} + tt_{(eb,bf)} \leq arrBF[b]$ **then** $\texttt{bm}[o] := b$; $bb{+}{+}$; $oo{+}{+}$ ;
  **else** $bb{+}{+}$ ;
**return** $bm$;

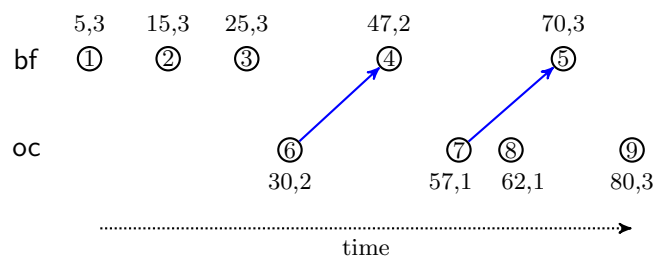


Figure 6: The backward matching for Example 1.1.

events are already sorted. Let $\texttt{bm} : M \to N \cup \{\infty\}$ denote the backward matching returned by Algorithm 5. Note that some of the last oxygen converter events can not be matched with any blast furnace event. We represent this case by a match to $\infty$.

**Example 3.3.** *Consider Example 1.1 from page 2. Then Algorithm 5 computes the backward matching as shown by the arrows in Figure 6, in which events 8 and 9 do not get a match.* □

**Theorem 3.4.** *Let $(N, M, G)$ be a feasible torpedo scheduling problem. Then there exists an optimal solution $S$ using the backward matching computed by Algorithm 5 for the reuse of torpedoes.*

*Proof.* Let $S'$ be an optimal solution. We can assume that $S'$ uses the departure times at the oxygen converter computed by Algorithm 2. We will construct a solution $S$ by swapping torpedoes in $S'$. Consider the first blast furnace event $b_1$ which uses a torpedo $t_1$ other than the assigned one $t_2$ in the backward matching $\texttt{bm}$. Without loss of generality, we assume that $b_2$ is the next blast furnace event that $t_2$ serves. Since $b_1$ is the earliest event that $t_2$ can serve after finishing its oxygen converter run, it holds $dd^{\mathsf{bf}}_{b_1} \leq dd^{\mathsf{bf}}_{b_2}$. Now, we distinguish regarding the origin of torpedo $t_1$. If the torpedo $t_1$ was never used before or returns from an emergency pit run then, clearly, we can swap the torpedoes for $b_1$ and $b_2$. If the torpedo $t_1$ returned from an oxygen converter trip then the departure time from the oxygen converter must be later than for $t_2$, otherwise $S'$ would deviate earlier from the backward matching. Since Algorithm 5 matched $t_2$ with the earliest possible blast furnace event, it holds that $dd^{\mathsf{bf}}_{b_1} \leq dd^{\mathsf{bf}}_{b_2}$. Therefore, the torpedoes can be swapped. □

Given a backward matching, it divides the blast furnace events into the set of *matched* events, *i.e.*, $V = \texttt{bm}(M) \setminus \{\infty\}$, and the set of *unmatched* events, *i.e.*, $U = N \setminus \texttt{bm}(M)$. Our solution method presented in the next section will extend this matching by matching torpedoes used for an emergency pit trip to unmatched events. As all departure and arrival times are known in the case of those trips, we reduce possible matchings to $R = \{(i,j) \in N \times U \mid dd_i^{\mathsf{bf}} + dur^{\mathsf{bf}} + tt_{(\mathsf{bf},\mathsf{eb})} + tt_{(\mathsf{eb},\mathsf{bf})} \leq \mathbf{arr}_j^{\mathsf{bf}}\}$, where $\mathbf{arr}_j^{\mathsf{bf}} = arrBF[j]$ computed by Algorithm 4.

**Example 3.4.** *Given the example from Example 3.3. Then,* $V = \{4,5\}$ *and* $U = \{1,2,3\}$. *The time cost for an emergency trip is from* $\mathsf{bf}$ *(including loading) back to it is* $5+20+1 = 26$. *Thus, no torpedo serving any blast furnace events would be able to return to* $\mathsf{bf}$ *in time for one of the unmatched one,* i.e.*,* $R = \emptyset$. *Therefore, the backward matching cannot be extended.*

## 4. Solution Method

The solution method is based on Logic-Based Benders Decomposition (Hooker & Ottosson, 2003). The idea of the Benders decomposition is to split the torpedo scheduling problem into an assignment problem used as the master problem and a scheduling problem. It alternates between solving the master and scheduling problem until an optimal solution is found or infeasibility is proven. Before we start the Benders decomposition, we preprocess an instance to determine the departure times at the oxygen converter and empty buffer, the arrival times at the blast furnace and empty buffer, and the backward matching `bm` as described in the previous section. We also perform an overload check at the blast furnace and oxygen converter provided with the corresponding times. If the check fails then we have proven the infeasibility of the instance. In the other case, we will start the Benders decomposition.

The master problem is formulated as a MIP, in which each oxygen converter event is assigned to a torpedo run, unmatched blast furnace events are matched with emergency pit trips, and the lexicographic objective of the problem is minimized. Then the remaining scheduling problem is split into smaller sub-problems using the optimal matching from the MIP solution. Each sub-problem is then solved as a constraint optimization problem minimizing the total time spent at the desulfurization station. If all sub-problems are feasible and the total time spent at the desulfurization station equals the corresponding lower bound in the MIP solution then we have found a globally optimal solution. If some sub-problems are not feasible, we compute minimal Benders cuts, add them to the MIP problem, and re-optimize the MIP. If some sub-problems require extra desulfurization time, we add optimality cuts, which forces the objective to take into account the extra desulfurization time, and re-optimize the MIP.[1] The optimality cuts can also make the MIP problem infeasible. In this case, it proves that the last found solution was the optimal one.

A brief overview of the notations introduced in the previous sections is shown in Table 1 on page 6. Table 2 lists the notations defined in this section.

1. Note that this case never occurred for generated instances and was only tested on handcrafted instances.

| Notation | Description |
|---|---|
| $\mathbf{x}_{ij}$ | A binary variable expressing whether the event $i \in N$ serves the event $j \in M$. |
| $\mathbf{r}_{ii'}$ | A binary variable expressing whether the torpedo of the event $i \in N$ is reused for the event $i' \in N$. |
| $\mathbf{obj}_1$ | The variable expressing the number of torpedoes used. |
| $\mathbf{obj}_2$ | The variable expressing the total time spent at the desulfurization station. |
| $eat_i^l$ | The earliest arrival (start) time of the event $i$ at the location $l$. |
| $eat_i^{(l,l')}$ | The earliest arrival (start) time of the event $i$ at the rail segment between locations $l$ and $l'$. |
| $ldt_i^l$ | The latest departure (completion) time of the event $i$ at the location $l$. |
| $ldt_i^{(l,l')}$ | The latest departure (completion) time of the event $i$ at the rail segment between locations $l$ and $l'$. |

Table 2: A brief overview of notations introduced in Section 4.

### 4.1 MIP Model

The MIP model tries to find an assignment from blast furnace events to converter events and reuse of torpedoes after emergency trips such that the number of torpedoes is minimized and for the minimal number of torpedoes, the lower bound on the desulfurization time is minimized.

Contrary to (Kletzander & Musliu, 2017; Geiger, 2017b), the idea of counting the number of torpedoes used is not based on how many torpedoes are doing an emergency pit or an oxygen converter trip at the same time, but rather modelled via the reuse of torpedoes. The backward matching `bm` already provides the reuse of torpedoes used for oxygen converter trips. Solving the MIP model just extends this backward matching for torpedoes used for emergency trips.

Besides the binary variables $\mathbf{ep}_i$ from the torpedo run, the MIP model uses the following binary variables. For each $(i,j) \in X$, we create a variable $\mathbf{x}_{ij} \in \{0,1\}$ expressing whether the torpedo run $i$ serves the demand of the oxygen converter event $j$. For each $(i,i') \in R$, the variables $\mathbf{r}_{ii'} \in \{0,1\}$ model whether the torpedo from torpedo run $i$ is reused for the blast furnace event $i'$.

$$\min \quad cap_{\mathsf{ds}} \cdot \left( \max_{j \in M} dd_j^{\mathsf{oc}} \right) \cdot \mathbf{obj}_1 + \mathbf{obj}_2 \tag{10}$$

$$\text{s.t.} \quad \mathbf{obj}_1 = |U| - \sum\nolimits_{(i,j) \in R} \mathbf{r}_{ij} \tag{11}$$

$$\mathbf{obj}_2 = \sum\nolimits_{(i,j) \in X} \mathbf{x}_{ij} \cdot \max(0, sul_i - sul_j) \cdot dur^{\mathsf{ds}} \tag{12}$$

$$\sum\nolimits_{(i,j) \in X} \mathbf{x}_{ij} = 1 \qquad \forall j \in M \tag{13}$$

$$\mathbf{ep}_i + \sum\nolimits_{(i,j) \in X} \mathbf{x}_{ij} = 1 \qquad \forall i \in N \tag{14}$$

$$\sum\nolimits_{(i,i') \in R} \mathbf{r}_{ii'} \leq \mathbf{ep}_i \qquad \forall i \in N \tag{15}$$

$$\sum\nolimits_{(i,i') \in R} \mathbf{r}_{ii'} \leq 1 \qquad \forall i' \in U \tag{16}$$

$$\mathbf{x}_{ij} \in \{0,1\} \qquad \forall i \in N, j \in M$$
$$\mathbf{r}_{ii'} \in \{0,1\} \qquad \forall i \in N, i' \in U$$
$$\mathbf{ep}_i \in \{0,1\} \qquad \forall i \in N$$

Constraint (10) states the objective of the MIP, which is split into two parts. The first part (11) models the minimization of the number of torpedoes, by maximizing the number of reused torpedoes for unmatched blast furnace events. We scale this objective by the product of the capacity of the desulfurization station and the *time horizon* — the latest time that a torpedo can be at the desulfurization station — in order to account for the lexicographic problem objective. Constraint (13) ensures each oxygen converter event is matched by one blast furnace event, whereas (14) matches each blast furnace event to an oxygen converter event or emergency trip. Constraint (15) models that a torpedo used for an emergency trip can be reused for at most one unmatched blast furnace event, whereas (16) ensures that at most one torpedo is reused for each unmatched blast furnace event. Note that the reuse of torpedoes for an oxygen converter trip is already determined by the backward matching `bm`, and thus can be left out of the model.

An assignment, **oc**, for a solution can be extracted from the MIP by setting for all $i \in N$, $\mathbf{oc}_i = \sum_{j \in M} \mathbf{x}_{ij} \cdot j$, this gives $\mathbf{oc}_i = 0$ whenever $\mathbf{ep}_i = 1$. All $\mathbf{ep}_i$ for the solution can be copied directly from the result of the MIP.

A MIP solution provides not only the matching of torpedo runs to oxygen converter events and the matching for the reuse of torpedoes, but also a lower bound on the desulfurization time, which is used as a quality measurement for the scheduling solution.

### 4.2 Partitioning the Scheduling Problem

Once, we have the assignment **oc**, the remaining scheduling problem is partitioned into smaller ones, which are then solved separately. We present two partitioning algorithms. The first one is opportunistic and very quick to compute, but may require additional steps to consolidate the individual schedules to a full one, whereas the second one is based on potential resource overloads and requires no consolidation steps, but is more complex to compute. In the remainder, we refer to locations and paths as resources.

Note that in both of these methods, we use the observation that emergency pit trips are not required to be passed in to the CP model in the same way as oxygen converter trips. Specifically, we can precompute the departure time from the blast furnace as $cap_{(\mathsf{bf},\mathsf{eb})} = cap_{\mathsf{eb}} = \infty$, hence they should leave the blast furnace as soon as they are filled, and their arrival times at the blast furnace are already precomputed by Algorithm 4. This means that for the one place where emergency pit trips interact with other trips, namely at the blast furnace, the resource capacity constraint can use these fixed values instead.

#### 4.2.1 Opportunistic Partitioning

The opportunistic partitioning splits the blast furnace events serving an oxygen converter event in chronological order, when it seems that a blast furnace event does not interfere in terms of time with any previous torpedo run at the desulfurization station. Algorithm 6

**Algorithm 6:** Computation of the sub-problems via opportunistic partitioning.

**Input** : $N$ an array of $n$ blast furnace events sorted in non-decreasing order of the due dates.
**Input** : **oc** an assignment from blast furnace events to oxygen converter events or 0 if not connected to any (*i.e.*, if $\mathbf{ep}_i = 1$).
**Output** : $B$ a partition of the $n$ blast furnace events.

$latestArrDS := -\infty$; $maxDur := \max(dur^{\mathsf{bf}}, tt_{(\mathsf{bf},\mathsf{fb})}, tt_{(\mathsf{fb},\mathsf{ds})}, tt_{(\mathsf{ds},\mathsf{oc})})$; $A := \emptyset$; $B := \emptyset$;
**for** $ii := 1$ **to** $n$ **do**
  **if** $\mathbf{oc}_{N[ii]} = 0$ **then continue**;
  $i := N[ii]$; $earliestArrDS := dd_i^{\mathsf{bf}} + dur^{\mathsf{bf}} + tt_{(\mathsf{bf},\mathsf{fb})} + tt_{(\mathsf{fb},\mathsf{ds})}$;
  **if** $latestArrDS + maxDur \leq earliestArrDS$ **and** $A \neq \emptyset$ **then**
    $B := B \cup \{A\}$; $A := \{i\}$;
  **else** $A := A \cup \{i\}$;
  $latestArrDS := \max(latestArrDS, dd_{\mathbf{oc}_i}^{\mathsf{oc}} - tt_{(\mathsf{ds},\mathsf{oc})} - dur^{\mathsf{ds}} \times \max(0, sul_{\mathbf{oc}_i}^{\mathsf{oc}} - sul_i^{\mathsf{bf}}))$;
**return** $B$;

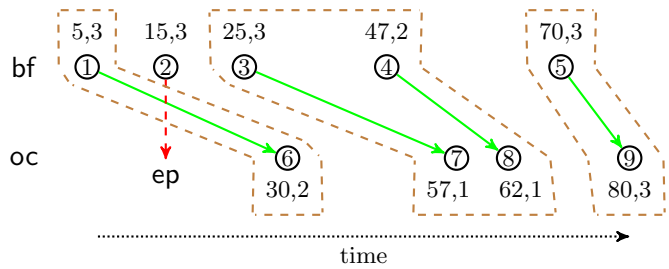


Figure 7: Partition of the scheduling given the shown matching.

computes all sub-problems.[2] It iterates over all blast furnace events in chronological order while skipping events going to the emergency pit. The set $A$ records all events belonging to the same sub-problem whereas the set $B$ collects all those sets. For each event $i$, it checks whether the earliest arrival time of its torpedo run at the desulfurization station is not before the latest arrival of any previous torpedo run. In that case, the current set $A$ is completed, $A$ is added to $B$, and a new set $A$ containing the event $i$ is created. In the other case, the set $A$ is expanded by $i$.

**Example 4.1.** *Consider Example 1.1 from page 2. Algorithm 6 will split the problem into three sub-problems as depicted in Figure 7.*

The condition used for partitioning ensures that a torpedo serving a blast furnace event $i$ does not interfere in time and space with any torpedo serving previous blast furnace events before the desulfurization station. But a time and space interference might occur at the desulfurization, on the way to the oxygen converter, and at the oxygen converter. Consequently, a composition of the schedules from two consecutive sub-problems may cause a resource overload. In such a case, we merge the two sub-problems and re-solve the now bigger problem. Note that this case has never occurred during our experiments.

2. Note that this algorithm is a correction of the one presented in the preliminary version in (Goldwaser & Schutt, 2017), in which we presented the wrong condition for partitioning that did not reflect the one used for our experiments.

#### 4.2.2 Overload Partitioning

The overload partitioning is based on the resource capacity of each location and each path in the network. It uses the earliest arrival and latest departure times for these places to compute maximum profiles, from which the sub-problems are derived. For each torpedo run $i$ with $\mathbf{ep}_i \neq 1$, we define the earliest arrival times by

$$
\begin{aligned}
eat_i^{\mathsf{bf}} &= \mathbf{arr}_i^{\mathsf{bf}}, & eat_i^{(\mathsf{bf,fb})} &= dd_i^{\mathsf{bf}} + dur^{\mathsf{bf}},\\
eat_i^{\mathsf{fb}} &= eat_i^{(\mathsf{bf,fb})} + tt_{(\mathsf{bf,fb})}, & eat_i^{(\mathsf{fb,ds})} &= eat_i^{\mathsf{fb}},\\
eat_i^{\mathsf{ds}} &= eat_i^{(\mathsf{fb,ds})} + tt_{(\mathsf{fb,ds})}, & eat_i^{(\mathsf{ds,oc})} &= eat_i^{\mathsf{ds}} + dur^{\mathsf{ds}} \times \max(0, sul_{\mathbf{oc}_i}^{\mathsf{oc}} - sul_i^{\mathsf{bf}}),\\
eat_i^{\mathsf{oc}} &= eat_i^{(\mathsf{ds,oc})} + tt_{(\mathsf{ds,oc})},
\end{aligned}
$$

and the latest departure times by

$$
\begin{aligned}
ldt_i^{\mathsf{bf}} &= ldt_i^{(\mathsf{bf,fb})} - tt_{(\mathsf{bf,fb})}, & ldt_i^{(\mathsf{bf,fb})} &= ldt_i^{\mathsf{fb}},\\
ldt_i^{\mathsf{fb}} &= ldt_i^{(\mathsf{fb,ds})} - tt_{(\mathsf{fb,ds})}, & ldt_i^{(\mathsf{fb,ds})} &= ldt_i^{\mathsf{ds}} - dur^{\mathsf{ds}} \times \max(0, sul_{\mathbf{oc}_i}^{\mathsf{oc}} - sul_i^{\mathsf{bf}}),\\
ldt_i^{\mathsf{ds}} &= ldt_i^{(\mathsf{ds,oc})} - tt_{(\mathsf{ds,oc})}, & ldt_i^{(\mathsf{ds,oc})} &= dd_{\mathbf{oc}_i}^{\mathsf{oc}},\\
ldt_i^{\mathsf{oc}} &= \mathbf{dep}_i^{\mathsf{oc}}.
\end{aligned}
$$

We extend these notations to torpedo runs $i$ going to the emergency pit, *i.e.*, $\mathbf{ep}_i = 1$.

$$
\begin{aligned}
eat_i^{\mathsf{bf}} &= \mathbf{arr}_i^{\mathsf{bf}} & eat_i^{(\mathsf{bf,fb})} &= eat_i^{\mathsf{fb}} = eat_i^{(\mathsf{fb,ds})} = eat_i^{\mathsf{ds}} = eat_i^{(\mathsf{ds,oc})} = eat_i^{\mathsf{oc}} = \infty\\
ldt_i^{\mathsf{bf}} &= dd_i^{\mathsf{bf}} + dur^{\mathsf{bf}} & ldt_i^{(\mathsf{bf,fb})} &= ldt_i^{\mathsf{fb}} = ldt_i^{(\mathsf{fb,ds})} = ldt_i^{\mathsf{ds}} = ldt_i^{(\mathsf{ds,oc})} = ldt_i^{\mathsf{oc}} = \infty
\end{aligned}
$$

The earliest arrival times and the latest departure times define the time intervals, in which a torpedo car uses up to one unit from the torpedo capacity of a resource over some time. If we overestimate the usage by assuming that a torpedo car requires one unit during his entire time intervals then we can compute *the maximum profile* for each resource. The maximum profile then provides an upper bound on the resource usages for each time over the planning horizon. The idea then is to look for time intervals for that the maximum profiles exceed the capacity and put the corresponding blast furnace events that may cause an overload in the same partition.

The overload partitioning then uses these maximum profiles retrieved from the earliest arrival and latest departure times at each location. The partition is computed by combining blast events that might caused a resource overload at any location at any time, *i.e.*, time units for that the maximum profile exceeds the resource capacity. Algorithm 7 shows a standard algorithm to compute the resource profile which is defined as a sorted array of triplets $(t, c, e)$ where $t$ is the time point when a change in the profile happens, $c$ the amount of the increase or decrease of the profile, and $e$ the event (or task) causing the change. The runtime complexity is dominated by the sorting (line 6), which is $\mathcal{O}(n \log n)$. A (linear) sweep from the start or end of the array is required for determining the height of the profile at a specific point in time. However, the partition algorithm described next has to perform a sweep anyway, so that it does not incur an additional cost.

**Algorithm 7:** getProfile($start$, $end$) : Computation of the resource profiles.

**Input** : $start$ an array of $n$ start times of a resource usage.
**Input** : $end$ an array of $n$ end times of a resource usage.
**Output** : $profile$ an array of at most $2n$ triplets $(t, u, i)$ sorted in non-descending lexicographic order, where $t$ is a time unit, $u$ the amount of change in the resource usage, and $i$ the task.

```
k := 1;
for i := 1 to n where start[i] ≠ ∞ do
    if start[i] ≠ end[i] then
        profile[k] := (start[i], 1, i);
        profile[k + 1] := (end[i], −1, i);
        k := k + 2;
sort profile in lexicographic ascending order of the triplets;
return profile;
```

**Algorithm 8:** Computation of the sub-problems via overload partitioning.

**Input** : $N$ an array of $n$ blast furnace events sorted in non-decreasing order of the due dates.
**Input** : $eat_i^r$ the earliest arrival time for all blast furnace events and resources.
**Input** : $ldt_i^r$ the latest departure time for all blast furnace events and resources.
**Output** : $B$ a partition of the $n$ blast furnace events.

```
for i := 1 to n do disjointSet.MakeSet(i);
for r ∈ L ∪ P do
    profile := getProfile(eat_·^r, ldt_·^r);
    height := 0; events := ∅;
    for k := 1 to profile.size do
        if 0 < profile[k].change then
            events := events ∪ {profile[k].event};
        else
            if height > cap_r then
                for two consecutive events i, j ∈ events do disjointSet.Union(i, j);
            events := events \ {profile[k].event};
        height := height + profile[k].change;
B := disjointSet.RetrievePartition();
return B
```

Algorithm 8 computes the overload partition using Algorithm 7 for computing the maximum profiles and disjoint-set data structure (Galler & Fisher, 1964; Tarjan, 1975) for finding and merging partitions. First, it initialises the disjoint-set data structure *disjointSet* by making each blast furnace event its own partition (line 1). Second, it then iterates over all resources (lines 2–12), *i.e.*, $\mathsf{bf}$, $(\mathsf{bf}, \mathsf{fb})$, $\mathsf{fb}$, $(\mathsf{fb}, \mathsf{ds})$, $\mathsf{ds}$, $(\mathsf{ds}, \mathsf{oc})$, and $\mathsf{oc}$. In the loop, it computes the profile of the resource (line 3) before performing a sweep over the profile (lines 5–12). During the sweep, it tracks the profile height *height* and the events *events* creating the height. If the height exceeds the resource capacity then the partitions to whom the events in *events* belong to are joined (lines 9, 10). Last, it retrieves the final partition and returns it (lines 13, 14).

### 4.3 The CP Model

Once we have an assignment **oc** and the opportunistic or overload partitioning of blast furnace events, each sub-problem is modeled and solved separately and the schedules obtained are combined to a schedule for the entire problem if possible.

Each sub-problem is modeled as a constraint optimization problem using the same model, but restricted to torpedo runs in the sub-problem, as in Definition 2.1 on page 7 except for the torpedo capacity constraints and an additional constraint enforcing an upper bound on the departure times at the blast furnace for avoiding an overload (17).

$$
\begin{aligned}
\min \quad & \sum\nolimits_{i \in N''} \mathbf{dep}_i^{\mathsf{ds}} - \mathbf{arr}_i^{\mathsf{ds}} \\
s.t. \quad & (3\text{–}7) \\
& \text{cumulative}((\mathbf{arr}_i^{\mathsf{bf}})_{i \in N'}, (\mathbf{dep}_i^{\mathsf{bf}} - \mathbf{arr}_i^{\mathsf{bf}})_{i \in N'}, (1)_{i \in N'}, cap_{\mathsf{bf}}) \qquad (17) \\
& \text{cumulative}((\mathbf{dep}_i^{l})_{i \in N''}, (\mathbf{arr}_i^{k} - \mathbf{dep}_i^{l})_{i \in N''}, (1)_{i \in N''}, cap_{(k,l)}) \qquad \forall (k,l) \in P' \\
& \text{cumulative}((\mathbf{arr}_i^{l})_{i \in N''}, (\mathbf{dep}_i^{l} - \mathbf{arr}_i^{l})_{i \in N''}, (1)_{i \in N''}, cap_{l}) \qquad \forall l \in L \setminus \{\mathsf{bf}\}
\end{aligned}
$$

where $N'$ are the blast furnace events of the sub-problem, $N''$ the blast furnace events of $N'$ going to the oxygen converter, $P' = \{(\mathsf{bf}, \mathsf{fb}), (\mathsf{fb}, \mathsf{ds}), (\mathsf{ds}, \mathsf{oc})\}$, and cumulative is global constraint modeling non-unary discrete resources in CP solvers (Aggoun & Beldiceanu, 1993) taking as arguments following parameters in this order: variable start times of tasks, variable durations of tasks, variable resource requirements of tasks, and the resource capacity. Note that Algorithms 2 and 4 already provide a non-overload schedule at $(\mathsf{eb}, \mathsf{bf})$ and $(\mathsf{oc}, \mathsf{eb})$ and therefore we can safely omit the corresponding constraints from the CP model.

We employ a sequential search over sub-searches, which represent a location or a path. Each sub-search branches over the duration of the torpedoes $i$ used for the location $l$ or the path $(k, q)$, *i.e.*, $\mathbf{dep}_i^l - \mathbf{arr}_i^l$ and $\mathbf{arr}_i^q - \mathbf{dep}_i^k$. The most constrained duration variable is selected first and its smallest possible duration is assigned to it. The sub-searches are explored in this order $\mathsf{ds}$, $(\mathsf{fb}, \mathsf{ds})$, $(\mathsf{ds}, \mathsf{oc})$, $\mathsf{oc}$, $(\mathsf{bf}, \mathsf{fb})$, and $\mathsf{bf}$. There are two important ingredients for this search. First, the search is objective driven, because it assigns the minimal duration spent at $\mathsf{ds}$ at first. Second, branching on the durations rather than on the departure or arrival times keeps the schedule flexible while providing some propagation on the departure and arrival time variables. Other searches tested, that did not follow both ingredients, were inferior.

**Caching** We recognized that from one MIP iteration to another one the assignment $\mathbf{oc}$ does not change very much. Thus, many sub-problems are the same. Instead of resolving the sub-problems from the scratch again, we cache them. For each sub-problem, we store the assignment and the solution. This storage is performed using a hash map of the actual assignment giving $\mathcal{O}(k)$ average case lookup, where $k$ is the length of the assignment.

### 4.4 Benders Cuts

The scheduling problem can have three possible outcomes. First, it is infeasible. Second, it is schedulable, but not with the lower bound on the desulfurization time from the MIP solution. Last, it is schedulable with the same desulfurization time. Only in the first two cases do we need to create Benders cuts in terms of the decision variables in the master problem. In the third case, the combined MIP and CP solution is an optimal solution of the entire problem.

We express the cuts in terms of the variables $\mathbf{x}_{ij}$ from the MIP problem. Let $N'$ be the set of blast furnace events in the sub-problem.

#### 4.4.1 Infeasibility Cuts

The sub-problem is infeasible, which is a direct result of the assignment $\mathbf{oc}$. Thus, $\sum_{i \in N'} \mathbf{x}_{i\mathbf{oc}_i} < |N'|$ is valid cut, because it forces the MIP solver to choose a different oxygen converter event for at least one torpedo run.

To strengthen the cut, we rerun the CP model $|N'|$-times, but with a small modification. For each rerun, we remove one torpedo run including the matched oxygen converter event from the model. If the model is still infeasible then this run does not contribute to the infeasibility and we can leave it out; otherwise it contributes to the infeasibility and we reinsert it. The removals are performed in chronological order of the blast furnace events. At the end of the process, we obtain a minimal unsatisfiable set of torpedo runs $N'' \subseteq N'$ leading to the stronger cut $\sum_{i \in N''} \mathbf{x}_{i\mathbf{oc}_i} < |N''|$, which is minimal too. In preliminary testing, this minimization resulted in an order of magnitude less MIP iterations.

We also investigated more general cuts by relaxing the conditions on the start time of a torpedo trip instead of removing it completely, but they were not beneficial for the overall runtime.

#### 4.4.2 Optimality Cuts

The sub-problem is schedulable with minimal desulfurization time $\beta$, but the desulfurization time $\alpha$ from the MIP solution is smaller, *i.e.*, $\alpha < \beta$. In this case, we introduce a new binary variable $b$ for the MIP model, add the term $(\beta - \alpha) \cdot b$ to the objective (10), and add the constraint $\sum_{i \in N'} \mathbf{x}_{i\mathbf{oc}_i} - (|N'| - 1) \leq b$ to the MIP model. The variable $b$ takes value 1 if and only if the MIP uses the same assignment $\mathbf{oc}$ for the sub-problem. In that case, the added objective term accounts for the difference in the desulfurization time derived by the CP model. If the variable $b$ takes value 0 then the MIP model is forced to take a different assignment due to the added constraint and the added objective term is zero.

### 4.5 Limited Forward Matchings

The size of the MIP model, *i.e.*, the number of constraints, variables, and the size of constraints, depends on the number of blast furnace and oxygen converter events. For example, the objective (12) has a quadratic size of $\mathcal{O}(nm)$. For large problems, the MIP model is so large that the MIP solver runs out of memory or is extremely slow. A way to reduce the size is to limit the oxygen converter events to which a blast furnace event can be matched and the reuse of torpedoes after emergency pit run. We look at two different ways.

**Event-based** For the event-based limited forward matching, we specify the absolute number of events, for example 10, that a blast furnace event can be matched to next closest reachable oxygen converter events, and the reuse of torpedo coming from the emergency trip can be matched to the next closest blast furnace events.

**Time-based** Instead of restricting forward matching by the absolute number of next closest event, we can impose a time restriction on the duration of how long a torpedo can take from picking up hot metal to dropping off it. Thus, we could enforce time restrictions for delivery of hot metal to the oxygen converter if existent. In comparison to the event-based limit, time-based limits provide more control over how long the delivery of hot metal can take, but less control over the model size.

However, in both cases, the model size is not only drastically smaller, but also significantly speeds up the solving time. The drawback is that we cannot prove the optimality of the original problem and the optimal solution of this relaxed problem can be worse than the one from the original problem. From a practical point of view, it might be the preferred mode because a matching of a blast furnace to an oxygen converter event far in the future can be seen as not preferable or sub-optimal due to cooling of the molten metal.

**Limits to Generate Cuts** Any cuts generated in runs with limited forward matchings are clearly valid globally, as they only talk about whether an assignment is feasible or how much extra desulfurization time beyond the lower bound it would need. This gives rise to a hybrid approach where the instance is first solved to completion using a limited forward matching, then the cuts used in that solution process are combined with the MIP model for an unlimited version. The benefit of this is that it will require fewer iterations of solving the large MIP without the forward limit, which dominates the runtime in most cases. This approach also conserves the proof of optimality as the limited forward matching is double checked with the unlimited matching version afterwards.

## 5. Experiments

We conducted experiments on a variety of instances, either taken from the ACP 2016 Challenge, created using the instance generator provided at the ACP Challenge website or, in the case of the extended network, our own instance generator. We group all instances into two benchmark sets called ACPNet and ExtNet. ACPNet contains all instances having the same capacities for the rail network and the blast furnace as they appear in ACP 2016 Challenge, all these capacities are unary. ExtNet contains all other instances. All instances are available at `https://github.com/AdGold/TorpedoSchedulingInstances`. We ran all

our experiments on Dell PowerEdge M630 machines having Intel Xeon E5-2660 V3 processors running at 2.6 Ghz with a 25 MB cache and 8GB RAM, unless otherwise stated. Our solution method was implemented in Python 3.5.1 interfacing Gurobi 7.0.2 using the Python library gurobipy and Chuffed using system calls. Gurobi was used for solving the MIP problem and executed with the default settings and on 2 cores. Chuffed (Chu, 2011) was used for solving the CP problems. The time-tabling propagator with explanation generation as described in (Schutt, 2011; Schutt, Feydy, Stuckey, & Wallace, 2011) was used for all cumulative constraints in the CP model. We impose a runtime limit of 5 minutes for Chuffed for solving any CP problem. No runtime limit was imposed for solving the MIP problem. If Chuffed timed out then we aborted the solution method. All solutions were verified using the ACP solution checker or our extended solution checker for instances in ExtNet.

### 5.1 Results on ACPNet Instances

We group the instances in this benchmark set as follows:

**small** 15 instances with 30 to 500 blast furnace events,

**comp** 6 instances with 850 to 2500 blast furnace events,

**medium** 19 instances with 1000 to 3000 blast furnace events, and

**large** 3 instances with 10000 blast furnace events.

**50k** 3 instances with 50000 blast furnace events, used only for looking at forward limit runtime.

The test **small** comprises all nine test instances from the ACP 2016 Challenge and six additional created instances, the test **comp** all competition instances from the challenge whereas the others were generated by using the provided instance generator. On these sets, we compare our solution method using unlimited and limit forward matchings, and opportunistic and overload partitioning.

We record following parameters in different tables. We list the instance name (Inst), the number of torpedoes (#T), the desulfurization time spent at ds (Desulf), the total runtime (RT), the percentage of the total runtime that was used by the MIP solver (MT), the number of iterations (#I), the cache hit rate for sub-problems (CHR), the number of total sub-problems stores (#SP), the success rate of sub-problems (SSR), *i.e.*, no cuts needed to be generated, the percentage of sub-problems with size 1 (S1), the average size of sub-problems with size greater than 1 (SAvg), the maximal size of sub-problems (SMax), and the percent reduction in maximal sub-problem size (RMSS).

#### 5.1.1 Unlimited Forward Matchings

Table 3 shows the results on the set **comp** for each instance when using opportunistic partitioning (Algorithm 6). All ACP challenge instances were optimally solved in less than 10 minutes, which is much quicker than the winning method presented in (Kletzander & Musliu, 2017). In addition, it is the first time that the optimality was proven. The results

| Inst | #T | Desulf | RT | MT | #I | CHR | #SP | SSR | S1 | SAve | SMax |
|---|---|---|---|---|---|---|---|---|---|---|---|
| instance01 | 4 | 7695 | 56s | 58% | 2 | 38% | 4 | 94% | 63.6% | 144 | 604 |
| instance02 | 4 | 5302 | 117s | 77% | 1 | 0% | 16 | 100% | 63.6% | 31 | 769 |
| instance03 | 3 | 27150 | 301s | 93% | 1 | 0% | 100 | 100% | 82.5% | 3 | 215 |
| instance04 | 3 | 10676 | 43s | 80% | 1 | 0% | 149 | 100% | 81.9% | 1 | 5 |
| instance05 | 4 | 16308 | 471s | 87% | 3 | 50% | 203 | 99% | 80.2% | 3 | 1747 |
| instance06 | 4 | 7755 | 360s | 90% | 1 | 0% | 10 | 100% | 67.7% | 75 | 1149 |

Table 3: Detailed results on **comp** using opportunistic partitioning.

| Inst | Method | #T | Desulf | RT | MT | #I | CHR | #SP | SSR |
|---|---|---|---|---|---|---|---|---|---|
| **small** | opp. part. | 3.5 | 362 | 3s | 33% | 1.1 | 3% | 9 | 99.3% |
| | over. part. | 3.5 | 362 | 3s | 33% | 1.1 | 3% | 10 | 99.5% |
| | lim. + unlim. | 3.5 | 362 | 3s | 28% | 2.1 | 2% | 10 | 99.2% |
| **comp** | opp. part. | 3.7 | 12481 | 225s | 81% | 1.5 | 14% | 80 | 99.1% |
| | over. part. | 3.7 | 12481 | 218s | 84% | 1.5 | 14% | 29 | 98.3% |
| | lim. + unlim. | 3.7 | 12481 | 210s | 71% | 2.5 | 14% | 80 | 99.1% |
| **medium** | opp. part. | 4.4 | 1223 | 271s | 92% | 1.1 | 4% | 34 | 99.6% |
| | over. part. | 4.4 | 1223 | 270s | 92% | 1.1 | 4% | 41 | 99.7% |
| | lim. + unlim. | 4.4 | 1223 | 287s | 79% | 2.2 | 6% | 22 | 99.2% |
| **large** | opp. part. | 4.5 | 6480 | 30679s | 99% | 2.0 | 9% | 131 | 99.5% |
| | over. part. | 4.5 | 6480 | 30939s | 99% | 2.0 | 9% | 169 | 99.7% |
| | lim. + unlim. | 4.5 | 6480 | 14845s | 98% | 3.0 | 25% | 117 | 99.6% |

Table 4: Results on each test set excluding infeasible instances.

also reveal that the MIP solver used the majority of the runtime and the sub-problems had almost 100% success, which lead to a very low number of iterations. In addition, most sub-problems were small and only a few contained 100s of blast furnace events. Note that the nogood learning solver Chuffed was essential for quickly solving the sub-problems. In particular on larger and infeasible ones, we had to terminate the process if using Gecode.

Table 4 presents the results on all ACPNet test sets excluding infeasible instances. For each entry it shows an average over the number of (feasible) instances. It compares three different variations of our solution method:

**opp. part.:** The solution method uses the opportunistic partitioning and unlimited forward matching.

**over. part.:** The solution method uses the overload partitioning and unlimited forward matching.

**lim. + unlim.:** At first, we run the solution method using opportunistic partitioning and event-based limited forward matching of 30 events. Afterwards we run the same method with unlimited forward matching, but including all cuts from the first run and restricting the upper bound on the objective with the objective value found in the first run.

The results of **opp. part.** on the different sets show a similar picture to the ACP challenge instances (**comp**) in Tab. 3, *i.e.*, majority time is spent on solving the MIP problem (except on the small instances), the number of iterations is one or two in the most cases, and almost all sub-problems are feasible. Note that for the large size instances in **large**, the runtime was more than 30 hours and it had to be run on a machine with extra RAM in order to be able to solve the MIP. This machine had the same specifications as the others except that it used 128GB of RAM instead.

Both **opp. part.** and **over. part.** both have very similar results on the ACPNet benchmarks. This is because there are few places with a capacity of more than one.

Except for **large**, there is no significant difference between the methods in terms of runtime. However, the hybrid solution method **lim. + unlim.** is twice as fast as the others for **large**.

The reason this method is very similar for the **small**, **comp** and **medium** benchmarks is that the MIP solving is quite quick, hence anywhere where time saved through not doing more unlimited runs of the MIP is cancelled out by having to run an initial limited forward matching run. However, on the **large** benchmark, the runtime is halved because the MIP takes many hours to solve and the number of times that it has to be solved in the unlimited case can be reduced from 2 to 1, essentially halving the solve time as the MIP takes up 99% of the runtime. The MIP also involved large objectives due to the constant used in the lexicographical ordering of objectives, however these did not exceed 15 million even on the largest cases tested so never caused problems.

The runtimes for most instances were slightly shorter than in (Goldwaser & Schutt, 2017), while a few took longer. The reason for this was due to the removal of a redundant constraint in the MIP which caused different assignments to be found, in most cases reducing the number of iterations needed to find a feasible assignment and in some cases making the subproblems simpler to solve.

#### 5.1.2 Limited Forward Matchings

Figures 8–10 show the development of the optimality gap on the desulfurization time spent, of the percentage of instances optimally solved, and of the runtime when the limit on the event-based forward matchings increases. The benchmark **50k** was here run with 40GB of memory and the same specifications for everything else. Note that Figure 10 uses logarithmic scale for the y-axis. The optimality gap on the desulfurization time spent converges quickly on each test set. Between a limit of 30 and 40 the last optimal solution was found even on the test set **large**.

The runtime could be reduced by orders of magnitude for medium and large scale problems, especially for large scale instances where the runtime was reduced to less than 30 minutes. All ACP challenge instances were solved in less than one minute, down from 10 minutes. In order to test the limit of our method, we created six new instances, three each with 50,000 and 100,000 blast furnace events, respectively. The average total runtime of the 50k instances were below 60 minutes except for a limit of 30 or more as shown in Figure 10. Interestingly, the same optimal solutions were generated with limits of at least 10. The 100k instances were solved between 70 minutes and 3.5 hours for a limit of 20.

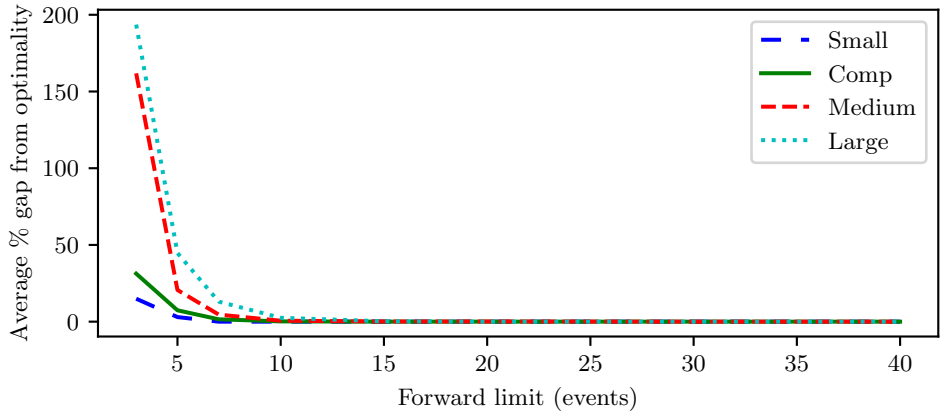


Figure 8: Optimality gap in desulfurization time using event based forward limits.

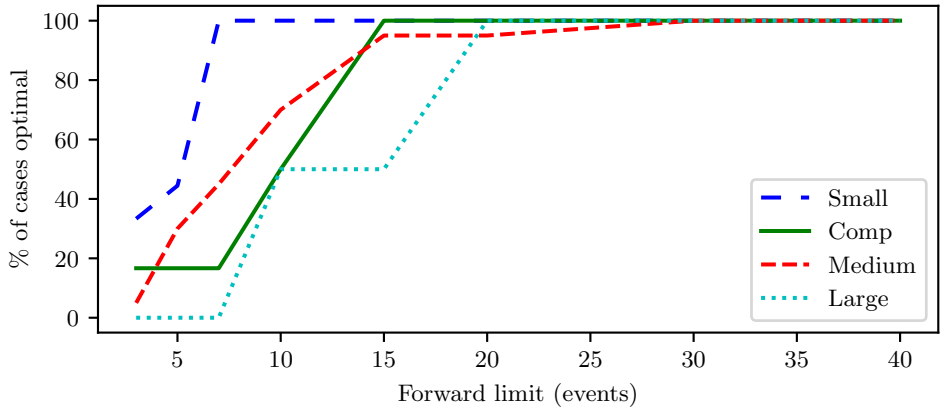


Figure 9: Percent of instances solved optimally. using event based forward limits.

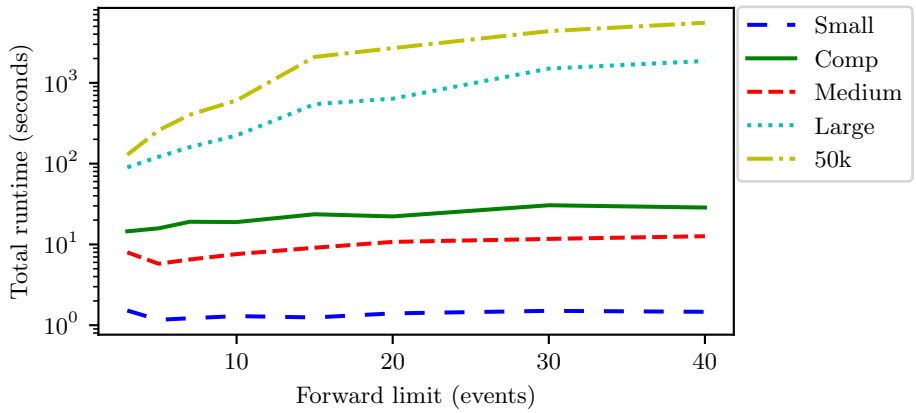


Figure 10: Total runtime using event based forward limits.

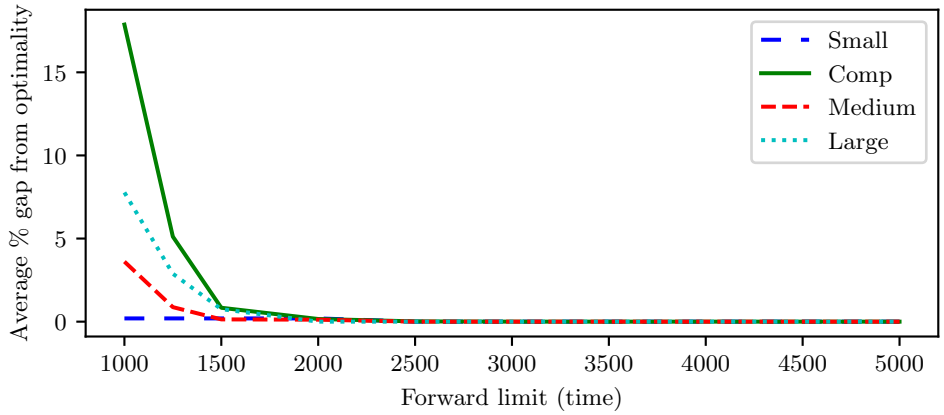


Figure 11: Optimality gap in desulfurization time using time based forward limits.

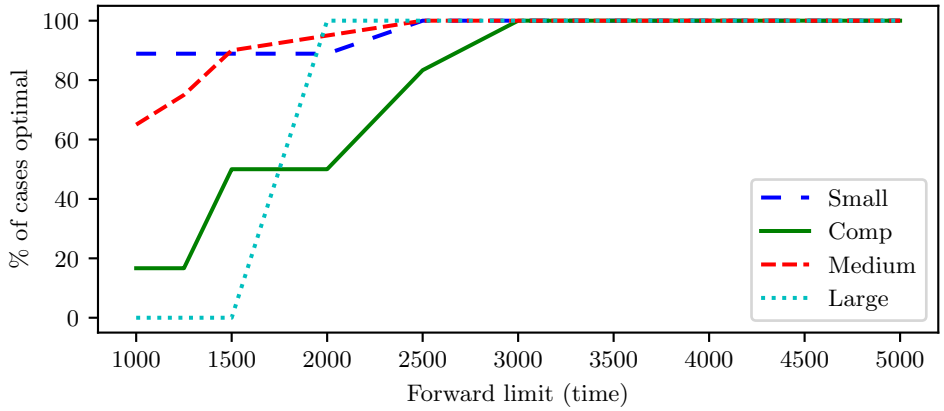


Figure 12: Percent of instances solved optimally using time based forward limits.

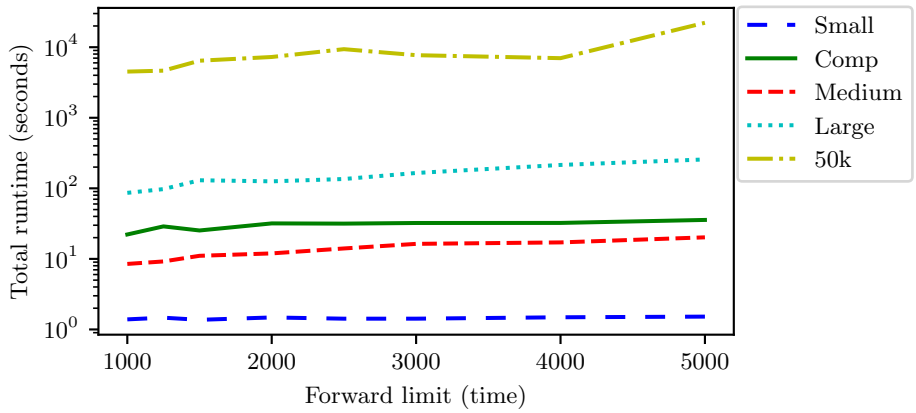


Figure 13: Total runtime using time based forward limits.

| Inst | Method | #T | Desulf | RT | MT | #I | #SP | SAve | RMSS |
|---|---|---|---|---|---|---|---|---|---|
| **small-2x** (51) | opp. part. | 6.5 | 624 | 14s | 37% | 1.1 | 18 | 33 | |
| | over. part. | 6.5 | 624 | 10s | 37% | 1.1 | 56 | 9 | 29.4% |
| **small-3x** (31) | opp. part. | 9.3 | 513 | 4379s | 43% | 104 | 354 | 391 | |
| | over. part. | 9.3 | 513 | 4285s | 43% | 104 | 752 | 711 | 26.9% |

Table 5: Results on instances from ExtNet solved by both methods using unlimited forward matching.

We also show, for comparison, similar graphs for time-based limits in Figures 11–13. Here it can be seen that there are a few differences. Optimality gaps start a lot lower, this is mostly a factor of the graph starting when all instances can be solved with that limit and event based limits being less dependent on the actual instance due dates in comparison to time based limits.

The other interesting point is that the **large** cases start at similar points but event-based limits grow much faster, rising up to over 20 minutes for a limit of 40, whereas time-based limits never took more than 5 minutes for the same cases. However, the opposite can be seen in the **50k** instances where time based limits were almost as slow with the smallest limit than event based ones were with the highest.

### 5.2 Results on ExtNet Instances

In order to test the solution method on denser instances, *i.e.*, instances having more blast furnace and oxygen converter events over a similar planning horizon, we created new instances based on the instances from the test set **small** having up to 300 blast furnace events. For each of these instances in **small**, we created ten new instances; five instances with the twice as many blast furnace and oxygen converter events and five instances with the triple the amount of those events. We copied the events with the same due dates and sulfur level, we then shifted the copied events by the same amount of time units backwards in the planning horizon, and finally, we individually shifted blast furnace and oxygen converter events by small amount of time units forward and backwards, respectively. We randomly picked the amount of time units to be shifted, but at most twice as the unloading duration of a torpedo at the converter. We grouped the benchmark set ExtNet into two test sets.

**small-2x** 60 instances with 60 to 600 blast furnace events created by doubling the number of events in instances from **small**, and

**small-3x** 60 instances with 90 to 900 blast furnace events created by tripling the number of events in instances from **small**.

Table 5 presents the results for both partitioning algorithms and unlimited forward matchings on 79 instances in ExtNet that were solved by our solution method with either partitioning algorithm. First, we observe that not all instances were solved as it was for their corresponding instances in ACPNet. Second, the number of solved instances decreases with an increase of the density. Both observations are not surprising, because if the density

| Inst | Method | #T | Desulf | RT | MT | #I | #SP | SAve | RMSS |
|---|---|---|---|---|---|---|---|---|---|
| **small-2x** (26) | opp. part. | 7.2 | 967 | 26s | 53% | 1.1 | 30 | 46 | |
| | over. part. | 7.2 | 967 | 19s | 53% | 1.1 | 88 | 12 | 31.3% |
| **small-3x** (12) | opp. part. | 10 | 722 | 11309s | 52% | 266 | 905 | 918 | |
| | over. part. | 10 | 722 | 11067s | 55% | 266 | 1913 | 1811 | 28.8% |

Table 6: Results on instances from ExtNet solved by both methods using unlimited forward matching, excluding those with a runtime of under 5s with either method.

increases then there are less opportunities to partition the scheduling problem and, consequently, the size of the scheduling sub-problems are larger as shown in the average size of sub-problems (SAve) in the table. Thus, the sub-problems are harder to optimally solve for the CP solver and more time has to be spent for the scheduling part. For example, the solution method spent more than 90% of the runtime for solving the CP part on instances that could not be solved within 40 seconds. Since the scheduling part becomes harder to solve our solution method deteriorates and hits its limitation, because it relies on fast solving of the scheduling problem.

Table 6 shows the same results as in Tab. 5 but excludes instances that were solved in under five seconds using either partitioning algorithm. In these tables, the number in the brackets after the test set shows the number of instances considered. This table provides a better comparison between opportunistic and overload partitioning when the easy-to-solve instances are excluded. It clearly shows that there is a runtime benefit when using the finer overload partitioning algorithm, which was able to reduce the size of the biggest sub-problems (RMSS). We note that an additional three instances from **small-3x** were solved when using overload partitioning. As expected, the average number of sub-problems and the average size of sub-problems are higher and lower respectively. The majority of the number of additional sub-problems for overload partitioning resulted from dividing sub-problems from the opportunistic partitioning, which could already be solved quickly. Thus, for those ones there was no or a negligible benefit as shown in Tab. 4 for instances in ACPNet. The difference lays in the ability to split the biggest sub-problems into smaller ones. Computing the overload partitioning was orders of magnitudes more costly, but even for the bigger instances a fraction of a second. To sum up, the overload partitioning is clearly preferable for more dense instances such as in ExtNet, due to the potential to generate smaller sub-problems.

## 6. Conclusion

We propose a logic-based Benders decomposition solution method for the industrial problem of torpedo scheduling in steel production and also extend this solution to a generalised version of the torpedo scheduling problem. The master problem was modeled as a MIP, which takes care of the assignment component of the problem and the lexicographical objective. The remaining scheduling problem was split into smaller sub-problems and solved by a CP solver with nogood learning. This solution method is the first exact one for the torpedo scheduling problem and is the first one, that could prove the optimality of all instances

from the ACP 2016 Challenge in less than 10 minutes. Thus, it outperforms the previous state of the art. A limited version of our method, which cannot guarantee optimality, could reduce the runtime by an order of magnitude and was able to find optimal solutions very quickly for even larger instances that we created. The success of the solution method relies in the fast solving of the scheduling sub-problems, which in general took no time compared to the master problem. In order to test the limitation of our method, we also generated small dense instances, which quickly became hard to solve for our solution method, because the maximal size of the scheduling sub-problems increased, so that it became hard to solve by the CP solver.

## Acknowledgments

A preliminary version of this paper appears as (Goldwaser & Schutt, 2017).

This work was partially supported by the Asian Office of Aerospace Research and Development grant 15-4016.

## References

Aggoun, A., & Beldiceanu, N. (1993). Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and computer modelling*, *17*(7), 57–73.

Chu, G. G. (2011). *Improving Combinatorial Optimization.* Ph.D. thesis, The University of Melbourne.

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.

Galler, B. A., & Fisher, M. J. (1964). An improved equivalence algorithm. *Commun. ACM*, *7*(5), 301–303.

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., Geiger, M. J., Rommelfanger, H., & Metzger, O. (Eds.), *Entscheidungsunterstützung in Theorie und Praxis: Tagungsband zum Workshop FEU 2016 der Gesellschaft für Operations Research e.V.*, pp. 63–86. Springer Fachmedien Wiesbaden, Wiesbaden.

Goldwaser, A., & Schutt, A. (2017). Optimal torpedo scheduling. In *International Conference on Principles and Practice of Constraint Programming*, pp. 338–353. Springer.

Hooker, J., & Ottosson, G. (2003). Logic-based Benders decomposition. *Mathematical Programming*, *96*(1), 33–60.

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 steelmaking plants. *IFAC Proceedings Volumes*, *44*(1), 9476 –9481.

Kikuchi, J., Konishi, M., & Imai, J. (2008). Transfer planning of molten metals in steel works by decentralized agent. In *Memoirs of the Faculty of Engineering*, Vol. 42, pp. 60–70. Okayama University.

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 for Combinatorial Optimization Problems – CPAIOR '17*, Lecture Notes in Computer Science, pp. 344–358. Springer International Publishing.

Kumakura, M. (2013). Advances in steel refining technology and future prospects. Tech. rep. 104, Nippon Steel, Steelmaking Technical Department.

Li, J.-q., Pan, Q.-k. P., & 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. Y., & Wang, G. S. (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 – CISIA 2015*, pp. 731–734. Atlantis Press.

Ohrimenko, O., Stuckey, P. J., & Codish, M. (2009). Propagation via lazy clause generation. *Constraints*, *14*(3), 357–391.

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`. Last accessed: 28 Apr 2017.

Schutt, A. (2011). *Improving Scheduling by Learning*. Ph.D. thesis, The University of Melbourne.

Schutt, A., Feydy, T., Stuckey, P. J., & Wallace, M. G. (2011). Explaining the cumulative propagator. *Constraints*, *16*(3), 250–282.

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.

Tarjan, R. E. (1975). Efficiency of a good but not linear set union algorithm. *J. ACM*, *22*(2), 215–225.

Wang, G., & Tang, L. (2007). A column generation for locomotive scheduling problem in molten iron transportation. In *2007 IEEE International Conference on Automation and Logistics*, pp. 2227–2233.