Optimal Torpedo Scheduling

Adrian Goldwaser[1,3] and Andreas Schutt[2,3]

[1] The University of New South Wales, Australia
[2] The University of Melbourne, Australia
[3] Decision Sciences, Data61, CSIRO, Australia
adrian.goldwaser@gmail.com,andreas.schutt@data61.csiro.au

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, the locations traversed and 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 20 minutes. In addition, we adapted our method to handle large-scale instances. 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 materials are melted in a blast furnace. In the second stage, the steel making, the hot metal is loaded in torpedo cars, or *torpedoes*, transported to different locations for improving its quality, and finally brought to an oxygen converter, in which it is poured. Once at the oxygen converter, the hot metal is further refined before the last two stages of continuous casting and hot trip mill. This work focuses on the rotation of the torpedoes between the blast furnace and oxygen converter in the steel making stage. 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. [12] for the ACP Challenge 2016. This problem focuses on the assignment of blast furnace events to oxygen converter events and the scheduling problem

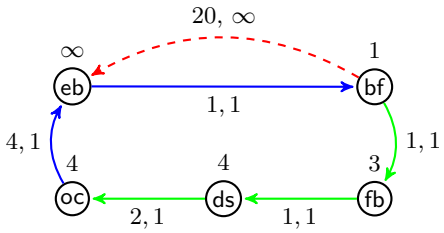

Figure 1. The graph for Ex. 1.

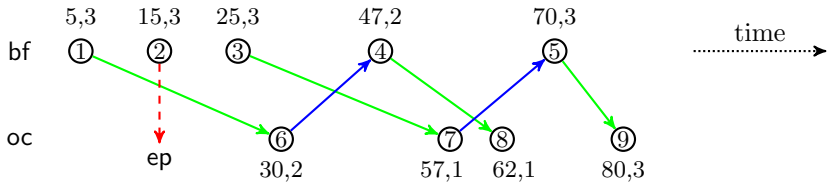

Figure 2. A small example of a torpedo scheduling problem.

of transporting the hot metal through different locations while satisfying all scheduling constraints and the quality constraint, sulfurization level, on the hot metal. Figure 1 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. Figure 2 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 Fig. 1, *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. For fulfilling the demands for oxygen converter events 6, 7, and 8, a total of 20 time units have to be spent for desulfurization. □

To best of our knowledge, the torpedo scheduling problem we study was proposed at the ACP Challenge 2016. From ten teams, who took part, we are only

aware of the publication of the winning team [8] and the third placed team [4]. Kletzander and Musliu [8] 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 to 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 [4] 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 [3]. In order to reduce the search tree size, Geiger removes infeasible assignments in a preprocessing step and after a solution is found. Both methods [4,8] 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 [2,7,10], the molten iron allocation problem [13], the molten scheduling problem [6,9], and the locomotive scheduling problem [14]. All those works use different solution methods such as local search, mixed integer programming, and column generation, but none use Logic-based Benders Decomposition [5] and Constraint Programming (CP) as we do in the present paper.

We propose a Logic-Based Benders Decomposition [5] method, in which the assignment problem and the lexicographical objective is handled in the master problem. The remaining scheduling is partitioned, 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 CP solver with Nogood Learning applying Lazy Clause Generation (LCG) [11]. 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 [8] 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.

2 Torpedo Scheduling

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 plant. The *oxygen converter trip* delivers

the hot metal to the converter and visits the locations in this order eb, bf, fb, ds, oc, and eb, whereas the *emergency pit trip* dumps the hot metal at the emergency pit and visits the locations in this order eb, bf, and 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 $cap_{\mathsf{bf}} = 1$ and $cap_{\mathsf{ep}} = \infty$. We extend this notation for edges $p \in P$, which is cap_p. All edges $p \in P \setminus \{(\mathsf{bf},\mathsf{eb})\}$ have a unit capacity $cap_p = 1$, whereas $cap_{(\mathsf{bf},\mathsf{eb})} = \infty$. A torpedo traversing the edge p requires a minimal *transition time* of tt_p.

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 periods at the blast furnace, while unloading takes $dur^{\mathsf{oc}} \in \mathbb{N}$ time periods at the oxygen converter. Reducing the sulfur level of hot metal by one unit requires $dur^{\mathsf{ds}} \in \mathbb{N}$ time periods at the desulfurization station.

Following [8], a *torpedo run* i is either a converter or emergency pit trip. In the former case, it is specified by variable departure times dep_i^l, variable arrival times arr_i^l for locations in $\{\mathsf{eb},\mathsf{bf},\mathsf{fb},\mathsf{ds},\mathsf{oc}\}$, and the variable converter event $oc_i \in M$, 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 ep_i whether it is a converter trip $ep_i = 0$ or an emergency pit trip $ep_i = 1$.

Definition 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. A solution satisfies the capacity constraints on each location (1) and on each edge (2),*

$$\sum\nolimits_{i \in S : arr_i^l \le t < dep_i^l} 1 \le cap_l \qquad \forall l \in L, \forall t \in \mathbb{N}^0 \tag{1}$$

$$\sum\nolimits_{i \in S : dep_i^l \le t < arr_i^k} 1 \le cap_{(l,k)} \qquad \forall (l,k) \in P, \forall t \in \mathbb{N}^0 \tag{2}$$

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

$$arr_i^k - dep_i^l \ge tt_{(l,k)} \qquad \forall i \in S : ep_i = 0, \forall (l,k) \in P \setminus \{(\mathsf{bf},\mathsf{eb})\} \tag{3}$$

$$arr_i^k - dep_i^l \ge tt_{(l,k)} \qquad \forall i \in S : ep_i = 1, \forall (l,k) \in \{(\mathsf{eb},\mathsf{bf}),(\mathsf{bf},\mathsf{eb})\} \tag{4}$$

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

$$arr_i^{\mathsf{bf}} \le dd_i^{\mathsf{bf}} \quad \wedge \quad dd_i^{\mathsf{bf}} + dur^{\mathsf{bf}} \le dep_i^{\mathsf{bf}} \qquad \forall i \in S \tag{5}$$

$$arr_i^{\mathsf{oc}} \le dd_{oc_i}^{\mathsf{oc}} \quad \wedge \quad dd_{oc_i}^{\mathsf{oc}} + dur^{\mathsf{oc}} \le dep_i^{\mathsf{oc}} \qquad \forall i \in S : ep_i = 0 \tag{6}$$

$$sul_i - \left\lfloor \frac{dep_i^{\mathsf{ds}} - arr_i^{\mathsf{ds}}}{dur^{\mathsf{ds}}} \right\rfloor \leq sul_{oc_i} \qquad \forall i \in S : ep_i = 0 \quad (7)$$

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 [8,4]. The secondary objective (9) is to minimize the total time spent at the desulfurization station.*

$$\min \max_{t \in \mathbb{N}^0} |\{i \in S \mid dep_i^{\mathsf{eb}} \leq t \wedge t < arr_i^{\mathsf{eb}}\}| \quad (8)$$

$$\min \sum\nolimits_{i \in S : ep_i = 0} dep_i^{\mathsf{ds}} - arr_i^{\mathsf{ds}} \quad (9)$$

Note that the solution does not provide an assignment of individual torpedoes to the torpedo runs. But such an assignment can be computed in polynomial time with respect to the number of torpedo runs using a stack for torpedoes in the empty buffer and a return queue of torpedoes sorted ascending by their arrival (return) times to the empty buffer. The algorithm would iterate over due dates of blast furnace events and the arrival times in the return queue in chronological order. Depending on the case, it either pops a torpedo from the stack and pushes it into the return queue or vice versa.

Moreover, as already observed in [8,4] 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 $X = \{(b, o) \in N \times M \mid dd_b^{\mathsf{bf}} + tt_{(\mathsf{bf},\mathsf{fb})} + tt_{(\mathsf{fb},\mathsf{ds})} + tt_{(\mathsf{ds},\mathsf{oc})} + dur^{\mathsf{ds}} \cdot \max(0, sul_b - sul_o) \leq dd_o^{\mathsf{oc}}\}$ 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 $dep_i^{\mathsf{bf}} = dd_i^{\mathsf{bf}} + dur^{\mathsf{bf}}$ and $arr_i^{\mathsf{eb}} = dep_i^{\mathsf{bf}} + tt_{(\mathsf{bf},\mathsf{eb})}$ if the torpedo run i goes to the emergency pit.

Example 2. Given the example from Ex. 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.

3 Preprocessing

Before solving the problem, we perform preprocessing steps in order to simplify the problem and setup the structure needed for our solution approach.

3.1 Departure Times from the Oxygen Converter

The empty buffer has unlimited capacity, this means that is it never suboptimal to get an empty torpedo there earlier rather than later as it can be reused earlier,

Algorithm 1: Computation of departure times from the oxygen converter.

Input : M an array of m oxygen converter events sorted in chronological order.

1 $j; = M[1]$; $depOC[j] := dd_j^{\mathsf{oc}} + dur^{\mathsf{oc}}$; $arrEB[j] := depOC[j] + tt_{(\mathsf{oc},\mathsf{eb})}$;
2 **for** $jj := 2$ **to** m **do**
3 $\quad j := M[jj]$;
4 $\quad depOC[j] := \max(arrEB[M[jj-1]], dd_j^{\mathsf{oc}} + dur^{\mathsf{oc}})$;
5 $\quad arrEB[j] := depOC[j] + tt_{(\mathsf{oc},\mathsf{eb})}$;

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, *i.e.*, $dd_i^{\mathsf{oc}} + dur^{\mathsf{oc}}$, and the arrival time of the previous torpedo at the empty buffer from the oxygen converter, *i.e.*, arr_j^{eb}.

Since the due dates for the oxygen converter events are known a priori, the departure dates 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 Alg. 1. Note that the order of torpedoes serving oxygen converter events remains unchanged by the algorithm. It is obvious that the following holds.

Proposition 1. *Algorithm 1 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.*

Example 3. Given the example from Ex. 1 from page 2. Then Alg. 1 respectively computes departure times 35, 62, 67, and 85 for the oxygen converter events 6, 7, 8, and 9. □

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 2 is symmetric to Alg. 1 for the arrival times at the blast furnace. It computes the times in reverse-chronological order of the blast furnace events. With similar arguments as in the oxygen converter case, the following claims hold.

Proposition 2. *Algorithm 2 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.*

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.

Algorithm 2: Computation of arrival times at the blast furnace.

Input : N an array of n blast furnace events sorted in chronological order.

1 $i := N[n]$; $arrBF[i] = dd_i^{\mathsf{bf}}$; $depEB[i] := arrBF[i] - tt_{(\mathsf{eb},\mathsf{bf})}$;
2 **for** $ii := n - 1$ **down to** 1 **do**
3 $\quad i := N[ii]$;
4 $\quad arrBF[i] := \min(depEB[N[ii+1]], dd_i^{\mathsf{bf}})$;
5 $\quad depEB[i] := arrBF[i] - tt_{(\mathsf{eb},\mathsf{bf})}$;

Algorithm 3: 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.

1 $dep :=$ Alg. 1(M);
2 **for** $o = 1$ **to** m **do** $\mathtt{bm}[o] := \infty$;
3 $bb := 1$; $oo := 1$;
4 **while** $bb \leq n$ **and** $oo \leq m$ **do**
5 $\quad b := N[bb]$; $o := M[oo]$;
6 $\quad$ **if** $dep[o] + tt_{(oc,eb)} + tt_{(eb,bf)} \leq dd_b^{bf}$ **then** $\mathtt{bm}[o] := b$; bb++; oo++ ;
7 $\quad$ **else** bb++ ;

Example 4. Given the example from Ex. 1 from page 2. Then Alg. 2 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, hence it follows that an optimal solution exists, which has the same arrival and departure times for the corresponding events as computed in Alg. 1 and 2. Thus, fixing the corresponding variables to those times removes symmetries from the problem.

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 $o \in M$ is used to serve the request for the blast furnace event $b \in N$. In other words, the torpedo used for o is *reused* for b.

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 3 computes a backward matching in linear time with respect to the number of blast furnace events, when these events and the oxygen converter events are already sorted. Let $\mathtt{bm} : M \to N \cup \{\infty\}$ denote the backward matching returned by Alg. 3. 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 ∞.

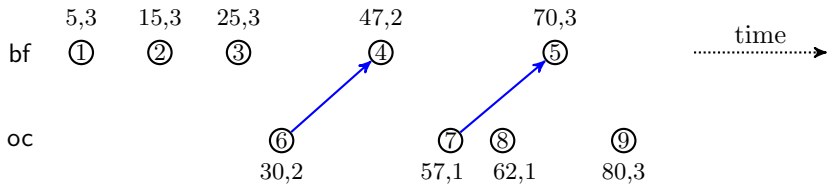

Figure 3. The backward matching for Ex. 1.

Example 5. Given the example from Ex. 1 from page 2. Then Alg. 3 computes the backward matching as shown by the arrows in Fig. 3, in which events 8 and 9 do not get a match. □

Theorem 1. *Let (N, M, G) be a torpedo scheduling problem. Then there exists an optimal solution S using the backward matching computed by Alg. 3 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 Alg. 1. 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 $\mathtt{bm}$. 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 $dd_{b_1}^{\mathsf{bf}} \leq dd_{b_2}^{\mathsf{bf}}$. 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 Alg. 3 matched t_2 with the earliest possible blast furnace event, it holds that $dd_{b_1}^{\mathsf{bf}} \leq dd_{b_2}^{\mathsf{bf}}$. Therefore, the torpedoes can be swapped. □

Given a backward matching, it divides the blast furnace events into the set of *matched* events, *i.e.*, $V = \mathtt{bm}(M) \setminus \{\infty\}$, and the set of *unmatched* events, *i.e.*, $U = N \setminus \mathtt{bm}(M)$. 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 $R = \{(i, j) \in N \times U \mid arr_i^{\mathsf{bf}} + dur^{\mathsf{bf}} + tt_{(\mathsf{bf},\mathsf{eb})} + tt_{(\mathsf{eb},\mathsf{bf})} \leq arr_j^{\mathsf{bf}}\}$.

Example 6. Given the example from Ex. 5. Then, $V = \{4, 5\}$ and $U = \{1, 2, 3\}$. The time cost for an emergency trip is from 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 bf in time for one of the unmatched one, *i.e.*, $R = \emptyset$. Therefore, the backward matching cannot be extended.

4 Solution Method

At first, we preprocess an instance for determining the various arrival and departure times, and the backward matching `bm` as described in the previous section. After that, we start the Benders decomposition, which alternates between solving the master and scheduling problems until an optimal solution is found. 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.

4.1 MIP model

The MIP model tries to find the mapping of 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 [8,4], 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 to model it 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 ep_i from the torpedo run, the MIP model uses the following binary variables. For each $(i, o) \in X$, we create a variable $x_{io} \in \{0, 1\}$ expressing whether the torpedo run i serves the demand of the oxygen converter event o. For each $(i, j) \in R$, the variables $r_{ij} \in \{0, 1\}$ models whether the torpedo from torpedo run i is reused for the blast furnace event j.

$$\min \quad dur^{\mathsf{ds}} \cdot n \cdot obj_1 + obj_2 \tag{10}$$

$$\text{s.t.} \quad obj_1 = |U| - \sum\nolimits_{(i,j)\in R} r_{ij} \tag{11}$$

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

$$obj_2 = \sum_{(i,o)\in X} x_{io} \cdot \max(0, sul_i - sul_o) \cdot dur^{\mathsf{ds}} \tag{12}$$

$$\sum_{(i,o)\in X} x_{io} = 1 \qquad \forall o \in M \tag{13}$$

$$ep_i + \sum_{(i,o)\in X} x_{io} = 1 \qquad \forall i \in N \tag{14}$$

$$\sum_{i\in N} ep_i = N - M \tag{15}$$

$$\sum_{(i,j)\in R} r_{ij} \leq ep_i \qquad \forall i \in N \tag{16}$$

$$\sum_{(i,j)\in R} r_{ij} \leq 1 \qquad \forall j \in U \tag{17}$$

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 number of blast furnace events and the duration for desulfurizing the hot metal by one sulfur level 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) ensures that there are the right number of emergency pit trips. Constraint (16) models that a torpedo used for an emergency trip can be reused for an unmatched blast furnace event, whereas (17) 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.

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 The CP model

Once, we have a mapping of blast furnace to oxygen converter events, we can split the remaining scheduling problem into several smaller ones. The idea is to split the problem at those blast furnace events serving an oxygen converter event, that do not interfere with any previous torpedo runs serving oxygen converter events. Algorithm 4 computes all sub-problems.

Example 7. Given the example from Ex. 1 from page 2. Algorithm 4 will split the problem into three sub-problems as depicted in Fig. 4.

Each sub-problem is then modeled as a constraint optimization problem using the same model, but restricted to torpedo runs in the sub-problem, as in Def. 1 on page 4 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 (18).

$$\min \quad \sum_{i\in S'} dep_i^{\mathsf{ds}} - arr_i^{\mathsf{ds}}$$

Algorithm 4: Computation of the sub-problems.

Input : N an array of n blast furnace events sorted in non-decreasing order of the due dates.
Input : oc a mapping from blast furnace events to oxygen converter events or ∞.

$latestDepDS := -\infty$; $A := \emptyset$; $B := \emptyset$;
for $ii := 2$ **to** n **do**
  if $oc(N[ii]) = \infty$ **then continue**;
  $i := N[ii]$; $earliestArrDS := dd_i^{\mathsf{bf}} + tt_{(\mathsf{bf},\mathsf{fb})} + tt_{(\mathsf{fb},\mathsf{ds})}$;
  if $latestDepDS \leq earliestArrDS$ **then**
    $B := B \cup \{A\}$; $A := \{i\}$;
  else $A := A \cup \{i\}$;
  $latestDepDS := \max(latestDepDS, dd^{\mathsf{oc}}_{\mathsf{oc}(i)} - tt_{(\mathsf{ds},\mathsf{oc})})$;
return B;

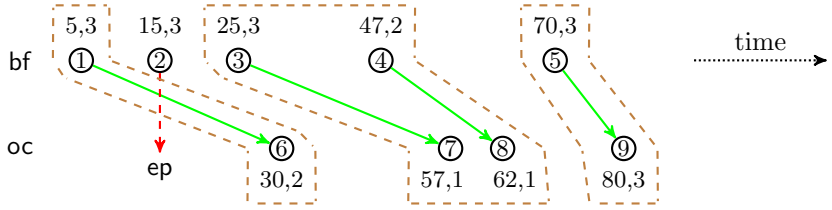

Figure 4. Partition of the scheduling given the shown matching.

s.t. (3–7)

$$dep_i^{\mathsf{bf}} \leq arr_{i+1}^{\mathsf{bf}} \qquad \forall i \in S' \setminus \{n\} \quad (18)$$

$$\text{disjunctive}((dep_i^l)_{i\in S'}, (arr_i^k - dep_i^l)_{i\in S'}) \qquad \forall (k,l) \in P'$$

$$\text{cumulative}((arr_i^l)_{i\in S'}, (dep_i^l - arr_i^l)_{i\in S'}, (1)_{i\in S'}, cap_l) \qquad \forall l \in L \setminus \{\mathsf{bf}\}$$

where S' are the torpedo runs of the sub-problem, $P' = \{(\mathsf{bf},\mathsf{fb}), (\mathsf{fb},\mathsf{ds}), (\mathsf{ds},\mathsf{oc})\}$, and disjunctive and cumulative are global constraints modeling unary and non-unary resources. Note that for each torpedo run $i \in S$, Alg. 1 and 2 provide the arrival times at bf and eb, and the departure times at oc and eb. In the case of an emergency trip, we also know dep_i^{bf}. Due to these pre-assigned times, the CP model only has to take care of oxygen converter trips from the blast furnace to the converter.

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.*, $dep_i^l - arr_i^l$ and $arr_i^q - dep_i^k$. 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 ds, $(\mathsf{fb},\mathsf{ds})$, $(\mathsf{ds},\mathsf{oc})$, oc, $(\mathsf{bf},\mathsf{fb})$, and bf. There are two important ingredients for this search. First, the first sub-search is objective driven, because it tries to minimize the duration spent at ds. 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.

Note that in order to avoid resolving sub-problems from scratch, we cached all sub-problems and their solution in a hash map.

4.3 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 x_{io} from the MIP problem. Let $\mathtt{oc}$ be the mapping from the MIP restricted to the sub-problem and N' the blast furnace events in the sub-problem.

Infeasibility Cuts The sub-problem is infeasible, which is a direct result of the mapping. Thus, $\sum_{i \in N'} x_{i\mathtt{oc}(i)} < |N'|$ 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 $|N'|$-times the CP model, 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 $\sum_{i \in N''} x_{i\mathtt{oc}(i)} < |N''|$, 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.

Optimality Cuts The sub-problem is schedulable with minimal desulfurization time β, but the desulfurization time α 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 $\sum_{i \in N'} x_{i\mathtt{oc}(i)} - (|N'| - 1) \leq b$ to the MIP model. The variable b takes value 1 if and only if the MIP uses the same mapping $\mathtt{oc}$ 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 mapping due to the added constraint and the added objective term is zero.

4.4 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, for example to the 10 next closest oxygen converter events which it can reach. The same is also done for the reuse of torpedoes after emergency trip events. Not only does this drastically shrink the model size, 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. However, 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. In the experiments, we show the sweet spot for the number of forward matchings.

5 Experiments

We conducted experiments on the ACP 2016 Challenge instances and created larger ones using the instance generator provided at the ACP Challenge website. All generated solutions were checked using the provided ACP solution checker. We grouped all instances in the test sets **small** having 15 instances with 30 to 500 blast furnace events, **comp** having 6 ACP challenge instances with 850 to 2500 blast furnace events, **medium** having 19 instances with 1000 to 3000 blast furnace events, and **large** having 3 instances with 10000 blast furnace events. All instances are available at `https://github.com/AdGold/TorpedoSchedulingInstances`. We ran all our experiments on a machine with an Intel(R) Core(TM) i7-5500U CPU at 2.40GHz and 8GB RAM unless otherwise stated. The solution was implemented in Python 3.5.2 interfacing Gurobi 7.0.1 using the Python library gurobipy. Gurobi was used for solving the MIP problem and Chuffed [1] for solving the CP problem. No runtime limit was imposed.

Unlimited Forward Matchings Table 1 shows the results on the set **comp** for each instance. We list 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, and the percentage of sub-problems with size 1 (S1), the average size of sub-problems with size greater than 1 (SAvg), and the maximal size of sub-problems (SMax). All ACP challenge instances were optimally solved in less than 20 minutes, which

Table 1. Detailed results on **comp**.

Inst	#T	Desulf	RT	MT	#I	CHR	#SP	SSR	S1	SAve	SMax
instance01	4	7695	41s	88%	1	0%	10	100%	70%	264	316
instance02	4	5302	119s	88%	1	0%	43	100%	67%	98	774
instance03	3	27150	415s	97%	1	0%	583	100%	84%	17	123
instance04	3	10676	35s	92%	1	0%	839	100%	84%	2	4
instance05	4	16308	575s	97%	3	50%	1074	99%	83%	14	410
instance06	4	7755	1134s	97%	2	48%	34	98%	62%	237	755

Table 2. Results on each test set excluding infeasible instances.

Inst	#T	Desulf	RT	MT	#I	CHR	#SP	SSR
small	3.5	362	2s	51%	1.1	3%	26	99.2%
comp	3.7	12481	386s	93%	1.5	16%	431	99.7%
medium	4.4	1224	484s	95%	1.2	4%	170	99.6%
large	4.5	6481	124093s	99%	2.0	31%	848	99.9%

is much quicker than the winning method presented in [8]. The results 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 2 presents the results on all test sets excluding infeasible instances. The table shows a subset of columns, but each entry is an average over the number of feasible instances. The results show a similar picture to the ACP challenge instances. However, for the large size instances in **large**, the runtime was more than 30 hours and we needed to run it on a machine with extra RAM in order to be able to solve the MIP. The machine used was an Intel(R) Xeon(R) CPU E5-2660 at 2.60GHz with 128GB RAM which is not practical in most cases.

Limited Forward Matchings Figures 5–7 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 forward matchings increases. Note that Fig. 7 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 10 minutes. All ACP challenge instances were solved in less than one minute, down from 20 minutes. In order to test the limit of our method, we created three instances with 50,000 and 100,000 blast furnace events, respectively. The average total runtime of the 50k instances were below

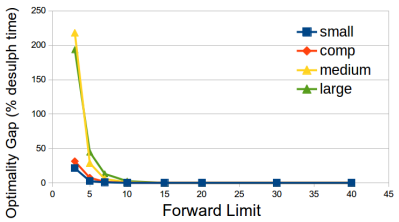

Figure 5. Optimality gap in desulfurization time.

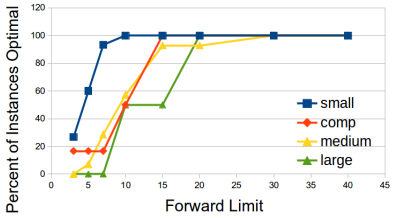

Figure 6. Percent of instances solved optimally.

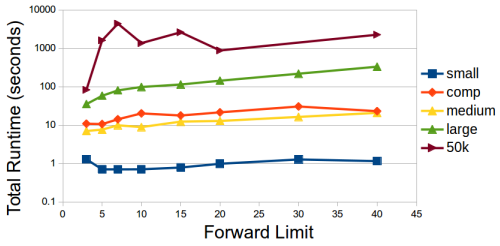

Figure 7. Total runtime.

20 minutes except for a limit of 7 as shown in Fig. 7. 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.

6 Conclusion

We propose a logic-based Benders decomposition solution method for the industrial problem of torpedo scheduling in the steel production. 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 20 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.

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

References

1. Chu, G.G.: Improving Combinatorial Optimization. Ph.D. thesis, The University of Melbourne (2011), http://hdl.handle.net/11343/36679
2. Deng, M., Inoue, A., Kawakami, S.: 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 (Aug 2011)
3. Geiger, M.J.: 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 (2017)
4. Geiger, M.J.: 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 (2017)
5. Hooker, J., Ottosson, G.: Logic-based benders decomposition. Mathematical Programming 96(1), 33–60 (2003)
6. Huang, H., Chai, T., Luo, X., Zheng, B., Wang, H.: Two-stage method and application for molten iron scheduling problem between iron-making plants and steelmaking plants. IFAC Proceedings Volumes 44(1), 9476 –9481 (2011)
7. Kikuchi, J., Konishi, M., Imai, J.: 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 (2008)
8. Kletzander, L., Musliu, N.: 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. pp. 344–358. Lecture Notes in Computer Science, Springer International Publishing (2017)
9. Li, J.q., Pan, Q.k.P., Duan, P.y.: An improved artificial bee colony algorithm for solving hybrid flexible flowshop with dynamic operation skipping. IEEE Transactions on Cybernetics 46(6), 1311–1324 (June 2016)
10. Liu, Y.Y., Wang, G.S.: 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 (2015)
11. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints 14(3), 357–391 (2009)
12. Schaus, P., Dejemeppe, C., Mouthuy, S., Mouthuy, F.X., Allouche, D., Zytnicki, M., Pralet, C., Barnier, N.: The torpedo scheduling problem: Description (2016), http://cp2016.a4cp.org/program/acp-challenge/problem.html, last accessed: 28 Apr 2017
13. Tang, L., Wang, G., Liu, J.: A branch-and-price algorithm to solve the molten iron allocation problem in iron and steel industry. Computers & Operations Research 34(10), 3001 – 3015 (2007)
14. Wang, G., Tang, L.: A column generation for locomotive scheduling problem in molten iron transportation. In: 2007 IEEE International Conference on Automation and Logistics. pp. 2227–2233 (Aug 2007)