A Multi-stage Simulated Annealing Algorithm for the Torpedo Scheduling Problem

Lucas Kletzander and Nysret Musliu

TU Wien,
Karlsplatz 13, 1040 Wien, Austria
{lkletzan,musliu}@dbai.tuwien.ac.at

Abstract. In production plants complex chains of processes need to be scheduled in an efficient way to minimize time and cost and maximize productivity. The torpedo scheduling problem that deals with optimizing the transport of hot metal in a steel production plant was proposed as the problem for the 2016 ACP (Association for Constraint Programming) challenge. This paper presents a new approach utilizing a multi-stage simulated annealing process adapted for the provided lexicographic evaluation function. It relies on two rounds of simulated annealing each using a specific objective function tailored for the corresponding part of the evaluation goals with an emphasis on efficient moves. The proposed algorithm was ranked first (ex aequo) in the 2016 ACP challenge and found the best known solutions for all provided instances.

Keywords: torpedo scheduling; simulated annealing; lexicographic evaluation function

1 Introduction

Production plants have a wide range of complex chains of processes that raise the need for optimization. To be competitive, cost needs to be minimized and efficiency and productivity need to be maximized. A selected scheduling problem in steel production was chosen for the 2016 ACP (Association for Constraint Programming) challenge [12]. The aim was to provide the best solutions for the torpedo scheduling problem where the transport of hot metal across various zones needs to optimized. The goal is to use as few transport vehicles (torpedoes) as possible and keep the time spent for a chemical process called desulfurization low while satisfying all deadlines and moving through the zones respecting capacity and duration constraints.

So far we know one other approach to the same problem by Geiger [4] ranking third in the competition. He used a branch and bound procedure to traverse feasible transport assignments and solved a resource-constrained scheduling problem based on variable neighborhood search to minimize desulfurization times. Other optimization problems in steel production have been researched in the past years. The molten iron allocation problem, modeled as a parallel machine scheduling problem [13] and the molten iron scheduling problem modeled as a flow shop

problem [6, 9] deal with assignments of torpedoes to machines. Various torpedo scheduling problems are defined for planning the transport of hot metal with the focus on vehicle routing across a network of rails and the objective to minimize transportation times [7, 2, 10]. Further production stages of steel making are also considered, e.g. steelmaking-continuous casting [14] which comes after the stage considered in this paper.

The approach we used is to utilize a multi-stage simulated annealing process adapted for the provided lexicographic evaluation function. The technique to simulate the physical process of annealing was introduced in [8] and is a widely used technique in many applications [3, 11]. Applications of multi-stage algorithms can be found in the domain of vehicle routing problems with time windows. These applications range back to [5] and also include application of simulated annealing [1], however, only for one stage. Several of these problems share a primary goal to minimize a number of vehicles, but pursue very different secondary objectives.

In our approach the first round of simulated annealing focuses on the primary goal of optimizing the number of torpedoes while the second round deals with the secondary goal of reducing the desulfurization time. The algorithm is designed to try a large number of moves in a short time by emphasis on efficient move calculations. Various parameters for the algorithm were determined by empiric evaluation. With this process it was possible to obtain the best solutions found in the competition for all given instances.

2 Problem Definition

The selected problem represents a part of the steel production process in a steel production plant. The *blast furnace (BF)* continually produces hot metal with a certain level of sulfurization that needs to be picked up at certain deadlines. From there the metal is transported via torpedoes which are cars suited for the transport of hot metal. There are two possible routes the torpedoes can take.

The first and standard route is to transition to a buffer zone (*full buffer, FB*) where torpedoes can wait for an arbitrary amount of time. Afterwards they reach the *desulfurization zone (D)* where the sulfurization level of the hot metal can be lowered. The time that needs to be spend at this station is proportional to the desired decrease in sulfurization levels. The delivery zone is at the *converter (C)* where the hot metal is unloaded from the torpedoes. The hot metal is required at certain deadlines at the converter. However, there is a maximum sulfuriziation level set for each deadline that the delivered hot metal must respect. Finally, the torpedo returns to another buffer zone (*empty buffer, EB*) where empty torpedoes wait for their next turn.

The other route can be used in order to get rid of excess hot metal that is not needed at the converter. It consists of a transition to the *emergency pit* where the torpedo disposes the hot metal and then returns to the buffer zone for empty torpedoes (EB). This route takes a fixed amount of time.

2.1 Instance Representation

For each instance of the problem several global parameters specifying the details of the production plant are given. *durBF* specifies the duration of filling a torpedo at the blast furnace. *durConverter* specifies the duration of unloading a torpedo at the converter.

The sulfurization level of hot metal is assumed to have five possible levels from 1 (low) to 5 (high). *durDesulf* specifies the time needed at the desulfurization zone to lower the sulfurization level by 1.

Further the three parameters specifying the amount of available slots in individual zones are *nbSlotsFullBuffer*, *nbSlotsDesulf* and *nbSlotsConverter*. These denote the maximum number of torpedoes that are allowed to stay at the zone at any given time. For the blast furnace and any transition between stations only one torpedo is allowed at the same time. There are no limits for the empty buffer and the emergency pit.

Finally the minimum transition times between any two adjacent zones are specified via the values $tt\langle Zone1\rangle To\langle Zone2\rangle$.

The goal of the algorithm is to schedule the routes of the torpedoes for an extended period of time. For this purpose, a series of blast furnace deadlines and converter deadlines is included in every instance.

Blast furnace deadlines are described as $BF\ \langle id\rangle\ \langle time\rangle\ \langle sulf\rangle$ where $\langle id\rangle$ identifies the individual deadline. $\langle time\rangle$ denotes the exact point in time when the hot metal will be released, therefore at this point in time a torpedo must be waiting to receive the metal. $\langle sulf\rangle$ denotes the sulfurization level the metal will have.

Converter deadlines are similarly described as $C\ \langle id\rangle\ \langle time\rangle\ \langle maxSulf\rangle$ where $\langle id\rangle$ identifies the individual deadline. $\langle time\rangle$ denotes the exact point in time when the hot metal has to be provided by a torpedo located at the converter. $\langle maxSulf\rangle$ denotes the maximum sulfurization level that can be accepted.

The instance files for the competition can be found on the web page.[1]

2.2 Solution Representation

The solution format contains the used number of torpedoes *nbTorpedoes*, followed by a list of torpedo tours. Each tour first contains three IDs. *idTorpedo* specifies the torpedo used for this tour in the range $[0, nbTorpedoes - 1]$. *idBF* specifies the ID of the blast furnace output the torpedo receives on this tour and *idC* denotes the ID of the converter demand the torpedo satisfies on this tour or -1 in case the metal is disposed at the emergency pit.

Next for each tour a set of time points is given, determining the exact sequence of the tour. For each zone $start\langle Zone\rangle$ and $end\langle Zone\rangle$ are specified according to the order the torpedo passes through the zones. The time points from *startFB* to *endC* are only specified for regular tours, but are missing for tours using the emergency pit.

[1] `http://cp2016.a4cp.org/program/acp-challenge-app/instance`

2.3 Evaluation Function

The function given for the evaluation of the solution quality considers the necessary number of torpedoes and the time spent in the desulfurization zone. First a solution is only valid if all deadlines at the blast furnace and the converter are met, the maximum sulfurization levels are respected, torpedoes move through the system correctly according to the specified transition and duration times and capacity constraints in the various zones are respected.

Then valid solutions are evaluated according to the lexicographic evaluation function that ranks solutions first based on the number of torpedoes and then on the total desulfurization time *timeDesulf*. More precisely the function

$$cost = nbTorpedoes + timeDesulf\,/(upperBoundTorpedo \cdot durDesulf) \qquad (1)$$

is used and the goal is to minimize the cost. Here *upperBoundTorpedo* is four times the number of converter deadlines as in the worst case every single torpedo tour to the converter needs to lower the sulfurization level from 5 (maximum) to 1 (minimum), therefore spending $4 \cdot durDesulf$ in this zone each time. This way for any reasonable solution the desulfurization time is normalized to the interval $[0, 1)$, the integer value of the cost represents the number of torpedoes and the part after the comma represents the desulfurization time, enforcing the lexicographic order.

3 Solution Approach

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

3.1 Representing a Solution Candidate

We model a possible solution with the concept of a *torpedo run*. Such a run represents one round trip of a torpedo and is tied to exactly one blast furnace deadline and either one converter deadline or the emergency pit. This is very similar to a tour in the required solution representation except that no specific torpedo ID is assigned. It allows to determine the amount of such runs directly from the given instance. The total amount of runs is equal to the number of blast furnace deadlines. The amount of runs targeting the converter is equal to the number of converter deadlines and the rest is targeting the emergency pit.

As a solution candidate we use an array of such torpedo runs that contains the correct number of runs targeting the converter and emergency pit. The algorithm is designed to respect the order of zones, prevent violations of zone durations and transition times and to always meet the blast furnace deadlines. All other

violations of the constraints previously described are possible in the algorithm. Instead of preventing them they are monitored and penalized by the objective functions used in the simulated annealing algorithm.

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

The first set of constraint violations regarding the converter demands can be described as follows given the notation that R_C is the set of torpedo runs targeting the converter and BF and C hold the corresponding blast furnace and converter deadline objects:

$$\sum_{i \in R_C} \max\{startC_i - C[idC_i].time, 0\} \tag{2}$$

$$diff_i = BF[idBF_i].sulf - \left\lfloor \frac{endD_i - startD_i}{durD} \right\rfloor - C[idC_i].maxSulf \tag{3}$$

$$\sum_{i \in R_C} \max\{diff_i, 0\} \tag{4}$$

Equation (2) counts the total converter deadline miss time by summing up the delay across all runs i that miss their deadline, in which case the start time at the converter $startC_i$ will be after the time of the assigned converter deadline.

Equation (3) first calculates the final sulfurization level for torpedo run i using the level at the blast furnace and the amount of time spend in the desulfurization zone. By subtracting the maximum allowed level at the converter it calculates the difference $diff_i$ between the maximum allowed level and the actual level. Values below 0 mean that the hot metal has lower sulfurization level than the allowed maximum indicating a potentially wasteful, but feasible pairing. A value of 0 exactly meets the requirement and values above 0 indicate sulfurization level misses and therefore constraint violations. Equation (4) sums up this difference in sulfurization levels across all runs that miss this requirement.

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

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

$$\sum_{0 \le t < T} \max\{X[t] - maxOccupation_X, 0\} \tag{5}$$

where $maxOccupation_X$ is either one of the given capacities *nbSlotsFullBuffer*, *nbSlotsDesulf*, *nbSlotsConverter* for the corresponding arrays or 1 in all other cases.

Monitoring Optimization Goals. To keep track of the number of torpedos another array spanning across the whole time span of the planning period is used. The array *occupation* counts for each point in time how many torpedo runs are currently active. Therefore the maximum value of this array represents the current value of the main evaluation goal.

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

$$occupationCount[i] = \text{count}\{t : occupation[t] = i\} , \tag{6}$$

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

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

$$desulfMismatch[i] = \text{count}\{j : BF[idBF_j].sulf - C[idC_j].maxSulf = i\} \tag{7}$$

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

3.2 The Simulated Annealing Algorithm

The simulated annealing process is done in two rounds using almost the same parameters, but very different objective functions. The general design of each round uses the usual process of simulated annealing:

```
generateInitialSolution();
for round = 0...1 {
  value = evaluateSolution();
  t = value / 10;
  for outer = 0...10000 {
    for inner = 0...innerIterations() {
      chooseMove();
      newValue = evaluateSolution();
      if (shouldAccept()) {
        acceptMove();
      } else {
        abortMove();
      }
    }
    t *= 0.998;
  }
}
```

Heuristic Generation of the Initial Solution Candidate. The concept of *generateInitialSolution*() is to take the ordered lists of blast furnace and converter deadlines and pair them according to this ordering. The torpedo runs are initialized to spend the minimum amount of time in the desulfurization zone that allows them to meet the sulfurization level requirement of the assigned converter. They arrive at the converter just at the time of the deadline or too late in case this pairing is actually not feasible and spend any required waiting time in the full buffer.

As long as emergency pit runs are available, they are set whenever it is possible to put the current blast furnace output to the emergency pit and still transport the next blast furnace output to the current converter deadline in time. As emergency pit runs tend to get scheduled a bit too early by this approach and cause converter deadline misses a few runs later, the algorithm includes backtracking on such deadline misses to remove earlier emergency pit runs again and schedule those later.

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

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

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

Iteration Parameters. For each round the algorithm uses a fixed number of outer iterations that was set to 10000. This value ensures that the algorithm converges to a stable result.

The temperature was set to start with a value of one tenth of the initial value of the objective function. In each outer iteration the temperature is decreased by multiplying with the factor 0.998. This choice, especially combined with the starting temperature and the number of inner iterations was one of the more critical choices for the quality of the results and therefore subject of detailed empirical evaluation. The initial solution is constructed in a way to already have a structure limiting the amount of constraint violations. While a certain increase in such violations is expected in the early stage of the simulated annealing process to prevent getting stuck too close to the initial solution, keeping the temperature

high for too long destroys the structure of the initial solution requiring extensive amounts of repair at lower temperatures that make the overall result worse. On the other hand, dropping the temperature too fast leads to getting stuck in local optima.

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

Efficient Moves. We proposed three moves that are chosen randomly with certain probabilities in *chooseMove*(). The first move is a switch between the assigned converter deadlines for two torpedo runs. It is chosen with a probability of 0.4 in the first round and 0.6 in the second round. The rather high probability is due to the fact that this is the move with the highest impact on the structure of the solution. The selection of the two runs is not randomized, but actually a sensitive choice. The reason is that choosing two runs at very different time points in the whole planning period will likely not be a good move as large deadline misses can be expected. On the other hand, only switching closely adjacent runs will likely end up in local optima too soon. Therefore selecting the distance of the two runs when sorted by their start times randomly between 1 and 10 showed to provide good results.

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

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

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

The important aspect is that all badness and goal tracking data can be updated incrementally. The sums or counts in (2), (4) and (7) allow easy removal and addition of torpedo runs. For (5) and (6) only the parts of the arrays X and *occupation* affected by the currently changed torpedo runs need to be updated, the effects on the sum and the count can easily be computed incrementally again.

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

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

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

3.3 Objective Functions

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

$$\sum_{0 \le i < 10} w_1[i] \cdot badness[i] + \sum_{i \ge 0} occupationCount[i] \cdot i^4 + cost \tag{8}$$

$$\sum_{0 \le i < 10} w_2[i] \cdot badness[i] + \sum_{i > fixed} 1000000 \cdot occupationCount[i] \cdot i^4 + \sum_{0 \le i < 5} 10000 \cdot desulfMismatch[i] \cdot i + timeDesulf \tag{9}$$

Equation (8) denotes the objective function for the first round and (9) the objective function for the second round. Again, the weights were chosen by experimental evaluation.

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

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

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

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

Number of Torpedoes. The next part of the objective functions uses the *occupationCount* array. For the first round each element $occupationCount[i]$ is weighted by i^4. This polynomial weighting strategy ensures a strong optimization towards a small number of currently active torpedo runs at any point in time with a special emphasis on eliminating areas using a high number of torpedoes. This showed to be a successful approach to optimize the first objective.

For the second round the goal of this part of the objective function is completely changed. This part is the reason for choosing to use two separate rounds of optimization in the first place. The key point is the structure of the solution created in the first round. The number of currently active torpedoes is kept as low as possible across the whole time span in order for the optimization to work. However, as only the maximum number of torpedoes counts for the value of the solution and this maximum value might only be reached at a small part of the whole process, this kind of optimization restricts the possibilities to optimize the desulfurization time more than necessary. Section 4.1 will highlight this in the results.

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

Desulfurization. In the first round the objective function is completed by adding the actual evaluation function used for the final solution as given by (1). This adds the optimization of desulfurization times as a low priority goal to the process.

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

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

4 Results

The algorithm was executed on an Intel Core i7-6700K with 4 x 4.0 GHz. Table 1 shows the main characteristics of the competition instances. The duration and capacity constraints were rather similar for the given instances while the main differences were in the number of blast furnace and converter deadlines and the time span covered. A single run for each instance used only 4 to 10 minutes on a single CPU core for all instances except 6. Here, even though the instance size is not that different, one run takes almost 30 minutes due to the structure of the instance creating longer torpedo runs.

4.1 Structure of a Solution

To see the importance of using two rounds of simulated annealing, data collected from one particular computation of instance 1 is presented after the creation of the initial solution and after each round of simulated annealing. The resulting distribution is similar in all instances, therefore it is only described for instance 1 to highlight the way the algorithms transforms the solution.

Table 1. Instance characteristics

Instance	1	2	3	4	5	6
Blast furnace deadlines	850	1500	2200	1000	1800	2500
Converter deadlines	800	1400	2100	1000	1780	2350
Time span	131100	144165	394723	133798	256216	251460

Table 2 shows the elements of *occupationCount* and *timeDesulf* for instance 1. All values $occupationCount[i]$ with $i > 5$ are 0.

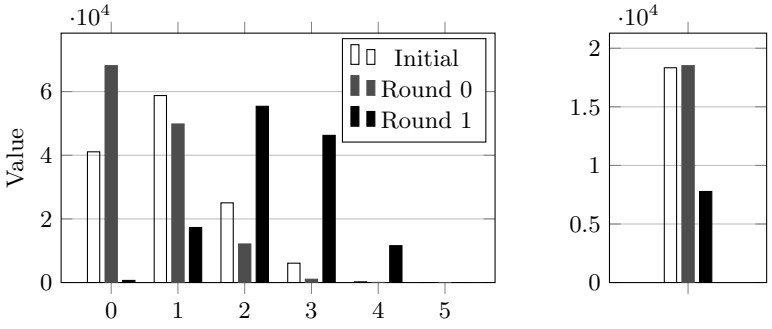
Table 2. Objective values in various stages of the algorithm (instance 1)

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

The initial solution generated by the heuristic typically uses only few torpedoes more than the final result, in this case 5 torpedoes are used. However, as to be expected, this solution is not feasible, capacity constraints are slightly violated at the transitions *FBtoD*, *DtoC* and *CtoE* as well as at the desulfurization zone. Figure 1 shows the distribution of the occupation values. The most frequent occupation at this stage is to have 0 or 1 torpedo active.

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

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



(a) *occupationCount*[*i*] (b) *timeDesulf*

Fig. 1. Objective values in various stages of the algorithm (instance 1)

4.2 Results for Competition Instances

For the competition 50 runs were performed for each instance and the best result was handed in. There were no restrictions on computation times.

Table 3 presents the results obtained by the 50 runs for each instance. Some runs did not find the best number of torpedoes. In case a previous run already got less torpedoes, the run was aborted after the first round of simulated annealing (Abort 1). Some runs resulted in minor levels of constraint violations and were discarded (Abort 2). The amount of times the best result (Minimum) was produced is also shown (Freq.). Maximum, mean and standard deviation are calculated from runs that were not aborted only.

Table 3. Computation results (50 runs for each instance)

Instance	Abort 1	Abort 2	**Minimum**	**Freq.**	Maximum	Mean	Std. dev.
1	1	4	**4.0890625**	**3**	4.09094907	4.089995	0.000484
2	1	0	**4.08607143**	**16**	4.08712662	4.086279	0.000217
3	16	0	**3.12928571**	**5**	3.12964286	3.129415	0.000080
4	0	3	**3.157**	**47**	3.157	3.157	0
5	6	1	**4.08483146**	**1**	4.08609551	4.085374	0.000282
6	1	0	**4.075**	**1**	4.07585106	4.075358	0.000172

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

the need to obtain the best results the algorithm can offer. With 50 runs per instance we were able to obtain the first place (ex aequo) on all instances.

The best result was produced in at least 3 out of the 50 runs for the first four instances making those results easily reproducible and increasing the confidence that these are the best results the algorithm can produce. For the last two instances, however, the best solution was only found in 1 out of 50 runs. These two turned out to be the most difficult instances in the competition. To gain more confidence, several recomputations of the whole 50 runs on these two instances were able to reproduce these results, even though only about every second recomputation for instance 5, but did not find any better results.

Constraint violations in the results were only encountered in a minor fraction of the runs (maximum of 4 out of 50). Moreover, for all aborted instances the violations only occurred at most at 2 time units across the whole planning period. Given the fact that penalties for capacity violations were deliberately set low in the second round of simulated annealing to focus further on the optimization of desulfurization times this is considered a good result.

Instance 3, a large instance with the longest time span, yet a very small amount of torpedoes in the solution, showed to be the hardest instance for the first round. Here 32% of runs did not find the best amount of torpedoes compared to at most 12% for any other instance.

Instance 4 was clearly the easiest instance, being one of the smaller instances, but also providing equal amounts of blast furnace and converter deadlines. This eliminates the need for emergency pit runs. In fact, 47 out of 50 runs on this instance produced the best found result.

5 Conclusion

This paper presented an approach using a multi-stage simulated annealing process to solve a scheduling problem in steel production plants. Utilizing efficient moves it optimizes results for the lexicographic evaluation function using two different objective functions each tailored for optimal progress towards the corresponding part of the evaluation in a two-stage simulated annealing process.

The results show that the approach is a valid and competitive method to solve the given problem. As the general idea is generic, it can also be adapted to various other problems utilizing lexicographic evaluation functions. Further the framework to track changing capacity violations in a fast manner showed to improve efficiency of the algorithm.

Future work could include the adaption of the approach to various other problems in this domain to see how well it competes with different approaches. Further the selection of critical parameters for simulated annealing, especially regarding automated parameter selection, could be the goal of research.

Acknowledgments. This work was supported by the Austrian Science Fund (FWF): P24814-N23.

References

1. Russell Bent and Pascal Van Hentenryck. A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows. *Transportation Science*, 38(4):515–530, November 2004.
2. Mingcong Deng, Akira Inoue, and Satoru Kawakami. Optimal path planning for material and products transfer in steel works using ACO. In *The 2011 International Conference on Advanced Mechatronic Systems*, pages 47–50. IEEE, 2011.
3. Kathryn A. Dowsland and Jonathan M. Thompson. Simulated Annealing. In Grzegorz Rozenberg, Thomas Bck, and Joost N. Kok, editors, *Handbook of Natural Computing*, pages 1623–1655. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
4. Marting Josef Geiger. Optimale Torpedo-Einsatzplanung – Analyse und Lösung eines Ablaufplanungsproblems der Stahlindustrie. In *Entscheidungsunterstützung in Theorie und Praxis - Tagungsband des gemeinsamen Workshops der GOR-Arbeitsgruppen "Entscheidungstheorie und -praxis", "Fuzzy Systeme, Neuronale Netze und Künstliche Intelligenz" sowie "OR im Umweltschutz" am 10. und 11. März 2016 in Magdeburg*. Springer-Verlag, in press.
5. Jörg Homberger and Hermann Gehring. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR: Information Systems and Operational Research*, 37(3):297–318, 1999.
6. Hui Huang, Tianyou Chai, Xiaochuan Luo, Binglin Zheng, and Hong Wang. Two-Stage Method and Application for Molten Iron Scheduling Problem between Iron-Making Plants and Steel-Making Plants. *IFAC Proceedings Volumes*, 44(1):9476–9481, January 2011.
7. Junji Kikuchi, Masami Konishi, and Jun Imai. Transfer Planning of Molten Metals in Steel Worksby Decentralized Agent. *Memoirs of the Faculty of Engineering, Okayama University*, 42(1):60–70, 2008.
8. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, May 1983.
9. Jun-qing Li, Quan-ke Pan, and Pei-yong Duan. 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. Y.Y Liu and G.S Wang. The Mix Integer Programming Model for Torpedo Car Scheduling in Iron and Steel Industry. In *International Conference on Computer Information Systems and Industrial Applications*, pages 731–734. Atlantis Press, 2015.
11. Duc Pham and Dervis Karaboga. *Intelligent optimisation techniques: genetic algorithms, tabu search, simulated annealing and neural networks*. Springer Science & Business Media, 2012.
12. P. Schaus, C. Dejemeppe, S. Mouthuy, F.-X. Mouthuy, D. Allouche, M. Zytnicki, C. Pralet, and N Barnier. The torpedo scheduling problem: Description. `http://cp2016.a4cp.org/program/acp-challenge/problem.html`, 2016. Accessed: 2017-02-02.
13. Lixin Tang, Gongshu Wang, and Jiyin Liu. A branch-and-price algorithm to solve the molten iron allocation problem in iron and steel industry. *Computers & Operations Research*, 34(10):3001–3015, October 2007.
14. Lixin Tang, Yue Zhao, and Jiyin Liu. An Improved Differential Evolution Algorithm for Practical Dynamic Scheduling in Steelmaking-Continuous Casting Production. *IEEE Transactions on Evolutionary Computation*, 18(2):209–225, April 2014.