Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 50-1 (2017) 12490–12495

An Efficient Algorithm for Scheduling a Flexible Job Shop with Blocking and No-Wait Constraints ⋆

A. Aschauer, F. Roetzer, A. Steinboeck, A. Kugi

Automation and Control Institute, Technische Universität Wien, Gusshausstraße 27-29, 1040 Vienna, Austria ({aschauer, roetzer, steinboeck, kugi}@acin.tuwien.ac.at)

Abstract: Optimal scheduling in industrial processes is crucial to ensure highest throughput rates and low costs. This paper presents the implementation of a scheduling algorithm in a hot rolling mill, which features several reheating furnaces and which is characterized by bidirectional material flow, blocking, and no-wait constraints. The scheduling problem is solved by a decomposition into a timetabling algorithm and a sequence optimization procedure. For the timetabling task, where the sequence of products is assumed to be fixed, a new recursive algorithm to generate a non-delay feasible schedule is developed. The sequence optimization procedure searches for the optimum product sequence and makes heavy use of the timetabling algorithm. A competitive starting sequence is generated by a construction heuristic and iteratively improved by a tabu search algorithm.

Keywords: scheduling, timetabling, flexible job shop, parallel machines, blocking, no-wait constraints, construction heuristic, tabu search, hot rolling mill, metal processing

1. INTRODUCTION

Hot rolling is a main cost driver in the production of metal plates. Here, the reheating is highly time and energy consuming. If larger series of equal products are produced, it is straightforward to schedule the reheating and rolling process. However, large series are becoming rare. The increasing diversification of product portfolios, individualization of products, lean inventory, and just-in-time production add to the complexity of the scheduling task. This is why an optimal scheduling algorithm is required. It should ensure best possible utilization of the available production facilities and highest throughput rates. In the considered plant, mainly molybdenum products are processed. This valuable metal requires an accurate scheduling algorithm to ensure that the material is rolled at the right time and with the right temperature.

1.1 Problem Description

Figure 1 shows an outline of the considered plant. The manipulator M conveys the raw material (slabs) from the charging station to one of the four furnaces F1-F4 for reheating. After reheating, the manipulator moves the product to the roller table with the reversing mill stand R, where it is rolled in several rolling passes. Many products require intermediate reheating steps or final annealing in one of the furnaces. For this purpose, the products have to be handled by the manipulator again. If furnaces are operated at the same temperature, they can be considered as identical. For every product, a production plan specifies

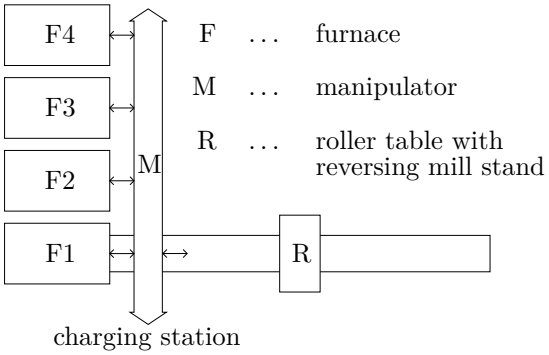

Fig. 1. Outline of the hot rolling mill.

the sequence of heating and rolling steps, the heating temperatures, and the process times or at least their lower and upper bounds.

Henceforth, a product is called a job and the single processing and manipulation steps are called tasks. For every job, the predefined sequence of tasks must be adhered to. Moreover, waiting times between tasks are not allowed due to the absence of space for the products to wait (no buffer) and the product quality would suffer (temperature decrease due to cooling and oxidation of the product surface). The processing time for heating tasks has a lower and an upper bound. If a heating task is scheduled to be longer than the lower bound time, the respective furnace is blocked longer than required, which may reduce the throughput rate. Manipulation and rolling tasks have a fixed duration, which should not be changed to avoid wrong rolling temperatures. The furnaces, the manipulator, and the roller table with the reversing mill stand are called machines. The furnaces are operated at constant temperatures, which are specified in the production plan. Furnaces with identical temperatures are called parallel machines. Each machine can only handle one task at a time

⋆ Great thanks are addressed to the industrial research partner Plansee SE supporting this work. Moreover, financial support from the EU project Power Semiconductor and Electronics Manufacturing 4.0 (SemI40), under grant agreement No 692466, is gratefully acknowledged. The project is co-funded by grants from Austria, Germany, Italy, France, Portugal, and - Electronic Component Systems for European Leadership Joint Undertaking (ECSEL JU).

10.1016/j.ifacol.2017.08.2056

and each job can only be processed by one machine at a time. The objective of the desired scheduling algorithm is the minimization of the makespan (total processing time) of a given number of jobs. In this minimization procedure, all constraints have to be satisfied.

1.2 Literature Review

The above problem is known in the literature as flexible job shop scheduling problem (FJSP) with blocking and no-wait constraints. The complexity of the task considered in this paper originates in the existence of parallel machines. For the classical job shop scheduling problem (JSP) with unlimited buffers between consecutive tasks, there exists a rich literature, e. g., (Pinedo, 2016; Błażewicz et al., 2001). A common way to represent this scheduling problem is a disjunctive graph. Pinedo (2016) described an exact branch-and-bound algorithm for a JSP with two machines. For larger problems, he recommends the so-called shifting bottleneck heuristic (Pinedo, 2016). There are also some studies on FJSPs, where certain tasks can be performed by several machines. Brucker and Schlie (1991) were among the first who addressed this problem. The complexity of a FJSP roots in the fact that additionally an assignment of tasks to machines is necessary. Important works have been made by Paulli (1995) and Dauzère-Pérès and Paulli (1997). Paulli (1995) presented a hierarchical approach which first assigns tasks to machines and second solves the classical JSP. Starting from this initial solution, this procedure is iteratively repeated. That is, tasks are reassigned and the classical JSP is solved again. Dauzère-Pérès and Paulli (1997) described an integrated approach of assignment and scheduling based on the help of an extended disjunctive graph.

The literature on FJSPs with blocking and no-wait constraints is rather scarce. Hall and Sriskandarajah (1996) and Allahverdi (2016) gave surveys of scheduling methods with no-wait constraints. Most works in this field consider flow shop scheduling problems. Nevertheless, some works on JSPs are mentioned. Many successful approaches handle the JSP with no-wait constraints by decomposing the problem into a timetabling and a sequencing part. Schuster and Framinan (2003) achieved excellent results with a non-delay timetabling algorithm and a tabu search algorithm for sequence optimization. Framinan and Schuster (2006) could further improve these results by a combined non-delay and inverse timetabling approach. Samarghandi et al. (2013) presented a study about different combinations of timetabling and sequencing algorithms. Raaymakers and Hoogeveen (2000) examined a FJSP with blocking and no-wait constraints. They decomposed the problem into timetabling, machine assignment, and sequencing. Mascis and Pacciarelli (2002) investigated the JSP with blocking and no-wait constraints by generalized disjunctive graph methods. They pointed out that these methods entail a trade off between feasibility and quality of the obtained solution.

1.3 Content

Feasibility is indispensable for the problem considered in this paper. Therefore, the well established decomposition into timetabling and sequence optimization is used in this work. The contribution of this paper is the simultaneous handling of no-wait and blocking constraints, where the blocking time can also be limited. A recursive timetabling algorithm is designed so that jobs are scheduled with minimal finishing time without violating any constraints by all of the jobs' tasks. Moreover, to the best knowledge of the authors, the assignment to parallel machines is done in a new way. In Section 2, the recursive timetabling algorithm is presented. Section 3 focuses on sequence optimization. A starting sequence is generated by a construction heuristic and further improved by a tabu search algorithm. In Section 4, scheduling results for data from an industrial production environment are given. Conclusions are drawn in Section 5.

t in s	$c_m(t)$
0	0
120	1
240	2
360	1
600	2
⋮	⋮
∞	0

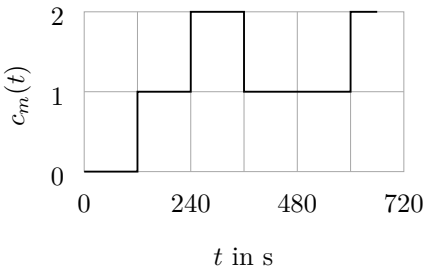

Fig. 2. Capacity utilization of a machine type m with $c_{m,max} = 2$.

2. TIMETABLING

In the course of timetabling, a given sequence of jobs is scheduled one by one so that all constraints are satisfied and the finishing time of each job is minimized. The total number of jobs is N_J. Each job $j \in \{1, \ldots, N_J\}$ is associated with $N_{T,j}$ tasks. The timetabling algorithm determines the starting times $t_{j,n}$ and the durations $d_{j,n}$, $\forall j$, $n = 1, \ldots, N_{T,j}$ of all tasks of all jobs. The durations $d_{j,n}$ of the tasks are bounded by the lower bounds $\underline{d}_{j,n}$ and by the upper bounds $\overline{d}_{j,n}$ which are specified by the production plan. For the manipulating and rolling tasks, these bounds are identical, i. e., $\underline{d}_{j,n} = \overline{d}_{j,n}$. The sequence of the jobs is stored in the so-called permutation vector $\mathbf{p} = (p_1, \ldots, p_i, \ldots, p_{N_J})$, $p_i \in \{1, \ldots, N_J\}$ with $p_i \neq p_j$ for $i \neq j$.

The proposed algorithm also assigns tasks to machines. To avoid an initial assignment of tasks to one of the parallel machines, which could finally lead to suboptimal solutions, so-called machine types m are defined. These machine types have a maximum capacity $c_{m,max}$ according to the number of parallel machines. The capacity utilization $c_m(t)$ over time is stored in a table of values for every machine type, see Fig. 2. Free time slots occur whenever $c_m(t) < c_{m,max}$ holds true. When the timetabling of all N_J jobs is finished, an assignment of the tasks to machines can be easily made.

Timetabling is done one by one for each job j. The tasks $n = 1, \ldots, N_{T,j}$ of each job j must satisfy the following constraints:

- No waiting times between consecutive tasks: $t_{j,n} + d_{j,n} = t_{j,n+1}$ for $n = 1, \ldots, N_{T,j} - 1$
- Valid durations of the tasks according to the lower and upper bounds: $\underline{d}_{j,n} \leq d_{j,n} \leq \overline{d}_{j,n}$, $\forall n$
- The maximum number of simultaneous tasks at the machine types: $c_m(t) \leq c_{m,max}$, $\forall m$

Therefore, the recursive function FINDSCHEDULE() which is summarized in Algorithm 1 is developed. The function tries to schedule the tasks according to the strategy *As early & short as possible, as late & long as necessary.*

The function FINDSCHEDULE() works as follows. Generally, local working variables are named without indices. Only the starting times $t_{j,n}$ and the durations $d_{j,n}$, which are the outcome of the algorithm, are denoted by the indices j and n. The function is explained from the point of view of one recursion layer which represents one task.

Its inputs are the job number j, the task number n, the earliest possible starting time t_{rl}, also referred to as release time, and the latest possible starting time t_{lst}. The numbers j and n are needed for identifying the task. The release time t_{rl} is the earliest possible end time of the preceding task and the latest possible starting time t_{lst} is the latest possible end time of the preceding task. Note that the latest possible starting time t_{lst} can also be infinity ($t_{lst} = \infty$) if the corresponding machine type is available for all future times. Only for $n = 1$ the release time t_{rl} and the latest possible starting time t_{lst} are set to default values, i. e., $t_{rl} = 0$, $t_{lst} = \infty$. The return values of the function FINDSCHEDULE() are the starting time $t_{j,n}$ and the duration $d_{j,n}$ of the task n of job j, as well as an error flag err.

First, the machine type m, the lower bound $\underline{d}$ and the upper bound $\overline{d}$ of the task duration, as well as the total number of tasks $N_{T,j}$ are retrieved from the production plan. Then, the function FINDSLOT() determines the earliest free time slot $[t_1, t_2)$ of machine type m, starting after the release time t_{rl} with a length greater or equal than the lower bound $\underline{d}$ of the task duration, i. e., $t_2 - t_1 \geq \underline{d}$. Moreover, the task duration $d_{j,n}$ and the error flag err are initialized, i. e., $d_{j,n} = \underline{d}$, $err = 0$.

The first if-clause holds true if $t_{lst} \geq t_1$, which means that the determined free time slot $[t_1, t_2)$ is principally feasible according to the preceding task. If this condition fails, the error flag is set ($err = 1$) and the beginning of the free time slot $[t_1, t_2)$ is returned as an earliest possible starting time, i. e., $t_{j,n} = t_1$. The function terminates.

If $t_{lst} \geq t_1$ is satisfied, and if the task n is the last task of the job, i. e., $n = N_{T,j}$, it is scheduled right at the beginning of the free time slot slot, i. e., $t_{j,n} = t_1$. The function terminates.

If n has a successor, the function FINDSCHEDULE() is recursively called for the subsequent task $n+1$ to determine the starting time $t_{j,n+1}$. The release time for the successor is $t_1 + \underline{d}$ and its latest possible starting time is the end time t_2 of the free time slot $[t_1, t_2)$.

If no error is returned by the recursive function call, i. e., $err_{+1} = 0$, three cases can happen:

- Ideally, if the latest possible starting time t_{lst} fulfills the condition $t_{lst} \geq t_{j,n+1} - \underline{d}$, task n can be scheduled directly before task $n+1$ with minimum duration $\underline{d}$, i. e., $t_{j,n} = t_{j,n+1} - \underline{d}$.
- A schedule is still feasible if the condition $t_{lst} \geq t_{j,n+1} - \overline{d}$ holds true. The duration $d_{j,n}$ is then greater than the minimum duration $\underline{d}$. The starting time and the duration follow as $t_{j,n} = t_{lst}$ and $d_{j,n} = t_{j,n+1} - t_{lst}$. This case is illustrated in Fig. 3.
- No schedule can be generated if both conditions fail. An error $err = 1$ and an earliest possible starting time $t_{j,n} = t_{j,n+1} - \overline{d}$ are returned.

In all three cases the function terminates.

In case that the recursive call of the function FINDSCHEDULE() for the task $n+1$ returns an error, i. e., $err_{+1} = 1$, a new free time slot $[t_1, t_2)$ has to be determined. Therefore, the function FINDSLOT() is called with the release time $\text{MAX}(t_{j,n+1} - \overline{d}, t_2)$. The function MAX() returns the largest of its input values. Thus, it is guaranteed that the same free time slot is not used again. To make sure that the new free time slot $[t_1, t_2)$ lasts at least till the earliest possible starting time $t_{j,n+1}$ of the succeeding task $n+1$, i. e., $t_2 \geq t_{j,n+1}$, a while-loop is installed. The while-loop with the condition $t_2 < t_{j,n+1}$, calls the function FINDSLOT() with the release time t_2 until a free time slot $[t_1, t_2)$ is found which satisfies the condition $t_2 \geq t_{j,n+1}$. With this new free time slot $[t_1, t_2)$, the outer while-loop is repeated. Note that this is the only branch, where the outer while-loop and therefore the function is not terminated.

Algorithm 1 The function FINDSCHEDULE().

```
function [t_{j,n}, d_{j,n}, err] = FINDSCHEDULE(j, n, t_rl, t_lst)

    m = MACHINETYPE(j, n);
    [d_, d¯] = TASKDURATION(j, n);
    N_{T,j} = NUMBEROFTASKS(j);
    [t_1, t_2] = FINDSLOT(m, t_rl, d_);
    d_{j,n} = d_;
    err = 0;

    while 1 do
        if t_lst ≥ t_1 then
            if n = N_{T,j} then
                t_{j,n} = t_1;
                return
            else
                [t_{j,n+1}, d_{j,n+1}, err_{+1}] =
                        FINDSCHEDULE(j, n + 1, t_1 + d_, t_2);
                if err_{+1} = 0 then
                    if t_lst ≥ (t_{j,n+1} − d_) then
                        t_{j,n} = t_{j,n+1} − d_;
                    else if t_lst ≥ (t_{j,n+1} − d¯) then
                        t_{j,n} = t_lst;
                        d_{j,n} = t_{j,n+1} − t_lst;
                    else
                        t_{j,n} = t_{j,n+1} − d¯;
                        err = 1;
                    end if
                    return
                else
                    [t_1, t_2] =
                      FINDSLOT(m,MAX(t_{j,n+1} − d¯, t_2), d_);
                    while t_2 < t_{j,n+1} do
                        [t_1, t_2] = FINDSLOT(m, t_2, d_);
                    end while
                end if
            end if
        else
            t_{j,n} = t_1;
            err = 1;
            return
        end if
    end while
end function
```

The FINDSCHEDULE() function determines a feasible schedule with minimal end time for the job j. When the function has terminated, the time slots $[t_{j,n}, t_{j,n} + d_{j,n})$, $n = 1, \ldots, N_{T,j}$ of all tasks of the job j are added to the tables of capacity utilization (cf. Fig. 2) of the respective machine types.

All jobs $j \in \{1, \ldots, N_J\}$ are scheduled in this manner according to the sequence defined in the permutation

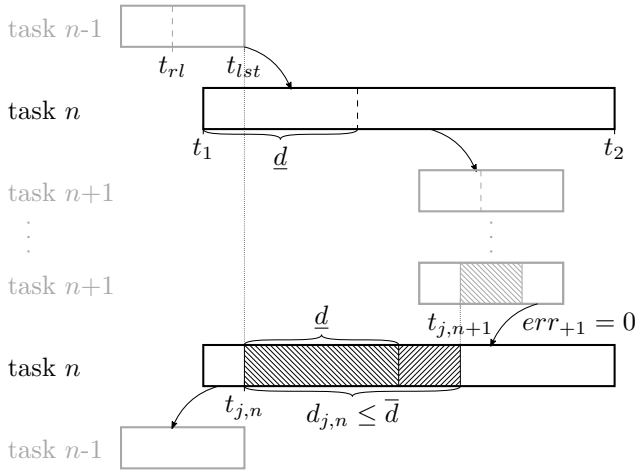

Fig. 3. Graphical example of scheduling a task n by the function FINDSCHEDULE().

vector $\mathbf{p}$, i. e., $j = p_1, \ldots, p_{N_J}$. The makespan T_m of the whole production lot can therefore be found as the time period between the starting time of the first job to be started and the end time of the last job to be finished.

3. SEQUENCE OPTIMIZATION

The goal of the sequence optimization procedure is to find the permutation vector $\mathbf{p}^*$ which minimizes the makespan T_m determined by the timetabling procedure from Section 2. The studies of Samarghandi et al. (2013) show that the tabu search algorithm performs well with most timetabling algorithms. Therefore, tabu search is also used in this work. This iterative algorithm implies the need of a starting permutation $\mathbf{p}_0$, which can be generated by a construction heuristic.

3.1 Construction Heuristic

The construction heuristic creates an initial sequence of jobs by choosing the best successor according to a set of criteria. The algorithm starts with an empty sequence. First, a starting job is chosen, appended to the sequence, and scheduled by the timetabling algorithm. All remaining jobs are examined with respect to the criteria. The job, which best fulfills the criteria, is chosen, appended to the sequence, and scheduled next by the timetabling algorithm. This is repeated until all jobs are assembled in the sequence.

For the determination of the criteria, all remaining jobs are experimentally scheduled by the function FINDSCHEDULE() from Section 2, but not added to the tables of capacity utilization of the machine types. The criteria are:

(1) The earliest starting time: The starting time of a job j is the starting time $t_{j,1}$ of its first task.
(2) The earliest starting time at the bottleneck machine type: For the considered problem, a bottleneck machine type is manually identified. The starting time of a job j at the bottleneck machine type is defined as the starting time $t_{j,n_{bf}}$ with n_{bf} being the first task of the job j to be processed at the bottleneck machine type. If there exists no such task, the starting time is infinity.
(3) The lowest relative lengthening of the job duration: The relative lengthening of the job duration is $\sum_{n=1}^{N_{T,j}} (d_{j,n} - \underline{d}_{j,n}) / \sum_{n=1}^{N_{T,j}} \underline{d}_{j,n}$.
(4) The longest duration of tasks after the bottleneck machine type: The duration of tasks after the bottleneck machine type is $\sum_{n=n_{bl}+1}^{N_{T,j}} d_{j,n}$ with n_{bl} being the last task of the job j to be processed at the bottleneck machine type. If no such task exists, n_{bl} is set to zero.
(5) The longest job duration: The job duration is $\sum_{n=1}^{N_{T,j}} d_{j,n}$.

The criteria are evaluated in increasing order. If the criterion (1) is ambivalent, criterion (2) is consolidated for the equivalent jobs according to criterion (1) and so on. The criteria (1)-(3) try to find the best fitting jobs, whereas the criteria (4) and (5) try to schedule unpleasant jobs as early as possible.

This construction heuristic is designed to reach a high degree of capacity utilization especially at the bottleneck machine type. The problem of selecting a good starting job is solved by executing the construction heuristic for all possible starting jobs and choosing the permutation with minimal makespan T_m.

3.2 Tabu Search

The starting permutation $\mathbf{p}_0$ should be further improved by a tabu search algorithm, cf. (Błażewicz et al., 2001). Special attention is given to the choice of the neighborhood. Therefore, a relation $\mathbf{p}_N = M(\mathbf{p}, k, l, g)$ is defined which determines a neighbor permutation $\mathbf{p}_N$ of $\mathbf{p}$ according to the parameters k, l, and g with $k < l$. The parameters k and l denote positions and g a group size. For the permutation vector

$$\mathbf{p} = [p_1, \ldots, p_k, \ldots, p_{k+g-1}, \ldots, p_l, \ldots, p_{l+g-1}, \ldots, p_{N_J}], \tag{1}$$

the relation $M(\mathbf{p}, k, l, g)$ yields the neighbor permutation $\mathbf{p}_N$ based on the current permutation $\mathbf{p}$ by exchanging groups of g jobs starting at the positions k and l, i. e.,

$$\mathbf{p}_N = M(\mathbf{p}, k, l, g) = [p_1, \ldots, p_{k-1}, p_l, \ldots, p_{l+g-1}, p_{k+g} \cdots \\ \cdots, p_{l-1}, p_k, \ldots, p_{k+g-1}, p_{l+g}, \ldots, p_{N_J}]. \tag{2}$$

The whole neighborhood $\mathcal{N}(\mathbf{p})$ of a permutation $\mathbf{p}$ is defined as the set of all possible neighbor permutations $\mathbf{p}_N$ up to a maximum group size g_{max}, i. e.,

$$\mathcal{N}(\mathbf{p}) = \{M(\mathbf{p}, k, l, g) \mid g \in \{1, \ldots, g_{max}\}, \\ k \in \{1, \ldots, N_J + 1 - 2g\}, \\ l \in \{k + g, \ldots, N_J + 1 - g\}\}. \tag{3}$$

In every iteration of the tabu search algorithm, the makespan T_m has to be determined for each element of the whole neighborhood $\mathcal{N}(\mathbf{p})$ of the current permutation $\mathbf{p}$. The algorithm starts with the starting permutation $\mathbf{p}_0$ and continues with the neighbor permutation $\mathbf{p}_N^*$ which achieves the minimal makespan T_m. Note that T_m may also increase in the course of the algorithm.

To prevent the algorithm from oscillating, a so-called tabu list stores recent neighbors, which are forbidden to go to for a certain number of iterations. In this work, the tabu list is a first in, first out list containing the positions k and l which led to $\mathbf{p}_N^*$. The positions k and l which are stored in the tabu list must not be used when searching for the next value of $\mathbf{p}_N^*$. The algorithm is executed for a fixed number of iterations. This is an important user-defined tuning parameter of the algorithm, just like the maximum group size g_{max} and the size of the tabu list. Glover and Laguna (1993) suggested a size of the tabu

list of $\lfloor\sqrt{N_J}+0.5\rfloor$. Here, two entries k and l are made in the tabu list in every iteration. Therefore, the size of the tabu list is $2\lfloor\sqrt{N_J}+0.5\rfloor$. Simulations have shown that, for the considered problem, the relation $M(\mathbf{p},k,l,g)$ performs better than a neighborhood relation which is shifting groups of jobs.

4. RESULTS

The proposed scheduling strategy is applied to a production lot from an industrial plant with four furnaces, two of which operate at the same temperature level. The considered production lot contains 101 jobs, where each job has between 4 and 28 tasks.

Table 1 contains the makespan T_m, the makespan relative to its theoretical lower bound $T_m/\underline{T}_m$, and the computational time[1] T_c for different stages of the sequence optimization. The makespan T_m is calculated by the timetabling algorithm from Section 2. The original sequence represents the actual production order in the plant, which was manually optimized by a senior operator. The table shows that after 500 iterations of the tabu search algorithm the makespan T_m is decreased by almost 10% compared to the original sequence. The quality of the obtained schedule may also be assessed based on a comparison of the makespan T_m with its theoretical lower bound $\underline{T}_m$. A theoretical lower bound $\underline{T}_m$ can be obtained by summing up the lower bounds of task durations $\underline{d}_{j,n}$ which require the most utilized machine type $\hat{m}$ divided by the maximum capacity $c_{\hat{m},max}$ of this machine type, i. e.,

$$\underline{T}_m = \frac{\sum_j \sum_{n\ requires\ \hat{m}} \underline{d}_{j,n}}{c_{\hat{m},max}}. \tag{4}$$

Here, the most utilized machine type $\hat{m}$ is known to be the furnaces which are operated at the same temperature level. The makespan T_m achieved after 500 iterations of the tabu search algorithm is only 0.84% higher than the theoretical lower bound $\underline{T}_m$.

Table 1 also shows that already the construction heuristic achieves an excellent result for the makespan T_m. This is due to the existence of a unique bottleneck machine type in the considered problem. Despite the long computational time of the tabu search algorithm, it can only slightly improve the result of the construction heuristic.

Table 1. Makespan T_m, makespan relative to its theoretical lower bound $T_m/\underline{T}_m$, and computational time T_c for different stages of the sequence optimization.

	original sequence	construction heuristic	tabu search 500 iterations
T_m	55.167 h	50.6 h	50.425 h
$T_m/\underline{T}_m$	110.32 %	101.19 %	100.84 %
T_c	-	2 s	1.5 min

Figure 4 shows the evolution of the makespan T_m during the tabu search algorithm with the starting permutation $\mathbf{p}_0$ generated by the construction heuristic. Hardly any improvement can be achieved due to the good starting permutation. Consequently, this means that the number of iterations of the tabu search algorithm could be reduced and computational time could be saved without loss of performance.

[1] The scheduling is done in MATLAB with partial C implementation on a personal computer with Intel i7 processor, 16GB RAM, and Windows 10 operating system. For faster execution, parallel computing is used for the evaluation of the neighborhood of the tabu search algorithm.

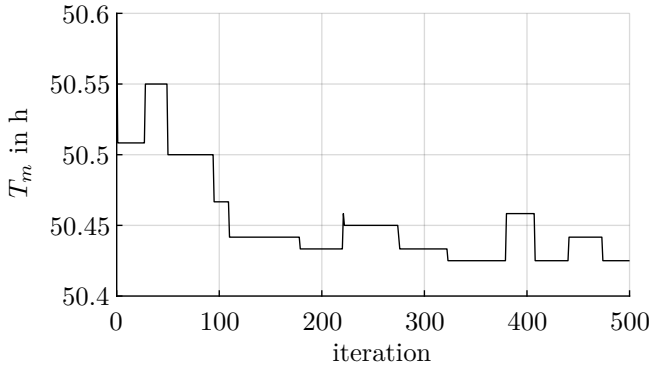

Fig. 4. Makespan evolution of the tabu search algorithm with a starting permutation generated by the construction heuristic.

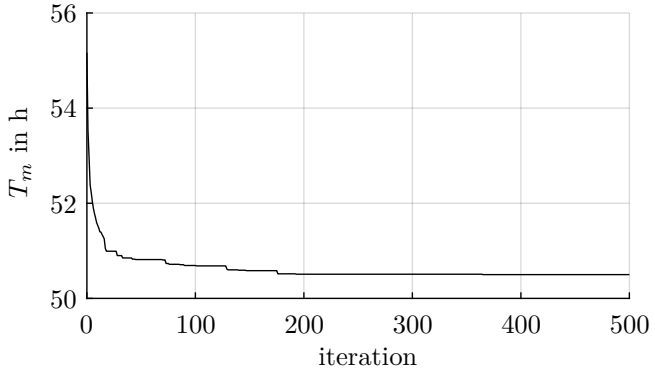

Fig. 5. Makespan evolution of the tabu search algorithm with the original sequence as starting permutation.

The effectiveness of the tabu seach algorithm is demonstrated in Fig. 5 by choosing the original sequence as starting permutation $\mathbf{p}_0$. In the beginning, the makespan T_m declines rapidly. Later on, it is tending to almost the same makespan T_m as reached by the combination of the construction heuristic and the tabu search algorithm. This indicates that the sequence optimization procedure is also suitable if the construction heuristic generates a poor starting permutation $\mathbf{p}_0$ which might be the case in the absence of a unique bottleneck machine type.

During the practical operation of the plant unforeseen errors may occur. Depending on the type of error, different strategies of troubleshooting are pursued. Minor errors, e. g., time delays at the roller table, can be handled by a rerun of the timetabling algorithm. This can be done almost instantaneously. Major errors, which change the configuration of the plant, e. g., failure of a furnace, require a complete rescheduling.

5. CONCLUSIONS

The aim of this work was to derive a scheduling algorithm for an industrial plant with a hot rolling mill and four reheating furnaces. Similar problems can be found in the literature but none of them systematically handles all constraints relevant in the considered plant. The decomposition into a timetabling and a sequencing part proved useful for this problem. For the timetabling, a recursive algorithm was developed to generate a feasible schedule.

It can also handle limited blocking times. An initial assignment to parallel machines is avoided by introducing so-called machine types. For the sequencing, an initial sequence is built up by a construction heuristic which tries to utilize a bottleneck machine type as much as possible. This sequence is further improved by a tabu search algorithm. Therefore, a special neighborhood structure is defined. The algorithm is applied to production data from the industrial plant. For this scenario, the proposed algorithm reduced the total processing time by almost 10% compared to the schedule that was actually used in the plant. Thus, the algorithm can increase the throughput rate by almost 10%. The achieved total processing time is only 0.84% higher than its theoretical lower bound.

REFERENCES

Allahverdi, A. (2016). A survey of scheduling problems with no-wait in process. *European Journal of Operational Research*, 255(3), 665–686.

Błażewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., and Węglarz, J. (2001). *Scheduling Computer and Manufacturing Processes*. Springer.

Brucker, P. and Schlie, R. (1991). Job-shop scheduling with multi-purpose machines. *Computing*, 45(4), 369–375.

Dauzère-Pérès, S. and Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70(0), 281–306.

Framinan, J.M. and Schuster, C. (2006). An enhanced timetabling procedure for the no-wait job shop problem: A complete local search approach. *Computers and Operations Research*, 33(5), 1200–1213.

Glover, F. and Laguna, M. (1993). Tabu search. In C.R. Reeves (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, 70–150. John Wiley & Sons, Inc.

Hall, N.G. and Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3), 510–525.

Mascis, A. and Pacciarelli, D. (2002). Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3), 498–517.

Paulli, J. (1995). A hierarchical approach for the FMS scheduling problem. *European Journal of Operational Research*, 86(1), 32–42.

Pinedo, M.L. (2016). *Scheduling*. Springer.

Raaymakers, W. and Hoogeveen, J. (2000). Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing. *European Journal of Operational Research*, 126(1), 131–151.

Samarghandi, H., ElMekkawy, T.Y., and Ibrahem, A.M.M. (2013). Studying the effect of different combinations of timetabling with sequencing algorithms to solve the no-wait job shop scheduling problem. *International Journal of Production Research*, 51(16), 4942–4965.

Schuster, C.J. and Framinan, J.M. (2003). Approximative procedures for no-wait job shop scheduling. *Operations Research Letters*, 31(4), 308–318.