Minimising total tardiness for a single machine scheduling problem with family setups and resource constraints

Oliver Herr[a], Asvin Goel[b]

[a]*Jacobs University, Bremen, Germany*
[b]*Kühne Logistics University, Hamburg, Germany*

Abstract

This paper considers a single machine scheduling problem in which each job to be scheduled belongs to a family and setups are required between jobs belonging to different families. Each job requires a certain amount of resource that is supplied through upstream processes. Therefore, schedules must be generated in such a way that the total resource demand does not exceed the resource supply up to any point in time. The goal is to find a schedule minimising total tardiness with respect to the given due dates of the jobs. A mathematical formulation and a heuristic solution approach for two variants of the problem are presented. Computational experiments show that the proposed heuristic outperforms a state-of-the-art commercial mixed integer programming solver both in terms of solution quality and computation time.

1. Introduction

In this paper we study the problem of scheduling jobs on a single machine with the goal of minimising total tardiness. Each job has a given processing time, a due date, and belongs to a given family. The machine can only process one job at a time and each job must be processed without preemption. A setup task has to be conducted between jobs belonging to different families and during this setup the machine cannot process any job.

In the problem studied in this paper, each job requires a certain amount of a common resource that is supplied through upstream processes. At any time, the

Email addresses: o.herr@jacobs-university.de (Oliver Herr), asvin.goel@the-klu.org (Asvin Goel)

cumulative consumption must not exceed the cumulative supply. Therefore, jobs may have to wait due to an insufficient availability of the resource.

Figure 1 illustrates the implication of the resource constraints. The figure shows the cumulative amount of resource supplied and the cumulative amount of resource required over time. The cumulative amount of resource supplied is shown as a linear curve with a constant supply rate. The cumulative resource demand over time is shown as a piece-wise linear curve that increases whenever a job is processed. The dotted vertical lines illustrate completion times of individual jobs. Horizontal segments of the demand curve illustrate times during which the machine has not yet started processing the next job, e.g. because a setup is conducted. As the cumulative amount of resource required must not exceed the cumulative amount of resource supplied at any time and because each job must be processed without preemption, the machine may also have to be idle before starting to process a job due to limited resource availability.

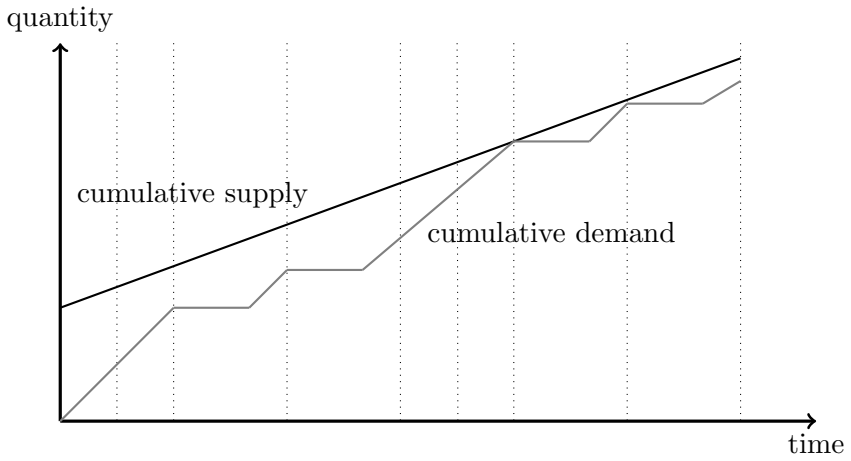

Figure 1: Depiction of the resource constraint in the context of scheduling with family setup times.

Our work is motivated by a practical problem arising in the continuous casting stage of steel production. A continuous caster is fed with ladles of liquid steel. Each ladle contains a certain steel grade and has orders allocated to it that determine a due date. Whenever two ladles of similar steel grade (within one setup family) are processed consecutively, no setup process is necessary. However, a setup is required whenever changing to a steel grade from another setup family. The liquid steel is produced from hot iron supplied by the blast furnace with a constant rate.

The sequence of ladles, including setups between ladles of different setup families, is not allowed to consume more hot metal then supplied by the blast furnace (see e.g. Box and Herbe 1988).

Similar situations occur in multi-stage production processes, where upstream work systems supply the common resource that is consumed by the jobs produced on the machine. Examples can be found in assembly processes where parts and components provided by an upstream stage are used to assemble different products (e.g. Drótos and Kis 2013, cutting pieces from a steel slab).

The remainder of this paper is organised as follows. Section 2 gives an overview of related work. Section 3 contains a detailed description of the problem and presents MIP formulations for two variants of the problem. In Section 4 we present an iterated local search approach for solving the problem. Section 5 presents computational experiments before final remarks are given in Section 6.

2. Related work

There are three main streams of literature that are of interest for the problem studied in this paper. First, scheduling with the target of minimising total tardiness. Second, single machine scheduling with setup considerations. Third, scheduling with resource constraints.

Comprehensive surveys for the minimum tardiness scheduling problem are presented by Koulamas (2010) and Sen et al. (2003). Even without setup considerations, the single machine total tardiness problem is proven to be NP-hard (Du and Leung 1990). Most methods developed for single machine problems that minimise total tardiness use properties developed by Emmons (1969) and Lawler (1977). Emmons (1969) describes conditions that need to be fulfilled in an optimal schedule, and Lawler (1977) introduced a decomposition approach that separates a problem into two mutually exclusive sub problems using the longest job to separate. For the problem of scheduling independent jobs on identical parallel machines, Shim and Kim (2007) present a branch and bound algorithm to minimise total tardiness. Schaller (2009) presents improved lower bounds that can be used in this branch-and-bound algorithm to reduce the computational effort required. Recently, Lee and Kim (2015) present a branch-and-bound algorithm for the problem of minimising the total tardiness of jobs in which for both of the two identical machines periodic maintenance activities are required during which the machine cannot process any job.Furthermore, Mensendiek et al. (2015) developed properties for optimal sequences with the total tardiness objective on parallel machines with fixed delivery dates and present heuristics for solving the problem.

The literature on scheduling with setup considerations is summarised e.g. in surveys of Allahverdi et al. (2008) and Potts and Kovalyov (2000). For the problem of minimising total tardiness on a single machine with sequence-dependent setup times, Gupta and Smith (2006) presented a GRASP multi-start heuristic as well as a space-based local search procedure, Liao and Juan (2007) developed a method based on ant colony optimization, Lin and Ying (2008) a hybrid of simulated annealing and tabu search, Ying et al. (2009) an iterated greedy heuristic based on local search, and Sioud et al. (2012) a hybrid genetic algorithm. An exact branch-and-bound algorithm for this problem class is presented by Bigras et al. (2008). For the variant of the problem where the goal is to mininise the weigthed tardiness of all jobs, Tanaka and Araki (2013) recently developed an exact procedure based successive sublimation dynamic programming and Subramanian et al. (2014) recently presented an iterated local search heuristic.

In family scheduling problems, all jobs are assigned to a certain family and a set of jobs of the same family that is produced consecutively without a setup is called a *batch*. While the allocation of jobs to families is given as a parameter, the allocation of jobs to batches for a certain setup family is part of the decision process. Under the *group technology assumption (GTA)* (see e.g. Potts and Van Wassenhove 1992) all jobs of the same setup family must be produced within exactly one batch, while several batches of the same family can be scheduled in the general case that is considered in this paper.

Gupta and Chantaravarapan (2008) and Schaller (2007) study a family scheduling problem in which the goal is to minimise total tardiness. Gupta and Chantaravarapan (2008) studied the problem under consideration of the GTA. They present a MIP formulation to solve small problem instances as well as a heuristic algorithm for larger instances. In their heuristic the authors separate the sequencing of jobs within a batch and the sequence of batches. Inside each batch they used a combination of neighbourhood operators previously developed by Holsenback and Russell (1992) and Panwalkar et al. (1993).

Schaller (2007) studied the family scheduling problem with and without the GTA. Based on the properties described by Emmons (1969), two optimal branch and bound procedures for both cases are developed. Furthermore, a heuristic based on five local search moves is proposed. These moves include the combining of two batches of the same setup family, moving jobs between batches of the same setup family, breaking a batch into two parts, and interchanging pairs of jobs. Furthermore, Schaller and Gupta (2008) study the minimisation of both earliness and tardiness for a single machine scheduling problem with family setups and propose exact and heuristic methods with and without the GTA. More recently, Schaller

(2014) presents several heuristic approaches for scheduling identical parallel machines with family setups for the problem of minimising total tardiness.

Grigoriev et al. (2005) provide a survey on scheduling problems with raw material constraints. They distinguish between three types of raw material usages: a) each job has its own raw material, b) a common resource is required by all jobs, and c) multiple common raw materials are required for each job. For all three cases they considered the objective of minimising maximum lateness and minimising makespan. Briskorn et al. (2010) study a single machine scheduling problem where jobs cannot be processed if the required resource is not available. Among the objectives considered are the objectives of minimising maximum lateness and minimising the number of tardy jobs. Györgyi and Kis (2014) study variations of the problem of minimising makespan with resource constraints using propositions from the knapsack and vertex cover problems to develop a polynomial-time approximation scheme. Briskorn et al. (2013) study the single machine problem with inventory constraints to minimise the weighted sum of completion times. The authors derived properties for an optimal solution and developed a branch and bound as well as a dynamic programming approach to solve the problem. They conclude that even for small problem instances with 20 jobs, exact approaches are not efficient enough and heuristic approaches are required. None of these works considers the objective of minimising total tardiness.

While previous work on family scheduling problems does not include resource constraints, papers published on scheduling problems with resource constraints do not consider family setups. To the best of our knowledge, the single machine scheduling problem with family setups and resource constraints has not been studied.

3. Problem Description

The machine scheduling problem studied in this paper can be described as follows. Let J denote a set of jobs to be processed by a single machine. Each job is characterised by a due date d_j, a processing time p_j, and the quantity q_j of a resource required by the machine to process the job. It is assumed that the resource is consumed at a constant rate q_j/p_j. Furthermore, each job belongs to a given setup family f_j. For any pair of jobs $i, j \in J$ with $f_i \neq f_j$, a setup of duration s_{ij} is required between processing jobs i and j. A single machine is available that can process one job at a time. The resource required is supplied with a constant rate of r per unit of time, and initially an amount of r^* of the resource is available. The goal is to find a production schedule that minimises total tardiness.

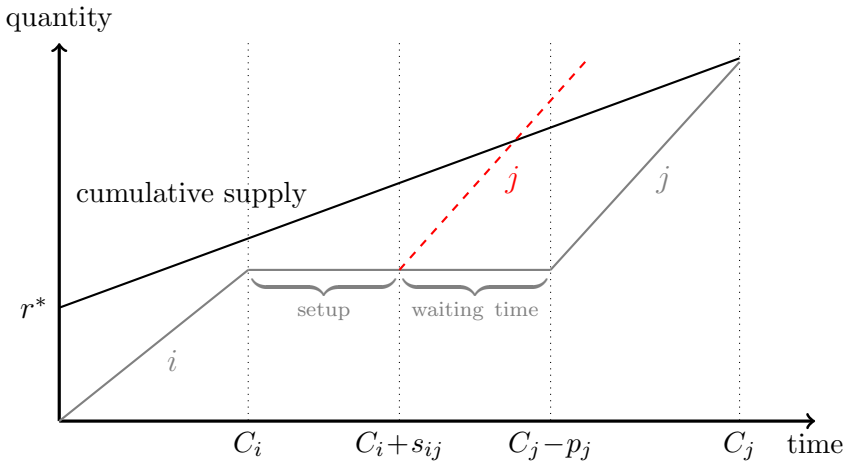

Figure 2: Example how required waiting time can result in an increased completion time.

The fundamental difference of this problem to the case without resource constraints is, that it may be necessary that the machine is idle because the required resource for the next job is net yet available, whereas in the case without resource constraints the machine is only idle for the time of the setups that may be required. In the case without resource constraints the optimal duration of any subsequence of jobs is always the sum of all processing times and the required setups. Figure 2 shows an example where the minimum duration of a sequence is larger than the sum of processing and setup times because the machine has to wait for the resource required. In the example shown in the figure, it is not possible to schedule job j immediately after job i and the completion of the setup, because the cumulative resource requirements exceeds the cumulative resource supply if no additional waiting time is scheduled.

Without resource constraints, any sequence of jobs of the same family can be reordered without impacting the completion time of this sequence and the tardiness of subsequent jobs. Therefore, it is possible to locally optimise the order in which jobs of the same family are scheduled. In the presence of resource constraints, however, any permutation of jobs may lead to an increase or decrease of the cumulative duration due to necessary waiting times. Figure 3 illustrates an example in which the sequence of jobs within a batch can impact cumulative duration due to necessary waiting times. In the sequence illustrated by the dashed line, job j_2 has to wait after completion of job j_3 due to insufficient resource avail-

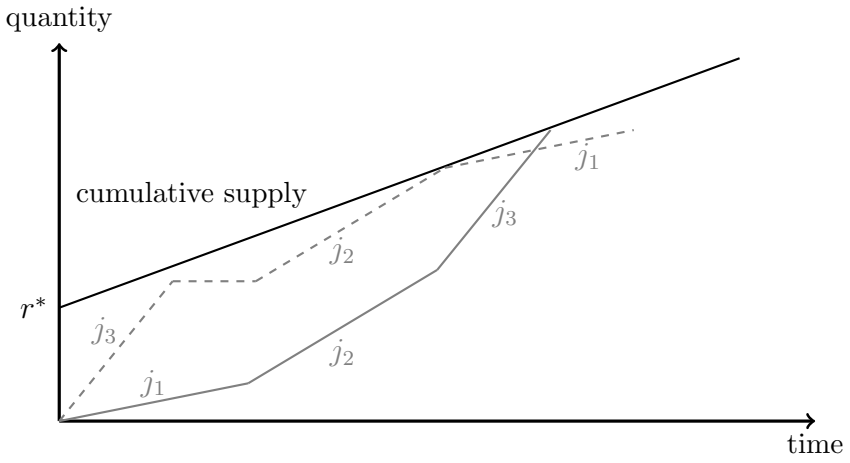

Figure 3: Example on how the sequence of jobs within a batch can increase the total completion time.

ability, whereas no waiting time is required in the sequence illustrated by the solid line. Thus, the tardiness of subsequent jobs may change if the order of jobs within a subsequence is modified.

For the first problem variant studied in this paper let us assume that the machine can be idle for any period of time after completion of one job and before starting to process the next job. For the ease of notation we assume that $s_{ij} = 0$ for any pair of jobs $i, j \in J$ with $f_i = f_j$. Furthermore, we assume that each schedule begins with a dummy job j^* that is included in J. This job has a due date large enough so that it will never be tardy, has zero precessing time and resource demand, and no setup time is required before processing any other job. Let x_{ij} denote a binary variable indicating whether job $j \in J$ is scheduled immediately after job i ($x_{ij} = 1$) or not ($x_{ij} = 0$).

For each job $j \in J$ let C_j, T_j and Q_j be variables indicating the completion time, the tardiness, and the accumulated amount of the resource requirements. Furthermore, let x_{ij} denote a binary variable indicating whether job i is processed immediately before job j or not.

The problem is

$$\text{minimise} \sum_{j \in J} T_j \tag{1}$$

subject to

$$\sum_{i \in J \setminus \{j\}} x_{ij} = 1 \text{ for all } j \in J \tag{2a}$$

$$\sum_{j \in J \setminus \{i\}} x_{ij} = 1 \text{ for all } i \in J \tag{2b}$$

$$C_{j^*} = 0 \tag{3a}$$

$$C_j \geq C_i + s_{ij} + p_j - (1 - x_{ij})M \text{ for all } i \in J, j \in J \setminus \{j^*\} \tag{3b}$$

$$T_j \geq 0 \tag{4a}$$

$$T_j \geq C_j - d_j \text{ for all } j \in J \tag{4b}$$

$$Q_{j^*} = 0 \tag{5a}$$

$$Q_j \geq Q_i + q_j - (1 - x_{ij})M \text{ for all } i \in J, j \in J \setminus \{j^*\} \tag{5b}$$

$$Q_j \leq r^* + rC_j \text{ for all } j \in J \tag{6}$$

$$x_{ij} \in \{0, 1\} \text{ for all } i, j \in J \tag{7}$$

The objective (1) is to minimise the sum of tardiness of all jobs in the production schedule. Constraints (2a) and (2b) require that each job is fulfilled exactly once. Constraints (3a) and (3b) require that the completion time of a job must be at least as large as the the completion time of the preceding job plus the required setup time (which may be zero), and the processing time. Constraints (4a) and (4b) restrict the tardiness of each job so that in the optimal solution we have $T_j = \max\{0, C_j - d_j\}$. Constraints (5a) and (5b) ensure that Q_j is at least as large as the cumulative resource requirements until job $j \in J$. The cumulative resource requirements must not exceed the amount available until completion of the job as required by constraint (6). Finally, constraint (7) restricts the domain of the binary decision variables.

Because of operational requirements it is not always possible that a machine can be idle between completion of one job and start of the next job. This is the

case, if a machine must be brought to a state in which it can remain idle, e.g. if cleaning is required. For such cases, we assume that the duration of a setup that may be required between jobs of different families can be increased by an arbitrary amount, however, an additional setup between jobs of the same family is required if the job does not start immediately after completion of the preceding job. For this second problem variant we thus have $s_{ij} > 0$ for jobs $i, j \in J$ with $f_i = f_j$. As this setup between jobs of the same family is not always necessary we have to add a binary variable y_{ij} indicating whether a setup is conducted between jobs i and j to our model. The following constraints are added to the problem

$$y_{ij} = x_{ij} \text{ for all } i, j \in J : f_i \neq f_j \tag{8a}$$
$$y_{ij} \leq x_{ij} \text{ for all } i, j \in J : f_i = f_j \tag{8b}$$

$$y_{ij} \in \{0, 1\} \text{ for all } i, j \in J \tag{9}$$

and constraints (3a) and (3b) are replaced by constraints

$$C_{j^*} = 0 \tag{10a}$$
$$C_j \geq C_i + y_{ij} s_{ij} + p_j - (1 - x_{ij})M \text{ for all } i \in J, j \in J \setminus \{j^*\} \tag{10b}$$
$$C_j \leq C_i + p_j + (1 - x_{ij} + y_{ij})M \text{ for all } i \in J, j \in J \setminus \{j^*\}. \tag{10c}$$

Constraint (8a) ensures that a setup is conducted if jobs of different families are processed after another, whereas constraint (8b) allows a setup to be conducted if jobs of the same family are processed after another. Constraint (9) gives the domain of the additional binary decision variables. Like constraint (3a) and (3b), constraints (10a) and (10b) require that the completion time of a job must be at least as large as the the completion time of the preceding job plus the setup time, if a setup is conducted, and the processing time. Furthermore, constraint (10c) ensures that job j is processed immediately after completion of a preceding job i if no setup is conducted.

Obviously, any feasible solution for this problem variant is also a feasible solution to the first problem variant described by (1) - (7). However, the sequences of jobs that are optimal for the two problem variants may differ. Let us consider the example with two jobs belonging to the same setup family with parameters $d_1 = 0$, $p_1 = 14$ and $d_2 = 15$, $p_2 = 12$. Furthermore, assume that at the beginning of the planning horizon there is an initial inventory that is sufficient to process either of the jobs without delay, but not both, and that after 30 units of time the

resource supply reaches a level that is sufficient to process both jobs. Furthermore, let us assume that for the second problem variant, the duration of a setup between the two jobs is sufficiently large, so that an optimal solution will not have a setup between the two jobs. Figure 4 illustrates the optimal sequences for both problem variants. The optimal sequence is illustrated by a solid line, whereas the inferior sequence by a dashed line. For the first problem variant, where the machine may be idle for any period of time after completion of one job and before starting to process the next job, the optimal sequence is to process job 1 before job 2 with $C_1 = 14$, $T_1 = 14$, $C_2 = 30$, $T_2 = 15$ and a total tardiness of 29. If job 2 is processed before job 1, we have $C_1 = 30$, $T_1 = 30$, $C_2 = 12$, $T_2 = 0$ and a total tardiness of 30. For the second problem variant, where an additional setup between jobs of the same family is required if the job does not start immediately after completion of the preceding job, it is better to delay the start of the first job in the optimal sequence instead of adding an additional setup. The optimal sequence is to process job 2 before job 1 with $C_1 = 30$, $T_1 = 30$, $C_2 = 16$, $T_2 = 1$ and a total tardiness of 31. If job 1 is processed before job 2, we have $C_1 = 18$, $T_1 = 18$, $C_2 = 30$, $T_2 = 15$ and a total tardiness of 33.

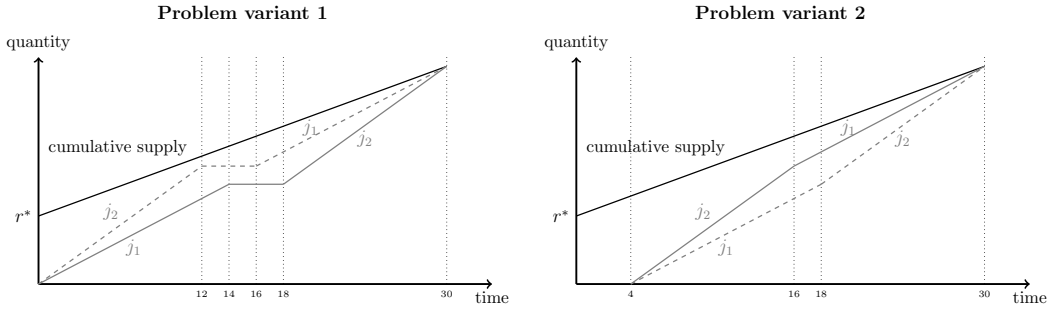

Figure 4: Example on how the optimal sequences can differ for both problem variants.

As the above example and the example of Figure 3 illustrate, the optimal sequence depends on both the resource availability as well as the operational details determining when necessary waiting times can be scheduled. Thus, the previously developed optimality conditions for minimising total tardiness, e.g. those presented by Schaller (2007), are not valid for scheduling with resource constraints. As a result, a solution approach cannot exploit these properties.

4. Solution approach

Solving the problem described above using a MIP-solver is in general too time consuming for problem instances with larger numbers of jobs. In fact, if all jobs belong to the same family and r^* is sufficiently large, the problem reduces to the single machine total tardiness problem studied by Du and Leung (1990) which is proven to be NP-hard. It follows that the problem studied in this paper is also NP-hard. Therefore, we present a heuristic approach based on simple operators for modifying the sequence of jobs. With this approach we are able to generate schedules with low total tardiness quickly.

```
1: set s := initialsolution()
2: set i := 1;
3: while i ≤ k do
4:   repeat
5:     select neighbourhood operator Ω
6:     for all schedules s' that can be generated by applying Ω to s do
7:       if tardiness(s') < tardiness(s) then
8:           set s := s'
9:       end if
10:    end for
11:  until s is locally optimal for all operators
12:  set s := perturbation(s)
13:  set i := i + 1;
14: end while;
```

Figure 5: Heuristic

The pseudocode of our heuristic is shown in Figure 5. The approach begins by generating an initial solution which can be any sequence of jobs. It then initialises an iteration counter and repeats the same steps until the iteration counter has reached a given limit. In each iteration, the approach selects the next neighbourhood operator from a randomly generated sequence of the operators. For each operator, all possible moves are examined. If an improving solution is found, this solution is accepted as the new incumbent solution. If no improvement can be obtained by any of the operators a locally optimal solution is found. Following the iterated local search framework of Lourenço et al. (2003), the search process is repeated in order to be able to escape from local optima of poor quality. Before

continuing with the next iteration, the incumbent solution is perturbed in order to obtain a new solution that potentially can be improved using the operators.

The solution approach comprises three main components: the evaluation of tardiness of a given sequence of jobs, different neighbourhood operators to modify sequences, and a method to perturb a solution.

4.1. Tardiness evaluation

To evaluate the tardiness of a particular sequence the approach has to compute completion times and tardiness values for each job in the sequence. For the first problem variant, in which the machine may be idle for any period of time after completion of one job and before starting to process the next job, optimal completion times for a given sequence of jobs can be computed very easily. For a job j in a given sequence, let Q_j denote the cumulative amount of resource required by all jobs in the sequence up to job j. The completion times of the jobs in the sequence can be computed as follows. The first job j in the sequence has completion time

$$C_j = \max\{p_j, \frac{Q_j - r^*}{r}\}.$$

For all other jobs j the completion time is

$$C_j = \max\{C_i + s_{ij} + p_j, \frac{Q_j - r^*}{r}\},$$

where job i is the predecessor of j. The tardiness of any job j is

$$T_j = \max\{0, C_j - d_j\}.$$

Now, let us consider the second problem variant, where an additional setup between jobs of the same family is required if the job does not start immediately after completion of the preceding job. If the values y_{ij} indicating whether there is a setup between i and j are known, completion times and tardiness can be scheduled analogously to the method described above. The solution approach presented in this paper, however, is based on operators changing the sequence of jobs and the decision whether a setup is taken or not is not explicitly taken by the operators. Instead, this decision is taken when calculating the completion times of the jobs in the sequence. The first job j in the sequence is tentatively given the completion time

$$C_j = \max\{p_j, \frac{Q_j - r^*}{r}\}.$$

This value may have to be increased if a subsequent job without intermediate setup cannot be processed due to the resource constraint.

For any other job j, we have to distinguish between several cases. If $f_i \neq f_j$, where job i is the predecessor of j, then a setup is required and the completion time job j is tentatively set to

$$C_j = \max\{C_i + s_{ij} + p_j, \frac{Q_j - r^*}{r}\}.$$

If $f_i = f_j$ then two alternatives must be considered. In the first alternative an additional setup is included and the completion time of job j is tentatively set to

$$C_j = \max\{C_i + s_{ij} + p_j, \frac{Q_j - r^*}{r}\}.$$

In the second alternative, no setup is made between i and j and the completion time of job j is tentatively set to

$$C_j = \max\{C_i + p_j, \frac{Q_j - r^*}{r}\}.$$

To eliminate possible idle time between jobs, the completion time of all jobs prior to j which are not separated by a setup is increased by

$$\Delta = \max\{0, \frac{Q_j - r^*}{r} - (C_i + p_j)\}.$$

As above, the tardiness of any job j is

$$T_j = \min\{0, C_j - d_j\}.$$

The approach can efficiently be implemented as follows. For any job in a given sequence of jobs let $l = (l^{\text{time}}, l^{\text{tardiness}})$ be a label where l^{time} denotes the completion time of the job and $l^{\text{tardiness}}$ denotes the cumulative tardiness until completion of the job. The approach begins with a label $l = (0, 0)$. Then it iterates through the given sequence of jobs and calculates completion times and tardiness values as described above. For each different alternative a new label is generated. By extending each alternative label, a tree of alternatives is generated. To reduce the number of alternative labels to be considered the following dominance criteria are used to reduce the number of labels.

Proposition 1. *Let l_1 and l_2 denote labels associated to schedules up to the same job i in a sequence of jobs and let j denote the next job in the sequence. Label l_1 dominates label l_2 if*

$$l_1^{\text{time}} + s_{ij} \leq l_2^{\text{time}}$$

and

$$l_1^{\text{tardiness}} \leq l_2^{\text{tardiness}}.$$

Proof: Let $\hat{l}_1$ denote the label associated to the schedule obtained by adding a setup of duration s_{ij} and job j to the schedule associated to l_1 and let $\hat{l}_2$ denote the label associated to a schedule obtained by extending the schedule associated to l_2 by job j. We have $\hat{l}_1^{\text{time}} \leq \hat{l}_2^{\text{time}}$ and $\hat{l}_1^{\text{tardiness}} \leq \hat{l}_2^{\text{tardiness}}$. Furthermore, for any extension of $\hat{l}_1$ and $\hat{l}_2$ adding setups at the same positions, every job in the schedule associated to the former will be completed earlier or at the same time compared to the same job in the schedule associated to the latter. Thus, the total tardiness of the former will be smaller or equal to the latter and l_1 dominates l_2. □

By eliminating all dominated labels, the size of the search tree built to evaluate total tardiness is reduced effectively.

It must be noted that total tardiness can similarly be evaluated for non-constant supply and demand rates. By replacing all occurrences of the term $\frac{Q_j - r^*}{r}$ with a function that calculates the earliest point in time when the job j can be completed subject to a sufficient resource availability, arbitrary supply and demand patterns can be considered.

4.2. Operators

The solution approach uses six different operators. The first two operators are directly based on the sequence of jobs, whereas the other operators are based on batches, i.e. subsequences without a setup. The advantage of such batch-based operators is that some structural properties of the current solution are maintained and unnecessary setups can be avoided.

4.2.1. Job Move

The *Job Move* operator selects a job and inserts it at another position in the sequence. Figure 6 illustrates all possible operator moves for a given job. After selecting a job, the operator iterates through the sequence of jobs and evaluates the insertion of the job at all possible positions. If total tardiness can be reduced, the job is moved to the position leading to the lowest total tardiness. This operator is a generalisation of the *Move procedure* of Schaller (2007), which only moves jobs within batches of the same family. Our operator also allows to remove a job from

a batch and generate a new batch containing only the removed job. This may be particularly beneficial if the due date of the job does not fit well to the due dates of other jobs of the same family.

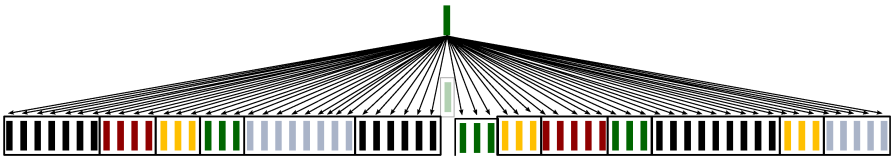

Figure 6: Example of all possible job move operations for a selected job.

4.2.2. Job Exchange

The *Job Exchange* operator selects a job and and exchanges its position with another job in the sequence. Figure 7 illustrates a single operator move. After selecting a job, the operator iterates through the sequence of jobs and evaluates the exchange of the job with any other job in the sequence. If total tardiness can be reduced, the positions of the two jobs leading to the lowest total tardiness reduction are exchanged. This operator is equivalent to the *JI procedure* proposed by Schaller (2007).

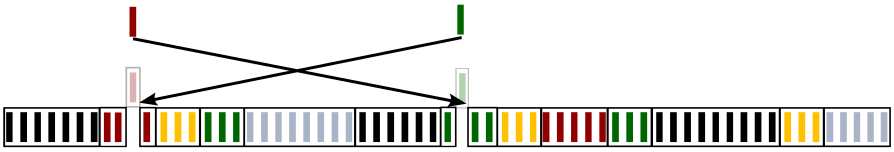

Figure 7: Example of exchanging two jobs.

4.2.3. Batch Move

The *Batch Move* operator selects a batch and inserts it at another position in the sequence. Figure 8 illustrates the operator move. After selecting a batch, the operator iterates through the sequence of batches and evaluates the insertion of the complete batch at any sequence position. If a reduction of the total tardiness can be achieved, the batch is moved to the position leading to the lowest total tardiness.

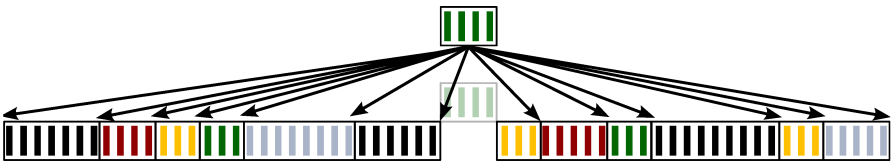

Figure 8: Example of all possible batch move operations for a selected batch.

4.2.4. Batch Exchange

The "Batch Exchange" operator selects a batch and and exchanges its position with another batch in the sequence of batches. Figure 9 illustrates a single operator move. After selecting a batch, the operator iterates through the sequence of batches and evaluates the exchange of the positions of the complete batches. If total tardiness can be reduced, the positions of the two batches leading to the lowest total tardiness reduction are exchanged. This operator is equivalent to the *BI procedure* proposed by Schaller (2007).

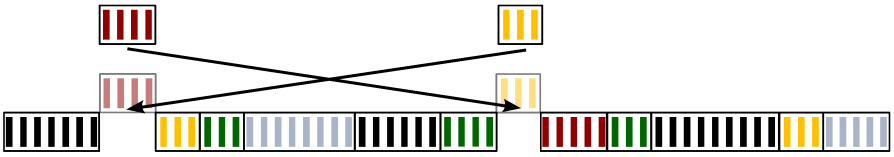

Figure 9: Example of exchanging the position of two batches.

4.2.5. Batch Combine

The *Batch Combine* operator selects a batch and combines the batch with the next batch of the same family. Figure 10 illustrates a single operator move. It removes the jobs of both batches and iterates through the sequence to evaluate whether reinsertion of the combination of both batches can reduce total tardiness. If total tardiness can be reduced, the combined batch is inserted at the position leading to the lowest total tardiness. This operator is equivalent to the *CONSOL procedure* proposed by Schaller (2007).

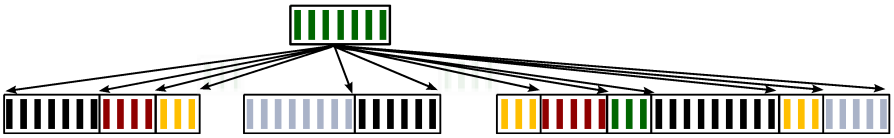

Figure 10: Example of all possible insertion positions for a combined batch.

4.2.6. Batch Break

The *Batch Break* operator selects a batch, breaks the batch into two parts and inserts both parts at a new position in the sequence. Figure 11 illustrates a single operator move. After selecting a batch, the operator sorts all the jobs within the batch according to their due date and divides the batch at the position of the largest due date difference. The operator then iterates over all possible sequence positions for both parts and evaluates the total tardiness. If total tardiness can be reduced, the parts are inserted at the positions leading to the lowest total tardiness.

Although the *Break procedure* proposed by Schaller (2007) can also be used to break a batch into two parts, both operators have conceptual differences. While the *Break procedure* moves one job at a time into a new batch as long as total tardiness is reduced, our operator breaks a batch at the position of the largest due date difference and inserts both parts at the best positions in the sequence. Especially, if jobs with inhomogeneous due dates are grouped together, our *Batch Break* operator can increase the likelihood that a new sequence with lower tardiness is found.

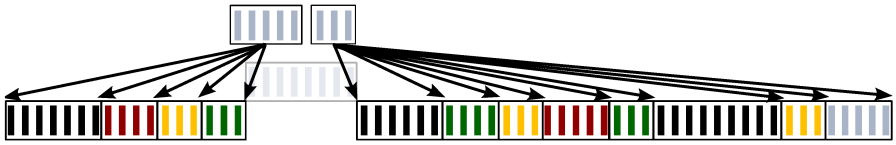

Figure 11: Example of all possible insertion positions for the two parts of a broken batch.

4.3. Perturbation

Our approach for perturbation is motivated by Lü and Hao (2009). Their critical element-guided perturbation strategy (CEGP) investigates a current best solution and calculates a scoring variable for each job. Based on the scores, which are derived from problem characteristics, a new initial sequence is derived.

The idea of our perturbation approach is to evaluate for each job the most promising sequence position in terms of tardiness minimisation. Based on the current solution we calculate the lateness of each job by

$$L_j = C_j - d_j.$$

A positive lateness value indicates that tardiness may be reduced by shifting this job towards the beginning of the sequence, whereas a negative lateness value indicates that the job can be shifted to a later position without increasing tardiness.

In order to estimate a promising shift in sequence positions, we divide each lateness value by the average processing time and calculate a new position index $k_j^\star$ by subtracting the calculated shift from the original position index k_j by

$$k_j^\star = k_j - \frac{|J|L_j}{\sum_{i \in J} p_i}.$$

A new initial solution is obtained by ordering the jobs according to their values k_j^* and using the original position index k_j as a tie breaker.

5. Computational experiments

This section presents test instances and computational experiments conducted to evaluate the proposed approach for both variants of the problem.

5.1. Test instances

Artificially instances are generated motivated from a real-world scheduling problem in the steel industry. The generated instances share common characteristics of the real-life problem, however, for confidentiality reasons, these characteristics cannot be described in detail.

For each instance, a set of jobs J and a set of families F are generated. Each family is randomly assigned one job and the remaining jobs are distributed across the families according to a random distribution derived from the real-world problem that motivated this work. Figure 12 gives an illustration of this distribution.

In the real-world problem, processing times of a job depend on the setup family and the production width. Therefore, for each setup family $f \in F$ processing times p_f are randomly generated in the range between 2400 and 3000. Then, for each job j in family f, the processing time p_j is randomly generated in the range between $\max(0.9p_f, 2400)$ and $\min(1.1p_f, 3000)$. Thus, processing times are similar for jobs of equal setup family, but still vary as a result of the production width.

For each pair of families, setup times between any pair of jobs of these families are randomly selected to be either 900 or 2700. This is also motivated from real world instances, where a reduced setup time is possible in case certain characteristics are given.

Motivated from the real problem instances, the resource demand q_j of job j is set to a random number in the range between 250 and 270.

The due date of jobs are set as follows. First, a sequence of jobs is generated by grouping all jobs belonging to one family together, and ordering the groups in

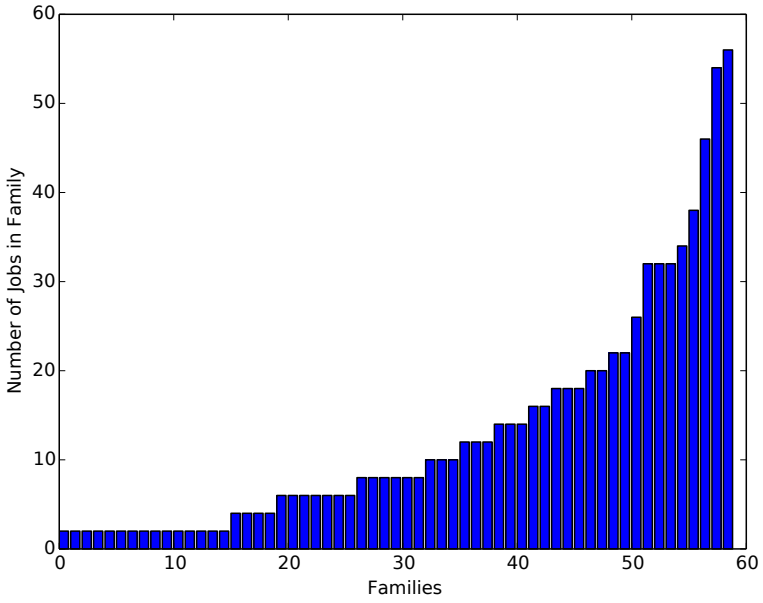

Figure 12: Distribution of jobs per setup family

descending number of jobs. This approach is similar to scheduling with the group technology assumption (GTA). The resulting sequence minimises the number of setups required. For this sequence we calculate the minimum total time t_1 required without resource constraints. Then, for each job j, the due date d_j is randomly set to a value between $-0.25t_1$ and $1.25t_1$. This random selection of due dates resembles the real-world situation, where some jobs are already delayed at the beginning of the planning horizon. As a negative due date will contribute equally to the total tardiness of any schedule, we replace negative due date values by zero.

In order to generate instances in which the resource constraint is not trivially satisfied and does not always require waiting times, the supply rate r and the initial inventory r^* are determined as follows. Another sequence is generated in which jobs are ordered according to their due dates, with the earliest due date first. For this sequence we calculate the minimum total time t_2 required without resource constraints. For all instances we have $t_1 < t_2$ because more setups are required. The supply rate r is chosen as $r = \frac{2}{t_1+t_2} \sum_{j \in J} q_j$. As shown in Figure 13, the resource supply (without initial inventory) grows at a rate that is between the average demand rates of the two sequences generated. The initial inventory

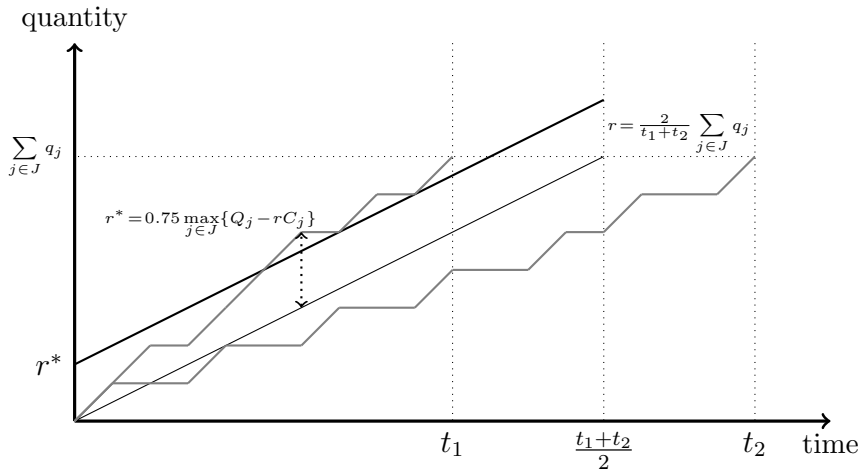

Figure 13: Supply rate and initial inventory are derived from two reference sequences.

is now set to 75% of the largest difference between the cumulative demand of the first sequence and the supply curve without initial inventory. As a result, the first sequence, which minimises the total number of setups, would be infeasible without additional waiting times. When searching for a solution with low total tardiness, a good tradeoff between minimising the number of setups and minimising waiting times thus has to be found.

5.2. Experiments

The solution approach presented in this paper is evaluated using the test instances described above which are available at http://www.telematique.eu/research/download. The experiments are conducted on an Intel Xeon(R) CPU W3530 @ 2.80GHz X 4 with UBUNTU 14.04 64-bit operating system. To obtain optimal solutions or lower bounds on the total tardiness the MIP formulated in Section 3 was solved using the commercial solver CPLEX with version 12.6, one thread, 2 GB RAM, and a 10 GB tree limit. Furthermore, a run time limit of 3600 seconds was used.

An initial sequence of jobs is obtained by ordering the jobs according to their due dates. This sequence is given to both the MIP solver and the heuristic solution approach as the initial solution. In the heuristic, the neighbourhood operators are selected in a random order and whenever no further improvement is possible with the chosen operator the next neighbourhood operator is selected. Based on

preliminary experiments we limited the iterated local search to three runs.

Tables 1 and 2 show average results for both variants of the problem and detailed results for the individual instances are provided in the Appendix. In Tables 1 and 2, each line shows average values of all instances with the same number of jobs. The first column indicates the number of jobs. The second column gives the average computation time (in seconds) required by the MIP solver. The third column indicates the solution quality by providing the average percentage by which the gap between the initial solution to the best lower bound is closed (GAP). This value is computed as $100(T^{\text{Start}} - T^{\text{UB}})/(T^{\text{Start}} - T^{\text{LB}})$, where T^{Start} denotes the total tardiness of the initial solution, T^{UB} denotes the total tardiness of the best found solution, and T^{LB} denotes the lower bound obtained by the MIP solver. A value of 100 indicates that the gap is closed completely, i.e. that the solution is optimal. The next two columns give the same information for the heuristic. The last column indicates the ratio between the GAP value for the heuristic divided by the GAP value for the MIP. A ratio larger than 1 indicates that the heuristic outperforms the MIP.

	MIP		**Heuristic**		
Jobs	**CPU**	**GAP**	**CPU**	**GAP**	**Ratio**
8	6.69	100.00	0.01	100.00	1.00
10	868.35	100.00	0.02	100.00	1.00
12	3331.11	40.07	0.04	40.07	1.00
15	3600.00	44.49	0.07	44.48	1.00
20	3600.00	49.57	0.23	49.76	1.01
30	3600.00	42.93	0.93	48.97	1.14
40	3600.00	29.99	2.61	48.86	1.87
50	3600.00	18.05	5.07	51.10	3.26

Table 1: Average results for first problem variant

The MIP manages to find optimal solutions for the smaller instances with 8 or 10 jobs in 20 out of 20 cases for the first problem variant and 19 out of 20 cases for the second problem variant. For all of these small-sized instances, the heuristic finds equally good solutions, however, the time required by the heuristic to find these solutions is magnitudes lower. For instances with more than 10 jobs, the MIP struggles closing the gap between the lower and upper bound and only finds the optimal solution for one of the instances with 12 jobs and 3 families and the first problem variant. This optimal solution is also found by the heuristic

	MIP		Heuristic		
Jobs	**CPU**	**GAP**	**CPU**	**GAP**	**Ratio**
8	17.63	100.00	0.17	100.00	1.00
10	1163.05	90.79	0.36	90.79	1.00
12	3326.04	39.06	0.62	39.06	1.00
15	3600.00	43.11	1.33	43.37	1.01
20	3600.00	47.53	3.54	48.66	1.03
30	3600.00	41.84	18.49	49.39	1.24
40	3600.00	29.36	61.70	49.43	1.98
50	3600.00	23.00	163.46	51.85	2.82

Table 2: Average results for second problem variant

approach.

In total, the heuristic is able to find equally good or better solutions for 77 of 80 instances for the first problem variant and for 79 of 80 instances for the second problem variant. As the computational effort required by the heuristic is much smaller than for the MIP solver, and typically takes only a few minutes or less, the heuristic is well suited for application scenarios where human decision makers want to obtain good schedules quickly.

6. Final remarks

This paper studies a single-machine family scheduling problem with sequence dependent setup times and resource constraints. This problem differs from similar problems without resource constraints because both the optimal sequence and the timing of jobs can be influenced by the resource constraint. We present a mathematical formulation for two variants of the problem that differ in the way necessary waiting times due to limited resource availability can be scheduled. We present a heuristic approach for both problem variants and show how tardiness can be evaluated. While tardiness evaluation is trivial if waiting times can be scheduled between any pair of jobs, the evaluation is non-trivial if waiting times can only be scheduled when a setup is performed. We present a labelling approach to determine the optimal scheduling of additional setups and waiting times. Computational experiments are conducted on instances derived from instances of a practical problem in steel production that motivated the research. Our experiments show that the proposed solution approach outperforms a state-of-the-art mixed integer programming solver both in terms of solution quality and compu-

tation time. Furthermore, the solution approach is fast enough to be used in practical scenarios where larger instances must be solved within a few minutes.

Throughout this paper we assumed that the resource is supplied and consumed at a constant rate. While these assumptions are essential for the mathematical formulation presented in this paper, the heuristic approach can easily be adapted to be used for different application scenarios with arbitrary supply and demand. Thus, the proposed solution approach can be used for a variety of multi-stage production processes in which schedulers must take into account that a sufficient amount of a required resource is supplied by upstream processes.

References

A. Allahverdi, C. Ng, T. Cheng, and M. Kovalyov. A survey of scheduling problems with setups times or costs. *European Journal of Operational Research*, 187:985–1032, 2008.

L.-P. Bigras, M. Gamache, and G. Savard. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization*, 5(4):685–699, 2008.

R. E. Box and D. G. J. Herbe. A scheduling model for LTV Steel's Cleveland Works' twin strand continuous slab caster. *Interfaces*, 18(1):42–56, 1988.

D. Briskorn, B. C. Choi, K. Lee, J. Leung, and M. Pinedo. Complexity of single machine scheduling subject to nonnegative inventory constraints. *European Journal of Operational Research*, 207(2):605–619, 2010.

D. Briskorn, F. Jaehn, and E. Pesch. Exact algorithms for inventory constrained scheduling on a single machine. *Journal of Scheduling*, 16:105–115, 2013.

M. Drótos and T. Kis. Scheduling of inventory releasing jobs to minimize a regular objective function of delivery times. *Journal of Scheduling*, 16:337–346, 2013.

J. Du and J. Y.-T. Leung. Minimizing Total Tardiness on One Machine is NP-Hard. *Mathematics of Operations Research*, 15(3):483–495, 1990.

H. Emmons. One-Machine Sequencing to Minimize Certain Functions of Job Tardiness. *Operations Research*, 17(4):701–715, 1969.

A. Grigoriev, M. Holthuijsen, and J. Van De Klundert. Basic scheduling problems with raw material constraints. *Naval Research Logistics*, 52:527–535, 2005.

J. N. D. Gupta and S. Chantaravarapan. Single machine group scheduling with family setups to minimize total tardiness. *International Journal of Production Research*, 46(6):1707–1722, 2008.

S. R. Gupta and J. S. Smith. Algorithms for single machine total tardiness scheduling with sequence dependent setups. *European Journal of Operational Research*, 175:722–739, 2006.

P. Györgyi and T. Kis. Approximability of scheduling problems with resource consuming jobs. *Mathematics of Operations Research*, pages 1–21, 2014.

J. Holsenback and R. M. Russell. A heuristic algorithm for sequencing on one machine to minimize total tardiness. *Journal of the Operational Research Society*, 43(1):53–62, 1992.

C. Koulamas. The single-machine total tardiness scheduling problem: Review and extensions. *European Journal of Operational Research*, 202(1):1–7, 2010.

E. L. Lawler. A "Pseudopolynomial" Algorithm for Sequencing Jobs to Minimize Total Tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977.

J. Lee and Y. Kim. A branch and bound algorithm to minimize total tardiness of jobs in a two identical-parallel-machine scheduling problem with a machine availability constraint. *Journal of the Operational Research Society (to appear)*, 2015. URL http://dx.doi.org/10.1057/jors.2014.122.

C. J. Liao and H. C. Juan. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. *Computers & Operations Research*, 34:1899–1909, 2007.

S.-W. Lin and K.-C. Ying. A hybrid approach for single-machine tardiness problems with sequence-dependent setup times. *Journal of the Operational Research Society*, 59:1109–1119, 2008.

H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated Local Search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, 2003.

Z. Lü and J.-K. Hao. A Critical Element-Guided Perturbation Strategy for Iterated Local Search. In C. Carlos and P. Cowling, editors, *Evolutionary Computation in Combinatorial Optimization*, pages 1–12. Springer, 2009.

A. Mensendiek, J. N. D. Gupta, and J. Herrmann. Scheduling identical parallel machines with fixed delivery dates to minimize total tardiness. *European Journal of Operational Research*, 243(2):514–522, 2015.

S. S. Panwalkar, M. Smith, and C. Koulamas. A heuristic for the single machine tardiness problem. *European Journal of Operational Research*, 70(3):304–310, Nov. 1993.

C. N. Potts and M. Y. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120(2):228–249, Jan. 2000.

C. N. Potts and L. N. Van Wassenhove. Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of the Operational Research Society*, 43(5):395–406, 1992.

J. E. Schaller. Scheduling on a single machine with family setups to minimize total tardiness. *International Journal of Production Economics*, 105(2):329–344, 2007.

J. E. Schaller. Note on Shim and Kim's lower bounds for scheduling on identical parallel machines to minimize total tardiness. *European Journal of Operational Research*, 197(1):422–426, 2009.

J. E. Schaller. Minimizing total tardiness for scheduling identical parallel machines with family setups. *Computers & Industrial Engineering*, 72:274–281, 2014.

J. E. Schaller and J. N. D. Gupta. Single machine scheduling with family setups to minimize total earliness and tardiness. *European Journal of Operational Research*, 187(3):1050–1068, June 2008.

T. Sen, J. M. Sulek, and P. Dileepan. Static scheduling research to minimize weighted and unweighted tardiness: A state-of-the-art survey. *International Journal of Production Economics*, 83:1–12, 2003.

S. O. Shim and Y. D. Kim. Scheduling on parallel identical machines to minimize total tardiness. *European Journal of Operational Research*, 177(1):135–146, 2007.

A. Sioud, M. Gravel, and C. Gagné. A hybrid genetic algorithm for the single machine scheduling problem with sequence-dependent setup times. *Computers & Operations Research*, 39(10):2415–2424, 2012.

A. Subramanian, M. Battarra, and C. N. Potts. An Iterated Local Search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, 52(9):2729–2742, 2014.

S. Tanaka and M. Araki. An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times. *Computers & Operations Research*, 40(1):344–352, 2013.

K. C. Ying, S. W. Lin, and C. Y. Huang. Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications*, 36:7087–7092, 2009.

Appendix

In this appendix detailed results for all instances are reported. Tables 3 to 10 give the results for the first problem variant and tables 11 to 18 give the results for the second problem variant. In the tables the first column provides the name of the instance. The names of the instances are formatted as $|F|\mathrm{X}|J|\_k$ where $k \in \{1, 2, 3, 4, 5\}$ is a counter to distinguish different instances with the same number of jobs $|J|$ and families $|F|$. The second column in the tables give the total tardiness of the initial solution. The next columns present the results for the MIP and the heuristic. The tables shows the calculation time (CPU), lower bound obtained by CPLEX (LB), the upper bound (UP), and the degree to which the gap between initial solution and lower bound is closed (GAP). The last column indicates the ratio between the GAP value for the heuristic divided by the GAP value for the MIP.

		MIP				**Heuristic**			
Instance	**Start**	**CPU**	**LB**	**UB**	**GAP**	**CPU**	**UB**	**GAP**	**Ratio**
2X8_1	34385	4.56	18877	18877	100.00	0.01	18877	100.00	1.00
2X8_2	56837	15.12	44955	44955	100.00	0.01	44955	100.00	1.00
2X8_3	48070	9.19	31017	31017	100.00	0.01	31017	100.00	1.00
2X8_4	18440	10.14	17980	17980	100.00	0.01	17980	100.00	1.00
2X8_5	12663	2.02	10976	10976	100.00	0.01	10976	100.00	1.00
3X8_1	31187	4.15	19301	19301	100.00	0.01	19301	100.00	1.00
3X8_2	13094	4.06	10206	10206	100.00	0.01	10206	100.00	1.00
3X8_3	23474	2.80	13428	13428	100.00	0.02	13428	100.00	1.00
3X8_4	72688	12.70	46737	46737	100.00	0.02	46737	100.00	1.00
3X8_5	26070	2.14	13288	13288	100.00	0.01	13288	100.00	1.00

Table 3: Results for first problem variant and instances with 8 jobs

		MIP				Heuristic			
Instance	**Start**	**CPU**	**LB**	**UB**	**GAP**	**CPU**	**UB**	**GAP**	**Ratio**
2X10_1	47382	904.40	32315	32315	100.00	0.02	32315	100.00	1.00
2X10_2	38918	272.97	32885	32885	100.00	0.02	32885	100.00	1.00
2X10_3	67083	2892.49	65092	65092	100.00	0.02	65092	100.00	1.00
2X10_4	15264	87.26	13267	13267	100.00	0.02	13267	100.00	1.00
2X10_5	41238	201.12	13912	13912	100.00	0.02	13912	100.00	1.00
3X10_1	76089	1823.74	69399	69399	100.00	0.02	69399	100.00	1.00
3X10_2	2474	0.01	2474	2474	100.00	0.02	2474	100.00	1.00
3X10_3	34244	32.45	20324	20324	100.00	0.02	20324	100.00	1.00
3X10_4	104453	2445.04	61680	61680	100.00	0.05	61680	100.00	1.00
3X10_5	11134	23.97	11134	11134	100.00	0.02	11134	100.00	1.00

Table 4: Results for first problem variant and instances with 10 jobs

		MIP				Heuristic			
Instance	**Start**	**CPU**	**LB**	**UB**	**GAP**	**CPU**	**UB**	**GAP**	**Ratio**
2X12_1	25118	3600.00	17852	25098	0.28	0.03	25098	0.28	1.00
2X12_2	47667	3600.00	23898	39105	36.02	0.04	39105	36.02	1.00
2X12_3	35935	3600.00	11893	28914	29.20	0.03	28914	29.20	1.00
2X12_4	178945	3600.00	31879	119712	40.28	0.03	119712	40.28	1.00
2X12_5	91028	3600.00	17237	56634	46.61	0.03	56634	46.61	1.00
3X12_1	43703	911.12	14311	14311	100.00	0.05	14311	100.00	1.00
3X12_2	93588	3600.00	7557	45907	55.42	0.05	45907	55.42	1.00
3X12_3	52192	3600.00	20071	37753	44.95	0.04	37753	44.95	1.00
3X12_4	99820	3600.00	33035	92382	11.14	0.04	92382	11.14	1.00
3X12_5	85266	3600.00	21865	61906	36.85	0.05	61906	36.85	1.00

Table 5: Results for first problem variant and instances with 12 jobs

		MIP				Heuristic			
Instance	**Start**	**CPU**	**LB**	**UB**	**GAP**	**CPU**	**UB**	**GAP**	**Ratio**
2X15_1	184932	3600.00	17470	129985	32.81	0.07	129985	32.81	1.00
2X15_2	140058	3600.00	15957	82638	46.27	0.06	82638	46.27	1.00
2X15_3	32087	3600.00	15442	19723	74.28	0.05	19723	74.28	1.00
2X15_4	99821	3600.00	19655	79116	25.83	0.06	77968	27.26	1.06
2X15_5	58282	3600.00	11719	38408	42.68	0.05	38408	42.68	1.00
3X15_1	71363	3600.00	17969	61801	17.91	0.08	61801	17.91	1.00
3X15_2	139214	3600.00	15113	69009	56.57	0.08	70371	55.47	0.98
3X15_3	91090	3600.00	18911	46056	62.39	0.09	46038	62.42	1.00
3X15_4	129206	3600.00	16917	97339	28.38	0.07	97339	28.38	1.00
3X15_5	120990	3600.00	13845	59049	57.81	0.09	59605	57.29	0.99

Table 6: Results for first problem variant and instances with 15 jobs

		MIP				Heuristic			
Instance	**Start**	**CPU**	**LB**	**UB**	**GAP**	**CPU**	**UB**	**GAP**	**Ratio**
3X20_1	213859	3600.00	12203	113884	49.58	0.14	113884	49.58	1.00
3X20_2	232565	3600.00	11406	120126	50.84	0.29	119947	50.92	1.00
3X20_3	34370	3600.00	8058	15877	70.28	0.20	15877	70.28	1.00
3X20_4	157437	3600.00	12156	133517	16.46	0.16	133213	16.67	1.01
3X20_5	298921	3600.00	10470	109031	65.83	0.32	109031	65.83	1.00
5X20_1	248468	3600.00	19956	112509	59.50	0.41	111518	59.93	1.01
5X20_2	29463	3600.00	10094	26839	13.55	0.15	26839	13.55	1.00
5X20_3	144960	3600.00	16530	73370	55.74	0.22	73370	55.74	1.00
5X20_4	164852	3600.00	14691	105378	39.61	0.25	102785	41.33	1.04
5X20_5	179782	3600.00	15168	57457	74.31	0.19	58338	73.78	0.99

Table 7: Results for first problem variant and instances with 20 jobs

		MIP				Heuristic			
Instance	**Start**	**CPU**	**LB**	**UB**	**GAP**	**CPU**	**UB**	**GAP**	**Ratio**
4X30_1	383922	3600.00	26344	289596	26.38	0.59	272091	31.27	1.19
4X30_2	474658	3600.00	18793	258057	47.51	0.72	211934	57.63	1.21
4X30_3	415739	3600.00	15754	214323	50.36	0.79	186776	57.24	1.14
4X30_4	585123	3600.00	33976	473219	20.30	0.74	467601	21.32	1.05
4X30_5	182577	3600.00	24333	120212	39.41	0.43	120016	39.53	1.00
5X30_1	473535	3600.00	20528	148943	71.65	0.86	128240	76.22	1.06
5X30_2	597635	3600.00	15373	253104	59.17	1.57	184523	70.95	1.20
5X30_3	138608	3600.00	18796	89143	41.29	1.31	89142	41.29	1.00
5X30_4	295655	3600.00	18512	193734	36.78	0.97	167339	46.30	1.26
5X30_5	370676	3600.00	17283	241883	36.44	1.30	201250	47.94	1.32

Table 8: Results for first problem variant and instances with 30 jobs

		MIP				Heuristic			
Instance	**Start**	**CPU**	**LB**	**UB**	**GAP**	**CPU**	**UB**	**GAP**	**Ratio**
4X40_1	448516	3600.00	16659	333716	26.58	2.43	204819	56.43	2.12
4X40_2	1087748	3600.00	25024	718270	34.77	1.93	517972	53.61	1.54
4X40_3	907995	3600.00	24433	594310	35.50	3.25	415286	55.76	1.57
4X40_4	737379	3600.00	24862	508421	32.13	1.84	328195	57.43	1.79
4X40_5	427109	3600.00	14346	395646	7.62	2.17	291773	32.79	4.30
5X40_1	541520	3600.00	21595	335475	39.63	4.39	266353	52.92	1.34
5X40_2	477325	3600.00	16843	383820	20.31	3.39	314847	35.28	1.74
5X40_3	542540	3600.00	30204	427408	22.47	2.39	371936	33.30	1.48
5X40_4	584405	3600.00	31807	398988	33.55	2.42	308586	49.91	1.49
5X40_5	854281	3600.00	24364	461592	47.32	1.90	346844	61.14	1.29

Table 9: Results for first problem variant and instances with 40 jobs

Instance	Start	MIP				Heuristic			Ratio
		CPU	LB	UB	GAP	CPU	UB	GAP	
4X50_1	1292112	3600.00	18109	1076345	16.94	6.60	564684	57.10	3.37
4X50_2	1545375	3600.00	27224	1233110	20.57	4.04	746021	52.65	2.56
4X50_3	1299467	3600.00	28791	924507	29.51	2.78	459813	66.08	2.24
4X50_4	240777	3600.00	13388	160716	35.21	3.55	74144	73.28	2.08
4X50_5	238753	3600.00	18431	227110	5.28	4.37	186472	23.73	4.49
5X50_1	1262438	3600.00	35471	1098889	13.33	8.79	559605	57.28	4.30
5X50_2	1112018	3600.00	27386	864738	22.80	3.82	462299	59.90	2.63
5X50_3	1044173	3600.00	26540	922890	11.92	4.67	574443	46.16	3.87
5X50_4	1307897	3600.00	28234	1211302	7.55	6.50	841724	36.43	4.83
5X50_5	379492	3600.00	15706	316076	17.43	5.64	239979	38.35	2.20

Table 10: Results for first problem variant and instances with 50 jobs

Instance	Start	MIP				Heuristic			Ratio
		CPU	LB	UB	GAP	CPU	UB	GAP	
2X8_1	34385	6.39	18877	18877	100.00	0.15	18877	100.00	1.00
2X8_2	56837	21.87	44955	44955	100.00	0.18	44955	100.00	1.00
2X8_3	48070	12.36	31517	31517	100.00	0.32	31517	100.00	1.00
2X8_4	18440	16.98	17980	17980	100.00	0.14	17980	100.00	1.00
2X8_5	12663	1.84	10976	10976	100.00	0.10	10976	100.00	1.00
3X8_1	31187	13.51	19838	19838	100.00	0.15	19838	100.00	1.00
3X8_2	13094	12.69	10206	10206	100.00	0.08	10206	100.00	1.00
3X8_3	23474	16.38	13428	13428	100.00	0.17	13428	100.00	1.00
3X8_4	72688	61.82	47171	47171	100.00	0.24	47171	100.00	1.00
3X8_5	26070	12.48	13832	13832	100.00	0.16	13832	100.00	1.00

Table 11: Results for second problem variant and instances with 8 jobs

Instance	Start	MIP				Heuristic			Ratio
		CPU	LB	UB	GAP	CPU	UB	GAP	
2X10_1	47382	1157.66	32315	32315	100.00	0.34	32315	100.00	1.00
2X10_2	38918	383.51	32885	32885	100.00	0.26	32885	100.00	1.00
2X10_3	68849	3600.00	37226	66365	7.86	0.80	66365	7.86	1.00
2X10_4	15264	120.43	13267	13267	100.00	0.29	13267	100.00	1.00
2X10_5	41238	303.09	13912	13912	100.00	0.38	13912	100.00	1.00
3X10_1	76089	2812.38	71879	71879	100.00	0.35	71879	100.00	1.00
3X10_2	2474	0.01	2474	2474	100.00	0.23	2474	100.00	1.00
3X10_3	34244	30.33	20324	20324	100.00	0.27	20324	100.00	1.00
3X10_4	104453	3189.47	62017	62017	100.00	0.38	62017	100.00	1.00
3X10_5	11134	33.63	11134	11134	100.00	0.33	11134	100.00	1.00

Table 12: Results for second problem variant and instances with 10 jobs

Instance	Start	MIP				Heuristic			Ratio
		CPU	LB	UB	GAP	CPU	UB	GAP	
2X12_1	25118	3600.00	15428	25098	0.21	0.34	25098	0.21	1.00
2X12_2	47963	3600.00	23240	39400	34.63	0.59	39400	34.63	1.00
2X12_3	35935	3600.00	11157	28914	28.34	0.54	28914	28.34	1.00
2X12_4	178945	3600.00	31407	121275	39.09	0.75	121275	39.09	1.00
2X12_5	91028	3600.00	16838	56634	46.36	0.41	56634	46.36	1.00
3X12_1	43703	860.41	14311	14311	100.00	0.63	14311	100.00	1.00
3X12_2	93588	3600.00	7509	45907	55.39	0.99	45907	55.39	1.00
3X12_3	52192	3600.00	17570	37753	41.71	0.59	37753	41.71	1.00
3X12_4	99820	3600.00	29121	93425	9.05	0.78	93425	9.05	1.00
3X12_5	85266	3600.00	20168	61906	35.88	0.56	61906	35.88	1.00

Table 13: Results for second problem variant and instances with 12 jobs

Instance	Start	MIP				Heuristic			Ratio
		CPU	LB	UB	GAP	CPU	UB	GAP	
2X15_1	184932	3600.00	17137	137271	28.40	1.81	136365	28.94	1.02
2X15_2	140058	3600.00	15296	85294	43.89	1.86	85294	43.89	1.00
2X15_3	32087	3600.00	15298	19723	73.65	1.17	19723	73.65	1.00
2X15_4	99821	3600.00	18892	79116	25.58	1.01	77968	27.00	1.06
2X15_5	58282	3600.00	11577	38408	42.55	0.98	38408	42.55	1.00
3X15_1	71363	3600.00	13672	61801	16.57	1.29	61801	16.57	1.00
3X15_2	139214	3600.00	14527	69009	56.30	1.13	68600	56.63	1.01
3X15_3	91090	3600.00	16869	46056	60.68	1.24	46038	60.70	1.00
3X15_4	129206	3600.00	15055	97339	27.92	1.42	97339	27.92	1.00
3X15_5	120990	3600.00	11004	59843	55.60	1.43	59605	55.81	1.00

Table 14: Results for second problem variant and instances with 15 jobs

Instance	Start	MIP				Heuristic			Ratio
		CPU	LB	UB	GAP	CPU	UB	GAP	
3X20_1	213859	3600.00	12198	121088	46.00	3.10	113884	49.58	1.08
3X20_2	232565	3600.00	11417	120211	50.80	4.11	119947	50.92	1.00
3X20_3	34370	3600.00	5351	16039	63.17	2.81	15877	63.73	1.01
3X20_4	157437	3600.00	12304	136330	14.54	2.94	133213	16.69	1.15
3X20_5	298921	3600.00	12262	111187	65.49	5.86	110890	65.59	1.00
5X20_1	248468	3600.00	19786	118413	56.87	3.96	113190	59.16	1.04
5X20_2	29463	3600.00	6501	26926	11.05	2.58	26839	11.43	1.03
5X20_3	144960	3600.00	16523	74693	54.71	3.11	73370	55.74	1.02
5X20_4	164852	3600.00	14034	103904	40.41	3.02	103848	40.45	1.00
5X20_5	179782	3600.00	13033	59238	72.29	3.91	57457	73.36	1.01

Table 15: Results for second problem variant and instances with 20 jobs

		MIP				Heuristic			
Instance	**Start**	**CPU**	**LB**	**UB**	**GAP**	**CPU**	**UB**	**GAP**	**Ratio**
4X30_1	383922	3600.00	28934	291619	26.00	13.01	272091	31.50	1.21
4X30_2	474658	3600.00	18904	267758	45.40	15.29	211934	57.65	1.27
4X30_3	415739	3600.00	13333	203917	52.64	17.44	186776	56.90	1.08
4X30_4	585123	3600.00	34416	478201	19.42	36.11	472810	20.39	1.05
4X30_5	182577	3600.00	28876	113325	45.06	13.22	112537	45.57	1.01
5X30_1	473535	3600.00	20542	157234	69.82	15.39	128240	76.23	1.09
5X30_2	597635	3600.00	15380	278580	54.80	27.95	173973	72.76	1.33
5X30_3	138608	3600.00	9742	91926	36.23	11.04	89142	38.39	1.06
5X30_4	295655	3600.00	21088	238879	20.68	18.29	167339	46.73	2.26
5X30_5	370676	3600.00	15848	199211	48.32	17.19	201250	47.75	0.99

Table 16: Results for second problem variant and instances with 30 jobs

		MIP				Heuristic			
Instance	**Start**	**CPU**	**LB**	**UB**	**GAP**	**CPU**	**UB**	**GAP**	**Ratio**
4X40_1	448516	3600.00	16753	260164	43.62	64.88	193123	59.15	1.36
4X40_2	1087748	3600.00	38729	631518	43.49	66.47	516971	54.41	1.25
4X40_3	907995	3600.00	28181	596225	35.44	80.53	415286	56.00	1.58
4X40_4	737379	3600.00	36086	401698	47.87	44.37	327955	58.38	1.22
4X40_5	427109	3600.00	23009	379259	11.84	49.44	291773	33.49	2.83
5X40_1	541520	3600.00	24964	414032	24.68	54.26	274822	51.63	2.09
5X40_2	477325	3600.00	18291	397658	17.36	82.36	314847	35.40	2.04
5X40_3	542540	3600.00	38382	443310	19.68	53.21	371825	33.86	1.72
5X40_4	584405	3600.00	32707	515776	12.44	61.66	308586	49.99	4.02
5X40_5	854281	3600.00	25470	545737	37.23	59.87	340627	61.97	1.66

Table 17: Results for second problem variant and instances with 40 jobs

		MIP				Heuristic			
Instance	**Start**	**CPU**	**LB**	**UB**	**GAP**	**CPU**	**UB**	**GAP**	**Ratio**
4X50_1	1292112	3600.00	20559	937265	27.91	185.03	564684	57.21	2.05
4X50_2	1545375	3600.00	28124	1110273	28.68	290.58	725538	54.03	1.88
4X50_3	1299467	3600.00	28136	921995	29.69	178.34	459813	66.05	2.22
4X50_4	240777	3600.00	1268	143026	40.81	108.90	67224	72.46	1.78
4X50_5	238753	3600.00	0	217513	8.90	79.78	173330	27.40	3.08
5X50_1	1262438	3600.00	38171	921350	27.86	214.52	547790	58.37	2.10
5X50_2	1112018	3600.00	17104	813778	27.24	103.29	450561	60.41	2.22
5X50_3	1044173	3600.00	27440	873744	16.76	194.33	571162	46.52	2.78
5X50_4	1307897	3600.00	31732	1246993	4.77	205.41	824250	37.90	7.94
5X50_5	379492	3600.00	13532	315834	17.39	74.48	239979	38.12	2.19

Table 18: Results for second problem variant and instances with 50 jobs