Hoyningen-Huene, Wiebke von

**Working Paper**

# Heuristics for an Integrated Maintenance and Production Scheduling Problem on Parallel Machines with Stochastic Failures and Non-Resumable Jobs

Arbeitspapiere des Instituts für Betriebswirtschaftslehre der Universität Kiel

# Heuristics for an Integrated Maintenance and Production Scheduling Problem on Parallel Machines with Stochastic Failures and Non-Resumable Jobs

*W. von Hoyningen-Huene*

**Abstract** Maintenance activities can have a large impact on the performance of a production process and promised delivery dates. Therefore, maintenance and production planning should be handled as an integrated problem. In this paper, the problem of scheduling $n$ jobs on $m$ identical parallel machines is solved such that the expected makespan is minimised. The machines are affected by stochastic machine failures which are assumed to result in long production stops. To avoid this, preventive maintenance activities are planned beforehand. If a failure cannot be averted, a corrective maintenance has to be performed. Furthermore, it is assumed that jobs, interrupted by machine failure, have to be repeated (non-resumable case). In this paper, three construction heuristics and an Ant Colony System (*ACS*) algorithm are developed for solving this integrated problem. The excellent performance of the *ACS* as well as the benefit solving the problem in an integrated manner instead of sequentially, is shown in a numerical study.

**Keywords:** Parallel Machine Scheduling, Maintenance, Stochastic Failures, Non-Resumable Jobs, Ant Colony System

# 1 Introduction and Literature Review

The integration of production and maintenance scheduling is a rather new research field, especially when it is assumed that the unavailability periods of a machine are not necessarily known in advance. Since in practice, unexpected machine failures and subsequent maintenance activities can have a considerable impact on the performance of a production process, a jointly planning of production and maintenance operations is reasonable. To avoid machine failures, preventive maintenance (PM) activities have to be planned in advance. During PM activities, a machine is unavailable for production. However, if machine failures occur, these failures require unplanned corrective maintenance (CM) activities which cause an even longer downtime of the machine. Thus, the planning of maintenance highly influences the completion times of the jobs. This is intensified if the jobs are non-resumable and therewith have to be repeated, if a machine fails during the production of a job. The non-resumable case can for example be observed in pharmaceutical industries or in industrial bakeries. The occurrence of a failure during the processing of sensitive ingredients causes that the semi-finished product is wasted and the job has to be restarted after the CM activity. Therefore, it is advisable to plan production and maintenance aspects coherently to avoid failures or to keep the probability of a failure low at the end of large jobs to prevent long repetitions.

A lot of research has been done on production scheduling and machine maintenance problems independently of one another. Representative overviews are Pinedo (1995) for production scheduling problems and Wang (2002) for a survey of maintenance optimisation models. Specialised on parallel machine scheduling problems, Mokotoff (2001) is a representative survey for deterministic problems. However, since few years the integration of these two fields attains an increasing attention, but with huge differences concerning the manner of modelling.

Some papers incorporate maintenance decisions in scheduling problems by using availability constraints. That means that periods of unavailability can be caused by PM activities or other events, but are known ‘a priori’ and are fixed in time. Overviews of such deterministic problems are provided by Lee (1996), Sanlaville and Schmidt (1998), Schmidt (2000) and Ma et al.

(2010). Another idea is to allow the starting times of unavailability periods to be flexible in a given time interval. Examples for this research field are: Qi et al. (1999), where multiple unavailability periods have to be planned; Graves and Lee (1999), which consider semi-resumable jobs on a single machine; Lee and Chen (2000), where parallel machines are considered and one unavailability period on each machine is assumed; Chen (2008) and Xu et al. (2009), which both assume the existence of an earliest starting time and a latest ending time for the unavailability periods and propose a heuristic based on the First Fit Decreasing approach to solve the problem in polynomial time.

Though all these contributions consider maintenance activities, they ignore the probability of unexpected machine failures. In contrast, Adiri et al. (1989) expect a single stochastic failure on a single machine, but do not regard the impact of PM activities. They proof that the Longest Processing Time (LPT) priority rule minimises the makespan if an Increasing Failure Rate (IFR) is assumed. Another handling of stochastic unavailability can be found in Berrichi et al. (2009). They include PM activities into the production schedule after a time interval while the makespan is minimised. Instead of assuming CM activities, a second objective function is generated to minimise the unavailability. They solve the problem with Genetic Algorithms and with an Ant Colony Optimisation approach in a later article (Berrichi et al. (2010)).

The integrated problem of scheduling jobs as well as corrective and preventive maintenance on machines when failures can occur stochastically is seldom discussed. Cassady and Kutanoglu (2003) present a model for the integrative problem with resumable jobs, minimising the weighted job tardiness on a single machine. They show that solving the job and the maintenance scheduling problem simultaneously outperforms the sequential consideration of these planning tasks. Another model is investigated in Cassady and Kutanoglu (2005), with the aim of finding integrated schedules while minimising the total weighted completion time. This model was solved in Sortrakul et al. (2005) by Genetic Algorithms and in Leng et al. (2006) by a Chaotic Particle Swarm Algorithm and was extended in Yulan et al. (2008) by further objective functions. Kuo and Chang (2007) prove analytically the optimality of several policies for the same problem as Cassady and Kutanoglu (2003) while assuming that the machine becomes ‘as good

as new' after a repair. In a recent article Cui et al. (2014) present an integrated model with similar assumptions as Cassady and Kutanoglu (2003), maximising the quality and solution robustness through the insertion of buffer times. They solve their problem with a three-phase heuristic. Other articles which are based on the model of Cassady and Kutanoglu (2003) integrate quality control with maintenance and production scheduling decisions, see for example Pandey et al. (2010) and Pandey et al. (2011). A survey regarding partial and complete integration of maintenance and production scheduling along with quality control is provided by Hadidi et al. (2012).

Only few articles address the non-resumable case regarding the integrated problem with stochastic failures. Examples are the paper of Adiri et al. (1989), which was mentioned before, and Lee and Lin (2001), where a similar problem is discussed as in the former. They schedule jobs on a single machine with the opportunity to insert a single PM activity and have modified job lengths after a maintenance activity.

In the papers mentioned above, delayed preventive maintenance activities do not result in breakdowns and the job sequence between two consecutive PM activities is of minor interest. In contrast, von Hoyningen-Huene and Kiesmüller (2015) developed a model for the integrated problem of scheduling maintenance activities and jobs on parallel identical machines under the assumption of non-resumable jobs. The objective function is to minimise the expected makespan. They consider stochastic failures and assume an IFR. As a consequence, corrective as well as preventive maintenance activities are performed. They found that already the evaluation of a solution by the exact objective function is a hard problem. Therefore, they developed two approximations for the objective function. Since until now no solution approaches are developed for their model, in this paper, three new construction heuristics and a problem-specific Ant Colony System (*ACS*) meta-heuristic are presented.

The paper is organised as follows: In the next section the optimisation model is introduced together with one of the approximated expressions for the objective function introduced by von Hoyningen-Huene and Kiesmüller (2015). In Section 3 the three construction heuristics and the

*ACS* algorithm are described. The results of a numerical study are presented in Section 4. The paper ends with a summary of the observed results.

## 2 Model and Approximated Objective Function

The problem is to allocate and sequence $n$ jobs on $m$ identical parallel machines. The processing time $p_j$ of a job $j$ is known, constant and independent from the allocated machine. Jobs cannot be split to different machines. All jobs are available at time zero and all machines start at the 'as good as new' state. With ongoing production the machines are assumed to wear out, such that the time to failure ($T$) of a new machine is modelled in an IFR mode. If a breakdown occurs, the interrupted job cannot be continued, but has to be processed from the beginning (non-resumable case).

The considered maintenance activities are preventive as well as corrective. The PM activities are planned concurrent to the production schedule and are not allowed to interrupt a job (non-preemptive case). PM activities restore the machine to be 'as good as new' after a service time $t_\mathrm{p}$. Since the problem takes place at the operational level, the existence of a frozen period is included in the scheduling process. That means, the schedule is determined before the production begins and the sequence of jobs and PM activities is not changed during the production process. In order to find the best sequence for the frozen period, the possibility of failures is considered as part of the problem. After a failure occurred, CM activities are performed to replace broken parts and restore the machine to become 'as good as new'. The interrupted job has to restart right after the repair and it is assumed that the machine does not fail again while repeating this job. A justification of this assumption conducted by a simulation study is done by von Hoyningen-Huene and Kiesmüller (2015). The duration $t_\mathrm{c}$ of a CM is assumed to be longer than $t_\mathrm{p}$. This leads to the main decision between risk avoiding PM or unplanned CM activities accompanied by many restarted jobs.

Summarised, the problem is to allocate $n$ jobs to $m$ machines, sequence these jobs and plan a variable amount of PM activities in order to minimise the expected makespan. The expected makespan is defined as the expected completion time of the latest finished job regarding all machines: $E(Cmax) = \max_l(\max_i(E(C_{il})))$ with $E(C_{il})$ being the expected completion time of the job on position $i$ (the $i$-th job) in the schedule of machine $l$. To model the formulated problem, the decision variables $x_{jil}$ and $y_{il}$ are defined. The variable $x_{jil}$ specifies the allocation of each job to a machine:

$$x_{jil} = \begin{cases} 1 & \text{if job } j \text{ is scheduled at the} \\ & i\text{-th position on machine } l \\ & \qquad\qquad j,\, i \;= 1,..,n,\, l = 1,..,m \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

To insert the preventive maintenance activities into the schedule, $y_{il}$ is defined as:

$$y_{il} = \begin{cases} 1 & \text{if a PM is performed just} \\ & \text{before the } i\text{-th job on machine } l \\ & \qquad\qquad i = 1,..,n,\, l = 1,..,m \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Let $n_l$ be the number of jobs on machine $l$:

$$n_l = \sum_{j=1}^{n} \sum_{i=1}^{n} x_{jil} \qquad\qquad l = 1,..,m \tag{3}$$

with $\sum_{l=1}^{m} n_l = n$. The expected makespan turns out to be

$$E(Cmax) = \max_l(E(C_{n_l l})). \tag{4}$$

The corresponding mathematical model for the problem is formulated as follows:

$$\text{Minimise} \quad \max_{l}(E(C_{n_l l})) \tag{5}$$

subject to:

$$\sum_{i=1}^{n}\sum_{l=1}^{m} x_{jil} = 1 \qquad j = 1,..,n \tag{6}$$

$$\sum_{j=1}^{n} x_{jil} \leq 1 \qquad l = 1,..,m,\ i = 1,..,n \tag{7}$$

$$\sum_{j=1}^{n} x_{jil} \leq \sum_{j=1}^{n} x_{ji-1,l} \qquad l = 1,..,m,\ i = 2,..,n \tag{8}$$

$$y_{il} \leq \sum_{j=1}^{n} x_{jil} \qquad l = 1,..,m,\ i = 1,..,n \tag{9}$$

$$y_{il} \in \{0,1\} \qquad l = 1,..,m,\ i = 1,..,n \tag{10}$$

$$x_{jil} \in \{0,1\} \qquad i,\ j = 1,..,n,\ l = 1,..,m \tag{11}$$

Equation (6) implies each job $j$ to be scheduled exactly once. Constraint (7) assigns a position on a machine to a job at most once. Since on each machine theoretically $n$ jobs could be scheduled, this constraint allows empty positions on a machine. Constraint (8) further assures that there is no idle time on the machines. (9) permits to schedule PM activities just in front of scheduled jobs.

Note, that a calculation of the objective function (5) is highly complex. The exact computation of the expected makespan is explained in von Hoyningen-Huene and Kiesmüller (2015) together with approximations for the expected makespan. One of these approximated objective functions ($E(Cmax^{\text{pol}})$) can be computed in polynomial time and leads to good results close to the exact calculation. Therefore it is used in this paper as part of the *ACS* and is explained in the following.

The approximated expected completion time of a machine $l$ is composed of four essential parts:

1. *the processing times of jobs allocated to the machine*
2. *the time units spent for the scheduled PM activities*
3. *the approximated expected time units spent for the CM activities*
4. *the approximated expected time units for the jobs which have to get processed twice due to job interruption*.

The first two parts are the same as for the exact calculation and are presented here briefly. The *first* term contains the sum of processing times on a machine $l = 1,..,m$:

$$\sum_{i=1}^{n_l} p_{il} \tag{12}$$

with $p_{il}$ as the processing time of the $i$-th job on machine $l$:

$$p_{il} = \sum_{j=1}^{n} x_{jil} \cdot p_j \qquad\qquad i = 1,..,n. \tag{13}$$

The *second* part, namely the time units spent for the PM activities, is written as

$$t_{\mathrm{p}} \sum_{i=1}^{n_l} y_{il}. \tag{14}$$

The *third* term, namely the expected time units spent for CM activities, is quite complex due to stochastic failures. The expected time for CM activities depends on the expected number of failures during the schedule. For an exact calculation, the failure probability of each job in every conceivable breakdown scenario has to be considered. The used approximation (*pol*) simplifies this by summing up the failure probability for each job, just assuming that the machine does not fail before this specific job. It results in an approximation for the number of expected failures

during the whole schedule. The *third* term is then:

$$t_c \sum_{i=1}^{n_l} F(p_{il} + z_{il} | z_{il}) \tag{15}$$

with $F(t)$ as the cumulative distribution function of $T$ and $z_{il}$ as the elapsed time since the end of the last maintenance activity up to the start of the job at the $i$-th position on machine $l$. Without any further restrictions, this would lead to higher failure probabilities for jobs at the end of the schedule than they are expected to have. For example in a scenario where previous jobs might fail with high probability, in the exact calculation an expected restart would lead to low failure probabilities for the subsequent jobs. In order to regard a restart in the approximation, the authors include an artificial renewal of a machine in the calculation of $z_{il}$:

$$z_{il} = (z_{i-1,l} \cdot g_{il} + p_{i-1,l}) \cdot (1 - y_{il}) \qquad i = 2,..,n_l \tag{16}$$

with

$$g_{il} = \begin{cases} 1 & \text{if } F(p_{i-1,l} + z_{i-1,l} | z_{i-1,l}) < 0.5 \qquad i = 2,..,n_l \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

and $z_{1l} = 0$. Then, if the conditional probability of a failure during the production of the $(i-1)$-th job is smaller than 0.5, the starting time of the $i$-th job, since the last maintenance, is composed of the starting and processing time of its direct predecessor. Otherwise, if the occurrence of a failure is very likely during the $(i-1)$-th job, a repair is assumed. The machine becomes 'as good as new' and the starting time of the $i$-th job is determined by the processing time of its direct predecessor. Additionally, if a PM activity is scheduled in front of the $i$-th job ($y_{il} = 1$), its starting time becomes zero.

The *fourth* term missing to determine the approximated expected makespan is the sum of expected time units of the interrupted jobs, which have to get processed twice. Let $R_{n_l}$ be the approximated expected number of repeated processing time units after $n_l$ jobs in the schedule of machine $l$. Since repeated time units just appear if a job is interrupted by a machine breakdown, the approximated expected number of repeated processing time units is derived as:

$$E(R_{n_l}) = \sum_{i=1}^{n_l} \int_{z_{il}}^{p_{il}+z_{il}} (u - z_{il}) \frac{f(u)}{F(p_{il}+z_{il}) - F(z_{il})} du \cdot F(p_{il}+z_{il}|z_{il}). \tag{18}$$

The expected repeated processing time units per job are weighted by the probability of a breakdown, again without having to incorporate every possible breakdown scenario and every possible number of failures.

The approximation (*pol*) for the expected makespan ($E(Cmax^{\text{pol}})$) is then generated by the approximated expected completion time of a machine $l = 1,..,m$, summing up all four terms (equations (12), (14), (15) and (18)):

$$E(C^{\text{pol}}_{n_l l}) = \sum_{i=1}^{n_l} p_{il} + t_{\text{p}} \sum_{i=1}^{n_l} y_{il} + t_{\text{c}} \sum_{i=1}^{n_l} F(p_{il}+z_{il}|z_{il}) + E(R_{n_l}). \tag{19}$$

## 3 Solution Approaches

It is known that already the classical parallel machine problem minimising the makespan ($Pm||Cmax$) is NP-hard (see Garey and Johnson (1979)). Therefore, we are looking for approaches solving the introduced problem heuristically. In this section, three new construction approaches ($List\&tau$, $Matching\ Batches$, $Batchfit$) are presented as well as an *ACS* algorithm.

### 3.1 List&tau

The first heuristic is a sequential approach, solving the production and the maintenance problem separately. In the first part of this approach, a simple heuristic is applied, which was made for the basic parallel machine problem. In the second part, a cost function is used to determine an interval length, declaring the maximum production time without a PM activity.
The heuristic of the first part is a list scheduling approach using the LPT rule of Graham (1969). The jobs are listed in non-increasing order of their processing times and assigned one after another to the machine with the least workload. Expected machine breakdowns are not considered. After this assignment a fixed PM interval is determined by the second part of the sequential approach. This time interval $\tau$ determines the maximum production time between two consecutive PM activities. $\tau$ is derived from a cost function which is often used in maintenance literature (see for example Barlow and Hunter (1960), Bosch and Jensen (1983), Beichelt (1993), Brandolese et al. (1996)). Since time aspects are of interest here, the modified function sums up the expected time spent for a preventive maintenance activity and the expected time spent for a corrective one. The 'cost' per time unit are:

$$C(\tau) = \frac{t_c \cdot F(\tau) + t_p \cdot (1 - F(\tau))}{\int_0^\tau (1 - F(t))dt}. \tag{20}$$

After deriving $\tau$ by minimising (20), PM activities are added to the previously developed production schedule. In detail, on each machine a PM is inserted in front of a job which processing time increases the cumulated processing time of scheduled jobs above $\tau$. The *List&tau* algorithm is summarised as follows:

***Listτ* algorithm**

Initialise parameters

**Step 1.** Assign $n$ jobs to the $m$ machines by LPT rule

**Step 2.** Determine $\tau$ by (20)

**For** each machine **do**

**Step 3.** Schedule PM in front of job that increases cumulated processing time above $\tau$

**End**

**Step 4.** Evaluate schedule with $E(Cmax)$

The runtime of the original list approach using LPT is bounded by $O(n\log n + nm)$ (see Graham (1969)) for sequencing the jobs and assigning them to the machines. Additionally, for the problem investigated here the placement of a PM activity is examined after each job $O(n)$, which still results in a runtime of $O(n\log n + nm)$.

Since a sequential solution of the production and the maintenance part with the help of a predetermined PM interval is often found in practice and also common in literature, this approach is supposed to serve as a benchmark for approaches which solve the two problems in cooperation with each other.

### 3.2 Matching Batches (MB)

The second heuristic tries to better balance the jobs and PM activities among the machines by determining the number of PM intervals in a first step. These intervals are named batches in the following. Every batch is bounded by two PM activities except at the beginning respectively the end of a machine schedule. The number of overall batches is calculated by:

$$B = \min\left(n, \left\lceil \frac{\frac{\sum_{j=1}^{n} p_j}{m}}{\tau} \right\rceil \cdot m\right) \tag{21}$$

with $\frac{\sum_{j=1}^{n} p_j}{m}$ as a Lower Bound (LB) to the expected makespan. The LB is the optimal solution for the basic parallel machine problem without considering breakdowns and maintenance but allowing preemption (see for example Mokotoff (2001)). Dividing this LB by $\tau$ and rounding up the result yields an approximated number of batches per machine. Multiplying this value by the number of machines gives the overall number of batches in (21).

Afterwards these batches are filled with jobs by using the previously stated list approach. Notice that $\tau$ is not serving as a batch limit anymore. After having assigned the jobs to the batches, these batches are assigned to the machines, again by using the idea of the list approach, taking the expected length of a batch as a job. PM activities are inserted between the scheduled batches. The *Matching Batches* algorithm:

| ***Matching Batches* algorithm** |
|---|
| Initialise parameters |
| **Step 1.** Determine number of batches $B$ by (21) |
| **Step 2.** Assign $n$ jobs to the batches by LPT rule |
| **Step 3.** Evaluate the length of batches with $E(Cmax)$ |
| **Step 4.** Assign $B$ batches to the machines by LPT rule and schedule PMs between them |

The runtime of the $MB$ approach is bounded by $O(n\log n+nB+B\log B+Bm)$ for sequencing jobs, scheduling them on $B$ batches, sorting these batches and finally assigning them to the machines.

### 3.3 Batchfit (BF)

The performance of the $MB$ heuristic depends on the quality of the estimated number of batches $B$. The $Batchfit$ approach takes therefore the idea of solving the problem several times, but using adjusted values for $B$ in every iteration. The number of scheduled batches is iteratively adjusted by orientating the expected lengths of the batches towards $\tau$ derived from (20). The

rest of the heuristic operates similar to the heuristic *MB*. At the beginning, the *MB* approach is run with the number of batches set to $B$ as given by (21). The number of batches in the next iteration is determined by decreasing respectively increasing it by the number of machines, meaning either one additional batch per machine or one less. If the generated solution results in batches, at least one having a length above $\tau$, the number of batches is increased in the next iteration, but maximally to $n$. Otherwise $B$ is decreased, but no further than down to $m$. This procedure is repeated until a lower and an upper limit are found, regarding the number of batches, which fulfil the following conditions: The lower limit of batches is denoted by $B_{min} = \hat{B}$ and have to result in a solution where at least one batch length is exceeding $\tau$. $B_{min}$ have to be at least equal to $m$ batches. The upper limit must have maximally $m$ additional batches, such that $B_{max} = min(\hat{B}+m, n)$, and have to result in a solution with none batch length exceeding $\tau$. In the next iterations $B_{max}$ is decremented in every step until either the number of batches equals $B_{min}$ or at least one batch has a length above $\tau$ or the maximum number of steps ($K$) is reached. $K$ is calculated by:

$$K = m + \left\lceil \frac{High - Low}{m} \right\rceil \tag{22}$$

with

$$Low = \min \left( n, \left\lfloor \frac{\frac{\sum_{j=1}^{n} p_j}{m}}{\tau} \right\rfloor \cdot m \right) \tag{23}$$

and

$$High = \min \left( n, \left\lceil \frac{\frac{2 \cdot \sum_{j=1}^{n} p_j}{m}}{\tau} \right\rceil \cdot m \right). \tag{24}$$

The *Batchfit* approach:

***Batchfit* algorithm**

Initialise parameters

**Step 1.** Run *MB* algorithm and set the iteration counter $k = 1$

**Repeat**

**Step 2. If** any length of a batch $> \tau$ **then**

$B = min(B+m, n)$

**else**

$B = max(B-m, m)$

**Step 3.** Run Step 2. - 4. of *MB* algorithm and set $k = k+1$

**Until** a solution with $B_{min} = \hat{B}$ (but at least $m$) batches has at least one with length $> \tau$ **and** a solution with $B_{max} = min(\hat{B}+m, n)$ batches has non length $> \tau$

**Step 4.** Set $B = B_{max} - 1$

**Repeat**

**Step 5.** Run Step 2. - 4. of *MB* algorithm and set $k = k+1$

**Step 6.** Set $B = B-1$

**Until** $B = B_{min}$ **or** a solution has at least one batch with length $> \tau$ **or** $k = K$

**Step 7.** Choose best found solution

Since the *BF* approach runs the *MB* heuristic up to $K$ times, the runtime is bounded by $O(n \log n + K(nB + B \log B + Bm))$.

### 3.4 Ant Colony System

The Ant Colony (AC) algorithm was introduced by Dorigo et al. (1996). The idea of this meta-heuristic is inspired by the behaviour of ants while they are searching for food. The ants are assumed to trail different ways to find a food source. If an ant is successful, it returns to the

nest and deposits pheromones on the used track. The shorter the trail an ant chooses, the faster this ant returns. Successive ants orientate themselves by the pheromone value on the tracks, thus using the pheromones as information where to get food. Sooner or later more ants are using the shorter trails.

The AC algorithm creates artificial ants, each representing a potential solution to the considered optimisation problem. The ants orientate themselves by the pheromone values previous ants deposited within the solution process. Additionally, these ants are affected by problem inspired hints (heuristic information). When and how these pheromone values are updated is different for distinct variations of the AC algorithm. One of these variations is the Ant Colony System (*ACS*) heuristic which was introduced by Dorigo and Gambardella (1997). Since an *ACS* algorithm is discussed here, only the implementation of this variant is explained in detail.

The algorithm is orientated towards the idea of the $Batchfit$ approach but uses the advantage of iterative improvement. A first ant colony fills an amount of batches with jobs and a second colony assigns these batches to machines. The *ACS* algorithm is outlined as follows and described in detail in the following subsections:

***ACS* algorithm**

Initialise parameters

**Repeat**

**For** each ant **do**

Set $pre =$ '0'

**Repeat**

**Step 1.** Select and schedule successor $j$ of $pre$ with (27) and (28)

**Step 2.** $pre = j$

**Until** $n$ jobs are scheduled in batches

**Step 3.** Evaluate the length of batches with $E(C^{\mathrm{pol}})$

**Repeat**

**Repeat**

**Step 4.** Select a machine $\psi$ with (30) and (31)

**Step 5.** Select a batch $v$ for machine $\psi$ with (34) and (35)

**Step 6.** Place the batch on the machine and insert PM activity, if necessary

**Until** all batches are scheduled on the machines

**Step 7.** $a_{\mathrm{M}} = a_{\mathrm{M}} - \delta$ from (32)

**Until** $w_{\mathrm{M}}$ ants of the second colony created a solution

**Step 8.** Replace, if necessary, the best found solution so far

**Step 9.** Do local pheromone update with (36)

**End**

**Step 10.** Do global pheromone update with (37)

**Until** the final iteration number is achieved

**Step 11.** Evaluate best found schedule with $E(Cmax)$

**Jobs to Batches**

In this first part of the algorithm, an ant decides in every step which job to schedule next in the current batch until all jobs are scheduled. To do so, without predetermining the number of batches, the set $S \subseteq \{1,..,n\}$ of jobs which are not scheduled yet, is extended by a dummy job '0' ($S_0 = \{\text{'0'}, S\}$) which has no processing time. If job '0' is chosen by the ant, the current batch is closed for further jobs and a new batch is initialised. Otherwise, the chosen job $j$ is placed right after the last scheduled job ($pre$) on the same batch. After each step the new scheduled job $j$ becomes the predecessor $pre$. The couple $(pre, j)$ represents a trail, that the ant uses. At the beginning is $pre =$ '0'. The decision which job $j \in S_0$ is chosen next is determined by the heuristic information ($\eta_{pre,j}$) of the possible trails and the pheromone value ($\xi_{pre,j}$) further ants have placed on these trails. The heuristic information is a problem specific hint for good solutions. We differentiate between the heuristic information of choosing a job $j \in S$ or the job '0'. $\eta_{pre,j}$ with $j \in S$ is determined as follows:

$$\eta_{pre,j} = \left| \min\left(1, \frac{z}{\tau}\right) - \frac{p_j}{p_{\max}} \right| \tag{25}$$

with $z$ as the sum of job lengths already scheduled in the current batch and $p_{\max}$ as the overall maximum job length. This computation prefers long jobs if the batch is rather empty and small jobs if the sum $z$ of processing times of already scheduled jobs in the batch is close to $\tau$. This serves the idea of avoiding long jobs at the end of a batch where the probability of a breakdown is high. The heuristic information of choosing a new batch is determined by:

$$\eta_{pre,0} = \min\left(1, \frac{\max(1, t_c + z + p_{\min} - MTTF) \cdot F(p_{\min} + z|z)}{t_p}\right) \tag{26}$$

with $p_{\min} = \min_{j \in S}(p_j)$ and $MTTF$ as the Mean Time To Failure of the IFR distribution. The minimal approximated penalty of a breakdown, assuming the smallest possible job to be scheduled next, is compared to the insertion of a PM activity and therewith to the initialisation of a new batch. The higher the time of a repair plus the exceedance of the $MTTF$ weighted by

the probability of a breakdown during the smallest selectable job, the higher is the heuristic information for a new batch.

The selection of a successor $j \in S_0$ is then done by a state transition rule called pseudo-random-proportional rule:

$$j = \begin{cases} \arg\max\limits_{s \in S_0}((\xi_{pre,s})^a \cdot (\eta_{pre,s})^b) & \text{if } q \leq q_0 \\ J & \text{otherwise.} \end{cases} \tag{27}$$

This rule integrates exploitation and exploration of the search process depending on a random number $q \in [0,1]$ and $q_0$ which is a preset parameter with $0 \leq q_0 \leq 1$. Parameters $a$ and $b$ weight pheromone value and heuristic information. Every time a job $j$ has to be chosen, a new random number $q$ is drawn. If $q \leq q_0$, the job with the maximal weighted product out of pheromone value and heuristic information is selected to favour exploitation. Otherwise, job $J$ is selected by help of Monte Carlo simulation, where $J$ is a random variable with probability distribution $P_{pre,j}$, favouring exploration:

$$P_{pre,j} = \begin{cases} \frac{(\xi_{pre,j})^a \cdot (\eta_{pre,j})^b}{\sum\limits_{s \in S_0} (\xi_{pre,s})^a \cdot (\eta_{pre,s})^b} & \text{if } j \in S_0 \\ 0 & \text{otherwise.} \end{cases} \tag{28}$$

This decision process is repeated for one ant until all jobs are scheduled in batches. Let $B$ be the number of resulting batches. Each batch is evaluated by its approximated expected completion time given through (19) where $m$ can be interpreted as the number of batches, and $n_l$ as the last job in a batch $l$. This is not a reliable indicator for the evaluation of the current solution, since the pending assignment of batches to machines has a strong impact on the overall solution quality. Therefore, these batches have to be planned on the machines in a further step. To find a good assignment, several loops are conducted for each ant of the first colony, whereby these loops form a second colony of ants (with $w_{\mathrm{M}}$ as the size of the second colony).

**Batches to Machines**

For each ant of the first colony (i.e. each set of batches) an ant of the second colony iteratively chooses a machine to get filled and then the batch that fits best to this machine. After the assignment of $B$ batches to $m$ machines, the next ant of the second colony creates its solution.

In a first step a machine is chosen by the help of the heuristic information $\omega_l$. Since the solutions of the ants of the second colony are different for every ant of the first colony, no pheromone values are used for this assignment problem. The selection of machines and batches is made by heuristic information only:

$$\omega_l = \frac{1}{\max(1, zM_l + yM_l \cdot t_\mathrm{p})} \qquad l = 1,..,m. \tag{29}$$

$zM_l$ is the sum of the expected lengths of the batches already placed on machine $l$. $yM_l$ is the number of PM activities already planned on machine $l$. Thus, machines are favoured, which have less workload so far.

The selection of machines is again done by a pseudo-random-proportional rule with a random number $q_\mathrm{M} \in [0,1]$. Let $q_\mathrm{M0}$ be a preset parameter with $0 \leq q_\mathrm{M0} \leq 1$. Then machine $\psi$ is selected by

$$\psi = \begin{cases} \arg\max\limits_{l=1..m} (\omega_l^{a_\mathrm{M}}) & \text{if } q_\mathrm{M} \leq q_\mathrm{M0} \\ \Psi & \text{otherwise.} \end{cases} \tag{30}$$

If $q_\mathrm{M}$ is less or equal $q_\mathrm{M0}$, the machine with the smallest weighted loading is chosen. This favours exploitation of the desired solution. Otherwise, through exploration of new solutions, machine $\Psi$ is selected by the help of Monte Carlo simulation, where $\Psi$ is a random variable with probability distribution $P_{\mathrm{M}\psi}$:

$$P_{\mathrm{M}\psi} = \frac{\omega_\psi^{a_\mathrm{M}}}{\sum\limits_{l=1}^{m} \omega_l^{a_\mathrm{M}}} \qquad \psi = 1,..,m. \tag{31}$$

To support exploration, the weight of the heuristic information ($a_{\mathrm{M}}$) is changed after each ant of the second colony, which increases the chance for the subordinate machines (respectively batches) to be chosen in every step. This lowers the greedy behaviour of the selection process. At the beginning $a_{\mathrm{M}}$ is set to a parameter $a_{\mathrm{M0}}$ and is reduced by $\delta$ until it ends with the value $a_{\mathrm{Mm}}$:

$$\delta = \frac{a_{\mathrm{M0}} - a_{\mathrm{Mm}}}{w_{\mathrm{M}}}. \tag{32}$$

In a second step a batch is selected to be assigned to the chosen machine $\psi$. The heuristic information for choosing batch $v$ is:

$$\varsigma_v = \frac{c_v}{c_{\max}} \qquad v \in S_{\mathrm{B}} \tag{33}$$

with $c_v$ to be the approximated expected completion time of batch $v$ already evaluated by (19), $c_{\max}$ the longest batch length of all $B$ batches and $S_{\mathrm{B}}$ the set of batches not yet chosen. This heuristic information favours the selection of long batches.

The selection of the batch is done by a pseudo-random-proportional rule with a random number $q_{\mathrm{BM}} \in [0,1]$. Let $q_{\mathrm{BM0}}$ be a parameter with $0 \leq q_{\mathrm{BM0}} \leq 1$. Then batch $v$ is selected by

$$v = \begin{cases} \arg\max\limits_{d \in S_{\mathrm{B}}} (\varsigma_d^{a_{\mathrm{M}}}) & \text{if } q_{\mathrm{BM}} \leq q_{\mathrm{BM0}} \\ V & \text{otherwise.} \end{cases} \tag{34}$$

If $q_{\mathrm{BM}}$ is less or equal $q_{\mathrm{BM0}}$, the batch with highest weighted expected length is chosen. This favours exploitation of the desired solution. Otherwise, by Monte Carlo simulation, batch $V$ is selected favouring exploration, where $V$ is a random variable with probability distribution $P_{\mathrm{BM}v}$:

$$P_{\mathrm{BM}v} = \begin{cases} \frac{\varsigma_v^{a_{\mathrm{M}}}}{\sum\limits_{d \in S_{\mathrm{B}}} \varsigma_d^{a_{\mathrm{M}}}} & \text{if } v \in S_{\mathrm{B}} \\ 0 & \text{otherwise.} \end{cases} \tag{35}$$

After assigning the chosen batch, $zM_l$ is updated, a PM activity is scheduled in front of the batch, except at the beginning of production, along with the increment of $yM_l$. Sequentially, machines are chosen by (29)-(31) and batches are assigned by (33)-(35) until all batches are scheduled. Then, $a_{\mathrm{M}}$ is updated and the next ant of the second colony creates a further schedule out of the same batches. This is repeated $w_{\mathrm{M}}$ times. If a new best solution is found, the best solution so far is replaced.

Before the next ant of the first colony creates its solution, the pheromone concentration on the passed trails $(pre, j)$ of the recent solution are updated by a local pheromone update. The pheromone values of all other trails stay as they are. Since this algorithm is sequential, the following ants of one population orientate towards prior ants. To avoid creating the same solution with succeeding ants, the local updating rule reduces the pheromone values of the used trails. It supports an exploration of the solution space. The local pheromone updating rule is formulated as follows:

$$\xi_{pre,j} = (1-\rho_{\mathrm{l}}) \cdot \xi_{pre,j} + \rho_{\mathrm{l}} \cdot \xi_0 \tag{36}$$

with $\rho_{\mathrm{l}}$ being the local pheromone evaporation rate ($0 \leq \rho_{\mathrm{l}} \leq 1$). All trails $(pre, j)$ have an initial pheromone concentration $\xi_0 = \frac{1}{n}$. To receive a shrinking local pheromone update, an increasing global pheromone update is needed ($\xi_{pre,j} > \xi_0$), which is conducted after all ants of the first colony created a solution. The global pheromone update is performed for the trails $(pre, j)$ which belong to the global best found solution so far. Since, with increasing number of iterations, the search process focuses on favourable solution paths, the global pheromone update supports exploitation. The global update is made according to

$$\xi_{pre,j} = (1-\rho_{\mathrm{g}}) \cdot \xi_{pre,j} + \rho_{\mathrm{g}} \cdot \triangle \tag{37}$$

with $\rho_g$ being the global pheromone evaporation rate ($0 \leq \rho_{\mathrm{g}} \leq 1$) and $\triangle$ as the positive intensification of pheromones. Let $G_{\mathrm{best}}$ be the approximated expected makespan of the recent global

best solution, then through

$$\triangle = \frac{\frac{\sum_{j=1}^{n} p_j}{m}}{G_{\text{best}}} \tag{38}$$

the global pheromone update is adjusted to the problem specific solution space. The amount of additional pheromone is dependent on the solutions quality, favouring small expected makespans.

When the maximum iteration number is achieved, the best found solution is evaluated by the exact objective function.

# 4 Computational Experiments

In this section, a numerical study is conducted to analyse the performance of the introduced heuristics. The heuristics are the decomposition approach *List&tau*, the two heuristics (*MB* and *BF*) based on the determination of the number of batches as well as the *ACS* meta-heuristic. For the *ACS* approach we consider two variants $ACS_{\text{MB}}$ and $ACS_{\text{BF}}$ which use *MB* respectively *BF* to generate starting solutions. These starting solutions are implemented by an initial pheromone update on the used trails according to equation (37). All heuristics are programmed using Matlab R2013a and run on a server with a Core$^{\text{TM}}$ i7-3770 CPU (3.4 GHz). In the next subsections, first the tested numerical examples are introduced as well as the parameter design tested for the *ACS* algorithm. Secondly, the experimental results are presented and discussed.

## 4.1 Numerical Examples and *ACS* Parameters

Ten different sets of processing times were created for each of the following couples $(m,n)$: (2,5), (2,6), (2,8), (2,10), (2,20), (3,20), (3,40), (3,60), (5,20), (5,40), (5,60), (8,40), (8,60). The processing times of the jobs are drawn from a discrete uniform distribution on the interval [1,50]. The time to failure ($T$) of a new machine is modelled as a Weibull distributed random variable, as common for deterioration problems, with shape parameter $\beta$, scale parameter $\alpha$ and

the cumulative distribution function:

$$F(t) = 1 - e^{-(\frac{t}{\alpha})^{\beta}} \qquad t \geq 0 \tag{39}$$

with $\beta > 1$ to ensure an increasing failure rate. Therewith the $MTTF$ in equation (26) becomes

$$MTTF = \alpha \cdot \Gamma\left(1 + \frac{1}{\beta}\right) \tag{40}$$

with $\Gamma$ being the complete Gamma function.
The values of $\beta$ are set to 7 and 8. $\alpha$ is varied between 80, 100, 110 and 120 to gain a compromise between relatively stable machines and therewith realistic settings (high value of $\alpha$) and the ability to show the effect of machine failures also in small problem instances (small value of $\alpha$). Additionally, the time for a preventive maintenance activity is fixed at $t_{\mathrm{p}} = 7$ time units while $t_{\mathrm{c}}$ is set to 10 and 20 time units. All mentioned parameter values are comparable to those of Cassady and Kutanoglu (2003). Altogether 2080 problem instances were investigated.

To determine good parameter values, used for the *ASC*, concerted values were selected and tested. In Table 1 all tested parameter values are listed with the best found values used for all following results. These superior values were determined by varying one parameter in every run, using a representative instance of the experimental design which was $(2,8), \alpha = 80, \beta = 7$, for $t_{\mathrm{c}} \in \{10,20\}$ and for each of the 10 runs, resulting in 20 runs for every parameter combination. It was counted how often the optimal solution, obtained by complete enumeration, was found. The parameter setting with the best performance was chosen. As an example Figure 1 shows the influence of the exponents $b$ and $a$ from (27) on the results. Here, $w$ is set to 10 and $w_{\mathrm{M}}$ to 20, which are the superior sizes for the two ant colonies, while the results are averaged over all other parameter values. Obviously, the solutions seem to get better the smaller $b$. $b$ weights the heuristic information of deciding which job or PM activity to schedule next. Since this heuristic information is $< 1$, we can state that the frequency of optimal solutions using the *ACS* algorithm increases, if the influence of the heuristic information increases, too. It was also found, that a smaller influence of the corresponding pheromone value, i.e. a high value of

Table 1: Superior values of parameters for *ACS*.

| parameter | tested values | superior value |
|---|---|---|
| $w$ | $\{10, 20, 40\}$ | 10 |
| $w_{\mathrm{M}}$ | $\{20, 40, 60\}$ | 20 |
| $a$ | $\{2, 3\}$ | 3 |
| $b$ | $\{0.5, 1, 2\}$ | 0.5 |
| $\rho_{\mathrm{l}}$ | $\{0.1, 0.2, 0.4\}$ | 0.2 |
| $\rho_{\mathrm{g}}$ | $\{0.2, 0.3, 0.5\}$ | 0.5 |
| $q_0$ | $\{0.4, 0.5\}$ | 0.4 |
| $q_{\mathrm{M0}}$ | $\{0.4, 0.7, 0.9\}$ | 0.9 |
| $q_{\mathrm{BM0}}$ | $\{0.4, 0.7, 0.9\}$ | 0.4 |
| $a_{\mathrm{M0}}$, $a_{\mathrm{Mm}}$ | | 0.3, 2 |

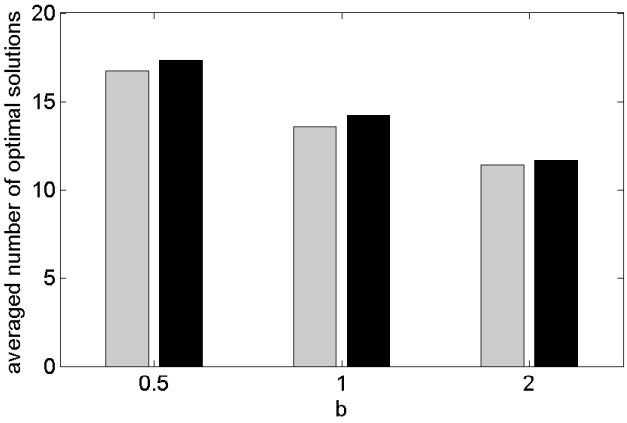



Figure 1: Results of parameter test for $b$ with $a = 2$ (*grey*) and $a = 3$ (*black*) while fixing $w = 10$ and $w_{\mathrm{M}} = 20$, showing the number of optimal solutions averaged over all other parameter values.

weight $a$, achieves better results. The final iteration number was always chosen so as to generate 100,000 solutions during the *ACS*. Since $w = 10$ and $w_{\mathrm{B}} = 20$, the value for the final iteration number is 500.

### 4.2 Experimental Results

For a first overview, the solutions of the algorithms are compared with the optimal solutions for the instances with $(m,n) \in \{(2,5),(2,6),(2,8)\}$ and with $\alpha \in \{80,100\}$ (for higher values of $\alpha$ no failures occurred in these small instances). The optimal values were again found by complete enumeration. Table 2 shows how often the optimum was found, the mean percentage

deviation to the optimum as well as the maximum of these deviations. The percentage deviation to the optimum is calculated by:

$$\text{'Dev to opt'} = \frac{E(Cmax(S_{heu})) - E(Cmax(S^*))}{E(Cmax(S^*))} \cdot 100\% \qquad (41)$$

$$heu \in \{List\&tau, MB, BF, ACS_{\text{MB}}, ACS_{\text{BF}}\}$$

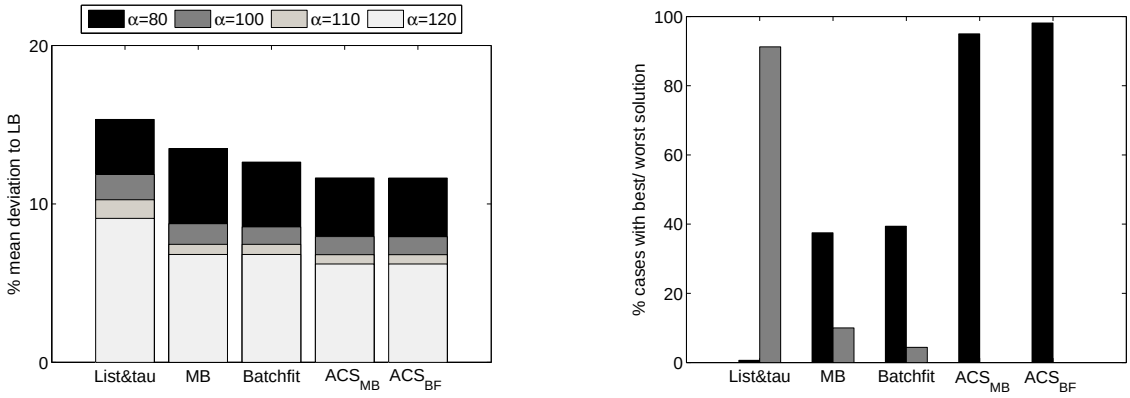
with $S^*$ being the optimal schedule and $S_{heu}$ the schedule created by a heuristic. To determine the mean percentage deviation, (41) was averaged over all described instances with fixed couple $(m,n)$. The decomposition approach *List&tau* results in poor solutions. Finding the optimal

Table 2: Average heuristic performance for instances with 2 machines and 5, 6, 8 jobs.

| (2,5) | % opt found | mean Dev to opt (in %) | max Dev to opt (in %) |
|---|---|---|---|
| *List&tau* | 42.50 | 2.75 | 11.13 |
| *MB* | 52.50 | 1.76 | 10.54 |
| *Batchfit* | 65.00 | 0.64 | 5.23 |
| $ACS_{\text{MB}}$ | 100.00 | 0.00 | 0.00 |
| $ACS_{\text{BF}}$ | 100.00 | 0.00 | 0.00 |
| (2,6) | | | |
| *List&tau* | 22.50 | 3.28 | 12.05 |
| *MB* | 37.50 | 1.26 | 6.52 |
| *Batchfit* | 45.00 | 0.99 | 5.14 |
| $ACS_{\text{MB}}$ | 96.25 | 0.00 | 0.35 |
| $ACS_{\text{BF}}$ | 96.25 | 0.00 | 0.35 |
| (2,8) | | | |
| *List&tau* | 17.50 | 3.35 | 10.82 |
| *MB* | 48.75 | 0.29 | 2.46 |
| *Batchfit* | 50.00 | 0.26 | 1.00 |
| $ACS_{\text{MB}}$ | 95.00 | 0.00 | 0.01 |
| $ACS_{\text{BF}}$ | 95.00 | 0.00 | 0.01 |

value in less than 20% of cases having only 8 jobs on 2 machines is no satisfying performance. Since all other approaches show better results, it can be claimed that solving the production and the maintenance problem in an integrated manner leads to better results than solving it sequentially. The *ACS* finds the optimal solution in most cases with nearly no average deviation. This seems to be independent of the starting solution. The other two approaches show moderate solution qualities.

The next instances imply at least 10 jobs. Here, a generation of the optimal solution is not possible in reasonable computation time. Therefore, the heuristic solutions are compared to the LB, which was also used to determine the number of batches in the construction heuristics. This LB is supposed to serve as a benchmark to easily interpret the overall performance of a heuristic, and to compare the approaches. However, since the LB does neither incorporate any maintenance or failure aspects nor assignment problems, it might not be very tight. Figure 2(a)




(a) Deviation to the LB separated for $\alpha$.

(b) Cases of being the best (*black bars*)/ worst (*grey bars*) solution.

Figure 2: Average heuristical performance for 10 jobs and 2 machines.

illustrates the mean percentage deviations to the LB having 10 jobs and 2 machines. The results are separated for the different values of $\alpha$, because the mean deviations are highly sensitive to this scale parameter, since the LB is not incorporating any maintenance activities. The more failures might occur during the production of a schedule (small $\alpha$), the more maintenance activities are conducted while the LB does not change. Figure 2(b) shows additionally how often an approach found the best and worst solution respectively compared to all other algorithms.

Table 3 states the described values for all cases with 20 jobs. Considering 10 and 20 jobs, we can identify differences in the performance quality of the applied heuristics by means of the average values. The decomposition approach *List&tau* performs poorly again. In most cases, its solution was not part of the best solutions, but in more than 90% part of the worst solutions. The other construction heuristics are also seldom part of the best performing approaches, but

Table 3: Average heuristic performance for instances with 20 jobs in %.

| | | | mean deviation to LB | | | |
|---|---|---|---|---|---|---|
| | best | worst | $\alpha$ | | | |
| (2,20) | solution | solution | 80 | 100 | 110 | 120 |
| *List&tau* | 0.00 | 95.00 | 16.29 | 13.55 | 12.11 | 10.17 |
| *MB* | 7.50 | 5.00 | 14.51 | 10.91 | 9.43 | 7.98 |
| *Batchfit* | 8.75 | 0.00 | 13.71 | 10.35 | 9.16 | 7.98 |
| $ACS_{\text{MB}}$ | 81.88 | 0.00 | 13.61 | 10.17 | 9.02 | 7.80 |
| $ACS_{\text{BF}}$ | 98.13 | 0.00 | 13.46 | 10.02 | 8.98 | 7.80 |
| (3,20) | | | | | | |
| *List&tau* | 0.00 | 95.63 | 16.47 | 13.80 | 11.35 | 9.48 |
| *MB* | 8.75 | 4.38 | 12.98 | 10.25 | 8.87 | 7.36 |
| *Batchfit* | 11.88 | 1.88 | 12.48 | 9.68 | 8.54 | 7.33 |
| $ACS_{\text{MB}}$ | 83.75 | 0.00 | 12.41 | 9.63 | 8.29 | 6.82 |
| $ACS_{\text{BF}}$ | 97.50 | 0.00 | 12.29 | 9.36 | 8.15 | 6.81 |
| (5,20) | | | | | | |
| *List&tau* | 4.38 | 91.25 | 16.76 | 14.13 | 13.59 | 11.37 |
| *MB* | 37.50 | 16.25 | 12.28 | 9.83 | 9.45 | 9.26 |
| *Batchfit* | 40.00 | 10.63 | 11.88 | 9.83 | 9.39 | 8.50 |
| $ACS_{\text{MB}}$ | 91.88 | 5.63 | 10.68 | 8.60 | 8.31 | 7.42 |
| $ACS_{\text{BF}}$ | 96.25 | 4.38 | 10.58 | 8.60 | 8.26 | 7.20 |

with lower mean deviations to the LB. The *ACS* algorithms again perform best. Interestingly, $ACS_{\text{BF}}$ now shows a slight advantage over $ACS_{\text{MB}}$.

Tables 4 and 5 show the results for the instances with 40 and 60 jobs. They support most of the findings stated above. The $ACS_{BF}$ is still the best heuristic in terms of the frequency of being the best approach and in terms of the deviation to the LB. The *BF* approach represents not often the best solution, but also rather seldom the worst solution and is in most instances able to outperform the $ACS_{\text{MB}}$ approach in terms of the mean deviation to the LB.

The averaged and the maximum computation time is presented in Table 6 for the four heuristics. The meta-heuristic requires much more time than the construction heuristics, but it is still done within a couple of minutes.

Table 4: Average heuristic performance for instances with 40 jobs in %.

| | | | mean deviation to LB | | | |
|---|---|---|---|---|---|---|
| | best | worst | $\alpha$ | | | |
| (3,40) | solution | solution | 80 | 100 | 110 | 120 |
| *List&tau* | 0.00 | 91.88 | 17.06 | 14.00 | 12.76 | 10.86 |
| *MB* | 0.00 | 8.13 | 15.21 | 11.66 | 10.11 | 8.86 |
| *Batchfit* | 0.00 | 1.25 | 14.29 | 10.73 | 9.76 | 8.78 |
| $ACS_{\text{MB}}$ | 73.13 | 0.63 | 15.06 | 11.24 | 9.86 | 8.55 |
| $ACS_{\text{BF}}$ | 98.13 | 0.00 | 14.18 | 10.63 | 9.65 | 8.55 |
| (5,40) | | | | | | |
| *List&tau* | 0.00 | 95.00 | 16.44 | 13.42 | 12.84 | 10.69 |
| *MB* | 0.00 | 5.00 | 13.92 | 9.94 | 8.86 | 8.49 |
| *Batchfit* | 0.00 | 0.63 | 13.11 | 9.85 | 8.72 | 8.14 |
| $ACS_{\text{MB}}$ | 84.38 | 0.00 | 13.69 | 9.53 | 8.41 | 8.01 |
| $ACS_{\text{BF}}$ | 97.50 | 0.00 | 12.89 | 9.56 | 8.35 | 7.71 |
| (8,40) | | | | | | |
| *List&tau* | 0.00 | 97.50 | 17.09 | 13.75 | 11.74 | 10.51 |
| *MB* | 28.75 | 2.50 | 13.95 | 8.69 | 7.77 | 7.34 |
| *Batchfit* | 31.25 | 1.88 | 12.99 | 8.69 | 7.77 | 7.34 |
| $ACS_{\text{MB}}$ | 91.25 | 0.00 | 13.07 | 8.14 | 7.33 | 6.84 |
| $ACS_{\text{BF}}$ | 98.13 | 0.00 | 12.41 | 8.18 | 7.33 | 6.84 |

Table 5: Average heuristic performance for instances with 60 jobs in %.

| | | | mean deviation to LB | | | |
|---|---|---|---|---|---|---|
| | best | worst | $\alpha$ | | | |
| (3,60) | solution | solution | 80 | 100 | 110 | 120 |
| *List&tau* | 0.00 | 94.38 | 17.43 | 14.42 | 13.11 | 11.46 |
| *MB* | 1.88 | 5.63 | 15.99 | 12.20 | 10.75 | 9.78 |
| *Batchfit* | 3.75 | 0.00 | 14.80 | 11.34 | 10.27 | 9.20 |
| $ACS_{\text{MB}}$ | 42.50 | 1.25 | 15.91 | 12.09 | 10.71 | 9.67 |
| $ACS_{\text{BF}}$ | 100.00 | 0.00 | 14.75 | 11.31 | 10.23 | 9.16 |
| (5,60) | | | | | | |
| *List&tau* | 0.00 | 98.13 | 17.08 | 14.12 | 12.62 | 11.18 |
| *MB* | 1.88 | 1.88 | 14.63 | 11.04 | 9.67 | 9.08 |
| *Batchfit* | 2.50 | 0.00 | 13.94 | 10.67 | 9.52 | 8.50 |
| $ACS_{\text{MB}}$ | 76.25 | 0.00 | 14.49 | 10.86 | 9.51 | 8.85 |
| $ACS_{\text{BF}}$ | 97.50 | 0.00 | 13.85 | 10.50 | 9.36 | 8.34 |
| (8,60) | | | | | | |
| *List&tau* | 0.00 | 96.25 | 16.37 | 14.22 | 12.20 | 10.96 |
| *MB* | 6.88 | 3.75 | 13.90 | 9.66 | 8.95 | 8.23 |
| *Batchfit* | 8.75 | 0.00 | 12.87 | 9.57 | 8.67 | 7.95 |
| $ACS_{\text{MB}}$ | 78.75 | 1.25 | 13.63 | 9.39 | 8.74 | 7.97 |
| $ACS_{\text{BF}}$ | 99.38 | 0.00 | 12.68 | 9.33 | 8.44 | 7.71 |

Table 6: Average and maximum computation time over all instances.

| | mean cpu (in sec) | max cpu (in sec) |
|---|---|---|
| *List&tau* | 0.09 | 0.24 |
| *MB* | 0.16 | 0.34 |
| *Batchfit* | 0.43 | 1.19 |
| *ACS* | 254.42 | 520.04 |

To give a summarising statement, Table 7 states the deviations of each approach to the best found solution averaged over all instances. Figure 3 additionally shows how often an algorithm resulted in the best respectively worst found solution. The *BF* approach does not represent the

Table 7: Average heuristic performance over all instances.

| | mean Deviation to best (in %) | max Deviation to best (in %) |
|---|---|---|
| *List&tau* | 3.34 | 12.01 |
| *MB* | 0.74 | 13.06 |
| *Batchfit* | 0.37 | 10.59 |
| $ACS_{\text{MB}}$ | 0.25 | 4.59 |
| $ACS_{\text{BF}}$ | 0.01 | 1.80 |

best solution quite often, but with small mean deviations. However, there are still cases with high deviations as one can see at the maximum deviation of nearly 11%. If one accepts the longer runtime of the $ACS_{\text{BF}}$, it is able to find the best solutions in most cases and or solutions with a maximum deviation of just 1.80%.

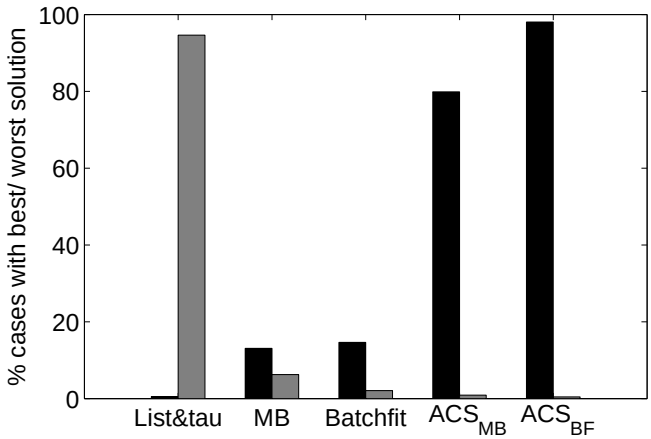



Figure 3: Overall performance of the algorithms regarding finding the best (*black bars*)/ worst (*grey bars*) solution.

## 5 Conclusion

In this paper, the problem of scheduling jobs and preventive maintenance activities is studied while failures can occur stochastically and jobs are non-resumable, minimising the expected makespan. The main goal of this paper is to design and evaluate solution approaches for this problem. Three construction heuristics and an Ant Colony System algorithm were developed. The computational results state the advantage of using the meta-heuristics in frequency of finding the optimal respectively the best found solution and the mean percentage deviation to the latter. Besides that, the sequential approach *List&tau* performed weak, compared to those heuristics that solve the production and maintenance scheduling problem jointly. With increasing instance size, the $ACS_{\mathrm{BF}}$ benefits from its starting solution. If technical conditions allow no fast running of the *ACS*, a recommendation would be to prefer the use of the iterative construction heuristic $Batchfit$, which performed well and fast. In all other cases, the $ACS_{\mathrm{BF}}$ is the best choice.

## References

Adiri, I., Bruno, J., Frostig, E., and Rinnooy Kan, A. H. G. (1989). Single Machine Flow-Time Scheduling with a Single Breakdown. *ACTA Informatica*, (26):679–696.

Barlow, R. and Hunter, L. (1960). Optimum Preventive Maintenance Policies. *Operations Research*, (8(1)):90–100.

Beichelt, F. (1993). *Zuverlässigkeits- und Instandhaltungstheorie*. Teubner, Stuttgart.

Berrichi, A., Amodeo, L., Yalaoui, F., Châtelet, E., and Mezghiche, M. (2009). Bi-Objective Optimization Algorithms for Joint Production and Maintenance Scheduling: Application to the Parallel Machine Problem. *Journal of Intelligent Manufacturing*, (20):389–400.

Berrichi, A., Yalaoui, F., Amodeo, L., and Mezghiche, M. (2010). Bi-Objective Ant Colony

Optimization Approach to Optimize Production and Maintenance Scheduling. *Computers and Operations Research*, (37):1584–1596.

Bosch, K. and Jensen, U. (1983). Instandhaltungsmodelle -Eine Übersicht: Teil 1. *OR Spektrum*, (5):105–118.

Brandolese, M., Franci, M., and Pozzetti, A. (1996). Production and Maintenance Integrated Planning. *International Journal of Production Research*, (34(7)):2059–2075.

Cassady, C. R. and Kutanoglu, E. (2003). Minimizing Job Tardiness Using Integrated Preventive Maitenance Planning and Production Scheduling. *IIE Transactions*, (35):503–513.

Cassady, C. R. and Kutanoglu, E. (2005). Integrating Preventive Maintenance Planning and Production Scheduling for a Single Machine. *IEEE Transactions on Reliability*, (54(2)):304–309.

Chen, J. (2008). Scheduling of Nonresumable Jobs and Flexible Maintenance Activities on a Single Machine to Minimize Makespan. *European Journal of Operational Research*, (190):90–102.

Cui, W., Lu, Z., and Pan, E. (2014). Integrated Production Scheduling and Maintenance Policy for Robustness in a Single Machine. *Computers and Operations Research*, (47):81–91.

Dorigo, M. and Gambardella, L. M. (1997). Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, (1(1)):53–66.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, (26(1)):29–41.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, NY.

Graham, R. L. (1969). Bounds on Multiprocessing Timing Anomalies. *SIAM Journal on Applied Mathematics*, (17):263–269.

Graves, G. H. and Lee, C. Y. (1999). Scheduling Maintenance and Semiresumble Jobs on a Single Machine. *Naval Research Logistics*, (46):845–863.

Hadidi, L., Al-Turki, U., and Rahim, A. (2012). Integrated Models in Production Planning and Scheduling, Maintenance and Quality: A Review. *International Journal of Industrial and Systems Engineering*, (10(1)):21–50.

Kuo, Y. and Chang, Z. (2007). Integrated Production Scheduling and Preventive Maintenance Planning for a Single Machine Under a Cumulative Damage Failure Process. *Naval Research Logistics*, (54):602–614.

Lee, C. Y. (1996). Machine Scheduling with an Availability Constraint. *Journal of Global Optimization*, (9):395–416.

Lee, C. Y. and Chen, Z. L. (2000). Scheduling of Jobs and Maintenance Activities on Parallel Machines. *Naval Research Logistics*, (47):61–67.

Lee, C. Y. and Lin, C. S. (2001). Single-Machine Scheduling with Maintenance and Repair Rate-Modifying Activities. *European Journal of Operational Research*, (135):493–513.

Leng, K., Ren, P., and Gao, L. (2006). A Novel Approach to Integrated Preventive Maintenance Planning and Production Scheduling for a Single Machine Using the Chaotic Particle Swarm Optimization Algorithm. *Proceedings of the 6th world congress on intelligent control and automation*, June 21-23, Dalian, China:7816–7820.

Ma, Y., Chu, C., and Zuo, C. (2010). A Survey of Scheduling with Deterministic Machine Availability Constraints. *Computers & Industrial Engineering*, (58):199–211.

Mokotoff, E. (2001). Parallel Machine Scheduling Problems: A Survey. *Asia-Pacific Journal of Operational Research*, (18):193–242.

Pandey, D., Kulkarni, M., and Vrat, P. (2010). Joint Consideration of Production Scheduling, Maintenance and Quality Policies: A Review and Conceptual Framework. *International Journal of Advanced Operations Management*, (2(1/2)).

Pandey, D., Kulkarni, M., and Vrat, P. (2011). A Methodology for Joint Optimization for Maintenance Planning, Process Quality and Production Scheduling. *Computers & Industrial Engineering*, (61):1098–1106.

Pinedo, M. (1995). *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, NJ.

Qi, X., Chen, T., and Tu, F. (1999). Scheduling the Maintenance on a Single Machine. *Journal of the Operational Research Society*, (50):1071–1078.

Sanlaville, E. and Schmidt, G. (1998). Machine Scheduling with Availability Constraints. *ACTA Informatica*, (35):795–811.

Schmidt, G. (2000). Scheduling with Limited Machine Availability. *European Journal of Operational Research*, (121):1–15.

Sortrakul, N., Nachtmann, H. L., and Cassady, C. R. (2005). Genetic Algorithms for Integrated Preventive Maintenance Planning and Production Scheduling for a Single Machine. *Computers in Industry*, (56):161–168.

von Hoyningen-Huene, W. and Kiesmüller, G. P. (2015). Evaluation of the Expected Makespan of a Set of Non-Resumable Jobs on Parallel Machines with Stochastic Failures. *European Journal of Operational Research*, (240):Pages 439–446.

Wang, H. (2002). A Survey of Maintenance Policies of Deteriorating Systems. *European Journal of Operational Research*, (139):469–489.

Xu, D., Yin, Y., and Li, H. (2009). A Note on "Scheduling of Nonresumable Jobs and Flexible Maintenance Activities on a Single Machine to Minimize Makespan". *European Journal of Operational Research*, (197):825–827.

Yulan, J., Zuhua, J., and Wenrui, H. (2008). Multi-Objective Integrated Optimization Research on Preventive Maintenance Planning and Production Scheduling for a Single Machine. *International Journal of Advanced Manufacturing Technology*, (39):954–964.