Proceedings of the 2024 Winter Simulation Conference
H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, eds.

SCHEDULING JOBS ON A SINGLE STRESS TEST MACHINE IN A RELIABILITY LABORATORY

Jessica Hautz[1], Andreas Klemmt[2], and Lars Mönch[3]

[1]Kompetenzzentrum Automobil- und Industrieelektronik GmbH (KAI), Villach, AUSTRIA
[2]Infineon Technologies Dresden GmbH, Dresden, GERMANY
[3]Dept. of Mathematics and Computer Science, University of Hagen, Hagen, GERMANY

ABSTRACT

We consider a single-machine scheduling problem with unequal job sizes and ready times. Several jobs can be processed at the same time on the machine if the sum of their sizes does not exceed the capacity of the machine. Only jobs of the same family can be processed at the same time. The machine can be interrupted to start a new job or to unload a completed job. A conditioning time is required to reach again the temperature for the stress test. The machine is unavailable during conditioning. Jobs that cannot be completed before conditioning have to continue with processing after the machine is available again. The makespan is to be minimized. A mixed-integer linear program and a constraint programming formulation are established, and a constructive heuristic and a biased random-key genetic algorithm are designed. Computational experiments based on randomly generated problem instances demonstrate that the algorithms perform well.

1 INTRODUCTION

Semiconductor manufacturing deals with producing integrated circuits, so-called chips. The production process is divided into front-end- and back-end production. The front-end consists of the wafer fabrication and sort stages whereas the backend is formed by the assembly and final test stages. The wafer fabrication stage takes place in wafer fabs where the chips are produced layer by layer on wafers, thin discs made of silicon, silicon carbide, or gallium arsenide. Afterwards, the wafers are diced, the defective chips are sorted out, and the chips of appropriate quality are assembled and packaged in an assembly facility. Various tests are performed on the final devices in test facilities (Mönch et al. 2013). Possible failure modes are anticipated in the early stages of technology development. This includes identifying an appropriate set of reliability test structures on the test chip. It must be ensured that the test chips are thoroughly tested when development lots are ready for testing. Trade-offs must be identified among conflicting requirements between component reliability and electrical performance throughout the design and development stages of a manufacturing process (El-Kareh and Hutter 2020). Reliability tests consist of a sequence of alternating measurement and stress investigations, executed under different temperatures and electrical stress conditions. They are conducted in reliability laboratories. The qualification of a product usually takes a few months to be completed.

In the present paper, we analyze and solve a scheduling problem for a stress test machine. The machine can process several jobs at the same time, i.e., it is a p-batching machine. However, in contrast to the common assumptions in the p-batching literature (Fowler and Mönch 2022), we assume job availability, i.e., the jobs are available for performing additional processing steps after finishing at the stress test machine, and it is not required that all jobs of the batch are available when the batch is started.

The paper is organized as follows. The scheduling problem at hand is described in the next section. This includes establishing a mixed integer linear programming (MILP) formulation and a discussion of relevant work. In Section 3, we propose a constraint programming (CP) formulation, a constructive heuristic

and a metaheuristic. Computational experiments with the proposed algorithms are discussed in Section 4. Finally, conclusions are future research directions are presented in Section 5.

2 PROBLEM SETTING

2.1 Problem Description

We consider a single batch processing machine (BPM). There are n jobs labeled by $j = 1, \dots, n$ to be scheduled on this machine. Each job j has a ready date $r_j \geq 0$, a processing time p_j, a size s_j measured in number of load boards to carry the chips, and a weight w_j which is used to express the importance of the job. Each job j belongs to a single incompatible family f_j. Only jobs of the same family can be batched together. Several jobs can be processed together in a batch on the machine until the sum of their job sizes does not exceed the maximum batch size B. After a job in a batch is completed it must be removed from the machine, i.e., we have job availability. To do so, the processing of all unfinished jobs of the current batch is stopped. This leads to a reduction of the stress temperature. After this, a condition time $cond$ is required to reach again the appropriate temperature for the test. A job can be added to an already processed batch if the job is ready for processing, belongs to the same family as the jobs of the current batch, and its size fits into the batch. After the job is added to an already processed batch again a conditioning time cond is required. Note that the condition time is enlarged if other jobs are finished during the time span of a conditioning activity or other jobs are added to the batch. The remaining jobs of the batch can continue with processing after the conditioning activity is finished, i.e., we have resumable jobs. We are interested in minimizing the makespan $C_{\max}$ of the schedule. It is the point in time, when the last job is completed on the machine. Using the three-field notation, the scheduling problem can be stated as follows:

$$1 \left| p - batch, incompatible, r_j, s_j, B, cond, r - a \right| C_{max} \tag{1}$$

where 1 indicates the single machine, $p - batch, incompatible$ refers to batch processing with incompatible families, and $cond, r - a$ describe the conditioning time and the resumable jobs, respectively.

Problem (1) is NP-hard since it is easy to see that for the special case $r_j \equiv 0, p_j \equiv 1, 1 \leq j \leq n,$ and $cond$=0 solving an instance of (1) is equivalent to solving an instance of the bin packing problem. Here, the items of the bin packing problem correspond to the jobs in (1), the size of the items to the size of the jobs, and the bins to the batches. Moreover, the batch size corresponds to the bin capacity. Clearly, the number of required bins is given by the $C_{\max}$ of the schedule. Since the bin packing problem is NP-hard (Martello and Toth 1990), (1) is also NP-hard.

2.2 MILP Formulation

We establish a MILP formulation for problem (1). The following sets and indices are used in the model:

N: set of all jobs, $N = \{1, \dots, n\}$

F: set of all incompatible job families, $F = \{1, \dots, f_{max}\}$

N_f: set of all jobs belonging to family $f \in F$

H: scheduling horizon, $H = \{1, \dots, T\}$

i, j: job index

f, k: family index

t: time slot index.

The following parameters are used in the model:

p_j: processing time of job j

r_j: ready time of job j

s_j: size of job j

B: maximum batch size

$cond$: conditioning time

T: sufficiently large number to describe the scheduling horizon H.

We define the following decision variables:

S_j: start time of job j

P_j: machine time of job j

$X_{jt} = \begin{cases} 1, & \text{if job } j \text{ is active in time slot } t \\ 0, & \text{otherwise} \end{cases}$

$avail_t = \begin{cases} 1, & \text{if the machine is not in a conditioning state at time slot } t \\ 0, & \text{otherwise} \end{cases}$

$prmtn_t = \begin{cases} 1, & \text{if the machine is preemted in time slot } t \\ 0, & \text{otherwise} \end{cases}$

$\chi_{jt} = AND(X_{jt}, X_{j,t+1})$

$Y_{jt} = AND(X_{jt}, avail_t)$.

The model can be formulated as follows:

$$\min C_{\max} \tag{2}$$

subject to

$$\sum_{t \in H} Y_{jt} = p_j, \qquad j \in N, \tag{3}$$

$$\sum_{t=1}^{T-1} \chi_{jt} = P_j - 1, \qquad j \in N, \tag{4}$$

$$\sum_{t \in H} X_{jt} = P_j, \quad j \in N, \tag{5}$$

$$\sum_{j \in N} s_j \cdot X_{jt} \leq B, \qquad t \in H, \tag{6}$$

$$S_j = X_{j1} + \sum_{t=1}^{T-1} (t+1) \cdot \left(X_{j,t+1} - \chi_{jt}\right), \qquad j \in N, \tag{7}$$

$$S_j \geq r_j, \; j \in N, \tag{8}$$

$$X_{jt} + X_{it} \leq 1, \qquad t \in H, j \in N_f, i \in N_k, f, k \in F, f \neq k, \tag{9}$$

$$prmtn_1 = OR_{j \in N}\left(X_{j1}\right), \tag{10}$$

$$prmtn_t = OR_{j \in N}\left(X_{jt} - 2 \cdot \chi_{j,t-1} + X_{j,t-1}\right), \; 2 \leq t \leq T, \tag{11}$$

$$avail_t = AND_{\tau \in \{\max\{1, t-cond+1\}, \ldots, t\}}(1 - prmtn_\tau), \qquad t \in H, \tag{12}$$

$$\chi_{jt} = AND\left(X_{jt}, X_{j,t+1}\right), \qquad j \in N, 1 \leq t \leq T-1, \tag{13}$$

$$Y_{jt} = AND\left(X_{jt}, avail_t\right), \qquad j \in N, t \in H, \tag{14}$$

$$C_{\max} \geq S_j + P_j, \qquad j \in N, \tag{15}$$

$$S_j, P_j, C_{\max} \in \mathbb{R}^+, X_{jt}, \chi_{jt}, Y_{jt} \in \{0,1\}, \; prmtn_t, avail_t \in \{0,1\}, j \in N, t \in H. \tag{16}$$

The AND and OR constructs in the MILP model can be linearized using standard techniques (cf. FICO Xpress Optimization 2024). The decision variable χ_{jt} is 1 if job j is processed at time slot t, but not finished at t. The Y_{jt} decision variables take a value of 1 if job j is processed on the machine at t and the machine is not in a conditioning state. Minimizing the makespan is the objective which is expressed by (2). Constraints (3) ensure that each job is active for exactly p_j time slots where the machine is not in a conditioning state, and constraint set (4) forces each job to be conducted without preemption. The machine time of job j is calculated by (5), and constraint set (6) ensures that the maximum batch size is not exceeded for any time slot. The job start times are calculated by the equalities (7). If job j starts processing in the first time slot, $X_{j1} = 1$ holds and the remaining summands are zero. Furthermore, j starts its processing in time slot $t+1$ if and only if $X_{j,t+1} = 1$ and $X_{jt} = 0$, i.e. $X_{j,t+1} - \chi_{jt} = 1$. The release dates of jobs are respected by the inequalities (8), and the incompatible families are assured with (9). Constraints (10) and (11) define the preemption variables. With (10), $prmtn_1 = 1$ must hold if and only if at least one job starts its processing in the first time slot, i.e., there is at least one job j with $X_{j1} = 1$. Furthermore, (11) assures that $prmtn_t = 1$ if and only if at least one job starts its processing at time slot $t > 1$, i.e., there exists at least one job j with $X_{j,t-1} = 0$ and $X_{jt} = 1$, or if a job finishes its machine time at time slot $t > 1$, i.e., there exists at least one job j with $X_{j,t-1} = 1$ and $X_{jt} = 0$. In both cases, $X_{jt} - 2 \cdot \chi_{j,t-1} + X_{j,t-1} = 1$ holds. Constraints (12) ensure that $avail_t = 0$ whenever the machine is in a conditioning period at time slot t, applying if the machine has been preempted within $[\max\{0, t - cond + 1\}, t]$. The binary variables χ_{jt} and Y_{jt} are defined by (13) and (14), respectively. Finally, the makespan is defined by the inequalities (15), and constraint set (16) defines the domain of the decision variables.

Since problem (1) is NP-hard, it can only be expected that small-sized problem instances are solved in a reasonable amount of computing time using the MILP model (2)-(16). However, the model can be applied to assess the performance of the proposed heuristics for small-sized problem instances and make sure that they are correctly coded.

2.3 Discussion of Related Work

For a recent survey of p-batching problems, we refer to Fowler and Mönch (2022). However, problems similar to (1) are not addressed in this survey. Schmidt (1984) studies a parallel-machine scheduling problem where each job has a deadline and each machine has different availability intervals. The goal is to

compute a feasible preemptive schedule whenever at least one exists. A pseudo-polynomial-time algorithm is provided. Lee (1996) discusses scheduling problems where the machine is unavailable during the period from s_j to t_j for $0 \leq s_j \leq t_j$. Resumable and nonresumable jobs are considered for several regular performance measures. However, the problem at hand is different since the unavailable periods are not given, they are a result of scheduling decisions. Schmidt (2000) reviews scheduling approaches for problems with limited machine availability. However, BPM problems are not considered. Ozturk (2022) studies a scheduling problem for a single BPM. Consecutive p-batches form a serial batch (s-batch), and all jobs of the same serial batch are considered as finished when the last job of the s-batch completes its processing. There is a fixed preparation time that occurs between consecutive s-batches. The total completion time and the total weighted tardiness are the performance measures. A column generation approach is proposed. Although this problem is similar to problem (1) we cannot reuse solution techniques since we allow resumable jobs and have a different objective function in problem (1). Overall, we conclude that to the best of our knowledge, problem (1) is not addressed so far in the literature.

3 SOLUTION APPROACHES

3.1 CP Approach

CP has become popular in solving complex scheduling problems, often outperforming MILP methods. The strength of CP lies in its global constraints and specialized algorithms that exploit the structure of common patterns found in scheduling problems and prune infeasible solutions efficiently. The direct representation of time and resources through interval variables is particularly advantageous as it avoids time granularity dependencies to decision variables. We refer to Apt (2003); Rossi et al. (2006) for CP details and the specific syntax used here. We use a parameter S to bound the maximum number of setups that can occur in a problem instance. To model problem (1) with CP, the following decision variables and cumulative functions are used:

J_j:	interval variable representing job j, no size is specified
CS_j, CE_j:	interval variable for conditioning periods of job j (start and end) – size is $cond$
C_s:	optional interval variable representing the conditioning periods s – no size
O_j:	integer variable representing the overlaps in the conditioning period of job j
$C_{\max}$:	integer variable representing the makespan
$cumul\ function\ C$:	$= \sum_{j=1}^{n} (pulse(CS_j, 1) + pulse(CE_j, 1))$
$cumul\ function\ S$:	$= \sum_{s=1}^{S} pulse(C_s, 1)$
$cumul\ function\ R$:	$= \sum_{j=1}^{n} pulse(J_j, s_j)$
$cumul\ function\ F$:	state function indicating the active job family.

Note that the w.l.o.g. decision variables C_{max} and O_j can also be modeled as decision expressions. The objective is given by $inimize\ C_{\max}$, and the constraints are:

$$startAtStart(J_j, CS_j),\ startAtEnd(CE_j, J_j), \quad j \in N, \tag{17}$$

$$alwaysIn(S, CS_j, 1,1),\ alwaysIn(S, CE_j, 1,1),\ j \in N, \tag{18}$$

$$alwaysIn(C, C_s, 1, B), \quad s \in \{1, \dots, S\}, \tag{19}$$

$$O_j = \sum_{s=1}^{S} overlapLength(J_j, C_s), \quad j \in N, \tag{20}$$

$$sizeOf(J_j) = p_j + O_j,\ j \in N, \tag{21}$$

$$alwaysEqual(F, J_j, f_j, 0,0), \quad j \in N, \tag{22}$$

$$R \leq B, \tag{23}$$

$$endOf(J_j) \leq Cmax, \quad j \in N, \tag{24}$$

$$presenceOf(C_{s2}) => presenceOf(C_{s1}), \quad s1, s2 = 1, \dots, S, s1 + 1 = s2, \tag{25}$$

$$endBeforeStart(C_{s1}, C_{s2}),\ s1, s2 = 1, \dots, S, s1 + 1 = s2. \tag{26}$$

Constraint set (17) ensure that a conditioning period is applied whenever a job starts or finishes its processing. With (18) and (19), a 1-n synchronization of this tasks to unique conditioning intervals C_s is performed. While (20) is measuring interval overlaps of Jobs J_j and conditioning times C_s, the actual machine time of the jobs J_j can be calculated by (21) defining interval J_j size finally. The batching constraints are given by (22). Constraints (23) ensure that the machine capacity is not exceeded, and (24) bounds the makespan value. To further strengthen the CP, the symmetry-breaking constraints (25), (26) are applied. The interesting aspect in modeling problem (1) in CP here is the fact that the actual machine time of the jobs is dependent on the schedule, meaning that the interval size is dynamic. The synchronization of interval variables via cumul function and the motivation for preemption modeling is shown in Figure1.

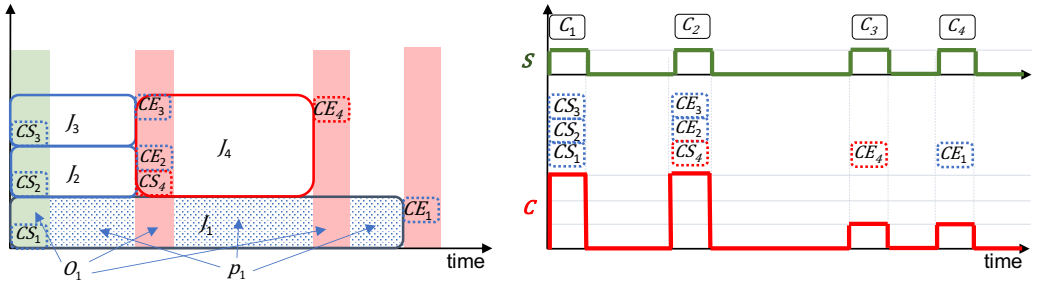

Figure 1: CP interval variable representation and synchronization via cumul function.

3.2 Constructive Heuristic

We introduce a constructive heuristic designed to compute schedules based on a given job permutation. It constructs schedules by arranging jobs according to a given permutation $\pi: N \to N$, where $\pi(i) = j$ indicates that job j is in the i-th position of the permutation. The schedule is constructed within two sequential algorithms. Algorithm 1 arranges jobs on the machine based on the permutation, starting each job as early as possible while considering the machine capacity and job families. Subsequently, Algorithm 2 builds upon the schedule obtained by Algorithm 1 and incorporates the machine conditioning times required whenever a job starts or ends. We use a capacity function $C: H \to \{0, \dots, B\}$ to describe the remaining capacity of the machine at time t, and a family function $F: H \to \{-1\} \cup \{1, \dots, f_{max}\}$ that keeps track of the active family at time $t \in H$. Initially, we set $C(t) := B$ and $F(t) := -1,\ t \in H$.

Algorithm 1:

We iterate over the jobs of the input permutation π, and the current job of iteration j is denoted by $\pi(j)$. For the current job $\pi(j)$, the job start time $S_{\pi(j)} := r_{\pi(j)}$ is set and a check is carried out whether the machine has enough capacity for the job's duration and whether the active family during the job's duration matches the family of the current job $f_{\pi(j)}$ or not. If the capacity and active family are sufficient for the entire duration of the job, the job start time is set, the capacity and family functions are updated accordingly and the next job in the permutation is considered. Otherwise, the job start time is incremented by one and the availability check is performed again until a feasible job start time of the current job is found.

Algorithm 2:

1. Algorithm 1 is executed with input permutation π, returning an array of job start times S_π. A second permutation μ is then defined, sorting the jobs in non-decreasing order of the derived job start times S_π. The set of job completion times of the previous iterations C_J is defined and initially set to $C_J := \emptyset$. The capacity and family functions are set to their initial values.
2. We iterate over the jobs of the input permutation μ, and the current job of iteration j is denoted by $\mu(j)$. For each iteration j, the set $J := \{\mu(i) \in N \mid i < j\}$ of already scheduled jobs is defined. For the current job $\mu(j)$, the job start time $S_{\mu(j)} := \max\{S_{\mu(j-1)}, r_{\mu(j)}\}$ is set and a check if the machine has capacity for the duration $[S_{\mu(j)}, S_{\mu(j)} + 1]$ and if the active family during this duration matches the family of the current job $f_{\mu(j)}$ is made. It is sufficient to check the first time slot only because the solution of Algorithm 1 already respects the capacity and family constraints and with the consideration of the conditioning times, the feasible solution is just shifted to the right. If the capacity and family constraints are met, the start time of the current job is set, the completion time is computed by $C_{\mu(j)} := S_{\mu(j)} + p_{\mu(j)} + cond$ and the capacity and family functions are updated accordingly. Then, the conditioning periods caused by job start times and job completion times are considered within Steps 3.a-c. Otherwise, the job start time is incremented by one and the availability check is performed again until a feasible job start time of the current job is found.
3. The current job can preempt the already scheduled jobs with its start and completion time. In addition, the current job can be preempted by the completion times of already scheduled jobs. Since the jobs are arranged in non-decreasing order of their start times, it is not possible that the current job is preempted by the start times of already scheduled jobs. Since the start time of the current job is already fixed, but the completion time can still vary, the conditioning periods have to be considered in the following order:

a. The completion times of jobs that are preempted by the start time of the current job are adapted. If $S_{\mu(j)} = S_{\mu(j-1)}$ or $S_{\mu(j)} \in C_J$ holds, then the preemption at this time slot was already considered in a previous iteration. Otherwise, the completion times of already scheduled jobs $i \in J$ with $S_{\mu(j)} < C_i$ have to be adapted. We denote the set of those jobs by $\tilde{J}$ and iterate over it. A parameter $OLC = 0$ is set to count the overlaps of the conditioning period caused by the current job with previous conditioning periods caused by jobs that are not preempted in the current iteration, i.e. jobs from $J \setminus \tilde{J}$. Subsequently, all job completion times of jobs in $\tilde{J}$ are incremented by $cond - OLC$ and the capacity and family functions, as well as the set C_J are updated accordingly.

b. The completion time of the current job is adapted, if it is preempted by the completion times of already scheduled jobs. A set $\widetilde{C_J} := \emptyset$ is used to store completion time preemptions that were already considered. The completion time of already scheduled jobs $i \in J$ preempts the current job if $S_{\mu(j)} < C_i < C_{\mu(j)}$ holds. We denote the set of those jobs by $\tilde{J}$ and iterate over it. If $C_i \in \widetilde{C_J}$ holds, then the preemption at this time slot was already considered in a previous iteration.

Otherwise, similar to Step 3.a, the completion time of the current job is incremented while taking overlaps into account, and the capacity and family functions, as well as the sets $\widetilde{C_J}$ and C_J are updated accordingly.

c. Finally, the completion times of jobs that are preempted by the completion time of the current job are adapted. If $C_{\mu(j)} \in C_J$ holds, then the preemption at this time slot was already considered in a previous iteration. Otherwise, the completion times of already scheduled jobs $i \in J$ with $C_{\mu(j)} < C_i$ have to be adapted. We denote the set of those jobs by $\tilde{J}$ and iterate over it. Similar to Step 3.a, the job completion times of jobs in $\tilde{J}$ are incremented while taking overlaps into account, and the capacity and family functions, as well as the set C_J are updated accordingly.

We sort the jobs with respect to the longest processing time (LPT) dispatching rule to provide π as input for Algorithms 1 and 2. It is well-known that the LPT rule leads to high-quality schedules for the $Pm||C_{max}$ problem (Pinedo 2008) where Pm refers to parallel identical machines. Since we consider a BPM which allows for processing several jobs at the same time, the problem at hand is similar to a parallel-machine scheduling problem. We abbreviate this heuristic based on Algorithm 2 as A2-LPT.

An illustrative example of the preemptions that have to be considered in Algorithms 2, 3.a-3.c is provided in Figure 2. The already scheduled jobs of the set J are colored green and the current job is colored red. In the example on the left, job 4 preempts job 1 with its start time, and job 4 is preempted by the completion time of job 1. In the example on the right, job 4 preempts job 1 with both its start and completion time. In both cases, the start time preemption of job 4 was already considered in previous iterations, since job 2 and 3 finish at the same time slot where job 4 starts.

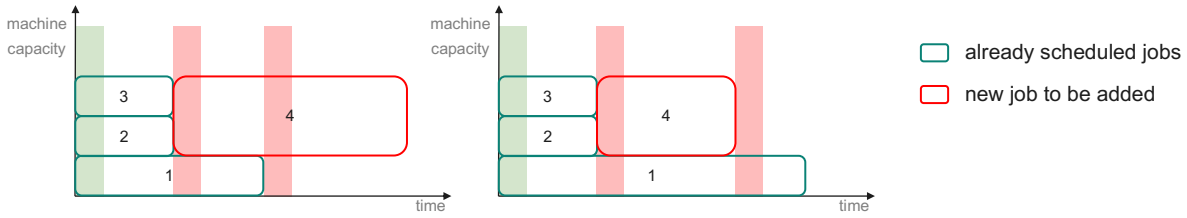

Figure 2: Example of preemptions.

3.3 BRKGA

A genetic algorithm (GA) maintains a set of solutions, and we refer to this set as population. GAs are iterative algorithms. A single iteration corresponds to a generation. Reproduction and mutation procedures are used to change the individuals of the current generation to form a new generation. It is likely that only the fittest individuals are selected for the new generation. GAs are successfully applied to solve computationally hard problems of operations management (Lee 2018). GAs based on the random-key representation, so-called random-key GAs (RKGAs), are appropriate to support sequencing decisions. Chromosomes are represented as vectors of randomly generated real numbers from $(0,1)$. Only problem-specific decoders must be designed to associate a chromosome with a solution of a scheduling problem at hand. Sorting the random keys is required to compute a sequence, i.e., a permutation. Starting from a randomly chosen population of random-key vectors, the fitness of a chromosome is determined by a decoder that implements the objective function of the optimization problem. The population consists of a small set of elite individuals and a set of non-elite individuals. Individuals that belong to the elite set have low objective function values in the case of a minimization problem. The elite individuals are taken over unchanged into the next generation. A certain fraction of randomly generated mutants is placed into the population. The remaining individuals of the population of the next generation are determined by crossover. Two individuals are randomly chosen from the population in a RKGA for crossover purposes. A

parameterized uniform crossover is applied. A biased coin is tossed for each gene to determine which parent will contribute to the allele. Biased RKGAs (BRKGAs) are RKGAs where the first parent chromosome is from the elite set and the probability of choosing from this parent is $\rho > 0.5$. BRKGAs converge often faster than RKGAs (Londe et al. 2024).

The random-key representation is appropriate to problem (1) since it is able to produce the job permutations $(\pi[1], \dots, \pi[n])$ to represent problem (1). The following encoding and decoding is therefore appropriate for problem (1). A chromosome is provided by the random-key vector $rk := [rk_1, \dots, rk_n]$ of real numbers where $rk_j \in (0,1), 1 \leq j \leq n$, and gene rk_j represents job j. We then use the Algorithms 1 and 2 from Subsection 3.2 as a decoder. Job permutations are transferred into feasible schedules.

4 COMPUTATIONAL EXPERIMENTS

4.1 Design of Experiments

We assess the performance of the proposed algorithms by randomly generated problem instances motivated by settings found for stress test machines in semiconductor reliability laboratories. We expect that the performance of the proposed algorithms depends on the number of jobs, the ready date setting, the maximum batch size, the condition time, and the number of incompatible families. For a first experiment, we generate medium- and large-sized instances. We compare the CP and the BRKGA relative to the performance of the A2-LPT which serves as a reference heuristic. The ratio $(1 - C_{max}(A)/C_{max}(A2 - LPT))100\%$ is applied to evaluate the performance of an algorithm A relative to the A2-LPT where $C_{max}(A)$ is the makespan of a schedule obtained by A. The design of experiments is summarized in Table 1. Here, $D[a, b]$ refers to a discrete uniform distribution over the set of integers $\{a, \dots, b\}$.

Table 1: Design of experiments for medium- and large-sized instances.

Factor	Level	Count
number of jobs, n	24,48,96,192	4
job sizes	$s_j \sim DU[1,13]$	1
maximum batch size, B	21 boards (3 jobs), 42 boards (6 jobs)	2
number of families, $f_{\max}$	1,3,6	3
number of jobs per family, n_f	n/ $f_{\max}$	1
processing time of the jobs p_j	$\sim DU[1, 100]$	1
ready times $r_j \equiv 0$	10%,20% of the jobs	2
ready times $r_j \geq 0$	$\sim DU[1, \lceil 50n/B \rceil]$	1
conditioning time	small: 1,2,3 (each with equal probability) large: 10,20,30 (each with equal probability)	2
Number of factor combinations		96
Number of independent problem instances		3
Total number of problem instances		288

In another experiment, we consider small-sized problem instances to check the correct implementation of the heuristics. The generation scheme is similar to the one of Table 1. But we consider only 6, 9, and 12 jobs. We use $B = 21$ and $f_{\max} \in \{1,3\}$, while the number of jobs per family is equally distributed. The job processing times follow $p_j \sim DU[1, 20]$ and the ready times $r_j \sim DU[1, \lceil 40n/B \rceil]$, while 10% of the jobs are ready at time 0. The conditioning time of the machine is set to 1,2, and 3, each with equal probability. The

remaining settings are the same as in Table 1. We have six factor combinations, and the number of independent problem instances per factor combination is 3. Overall, we solve 18 small-sized instances. We use a maximum computing time per instance for each of the considered algorithms except the A2-LPT.

4.2 Parameter Setting and Implementation Issue

The following parameter settings are used for the BRKGA. The population size is 400 in all experiments. We use $\rho_e = 0.7$ as the probability of choosing a parent from the elite set. The fraction of the population to be replaced by mutants is $p_m = 0.1$, and the fraction that belongs to the elite set is 0.2. These settings are found by recommendations from Toso and Resende (2011) and some preliminary experiments with a small number of instances applying a trial and error strategy. The MILP and the heuristics are coded using the C++ programming language. The MILP instances are solved using Gurobi 11.0.1. The CP Optimizer in IBM ILOG CPLEX Optimization Studio 22.1.1 (IBM 2024) is used to solve the CP instances using default settings, i.e. the parametrization of search types, propagation settings, and search phases defining instantiation strategies are omitted. The brkgaAPI framework (Toso and Resende 2015) is used to code the BRKGA. We apply the parallel decoding capabilities of the brkgaAPI using OpenMP. The experiments are performed on a workstation with Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz.

4.3 Results

The obtained $C_{\max}$ values for the small-sized instances are shown in Table 2. The time limit for the MILP approach is 6 hours per instance, and the BRKGA and CP approach are performed for 120 seconds per instance. We also report the relative MIP gap. Best known or even optimal $C_{\max}$ values are marked bold.

Table 2: Computational results for small-sized instances.

Benchmark			**A2-LPT**	**MILP (6h)**		**BRKGA (120s)**	**CP (120s)**
N	F	Instance	$C_{\max}$		Gap (%)		
6	1	1	49	**47**	0.00	48	**47**
6	1	2	50	**42**	0.00	**42**	**42**
6	1	3	31	**29**	0.00	31	**29**
6	3	1	64	**58**	0.00	59	**58**
6	3	2	68	**63**	0.00	**63**	**63**
6	3	3	70	**67**	0.00	69	**67**
9	1	1	53	**48**	0.00	**48**	**48**
9	1	2	55	**53**	0.00	**53**	**53**
9	1	3	47	**41**	0.00	47	**41**
9	3	1	93	**87**	0.00	**87**	**87**
9	3	2	76	**72**	0.00	**72**	**72**
9	3	3	77	**72**	0.00	**72**	**72**
12	1	1	73	**60**	0.00	61	61
12	1	2	77	68	25.00	**63**	64
12	1	3	56	**51**	0.00	53	**51**
12	3	1	80	**69**	0.00	70	70
12	3	2	101	**88**	9.09	91	89
12	3	3	77	**63**	0.00	**63**	**63**

Table 2 shows that the MILP struggles to find optimal solutions for problem instances with $n = 12$. This shortfall results from the time-indexed formulation of the problem presented in Subsection 2.2. The results of the BRKGA showcase the limitation of the algorithm in finding an optimal solution.

The tendency to schedule jobs as early as possible is efficient, but may lead to suboptimal decisions in certain situations, as highlighted in Figure 3. This instance contains $n = 7$ jobs with $p_j = \{1,2,3,1,2,3,2\}$, $s_j = \{1,1,1,1,1,1,1\}$, $r_j = \{0,1,2,5,6,7,8\}$, $B = 3$, and $cond = 2$. The schedule on the left is derived by considering the jobs in non-decreasing order of their ready times using Algorithm 2, the schedule on the right displays an optimal MILP solution. The red area reflects the conditioning time. We observe that it can be beneficial to start the jobs later and at the same time as other jobs are started or completed resulting in less conditioning periods and a smaller $C_{\max}$. CP demonstrates its ability to determine an optimal schedule for most problem instances within a maximum computing time of 120 seconds per instance. This efficiency of the CP compared to the MILP is enhanced by the utilization of interval variables to model the processing times of jobs.

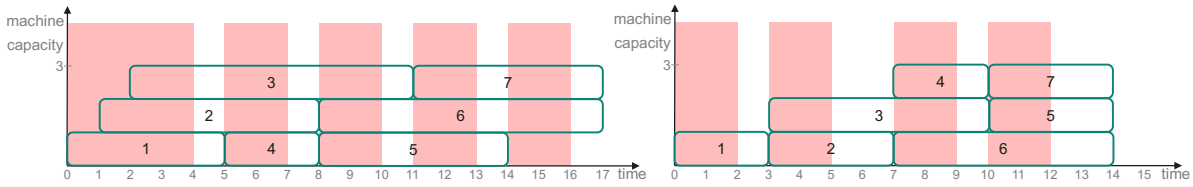

Figure 3: Worst-case example for the behavior of Algorithm 2.

We compare the A2-LPT, the BRKGA, and the CP for the instances from Table 1 in Table 3. Instead of presenting all results individually, average results are grouped by different factor levels. All values are relative to the $C_{\max}$ values found by the A2-LPT. Best results in each row are marked in bold.

Table 3: Computational results for medium- and large-sized instances.

Compare	***n***				***B***		***F***			***Cond***		**Overall**
	24	48	96	192	21	42	1	3	6	1,2,3	10,20.30	
BRKGA												
30s	**0.908**	**0.898**	**0.948**	1.019	**0.935**	**0.950**	**0.932**	**0.935**	**0.959**	0.947	**0.936**	**0.942**
60s	**0.907**	**0.894**	**0.931**	**0.998**	**0.926**	**0.938**	**0.924**	**0.925**	**0.947**	**0.940**	**0.923**	**0.932**
120s	**0.905**	**0.890**	**0.925**	**0.980**	**0.920**	**0.930**	**0.917**	**0.918**	**0.941**	0.935	**0.913**	**0.925**
300s	**0.905**	**0.887**	**0.920**	**0.969**	**0.917**	**0.925**	**0.913**	**0.914**	**0.936**	0.933	**0.907**	**0.921**
CP												
30s	0.916	0.909	0.981	**1.000**	0.947	0.951	0.947	0.939	0.961	**0.945**	0.953	0.950
60s	0.913	0.902	0.965	**0.998**	0.942	0.942	0.942	0.933	0.953	**0.940**	0.945	0.943
120s	0.912	0.897	0.952	0.998	0.938	0.937	0.939	0.927	0.947	0.936	0.940	0.938
300s	0.911	0.893	0.938	0.994	0.933	0.932	0.934	0.921	0.941	**0.932**	0.932	0.933

We see from Table 3 that the BRKGA and the CP generally outperform the A2-LPT, particularly when a minimum of 60 s of computing time is given. This trend is observable for the instances with $n = 192$, where the approaches struggle to find superior solutions within just 30 s per instance. The BRKGA obtaines the best results due to its problem-tailored design, but the CP approach also consistently yields high-quality solutions even without customization.

5 CONCLUSIONS AND FUTURE RESEARCH

In this paper, we formulated and analyzed a single-machine scheduling problem which can be found in semiconductor reliability laboratories. A MILP and a CP approach were established. Moreover, a constructive heuristic for a given job permutation and a metaheuristic approach were designed. Computational experiments were carried out that demonstrate that the metaheuristic approach and the CP

perform well, even for a small amount of computing time. There are several directions for future research. First of all, it is desirable to design neighborhood search-based metaheuristics, such as variable neighborhood search or iterated local search. It is expected that appropriate neighborhood structures can avoid the limitations of Algorithm 2. Moreover, it is desirable to consider parallel machines. Finally, it is interesting to consider complex job scheduling problems for semiconductor reliability laboratories where the problem studied in the present paper is a subproblem.

ACKNOWLEDGMENTS

This work was funded by the Austrian Research Promotion Agency (FFG, Project No. FO999900268).

REFERENCES

Apt, K. 2003. *Principles of Constraint Programming*. Cambridge: Cambridge University Press.

El-Kareh, B. and L. N. Hutter. 2020. *Silicon Analog Components: Device Design, Process Integration, Characterization, and Reliability.2nd ed.*, Cham: Springer.

FICO Xpress Optimization. 2024. "MIP Formulations and Linearizations". Quick Reference. https://msi-jp.com/xpress/-learning/square/10-mipformref.pdf, accessed 29th April 2024.

Fowler, J. W. and L. Mönch. 2022. "A Survey of Scheduling with Parallel Batch (p-batch) Processing". *European Journal of Operational Research* 298(1): 1-24.

IBM. 2024. "IBM ILOG CPLEX Optimization Studio v20.1". https://www.ibm.com/products/ilog-cplex-optimization-studio, accessed 29th April 2024.

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

Lee, C. K. H. 2018. "A Review of Applications of Genetic Algorithms in Operations Management". *Engineering Applications of Artificial Intelligence* 76: 1-12.

Londe, M. A., L.S. Pessoa, C. E. Andrade, and M. G. C. Resende. 2024. "Biased Random-key Genetic Algorithms: a Review". *European Journal of Operational Research*. In press.

Martello S. and P. Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. Chichester: Wiley.

Mönch, L., J. W. Fowler, and S. J. Mason. 2013. *Production Planning and Control for Semiconductor Wafer Fabrication Facilities: Modeling, Analysis, and Systems*. New York: Springer.

Ozturk, O. 2022. "When Serial Batch Scheduling Involves Parallel Batching Decisions: A Branch and Price Scheme". *Computers & Operations Research* 137: 105514.

Pinedo, M. L. 2008. *Scheduling: Theory, Algorithms, and Systems. 3rd edition*. New York: Springer.

Rossi, F., P. van Beek, and T. Walsh. 2006. *Handbook of Constraint Programming*. Amsterdam: Elsevier Science.

Schmidt, G. 1984. "Scheduling Independent Tasks with Deadlines on Semi-identical Processors". *Journal of the Operational Research Society* 39: 271-277.

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

Toso, R. F. and M. G. C. Resende. 2011. "A C++ Application Programming Interface for Biased Random-key Genetic Algorithms". http://mauricio.resende.info/doc/brkgaAPI.pdf, accessed 29th April 2024.

Toso, R. F. and M. G. C. Resende. 2015. "A C++ Application Programming Interface for Biased Random-key Genetic Algorithms". *Optimization Methods and Software* 30(1): 81-93.

AUTHOR BIOGRAPHIES

JESSICA HAUTZ is a PhD student at the University of Hagen. She received her master's degree in Mathematics in 2022 from the University of Klagenfurt. She works as a PhD researcher at KAI GmbH in the data science team. Her research interests are combinatorial optimization, mathematical programming, and scheduling. Her email adress is Jessica.Hautz@k-ai.at.

ANDREAS KLEMMT is a Lead Principal Engineer at Infineon Technologies. He received his master's degree in Mathematics in 2005 and Ph.D. in Electrical Engineering in 2011 from Dresden University of Technology. He works as vertical integration solution architect in the Global Factory Integration Department of Infineon. His research interests are scheduling, mathematical programming, capacity planning, production control, and simulation. His email adress is Andreas.Klemmt@infineon.com.

LARS MÖNCH is full professor of Computer Science at the University of Hagen where he heads the Chair of Enterprise-wide Software Systems. He holds M.S. and Ph.D. degrees in Mathematics from the University of Göttingen. His research and teaching interests are in information systems for production and logistics, simulation, scheduling, and production planning. His email address is Lars.Moench@fernuni-hagen.de. His website is https://www.fernuni-hagen.de/ess/team/lars.moench.shtml.