An Open Challenge for Exact Job Scheduling with Reticle Batching in Photolithography

Thomas Eiter[1], **Tobias Geibinger**[1], **Andrej Gisbrecht**[2], **Nelson Higuera Ruiz**[1], **Nysret Musliu**[1,3], **Johannes Oetsch**[1], **Daria Stepanova**[4]

[1] Institute for Logic and Computation, TU Wien, Favoritenstraße 9–11, 1040 Vienna, Austria,
[2] Robert Bosch GmbH, Tübinger Straße 123, 72762, Reutlingen,
[3] CD-Lab Artis, TU Wien, Favoritenstraße 9–11, 1040 Vienna, Austria,
[4] Bosch Center for AI, Robert Bosch Campus 1, 71272 Renningen, Germany
{eiter, nhiguera, oetsch}@kr.tuwien.ac.at, {tgeibing, musliu}@dbai.tuwien.ac.at,
{andrej.gisbrecht, daria.stepanova}@de.bosch.com

Abstract

We consider scheduling solutions for photolithography, an important sub-task in semi-conductor production, where patterns are transferred to wafers using reticles. The problem can be modelled as job scheduling on unrelated parallel machines with sequence-dependent setup times and release dates. The reticles add auxiliary-resource constraints for processing jobs. Equipping machines with the right reticles using transport robots from stockers in time renders this problem extremely difficult for exact solvers that use a declarative model. The latter would be attractive as such models tend to be compact and easy to maintain. We present a solver-independent MiniZinc model and provide 500 new benchmark instances. However, only small instances can be solved with state-of-the-art MIP and CP solvers. Consequently, we present this problem as an open challenge with considerable potential for driving improvements towards industrial applications.

1 Introduction

Semi-conductor production involves a number of rather complicated processing steps. A crucial sub-task is *photolithography*, where patterns are transferred to a wafer using photomasks, also called *reticles*, by exposing them to UV-light. This is often a bottleneck for production due to the limited number of machines, the expensive equipment that is required, and the large number of wafers.

We are interested in scheduling solutions for photolithography to improve throughput by minimising the *makespan* of the schedule, i.e., the completion time of the last job on any machine. This problem can be modelled as job scheduling on unrelated parallel machines with machine dedication, sequence-dependent setup times, and release dates. Each job needs to be non-preemptively processed by exactly one machine that is capable of doing so (*machine dedication*). Every job has a *release date*, which is its earliest start time, and a *duration*. Also, each job requires a setup that depends on the previous job on the machine, which we model with *sequence-dependent setup times*. All times are machine dependent.

The problem so far is already quite challenging as we need to simultaneously solve an assignment and a sequencing problem. A further complication are the required reticles, which represent additional auxiliary-resource constraints: every job needs a specific reticle to be processed. As reticles are expensive, their number is limited, and jobs that require the same reticle cannot be processed at the same time. While there is some work on this problem, cf. the article by Bitar, Dauzère-Pérès, and Yugma (2021) and the literature review therein, a common assumption is that reticles can be moved individually between machines. This is not realistic in larger workcenters as reticles are quite fragile and often transportation systems for groups of reticles are in place. Our problem has, to the best of our knowledge, not been studied yet: we assume that pods of fixed size are used to safely store and transport reticles to machines. Transport robots are used for equipping entire pods with the right reticles from reticle stockers in time. As robots move slowly and pod changes are costly, jobs need to be carefully batched into groups on machines that only require reticles from a single pod.

Exact constraint programming (CP) or mixed-integer programming (MIP) solvers that use a declarative model are attractive as such models tend to be compact and easy to maintain when requirements change. Furthermore, valuable knowledge about the application is explicitly represented. We present a first solver-independent MiniZinc model for this problem, which is a high-level modelling language for constraint satisfaction and optimisation problems that can be used by a wide range of solvers (Nethercote et al. 2007). We evaluate this model on a new benchmark set consisting of 500 randomly generated instances of realistic structure.

Unfortunately, the requirements around reticles and their transportation renders this problem extremely difficult for state-of-the-art MIP and CP solvers (we used IBM CP Optimizer, IBM CPLEX, and Google OR-Tools in our experiments), and it turns out that only small instances can be solved. Consequently, we present this problem together with the benchmark instances as an *open challenge*. We believe that making progress on it has considerable potential for driving improvements in optimisation-based scheduling for this relevant industrial problem.

2 Related Work

For an overview on scheduling problems in semi-conductor manufacturing, we refer to the survey article by Mönch et al.

(2011). The problem of unrelated parallel machine scheduling with setup times and no reticles as auxiliary resources has been studied rather extensively (Allahverdi 2015). The benchmark instances we introduce in this work extend previous ones from recent work (Eiter et al. 2021) on a job scheduling problem with setup times but without reticle batching.

A problem quite similar to ours has been studied by Bitar, Dauzère-Pérès, and Yugma (2021). They considered setup times as well as auxiliary resource constraints, but reticles can be moved individually between machines. In particular, they modeled moving an auxiliary resource from one machine to another with a unitary transport time. Furthermore, release dates were not considered. One of their objectives is to minimise the number of jobs completed before a fixed horizon, and they provide a respective ILP model. A memetic algorithm for this problem has been introduced in previous work (Bitar et al. 2016).

An approach with dispatching rules and MIP for a problem similar to ours with auxiliary constraints and release dates but on identical machines is due to Cakici and Mason (2007). They minimised the weighted sum of completion times as the objective. A MILP model for unrelated parallel machine scheduling with unspecific resources for minimising makespan but without release dates was studied by Fanjul-Peyro (2020). Soares and Carvalho (2022) used a hybrid evolutionary algorithm for makespan minimisation on identical machines, but also without release dates.

Ham and Cho (2015) integrated real-time dispatching rules with linear programming techniques to solve photolithography scheduling problems. Their approach proceeds in two stages: first, they find the least-cost assignment of jobs to reticles and machines using a transportation model and MIP. Second, a sequencing module uses dispatching business rules to provide sequencing decisions.

To the best of our knowledge, considering pods of limited capacity for reticle storage and transportation from stockers is a novel and practically important aspect of this work that has not been addressed for exact solvers so far.

3 Formal Problem Statement

We study the following scheduling problem. Given m machines and n jobs, every job needs to be processed by a single machine, and every machine can process at most one job at a time; preemption is not allowed. Some machines can only handle certain jobs, such that from the view of the latter, $cap(j)$ is the set of machines that can process job j.

We assume that a *release date* $\alpha_{j,k}$ is specified for every job j and machine k as a non-negative integer. Release dates are machine dependent because transportation times for jobs to the machines depend on the transport system and their location. No job can start before its release date.

A specified amount of time may be required to change from one job to the next one. Specifically, we assume that $\beta_{i,j,k}$ is the time needed to set up job j directly after job i on machine k. Consequently, these times are referred to as *sequence-dependent setup times*. Every job j has a positive *duration* $d_{j,k}$ that depends on the machine k it is assigned to.

Furthermore, we consider o reticles and l stockers. Each reticle i is assigned to a stocker, denoted by stk_i. For each stocker i and machine k, we assume a transportation time $t_{i,k}$. Reticles are auxiliary resources required by jobs. Each job i requires exactly one reticle denoted by r_i. Finally, we also assume a pod capacity b denoting the number of different reticles that can be held by any machine and a *pod change time* γ_k for each machine k.

A *schedule* S for a problem instance is defined by:

1. An assignment a that maps each job j to a machine $k \in cap(j)$ capable of processing it;
2. A start time st_i for each job i. For each machine k, the start times of the jobs J assigned to k via a determine a total order $\preceq_k$ on J. Relation $\preceq_k$ determines the sequence in which the jobs in J are processed on k;
3. For each machine k, an equivalence relation $\sim_k$ on the set J_k of jobs assigned to the machine via a such that whenever $j_1 \sim_k j_2$ then there is no $i \not\sim_k j_1$ such that $j_1 \preceq_k i \preceq_k j_2$.

The equivalence classes of $\sim_k$ are also called the *batches* of machine k. Informally, a batch is a group of jobs that can be processed consecutively without having to change the reticle pod. For a batch i, the set of its reticles is given by $ret(i) := \{r_j \mid j \in i\}$. We enforce the pod capacity by requiring that $|ret(i)| \leq b$ for each batch i. Additionally, we require that all reticles in a batch come from the same stocker, i.e, $stk_{r_1} = stk_{r_2}$ for each $r_1, r_2 \in ret(i)$.

Assume that $j_1 \preceq_k \ldots \preceq_k j_l$ is the processing sequence of the jobs assigned to machine k in a given schedule. The *processing time* p_{j_i} of a job j_i is its duration plus the setup time for its predecessor (if one exists); i.e., $p_{j_1} = d_{j_1,k}$ and $p_{j_i} = \beta_{j_{i-1},j_i,k} + d_{j_i,k}$, for $i > 1$. The *completion time* c_{j_i} of job j_i is then $st_{j_i} + p_{j_i}$. We require that each job starts after its predecessor, i.e., $st_{j_i} \geq c_{j_{i-1}}$ for $i > 1$. Furthermore, to ensure each job starts after its release date on k, $st_{j_i} \geq \alpha_{j_i,k}$ has to hold.

For the schedule to be feasible, we also need to ensure that batches using the same reticle do not overlap and that the respective transportation times for reticles are considered. By abuse of notation, let $st_i := min(\{st_j \mid j \in i\})$ and $c_i := max(\{c_j \mid j \in i\})$ denote the start and the completion time of a batch i, respectively. Then, for any batches $i_1 \neq i_2$ where $ret(i_1) \cap ret(i_2) \neq \emptyset$, it holds that either $st_{i_1} \geq c_{i_2} + t_{stk,k_2} + t_{stk,k_1}$ or $st_{i_2} \geq c_{i_1} + t_{stk,k_1} + t_{stk,k_2}$, where k_1 and k_2 are the machines of i_1 and i_2 respectively, and stk is the reticle stocker containing the reticles of both batches. Furthermore, for consecutive batches on the same machine, we enforce a *pod change time* between them. Formally, for any two batches $i_1 \neq i_2$ on machine k, either $st_{i_1} \geq c_{i_2} + \gamma_k$ or $st_{i_2} \geq c_{i_1} + \gamma_k$.

4 MiniZinc Model

We implemented a solver-independent model for schedule optimisation in the well-known high-level modelling language MiniZinc for constraint satisfaction and optimisation problems (Nethercote et al. 2007). MiniZinc models, after being compiled into FlatZinc, can be used by a wide range of solvers. Our model of the problem statement from Section 3 and the objective function are as follows.

For each job $i \in \{1, \ldots, n\}$, we use the following decision variables: $a_i \in \{1, \ldots, m\}$ for its assigned machine,

$p_i \in \{1, \dots, n\} \cup \{-k \mid 1 \leq k \leq m\}$ representing its predecessor or its negative machine assignment if it has none, and $c_i \in \{0, \dots, h\}$ denoting its completion time, where h is the *scheduling horizon*, i.e., the latest considered timepoint.

Furthermore, we utilize some auxiliary decision variables: $s_i \in \{1, \dots, n\} \cup \{-k \mid 1 \leq k \leq m\}$ represents a job's successor or its negative machine assignment if it has none, $\sigma_i \in \{0, 1\}$ indicates whether a job starts a batch, and $\rho_{i,k} \in \{0, 1\}$ denotes that reticle k is used in the batch of job i.

The constraints of the MiniZinc model are given in Fig. 1. The global constraint (1) ensures that no two jobs have the same predecessor, and the next constraint (2) enforces that a job and its predecessor are assigned to the same machine. First jobs on machines which have no predecessor have their negative machine assignment as a dummy predecessor as required by constraint (3). Thus, those jobs do not interfere with constraint (1). Constraints (4) and (5) link the successor and predecessor variables. Constraint (6) expresses that no job is its own predecessor, while (7) ensures that every job is assigned to a capable machine. Constraint (8) ensures that each job starts after its predecessor, and (9) enforces that every job starts after its release date.

Batching is modelled by pod changes and blocked reticles. Constraint (10) enforces that the required reticle of a job is always blocked by said job. Whenever a job does not start a batch, its blocked reticles are passed on to its predecessor; this is modelled by constraint (11). Similarly, by constraint (12) blocked reticles are passed forward if the successor of a job does not start a batch. Maximum pod capacity is constrained by (13), and constraint (14) defines the three cases when a pod change is required. Namely (i) whenever a job is the first on its machine, (ii) the required reticle of the predecessor comes from a different stocker, or (iii) the preceding job already blocks the maximum number of reticles that can be in a batch, and the current one requires a new reticle.

To ensure that no reticle is used by multiple jobs at the same time, we use a global cumulative constraint (15). It takes as input the start times, durations, and resource requirements of a list of jobs and ensures that their resource assignments never exceed the given bound of 1. To ensure that idle times in a batch still block all reticles, $batchStart(i)$ is defined as the completion of the predecessor unless the job i starts the batch. For transportation, $batchDuration(i)$ adds the respective time for the first and last jobs of a batch.

The objective $min\ max_{1 \leq i \leq n}(c_i)$ expresses the standard makespan objective for the scheduling problem.

5 Experiments

Next, we present an evaluation of different solvers with our MiniZinc model on a set of randomly generated benchmark instances. The instance generator, the MiniZinc model, as well as the logs from our experiment are available online.[1]

Problem Instances

To create our benchmark suite, we amended instances previously introduced for the related problem of scheduling jobs on unrelated parallel machines with release dates and sequence-dependent setup times but without batching requirements (Eiter et al. 2021). The 500 benchmark instances for that problem have different sizes and were generated randomly but reflect relevant properties of the real instances.

The machine capabilities were assigned uniformly at random for half of the instances: for each job, a random number of machines were assigned as capable. For the other half, we assigned the capabilities such that 80% of the jobs can only be performed by 20% of the machines. The latter setting reflects high machine dedication and the former models low machine dedication.

For each job j and any machine k, the duration $d_{j,k}$, setup time $\beta_{j,i,k}$ for any other job i, and release date $\alpha_{j,k}$ were drawn uniformly at random from $[10, 500]$, $[0, 100]$, and $[0, r_{max}]$, respectively, where

$$r_{max} = \frac{1}{m} \sum_{1 \leq j \leq n} \frac{1}{|cap(j)|} \Big(\sum_{k \in cap(j)} d_{j,k} + \sum_{1 \leq j' \leq n, k \in cap(j')} \beta_{j',j,k} \Big).$$

We then set the number of reticles to $o = \lceil 0.8n \rceil$ and the number of stockers to $l = \lceil 0.4m \rceil$. The required reticle of a job as well as the stocker of a reticle are assigned uniformly at random. Lastly, for each stocker i and machine k, the transportation time $t_{i,k}$ was drawn uniformly from $[10, 100]$, the pod capacity of an instance was taken from $[5, 10]$, and the pod change times from $[50, 100]$.

Preliminary Results

We conducted all experiments on a cluster with 13 nodes, where each node has two Intel Xeon CPUs E5-2650 v4 (max. 2.90GHz, 12 physical cores, no hyperthreading) and 256GB RAM. For each run, we set a memory limit of 20GB, and all solvers only used one solving thread.

To solve our model we used the CP solvers IBM CP Optimizer 20.1[2], Google OR-Tools 7.8[3], as well the MIP solver IBM CPLEX 12.10[4].

Solver	#feasible	#best	#optimal
Google OR-Tools	45	45	26
CP Optimizer	55	39	23
CPLEX	36	16	14

Table 1: Number of instances for which feasible, best, and optimal solution were found for each solver with 15 minutes time limit per instance.

Table 1 gives an overview of the results for all 500 instances with a time limit of 15 minutes per instance. This rather short time limit is required by the workcenter because they need to reschedule frequently to react to incoming jobs or machine failures. Our experiments indicate that this is indeed a hard scheduling problem: no solver could find feasible solutions for more than 59 instances and at most 26 could be optimally solved. The largest instance any solver could

[1]http://www.kr.tuwien.ac.at/research/projects/bai/keps22.zip.

[2]https://www.ibm.com/analytics/cplex-cp-optimizer.

[3]https://developers.google.com/optimization.

[4]https://www.ibm.com/analytics/cplex-optimizer.

$$\texttt{alldifferent}(\langle p_i \rangle_{1 \leq i \leq n}) \tag{1}$$
$$p_i > 0 \rightarrow a_i = a_{p_i} \qquad 1 \leq i \leq n \tag{2}$$
$$p_i < 0 \rightarrow p_i = (-1) \cdot a_i \qquad 1 \leq i \leq n \tag{3}$$
$$p_i = j \leftrightarrow s_j = i \qquad 1 \leq i, j \leq n, i \neq j \tag{4}$$
$$s_i < 0 \rightarrow s_i = (-1) \cdot a_i \qquad 1 \leq i \leq n \tag{5}$$
$$p_i \neq i \qquad 1 \leq i \leq n \tag{6}$$
$$a_i \in cap(i) \qquad 1 \leq i \leq n \tag{7}$$
$$c_i \geq (max(c_j,\ \alpha_{i,a_i}) + \beta_{j,i,a_i} + d_{i,a_i} + (\gamma_i \cdot [\sigma_i = 1])) \cdot [p_i = j] \qquad 1 \leq i, j \leq n, i \neq j \tag{8}$$
$$c_i \geq \alpha_{i,a_i} + d_{i,a_i} \qquad 1 \leq i \leq n \tag{9}$$
$$\rho_{i,r_i} = 1 \qquad 1 \leq i \leq n \tag{10}$$
$$(\sigma_i = 0 \wedge \rho_{i,k} = 1) \rightarrow \rho_{p_i,k} = 1 \qquad 1 \leq i \leq n, 1 \leq k \leq o \tag{11}$$
$$(p_j = i \wedge \sigma_i = 0 \wedge \rho_{i,k} = 1) \rightarrow \rho_{j,k} = 1 \qquad 1 \leq i, j \leq n, i \neq j, 1 \leq k \leq o \tag{12}$$
$$\sum_{1 \leq k \leq o} \rho_{i,k} \leq b \qquad 1 \leq i \leq n \tag{13}$$
$$\Big(p_i < 0 \vee (p_i > 0 \wedge stk_{r_i} \neq stk_{p_i}) \vee (p_i > 0 \wedge \sum_{1 \leq k \leq o} \rho_{p_i,k} = b \wedge \rho_{p_i,r_i} = 0)\Big) \leftrightarrow \sigma_i = 1 \qquad 1 \leq i \leq n \tag{14}$$
$$\texttt{cumulative}(\langle batchStart(i) \rangle_{1 \leq i \leq n}, \langle batchDuration(i) \rangle_{1 \leq i \leq n}, \langle \rho_{i,k} \rangle_{1 \leq i \leq n}, 1) \qquad 1 \leq k \leq o \tag{15}$$

$$batchStart(i) := \begin{cases} c_i - d_{i,a_i} - [p_i > 0] \cdot \beta_{p_i,i,a_i} & \text{if } \sigma_i = 1, \\ c_{p_i} & \text{otherwise.} \end{cases}$$

$$batchDuration(i) := \begin{cases} d_{i,a_i} + t_{l_{r_i},a_i} + \beta_{p_i,i,a_i} & \text{if } (\sigma_i = 1 \wedge p_i > 0) \vee s_i < 0 \vee \sigma_{s_i} = 1, \\ d_{i,a_i} + [p_i > 0] \cdot \beta_{p_i,i,a_i} & \text{otherwise.} \end{cases}$$

Figure 1: Solver-independent MiniZinc model (for a proposition P, $[P] = 1$ if P is true, and $[P] = 0$ otherwise).

find solutions for contains only 24 jobs. A similar picture was obtained when using a run time limit of one hour on a representative selection consisting of 10% of the instances.

It seems that modelling that batches which use the same reticle do not overlap is difficult and has a big impact on both model compilation and search performance. For example, our way of modelling blocked reticles with an auxiliary variable for each job appears to be costly in terms of the number of constraints it generates. This leads to memory outage for bigger instances. The cumulative constraint (15) does not contribute to problems in compilation, but our intuition is that it makes it significantly harder for solvers to find solutions.

6 Discussion

We deal with optimisation-based scheduling for the practically relevant photolithography problem. Besides machine dedication, sequence-dependent setup times, and release dates, a characteristic feature that makes this problem challenging are auxiliary-resource constraints in the form of reticles. We study this problem under the realistic assumption that reticles need to be transported together such that given pod capacities are respected. In particular, we (i) present a formal problem description, (ii) give a solver-independent MiniZinc model, and (iii) provide new benchmark instances for this problem. Our experiments with MIP and CP solves confirm that the problem is hard for exact solvers, and we present it consequently as an open challenge.

Fur future work, there are different possible paths forward to address this challenge. Meta-heuristics, like simulated annealing or tabu search, are often successful for machine scheduling. However, if we want to use declarative models, options are more limited. Direct encodings instead of the MiniZinc model, other models as well as optimising solver settings might result in some improvements. Another direction is to rephrase the problem as online scheduling and use stream reasoning over a sliding time window to only schedule newly arriving jobs. Alternatively, we can use a selection strategy to schedule only a subset of jobs within a limited horizon. Also, a two-phase approach where we separate the model for machine assignments and reticles from the sequencing of jobs as in related work by Ham and Cho (2015) may be promising. Another idea is to use a large-neighbourhood search on top of a declarative model, possibly together with a greedy construction heuristic to start search, as demonstrated for a similar problem (Eiter et al. 2022).

Acknowledgements

This work was supported by funding from the Bosch Center for AI in Renningen, Germany. We would like to thank Sebastian Bayer, Michel Janus, and Matthias Werner for fruitful discussions on the considered photolithography problem.

References

Allahverdi, A. 2015. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2): 345–378.

Bitar, A.; Dauzère-Pérès, S.; and Yugma, C. 2021. Unrelated parallel machine scheduling with new criteria: Complexity and models. *Computers & Operations Research*, 132: 105291.

Bitar, A.; Dauzère-Pérès, S.; Yugma, C.; and Roussel, R. 2016. A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing. *Journal of Scheduling*, 19(4): 367–376.

Cakici, E.; and Mason, S. 2007. Parallel machine scheduling subject to auxiliary resource constraints. *Production Planning and Control*, 18(3): 217–225.

Eiter, T.; Geibinger, T.; Musliu, N.; Oetsch, J.; Skocovsky, P.; and Stepanova, D. 2021. Answer-Set Programming for Lexicographical Makespan Optimisation in Parallel Machine Scheduling. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning* (*KR 2021*), volume 18, no. 1, 280–290.

Eiter, T.; Geibinger, T.; Ruiz, N. H.; Musliu, N.; Oetsch, J.; and Stepanova, D. 2022. Large-Neighbourhood Search for Optimisation in Answer-Set Solving. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence* (*AAAI-22*). http://www.kr.tuwien.ac.at/research/projects/bai/aaai22.pdf.

Fanjul-Peyro, L. 2020. Models and an exact method for the unrelated parallel machine scheduling problem with setups and resources. *Expert Systems with Applications: X*, 5: 100022.

Ham, A. M.; and Cho, M. 2015. A practical two-phase approach to scheduling of photolithography production. *IEEE Transactions on Semiconductor Manufacturing*, 28(3): 367–373.

Mönch, L.; Fowler, J. W.; Dauzère-Pérès, S.; Mason, S. J.; and Rose, O. 2011. A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of Scheduling*, 14(6): 583–599.

Nethercote, N.; Stuckey, P. J.; Becket, R.; Brand, S.; Duck, G. J.; and Tack, G. 2007. MiniZinc: Towards a Standard CP Modelling Language. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming* (*CP 2007*), volume 4741 of *Lecture Notes in Computer Science*, 529–543. Springer.

Soares, L. C.; and Carvalho, M. A. 2022. Application of a hybrid evolutionary algorithm to resource-constrained parallel machine scheduling with setup times. *Computers & Operations Research*, 139: 105637.