Knopp, S., Dauzère-Pérès, S., & Yugma, C. (2017). A batch-oblivious approach for complex job-shop scheduling problems. *European Journal of Operational Research, 263*(1), 50-61 DOI: https://doi.org/10.1016/j.ejor.2017.04.050

A Batch-Oblivious Approach for Complex Job-Shop Scheduling Problems

Sebastian Knopp[1,3,*], Stéphane Dauzère-Pérès[1,2], Claude Yugma[1]

[1] *École des Mines de Saint-Étienne, CMP Georges Charpak*
Department of Manufacturing Sciences and Logistics
CNRS UMR 6158 LIMOS, F-13541, Gardanne, France

[2] *Department of Accounting, Auditing and Business Analytics*
BI Norwegian Business School, Oslo, Norway

[3] *AIT Austrian Institute of Technology*
Center for Mobility Systems
Dynamic Transportation Systems, Vienna, Austria

Abstract

We consider a Flexible Job-Shop scheduling problem with batching machines, reentrant flows, sequence dependent setup times and release dates while considering different regular objective functions. Semiconductor manufacturing is probably one of the most prominent practical applications of such a problem. Existing disjunctive graph approaches for this combined problem rely on dedicated nodes to explicitly represent batches. To facilitate modifications of the graph, our new modeling reduces this complexity by encoding batching decisions into edge weights. An important contribution is an original algorithm that takes batching decisions "on the fly" during graph traversals. This algorithm is complemented by an integrated move to resequence and reassign operations. This combination yields a rich neighborhood that we apply within a local search and a Simulated Annealing (SA) metaheuristic. The latter is embedded in a Greedy Randomized Adaptive Search Procedure (GRASP) which is the most efficient approach. Numerical results for benchmark instances of different problem types show the generality and applicability of our approach. The conciseness of our idea facilitates extensions towards further complex constraints needed in real-world applications.

Keywords: Scheduling, Flexible Job-Shop Scheduling with Batching, GRASP, Metaheuristics, OR in Semiconductor Manufacturing

[*] Corresponding author, Tel.: +43 (0) 664 6207692.
Email addresses: sebastian.knopp@ait.ac.at (Sebastian Knopp[1,3,]), dauzere-peres@emse.fr (Stéphane Dauzère-Pérès[1,2]), yugma@emse.fr (Claude Yugma[1])

1. Introduction

The production of microelectronic devices is a highly complicated and cost intensive process—particularly in the front-end where the fabrication of wafers takes place. A wafer is a thin slice of semiconductor material on which chips are manufactured. In a semiconductor manufacturing facility (*fab*), up to 600 processing steps in different work areas are required for the production of a single lot of wafers. We consider high-mix fabs where various types of products are in progress at the same time. Here, scheduling decisions have a strong impact on key performance indicators such as throughput and cycle time. The presence of batching machines within a Job-Shop environment characterizes our scheduling problem and leads to a general formulation with a wide range of applications not only in semiconductor manufacturing but also in other fields, in particular in manufacturing processes where furnaces are required, e.g. mould manufacturing. In our setting, a batching machine can simultaneously process multiple lots (up to a given maximum capacity) using the same processing duration. In this paper, we concentrate on the diffusion and cleaning area of a semiconductor manufacturing facility. This work area contains machines for wet cleaning, wet etching, thermal processes, deposition and oxidation. Its properties lead to a Flexible Job-Shop scheduling problem with p-batching, reentrant flows, sequence dependent setup times and release dates. We optimize different regular mono-objective functions from the literature such as makespan, total weighted tardiness and maximum lateness as well as other performance indicators that are relevant in industrial applications. A formal definition of the problem is given in Section 3.

The considered problem is NP-hard since it generalizes both the NP-hard classical Job-Shop scheduling problem as well as the NP-hard single-machine scheduling problem with total weighted tardiness objective (see Garey et al. (1976)). We aim to solve industrial instances with more than 80 machines (number of machines in the diffusion and cleaning area) and hundreds of jobs, each job consisting of up to ten operations. We develop a heuristic method since we want to solve large instances of an NP-hard problem.

Most existing solution approaches for Complex Job-Shop scheduling problems with batching machines rely on the disjunctive graph representation of Ovacik and Uzsoy (1997). This representation introduces dedicated nodes to represent batching decisions. We propose a novel batch-oblivious modeling which avoids additional batching nodes. Instead, we encode batching decisions in the weights of edges to reduce the structural complexity of the graph. This modeling is explained in detail in Section 4. Then, we introduce a novel integrated algorithm to compute start dates and to create batches that takes advantage of the batch-oblivious representation. This representation allows our integrated algorithm to make batching decisions during graph traversal. This can be used to "fill up" underutilized batches by applying a combined resequencing and reassignment strategy. In addition, an integrated batch-oblivious move is proposed to relocate individual operations. The combination of the algorithm and the batch-oblivious move yields a neighborhood that implicitly comprises specific moves known from the literature such as the swapping of batches. This neighborhood also includes more moves generated by the interplay of the algorithm and the integrated batch-oblivious move. These building blocks for heuristic methods are detailed in Section 5. We apply them within a GRASP based approach (Feo and Resende (1995)). We randomize the construction of initial solutions by successively inserting jobs using a randomly perturbed ordering of jobs. Solutions are improved using a Simulated Annealing heuristic. This GRASP based metaheuristic approach is presented in Section 6. Computational experiments using a parallelized implementation yield very good results for various types of instances. Numerical results are presented and discussed in Section 7, where new public industrial benchmark instances are proposed.

2. Related Work

Scheduling problems with batching are reviewed in the articles of Mathirajan and Sivakumar (2006) and Potts and Kovalyov (2000). An overview of scheduling challenges in semiconductor manufacturing is provided in Mönch et al. (2013) and Mönch et al. (2011). The scheduling of parallel batching machines and variants of the Job-Shop scheduling problem are well-studied problems whereas their combination is rarely considered. Starting with the work of Ovacik and Uzsoy (1997), several approaches for Complex Job-Shop scheduling problems are based on the shifting bottleneck heuristic of Adams et al. (1988). This heuristic decomposes the problem into multiple parallel-machine scheduling problems and subsequently applies appropriate subproblem solution procedures. For this setting, Ovacik and Uzsoy (1997) introduce a disjunctive graph representation that represents batches using dedicated nodes. This representation was also used in Mason et al. (2005) and Mönch and Rose (2004), where the authors show that a modified shifting bottleneck heuristic outperforms classical dispatching rules. Similar approaches are proposed in Upasani et al. (2006) and Sourirajan and Uzsoy (2007). Results were improved in Mönch et al. (2007) by using a genetic algorithm in the subproblem solution procedure. Another approach is presented in Yugma et al. (2012) which relies on batch specific moves, e.g. moving one batch or swapping operations from different batches. Again, batches are represented using dedicated nodes. In distinction to all these approaches, our approach uses a less complex disjunctive graph model without dedicated nodes, enabling a more holistic integration of batching decisions.

A mixed integer formulation for Complex Job-Shops with total weighted tardiness objective is given in Mason et al. (2005). Bilyk et al. (2014) propose an improved method to solve parallel-machine scheduling problems which appear as subproblems of shifting bottleneck based approaches. A sequential decomposition approach for Complex Job-Shops is presented in Guo et al. (2012) which apply an ant colony optimization heuristic. Scheduling in the diffusion and cleaning area of semiconductor manufacturing with its particular constraints is also addressed by Yurtsever et al. (2009), Kim et al. (2010) and Jung et al. (2013), however, they do not consider a Job-Shop environment. Job-Shop scheduling problems with sequence dependent setup times are an important subproblem of Complex Job-Shop scheduling problems. Good results for such problems were achieved by a tabu search based approach in González et al. (2013).

Hence, to our knowledge, our approach is the first one to handle the full complexity of complex job-shop scheduling problems with batching using disjunctive graphs and no dedicated nodes. Moreover, as shown in Section 4, we propose an original algorithm to build batches while traversing a conjunctive graph.

3. Problem Description

This section provides a formal definition of the considered Flexible Job-Shop scheduling problems with batching, reentrant flows, sequence dependent setup times and release dates (*Complex Job-Shops*). We aim to optimize regular objective functions which are formally defined later in this section. Using Graham's $\alpha|\beta|\gamma$ notation, this class of scheduling problems can be denoted as $FJc|r_j, s_{i,j}, B, recr|reg$. We consider batching and its embedding in a Job-Shop environment to be the main characteristics of our problem. In this paper, batching lots in a machine (up to the batching capacity of the machine) means that all lots in the batch are processed together, i.e. they have the same start times and completion times. Also, the processing duration does not depend on the number of lots in the batch. This type of batching is also called p-batching in the literature.

We are given a set of *jobs* J which have to be processed using a given set of *machines* M. For each job $j \in J$, we are given a set of *operations* $O_j = \{o_{1,j}, o_{2,j}, \dots, o_{|O_j|,j}\}$, and a *release date* $r_j \in \mathbb{Z}$. The disjoint union $O = O_1 \dot\cup O_2 \dots \dot\cup O_{|J|}$ denotes all given operations. For a given set of *recipes* R, each recipe $r \in R$ prescribes a machine $m_r \in M$, a *processing duration* $p_r \in \mathbb{N}_0$ and a *batching capacity* $b_r \in \mathbb{N}_{>0}$. In our industrial case, the recipe of a process operation (see Johnzén et al. (2011)) precisely defines how the machine should conduct the process: Duration, temperatures, gas flow and pressure, etc. Note that a recipe is associated to only one machine. For a given set of *families* F, each family $f \in F$ specifies a subset of eligible recipes $R_f \subset R$. Each operation $o_{i,j} \in O$ is assigned to a family $f_{i,j} \in F$. We denote by $R_{i,j}$ the eligible recipes for a family $f_{i,j} \in F$ of an operation $o_{i,j} \in O$. In our industrial case, a family helps to identify operations that have similar characteristics, and thus lots that can be grouped in the same batch. Hence, we have to select for each operation $o_{i,j} \in O$ one recipe out of $R_{i,j}$. A given mapping $s : F \times F \to \mathbb{N}_0$ prescribes *sequence-dependent setup times* between operations that are scheduled on the same machine.

A *schedule* is completely characterized by selecting recipes $q_{i,j} \in R_{i,j}$ and *start times* $S_{i,j} \in \mathbb{Z}$ for all given operations $o_{i,j} \in O$. We denote the machines, processing durations and batching capacities related to this selection as $m_{i,j}$, $p_{i,j}$ and $b_{i,j}$, respectively. To describe a schedule that is *feasible*, selected recipes $q_{i,j}$ and start dates $S_{i,j}$ of operations $o_{i,j}$ have to respect several constraints that are detailed in the following. Preemption is not allowed: Once the processing of an operation has begun, it cannot be interrupted. Thus, the *completion time* of an operation is given by $C_{i,j} = S_{i,j} + p_{i,j}$. Operations belonging to the same job have to be performed in the order given by the *route* of the job. So, $C_{i,j} \leq S_{i+1,j}$ has to be fulfilled for all $o_{i,j} \in O$ with $i < |O_j|$. Operations performed on the same machine must not overlap. Hence, for two operations $o_{i,j}, o_{k,l} \in O$ with $m_{i,j} = m_{k,l}$, either $S_{i,j} = S_{k,l}$ or $S_{i,j} \geq C_{k,l}$ or $C_{i,j} \leq S_{k,l}$ must hold. Only operations of the same family can be included in a common batch. So, for two operations $o_{i,j}, o_{k,l} \in O$ with $f_{i,j} \neq f_{k,l}$ and $m_{i,j} = m_{k,l}$, we require $S_{i,j} \neq S_{k,l}$. Batching capacities limit the number of operations per batch. Thus, we require $|\{o_{k,l} \in O \mid S_{k.l} = S_{i,j} \wedge m_{k,l} = m_{i,j}\}| \leq b_{i,j}$ for all operations $o_{i,j} \in O$. The first operation $o_{1,j} \in O_j$ of each job cannot be processed before its release date, so $S_{1,j} \geq r_j$ must hold for all $j \in J$. To respect sequence-dependent setup times, for all operations $o_{i,j}, o_{k,l} \in O$ with $m_{i,j} = m_{k,l}$ and $S_{i,j} \neq S_{k,l}$ either $C_{i,j} + s(f_{i,j}, f_{k,l}) \leq S_{k,l}$ or $C_{k,l} + s(f_{k,l}, f_{i,j}) \leq S_{i,j}$ must hold.

Our goal is to determine schedules that optimize regular objective functions. An *objective function* is a function $f : \mathbb{R}^{|O|} \to \mathbb{R}$ that maps tuples of operation start dates to a real number. We call an objective function *regular* (see Conway et al. (1967) and Brucker (2007)), if for two given tuples of start dates $(S_1, \ldots, S_{|O|}), (S'_1, \ldots, S'_{|O|}) \in \mathbb{R}^{|O|}$ with $S_1 \leq S'_1 \wedge \cdots \wedge S_{|O|} \leq S'_{|O|}$ it follows that $f(S_1, \ldots, S_{|O|}) \leq f(S'_1, \ldots, S'_{|O|})$. Intuitively speaking, the quality of a schedule cannot deteriorate by advancing the start date of some of its operations. Most objective functions considered in the scheduling literature (e.g., makespan, maximum lateness, total weighted completion time, or total weighted tardiness) are regular objective functions.

We have provided a concise formal definition of a Complex Job-Shop scheduling problem which generalizes several scheduling problems defined in the literature: It reduces to a Flexible Job-Shop scheduling problem if all batching capacities are equal to one, and to a scheduling problem with parallel batching machines if the routes of all jobs contain only a single operation. In the latter case, the flexibility of batching machines allows each job to be assigned to any machine. Recurrent flows are comprised in the definition: No constraint forbids to reuse a machine for multiple operations of the same job. Note that some objective functions might depend on due dates $d_j \in \mathbb{Z}$ or weights $w_j \in \mathbb{R}$ which may be associated to each job $j \in J$. These parameters were not explicitly included in the formal definition above since they do not impose any hard constraint on schedules. Yet they can be integrated in the definition of an objective function.

4. Disjunctive Graph Modeling

Disjunctive graphs, introduced by Roy and Sussmann (1964), allow combinatorial properties of schedules to be represented in a concise way and have been applied to solve a broad range of scheduling problems. To tackle the inclusion of p-batching within Job-Shop environments, we introduce in this section a *batch-oblivious* disjunctive graph representation which is designed to facilitate decision-making on batches during graph traversals. First, we recall the disjunctive graph model for Complex Job-Shops. Then, two alternative representations for batching decisions are described: In Section 4.1, we recall and discuss an established representation which inserts dedicated batching nodes into the graph (see Ovacik and Uzsoy (1997)). In Section 4.2, a novel, batch-oblivious representation is introduced which modifies edge weights instead of introducing auxiliary nodes. This batch-oblivious representation helps us to make batching decisions on the fly (during graph traversal). This idea provides the foundation for the scheduling approach proposed in this paper.

Disjunctive graphs represent structural properties of schedules and model assignment, sequencing or batching decisions. *Conjunctive graphs* encode all decisions to be taken and are the principal tool for our algorithms. The basic graph representation corresponds to the one introduced in Dauzère-Pérès and Paulli (1997) for flexible (called multiprocessor in Dauzère-Pérès and Paulli (1997)) job-shop scheduling problems. Let us briefly recapitulate those graph types. In both cases, each node represents an operation and each edge represents a dependency induced by either the route of a job or the sequencing decisions for two operations assigned to the same machine. Disjunctive graphs model all possible assignments of operations to machines and all possible sequences of operations on the machines using undirected edges. By replacing undirected by directed edges while satisfying some feasibility constraints, a conjunctive graph is constructed which corresponds to an assignment of each operation to one machine and to a sequencing of the assigned operations on each machine. Redundant directed edges are removed in the conjunctive graph. Next, we provide a definition of a basic conjunctive graph representation that still neglects the representation of batching decisions.

A *conjunctive graph* $G = (V, E)$ is an acyclic directed graph with nodes $V = O \cup \{0, *\}$ that correspond to the given operations O plus an artificial start node 0 and an artificial end node $*$. For each job and each machine, the graph contains one path from the artificial start node 0 to the artificial end node $*$. The disjoint union of those paths yields all edges of the graph. Each node $v \in O$ is part of exactly two paths: One corresponding to the route of its job and one corresponding to the sequence of the machine it is assigned to. For a node $v \in O$, we denote its *route successor* by $r(v) \in V \setminus \{0\}$ and its *machine successor* by $m(v) \in V \setminus \{0\}$. Analogously, its predecessors are denoted by $r^{-1}(v) \in V \setminus \{*\}$ and $m^{-1}(v) \in V \setminus \{*\}$. The artificial start node 0 has $|J| + |M|$ outgoing edges and no incoming edges. Analogously, the artificial end node $*$ has $|J|+|M|$ incoming edges and no outgoing edges. Overall, the graph contains $|E| = 2\,|O| + |J| + |M|$ edges.

A conjunctive graph can be used to determine start dates S_o of operations $o \in O$. A weight $l_{u,v} \in \mathbb{N}_0$ is assigned to each edge $(u, v) \in E$ in order to ensure a minimum duration between the beginning of adjacent operations: $S_v \geq S_u + l_{u,v}$ for each edge $(u, v) \in E$. Having this, start dates of operations correspond to distances of longest paths from the artificial start node with respect to those edge weights. We denote the distance of a longest path from a node $v \in V$ to a node $w \in V$ by $L(v, w) \in \mathbb{N}_0$. For each operation $v \in O$, its start date is determined by $S_v = L(0, v)$. To reflect the given constraints, we define edge weights as follows. For edges $(0, o_{1,j}) \in E$ connecting the artificial start node 0 with the initial operation $o_{1,j}$ of a job j, the edge weight is set to the release date r_j of job j. For edges $(0, o_m) \in E$ connecting the artificial start node 0 with the initial operation o_m scheduled on machine $m \in M$, the edge weight is set to zero. For route edges $(v, r(v)) \in E$ with $v \neq 0$, the edge weight is set to the processing duration p_v of operation v. For machine edges $(v, m(v)) \in E$ with $v \neq 0$ of non-batching machines, the edge weight is set to the sum $p_v + s(v, m(v), m_v)$ of the processing duration of v and the sequence-dependent setup time between v and $m(v)$ on machine $m_v = m_{m(v)}$.

Now, what remains is to provide a representation for batching machines. They can be either modeled by modifying the structure of the graph (*batch-aware*) or by adapting the weights of edges (*batch-oblivious*). The following two subsections present both alternatives. Recall that each batch has to respect the capacity of the machine as well as the equality of involved families. The adherence to those constraints has to be guaranteed for each schedule. The related checks are not detailed in this section in order to focus on the essential parts of both representations.

4.1. Batch-Aware Conjunctive Graphs

This section reviews a batch-aware conjunctive graph representation that was introduced by Ovacik and Uzsoy (1997). All solution approaches for Complex Job-Shop scheduling problems that we are aware of make use of this type of representation (e.g., Mason et al. (2005), Mönch et al. (2003), or Yugma et al. (2012)).

A batch is a set of operations $B \subset O$ that is processed simultaneously on the same machine. In the batch-aware representation, for each batch, an additional node b is added to the graph. The start date of this batching node is taken as the common start date for all operations contained in the batch. A batch requires all of its operations to be ready before it can begin processing. To reflect this, each operation node $v \in B$ is connected to the batching node b via an edge (v, b) of weight zero. Then, operations following in the routes of involved jobs are connected as follows: For each operation $v \in B$ of the batch, an edge $(b, r(v))$ from the batching node to the route successor $r(v)$ of v is introduced. Those edges are given the processing duration of p_v as their weight. Two additional edges $(m^{-1}(b), b)$ and $(b, m(b))$ are introduced to order the batch in the sequence of operations on machine m_b. Analogously to the non-batching case, the weight of each machine edge (u, w) is defined by the sum $p_u + s(u, w, m_u)$. Each operation node $v \in B$ has exactly one incoming edge and one outgoing edge. The batching node b has $|B| + 1$ incoming edges and $|B| + 1$ outgoing edges.

Batch-aware conjunctive graphs represent dependencies stemming from batching decisions in a structural way. The number of nodes in those graphs depends on the number of batches. This structure renders modifications of batching decisions complicated to handle: The number of nodes in the graph must be adapted and several edges have to be manipulated while the acyclicity of the graph must be preserved.

4.2. Batch-Oblivious Conjunctive Graphs

In the following, we introduce a novel representation for batching decisions in conjunctive graphs which is non-intrusive regarding the structure of the graph. No dedicated batching nodes are introduced and the basic representation presented at the beginning of this section can remain as is. Our only means to represent batching decisions is to adapt the weights of machine edges $(v, m(v)) \in E$. The weight of a machine edge is set to zero if its adjacent operations should be processed in the same batch. Otherwise, the edge weight is set to $p_v + s(v, m(v), m_v)$, as in the non-batching case. Unfortunately, it is not that simple: $l_{v,m(v)} = 0$ only guarantees that $S_v \leq S_{m(v)}$ but not that $S_v = S_{m(v)}$. Since the start dates of all operations in a batch must be equal, setting edge weights to zero can lead to infeasible solutions. In the following, we develop a simple criterion that decides on the feasibility of zero weighted machine edges.

First, let us reconsider a general property of longest paths in directed acyclic graphs. The start date of a node $v \in V$ directly depends on the start dates of its predecessors as follows:

$$S_v = \max_{(u,v) \in E} (S_u + l_{u,v}). \tag{1}$$

Now, consider two operations $v \in O$ and $m(v) \in O$ that might be scheduled in the same batch. The node $m(v) \in V$ has two incoming edges coming from its machine predecessor v and its route predecessor $w = r^{-1}(m(v))$. We can apply equation (1) to obtain

$$S_{m(v)} = \max(S_v + l_{v,\,m(v)},\ S_w + l_{w,\,m(v)}). \tag{2}$$

If the length $l_{v,\,m(v)}$ of the machine edge $(v, m(v)) \in E$ is set to zero, we want to obtain $S_v = S_{m(v)}$ from a longest path computation. So, let us assume that $l_{v,\,m(v)} = 0$ and $S_v = S_{m(v)}$. With equation (2), we obtain

$$S_v = \max(S_v,\ S_w + l_{w,\,m(v)}) \iff S_v \geq S_w + l_{w,\,m(v)}. \tag{3}$$

This means (3) is a necessary condition to combine v and $m(v)$ in the same batch. Thus, in batch-oblivious disjunctive graphs, we require the *invariant*

$$\big(l_{v,\,m(v)} = 0 \ \wedge\ S_v \geq S_w + l_{w,\,m(v)}\big) \ \vee\ \big(l_{v,\,m(v)} = p_v + s(v, m(v), m_v)\big) \tag{4}$$

to be fulfilled for all nodes $v \in V$ and $w \in V$ with $w = r^{-1}(m(v))$. It follows that, for each operation $v \in V$, a longest path computation schedules the machine successor operation $m(v)$ either at the same time as v or at a later point in time where processing durations and sequence-dependent setup times are satisfied. This property propagates in a natural way: Multiple operations belonging to the same batch are connected in a path of zero weighted machine edges. Next, we want to show that each optimal schedule can be represented using our batch-oblivious conjunctive graph representation.

Theorem 1. *For any given regular criterion, there exists a batch-oblivious conjunctive graph G with edge weights $l : V \to \mathbb{N}_0$ such that longest paths in this graph represent an optimal schedule.*

Proof. Consider a feasible schedule that is optimal with respect to the given regular criterion. We denote the operation start dates of this optimal schedule by S_v. Now, we construct a batch-oblivious conjunctive graph that defines the assignment and ordering of operations on the machines as follows:

a) The graph respects all machine assignment decisions of the optimal schedule.
b) Ordering decisions on the machines respect the start dates of the optimal schedule: If $S_v < S_w$ for $v \in V$ and $w \in V$, then v is ordered before w.
c) Nodes $v \in V$ and $w \in V$ that are part of the same batch (i.e. $S_v = S_w$) are ordered as follows: If $S_{r^{-1}(v)} + l_{r^{-1}(v),\,v} < S_{r^{-1}(w)} + l_{r^{-1}(w),\,w}$, then v is ordered before w.
d) For two nodes $v \in V$ and $w \in V$ of the same batch with $S_{r^{-1}(v)} + l_{r^{-1}(v),\,v} = S_{r^{-1}(w)} + l_{r^{-1}(w),\,w}$, their relative order can be arbitrarily decided as long as no cycle is introduced.

Since those rules are derived from a feasible schedule, this graph is constructed without any cycle. Edge weights are set according to the batching decisions in the given optimal schedule. Property c) guarantees that, for all adjacent nodes $v \in V$ and $m(v) \in V$ of the same batch, $S_{r^{-1}(m(v))} + l_{r^{-1}(m(v)),\, m(v)} \leq S_v$ holds. Thus, invariant (4) holds for all edges of the graph. □

Figure 1 shows an example that allows to compare batch-aware and batch-oblivious representations. It shows a schedule with three jobs A, B and C using two machines. We see two batches processed on machine 2, each consisting of two operations: One is composed of operation 1 plus operation 4, another one is composed of operation 5 plus operation 8. For brevity of notation, sequence-dependent setup times have been omitted and let us denote $p_{1,4} = p_1 = p_4$ and $p_{5,8} = p_5 = p_8$. Note that invariant (4) is not visualized in Figure 1 (b), so assume that $S_1 \geq r_B$ and $S_5 \geq S_7 + p_7$.

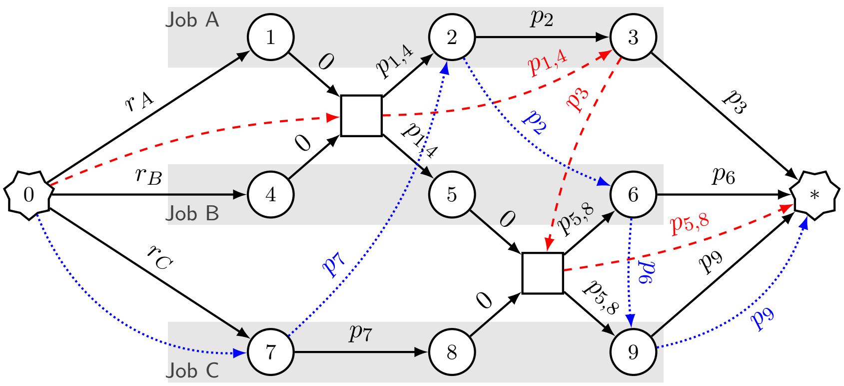
5. Building Blocks for Integrated Batching Decisions

This section develops the foundation of our heuristic approach. First, we describe in Section 5.1 how start dates can be computed from a given batch-oblivious conjunctive graph. The graph will remain structurally unchanged in this first version. Then, we define a general move in Section 5.2. It moves individual operations and is designed to be complemented by the interleaved start date computation and graph modification that we introduce in Section 5.3. This interleaved computation advances suitable nodes to "fill up" incomplete batches. Overall, our method integrates adaptive batching decisions with one general move to resequence and reassign operations. It can be used as a building block for metaheuristic approaches.

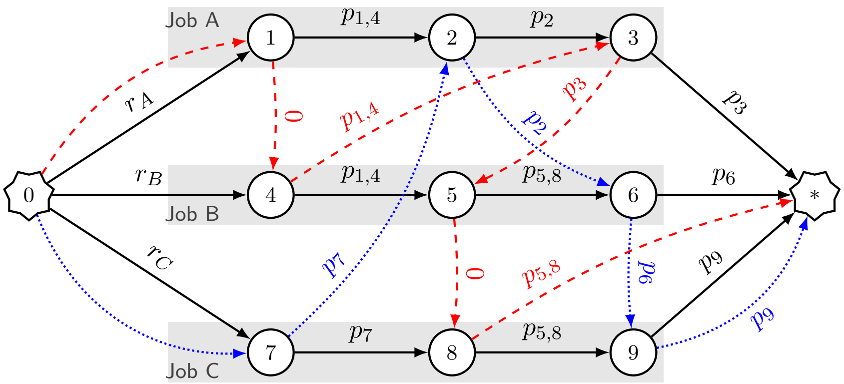
5.1. Static Start Date Computation

Let us first describe how start dates of operations can be computed from a given batch-oblivious conjunctive graph. For this, batching decisions are taken dynamically ("on the fly") during a traversal of the graph by deciding the weights of edges. Thereby, it is important to preserve invariant (4) introduced in Section 4. In contrast to the adaptive start date computation presented in Section 5.3, this static algorithm does not modify the graph itself: It preserves all ordering and assignment decisions inherent in the given conjunctive graph. The ordering is relaxed in the sense that a directed edge $(u, v) \in E$ requires only that operation v must not be processed before operation u. So, u and v might start at the same time which means they are part of the same batch.

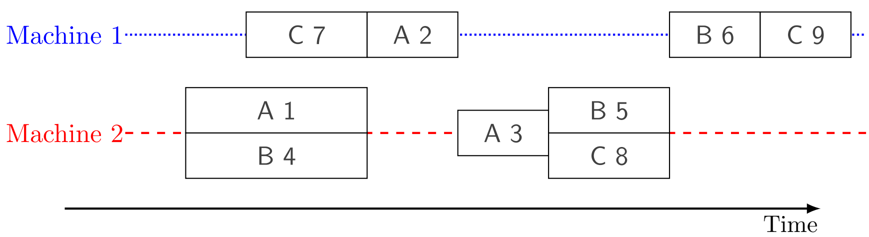
The computation is based on topological orderings. In an acyclic directed graph, a *topological ordering* can be defined as a relation $\prec \subseteq V \times V$ with $v \prec w \Rightarrow \nexists$ a path from w to v. Thus, traversing a conjunctive graph in



(a) Batch-Aware Conjunctive Graph



(b) Batch-Oblivious Conjunctive Graph



(c) Gantt Chart

Figure 1: A comparison of alternative representations of the same schedule

topological order guarantees that, for each node $v \in O$, both predecessors $r^{-1}(v)$ and $m^{-1}(v)$ are visited before v. Hence, the inductive formula for S_v given in equation (1) can be applied.

Algorithm 1 A static batching algorithm for a given conjunctive graph G

computeStartDatesStatically (G)
  $S_0 \leftarrow 0$
  $\beta_v \leftarrow 1 \quad (\forall\, v \in V)$
  for $v \in \mathit{computeTopologicalOrdering}\,(G \setminus \{0\})$
    if $(S_{r^{-1}(v)} + p_{r^{-1}(v)} \leq S_{m^{-1}(v)}$ **and** $f_{m^{-1}(v)} = f_v$ **and** $\beta_{m^{-1}(v)} < b_v)$
      $S_v \leftarrow S_{m^{-1}(v)}, \quad \beta_v \leftarrow \beta_{m^{-1}(v)} + 1$
    else
      $S_v \leftarrow \max\left(S_{r^{-1}(v)} + p_{r^{-1}(v)},\ S_{m^{-1}(v)} + p_{m^{-1}(v)} + s(m^{-1}(v), v)\right)$

Algorithm 1 provides the pseudo-code for a static graph evaluation algorithm. It tracks the used capacity β for each node and checks if the families $f_v \in F$ and $f_{m^{-1}(v)} \in F$ of consecutive operations are equal. The algorithm greedily creates batches while preserving the invariant for batch-oblivious conjunctive graphs. This corresponds to a longest path computation with dynamically specified edge weights. The computation takes $O(|E|)$ time since a topological ordering in a conjunctive graph can be computed in $O(|E|)$ and each node is visited exactly once. Batching decisions are taken greedily and strongly depend on the structure of the given graph.

5.2. An Integrated Batch-Oblivious Move

In order to develop heuristic algorithms, we want to introduce moves which modify a given batch-oblivious conjunctive graph. Known heuristic approaches we are aware of employ specific knowledge about previous batching decisions. E.g. they explicitly displace, combine or split entire batches, or they exchange operations belonging to different batches (Bilyk et al. (2014); Yugma et al. (2012)). To keep it simpler, we follow a different strategy and maintain the batch-obliviousness of our graph also for our moves. An observation from Section 4 is that, except of its edge weights, our conjunctive graph representation does not differ from a conjunctive graph representation for Flexible Job-Shop scheduling problems without batching. This allows us to apply the move introduced in Dauzère-Pérès and Paulli (1997) which integrates the resequencing and reassignment of operations. We include a detailed description of this move in this paper not only for completeness, but also to adapt it to our notation and to show that it remains valid for redefined edge weights.

Assume that all batching decisions have been taken and thus edge weights and start dates have been fixed. An operation v is moved after another operation w as follows: First, the machine related conjunctive edges $(m^{-1}(v), v) \in$

E and $(v, m(v)) \in E$ of operation v are replaced by an edge $(m^{-1}(v), m(v))$. Then, operation v is reinserted after operation w by replacing the edge $(w, m(w)) \in E$ with two edges (w, v) and $(v, m(w))$. In the graph that is created by executing the move, we then have $m(w) = v$. Recall that m_w must be a machine for which v is qualified to be processed on (i.e. $\exists\, q \in R_v : m_q = m_w$). To be computed efficiently, the feasibility check of a move relies on start dates of operations as shown in the following. Let us denote by $l_v = \min\left(l_{v,\,m(v)}, l_{v,\,r(v)}\right)$ the minimum weight of the outgoing edges of a node $v \in O$.

Theorem 2. *(Dauzère-Pérès and Paulli (1997)) An operation $v \in O$ can be moved between two operations w and $m(w)$ with $w \neq r(v)$ and $m(w) \neq r^{-1}(v)$ if $S_{r(v)} + l_{r(v)} > S_w$ and $S_{m(w)} + l_{m(w)} > S_{r^{-1}(v)}$.*

Proof. When v is removed, the edges $(m^{-1}(v), v)$ and $(v, m(v))$ are replaced by an edge $(m^{-1}(v), m(v))$ which cannot introduce a cycle. When v is reinserted after w, there are only two possible ways to create a cycle:

a) There was a path from operation $r(v)$ to operation w before moving v. This implies $S_{r(v)} + l_{r(v)} \leq S_w$, which contradicts the first assumption.

b) There was a path from operation $m(w)$ to operation $r^{-1}(v)$ before moving v. This implies $S_{m(w)} + l_{m(w)} \leq S_{r^{-1}(v)}$, which contradicts the second assumption.

□

Note that the original theorem has been adapted to include redefined edge weights. This move has been successfully applied to solve Flexible Job-Shop scheduling problems without batching and is not restricted to a particular objective function. However, in our case which includes batching, those moves might tear apart batches. This might lead to poor solutions containing unfavorable batches of only a single operation. Thus, to escape from a local optimum, a sequence of moves might need to strongly deteriorate a given solution before it can improve it again. The following subsection shows how we tackle this problem.

5.3. Adaptive Start Date Computation

To improve batching decisions, we interleave the computation of start dates with modifications of the batch-oblivious conjunctive graph. In particular, we want to improve schedules created by the moves described in Section 5.2 by "filling up" batches with remaining machine capacity: We advance suitable nodes by removing and reinserting them in the graph. In Algorithm 1 of Section 5.1, a topological ordering is computed first and then all nodes are traversed in this order. This is not viable anymore if we modify the graph while traversing it. Thus, we propose to interleave the computation of a topological ordering with a dynamic modification of the graph. We will see in the following that this idea can be described in terms of unidirectional cuts. During the course of our algorithm, unidirectional cuts distinguish unsettled nodes that still can be modified from settled nodes that are fixed. Finally, we propose in Section 5.4 different strategies for such adaptive graph modifications.

Definitions and Notation. For a given batch-oblivious conjunctive graph $G = (V, E)$, we consider a *cut* $V_s \subseteq V$ that partitions the graph into a subset of *settled* nodes V_s and a subset of *unsettled* nodes $V_u = V \setminus V_s$. A cut V_s is called *unidirectional* if there are no edges from an unsettled node to a settled node, i.e. $E \cap (V_u \times V_s) = \emptyset$. Let us denote by $G_s = (V_s, E_s)$ and $G_u = (V_u, E_u)$ the resulting subgraphs. The edges of each graph $G_\star \in \{G_s, G_u, G\}$ are given by $E_\star = E \cap (V_\star \times V_\star)$. Let us denote for a node $v \in V_\star$ its indegree in $G_\star$ by $deg^-_\star(v)$ and its outdegree in $G_\star$ by $deg^+_\star(v)$. A node $v \in V_\star$ without incoming edges (i.e. $deg^-_\star(v) = 0$) is called a *root node* of $G_\star$. A node $v \in V_\star$ without outgoing edges (i.e. $deg^+_\star(v) = 0$) is called a *leaf node* of $G_\star$.

Proposition 1. *In a conjunctive graph $G = (V, E)$, $V_s = \{0\}$ is a unidirectional cut.*

Proof. Since the only settled node $0 \in V_s$ is a root in G, no edges can end in a settled node. □

To *settle* a node $v \in V_u$ with $r^{-1}(v) \in V_s$ after a leaf node $w \in V_s$ of G_s, v is removed from G_u and appended to G_s. If $m^{-1}(v) = w$, the operation remains assigned to machine m_v sequenced after the same machine predecessor w. In this case, no edges need to be modified. Otherwise, if $m^{-1}(v) \neq w$, we modify the graph G as follows: The machine related conjunctive edges $(m^{-1}(v), v) \in E$ and $(v, m(v)) \in E$ of operation v are replaced by an edge $(m^{-1}(v), m(v))$ and the edge $(w, m(w)) \in E$ is replaced by two edges (w, v) and $(v, m(w))$. Settling a node does not change any route edges. If $m^{-1}(v) \neq w$ and $m_v = m_w$, then v is *resequenced.* If $m^{-1}(v) \neq w$ and $m_v \neq m_w$, then v is *reassigned.* Note that we require for a node $v \in V_u$ to be reassigned after a node $w \in V_s$ that $\exists\, q \in R_v$ with $m_q = m_w$.

Theorem 3. *Let $G = (V, E)$ be a conjunctive graph and let V_s be a unidirectional cut in G. When a node $v \in V_u$ with $r^{-1}(v) \in V_s$ is settled after a leaf node $w \in V_s$ of G_s, the modified graph $G' = (E', V')$ does not contain any cycle and $V'_s = V_s \cup \{v\}$ is a unidirectional cut in G'.*

Proof. When $v \in G_u$ is settled, three edges from $E \setminus E_s$ are deleted and the edges $(m^{-1}(v), m(v))$, (w, v) and $(v, m(w))$ are inserted. Edge deletions can neither introduce a cycle, nor invalidate any unidirectional cut. Since V_s is a unidirectional cut in G and $v \in V_u$, it follows that $m(v) \in V_u$ and $r(v) \in V_u$. With $r(v) \in V'_u$ and $m(w) \in V'_u$, we conclude that v is a leaf in G'_s. Since the predecessors of v in G' are settled, i.e. $r^{-1}(v) \in V'_s$ and $w \in V'_s$, edges adjacent to v cannot invalidate the unidirectional cut V'_s. The only inserted edge that is not adjacent to v in G', $(m^{-1}(v), m(v))$, does not invalidate the unidirectional cut since $m(v) \in V_u$. Thus, V'_s is a unidirectional cut in G'.

It remains to show that no cycle is introduced in G'. Since $v \in V'_s$, the edge $(m^{-1}(v), m(v))$ is the only inserted edge that might be contained in the subgraph G'_u. It cannot introduce a cycle since it replaced the edges $(m^{-1}(v), v)$ and $(v, m(v))$. Thus, the subgraph G'_u is acyclic. Both edges $(r^{-1}(v), v) \in E'$ and $(w, v) \in E'$ that are added to G'_s end in the node v. Since v is a leaf in G'_s, this cannot introduce a cycle in G'_s. Thus, the subgraph G'_s is acyclic. Overall, since G'_u and G'_s are acyclic, a cycle in G' must include an edge from V'_u to V'_s. Such an edge cannot exist since V'_s is a unidirectional cut in G'. □

Algorithm 2 An adaptive batching algorithm for a given conjunctive graph G

computeStartDatesAdaptively (G)
  $S_0 \leftarrow 0$
  $V_s = \{0\}$
  $\beta_v \leftarrow 1 \quad (\forall\, v \in V)$
  while $V_s \neq V$
    $v,\, w \leftarrow$ select $(v \in V \setminus V_s,\, w \in V_s)$
    assert $(r^{-1}(v) \in V_s$ **and** $deg_s^+(w) = 0)$
    settle v after w
    if $(S_{r^{-1}(v)} + p_{r^{-1}(v)} \leq S_{m^{-1}(v)}$ **and** $f_{m^{-1}(v)} = f_v$ **and** $\beta_{m^{-1}(v)} < b_v)$
      $S_v \leftarrow S_{m^{-1}(v)}, \quad \beta_v \leftarrow \beta_{m^{-1}(v)} + 1$
    else
      $S_v \leftarrow \max\big(S_{r^{-1}(v)} + p_{r^{-1}(v)},\ S_{m^{-1}(v)} + p_{m^{-1}(v)} + s(m^{-1}(v), v)\big)$
    $V_s \leftarrow V_s \cup \{v\}$

Algorithm 2 shows how the results on unidirectional cuts can be applied to interleave the computation of start dates with modifications of the graph. Initially, only the artificial start node 0 is considered to be settled. Then, nodes that meet the criteria of Theorem 3 can be successively settled without introducing any cycle. The start dates of settled nodes are calculated as proposed in Algorithm 1. The quality of the resulting schedule and the efficiency of the algorithm strongly depends on the selection of the nodes v and w. In the following, we propose and analyze three selection strategies.

5.4. Strategies for Selecting Nodes

A Static Selection Strategy. A straightforward selection strategy chooses in each step a root node $v \in V_u$ of G_u and settles it after its machine predecessor $w = m^{-1}(v)$. This strategy does not modify the graph and iterates the nodes in topological order. Algorithm 2 with this static selection strategy is equivalent to Algorithm 1 presented in Section 5.1. In order to implement this strategy, we need to determine root nodes of G_u efficiently. This has been done by applying the approach of Kahn (1962). It maintains the indegree in G_u for each node of the graph G and a list containing all root nodes in G_u. When a node is settled, it is removed from the list of root nodes in G_u and, for each of its successor nodes, the number of incoming edges in G_u is decreased. These successor nodes $v \in V_u$ are added to the list of root nodes when their indegree in G_u becomes zero. Since these auxiliary data structures can be updated in constant time, the runtime of the algorithm is linear in the number of edges of G.

A Resequencing Selection Strategy. The idea of this strategy is to "fill up" batches that underutilize the available batching capacity. This is done by advancing suitable operations on their assigned machines and can be implemented as follows: As in the static strategy, we first determine a root node $v \in V_u$ in G_u. If it can be included in the same batch as its machine predecessor $w = m^{-1}(v)$ or if no batching capacity is remaining for w, v is settled after w as in the static selection strategy. Otherwise, we iterate through the machine successors of v until we find an operation $u \in V_u$ with $r^{-1}(u) \in V_s$ and $q_u = q_w$ for which invariant (4) is fulfilled. If such an operation is found, u is settled after w, and is combined in a batch with operation w by Algorithm 2. If no such operation exists, we fall back to settling $v \in V_u$ after w. Again, the auxiliary data structures can be updated in constant time. We omit the details of this updating procedure due to a shortage of space.

A Reassigning Selection Strategy. We can enhance the resequencing selection strategy by extending the search for suitable "batch-filling" operations to other machines. If no resequenceable operation is found, we continue to search on other machines for suitable operations to be reassigned: We search in turn, starting from root nodes $y \in V_u$ in G_u with $m_y \neq m_v$. Again, we successively search machine successors of y until an operation $u \in V_u$ is found such that $r^{-1}(u) \in V_s$ and $\exists\, q \in R_u : q = q_w$ and which fulfills invariant (4). If such an operation is found, it is settled after w. Otherwise, we fall back to settling $v \in V_u$ after w.

Analysis. We proposed three selection strategies which differ in their effort to "fill up" underutilized batches. These strategies offer a solid baseline to evaluate our algorithmic framework. However, finding improved strategies might be an interesting challenge for future research. In the worst case, both the resequencing and the reassignment strategies explore $O(|V|)$ operations to select a node. This increases the runtime bound of Algorithm 2 to $O(|E| \cdot |V|)$. However, in the average case, as observed in the numerical experiments of Section 7, a much better behavior is obtained since only few batches are underutilized and only those will trigger a search.

An interesting property of our method is that it includes various classical moves. Consider as an example the swapping of adjacent batches of different families. First, an integrated move could displace a single operation of the second batch before the first batch. Then, the resequencing selection strategy fills up that newly created batch with all operations that had been part of the second batch. In the end, both batches are swapped. Note that this is only a simple example of possible interactions. We observe much more complex rearrangements in practice.

6. Heuristic Approaches

In this section, we apply the building blocks developed in Section 5 within different heuristics. Since our batch-oblivious methodology is not bound to one specific solution approach, we deploy it within classical heuristic frameworks in order to evaluate its performance. In the following, we describe a construction heuristic, a local search method, a Simulated Annealing metaheuristic and a Greedy Randomized Adaptive Search Procedure (GRASP) based metaheuristic.

First, we define a *construction heuristic* which adapts the methods presented in Yugma et al. (2012) and Knopp et al. (2014). If due dates and weights are given, jobs are initially sorted in decreasing order of their ratio $\frac{w_j}{d_j}$ (weight divided by due date). Otherwise, jobs are initially sorted in decreasing order of the sum of the shortest processing durations of their operations. The heuristic then iterates over the sorted list of jobs and successively inserts all operations of the current job. The operations of a job are greedily inserted, starting from the first operation, by selecting the best insertion position for each operation. The best insertion position is determined by the objective function value of the partial solution obtained by actually inserting the considered operation. The construction is completed when all operations of all jobs have been inserted.

In both local search and Simulated Annealing, we combine the batch-oblivious move from Section 5.2 with the adaptive start date computation from Section 5.3 as follows. After a batch-oblivious move is performed, an adaptive start date computation follows in order to determine start dates and batching decisions. The combined result of both modifications is considered as one single move. If such a move is rejected, all involved changes are collectively reverted. The *local search* starts with the solution found by the construction heuristic, and explores the neighborhood using steepest descent. All moves reachable from the current solution are evaluated and the one leading to the best solution is selected. The local search continues until no strictly better solution is found.

Our *Simulated Annealing* metaheuristic is based on the same integrated move and also starts with the solution found by the construction heuristic. In each step, one node is randomly chosen to be moved, its feasible insertion positions are computed, and one of them is randomly selected and performed. We use a geometric cooling schedule that maintains a temperature T which is multiplied by a cooling factor $P_c < 1$ after each iteration. At iteration n, the move is immediately accepted if the current value of the objective function f_n improves the previous objective function value f_{n-1}. Otherwise, the new solution is accepted with a probability of $exp(\frac{-\Delta}{T})$, where $\Delta = f_n - f_{n-1}$. If the new solution is not accepted, all changes related to the move are reversed. The search is stopped if the best solution does not improve during a specified number of iterations P_m. The initial temperature is determined by sampling a fixed number P_s of random moves. For each random move r, we evaluate its influence $\Delta = f_r - f_i$ on the objective function value f_i of the initial solution. Then, for a tuning parameter P_p, the P_p-th percentile of these values is selected as initial value for T.

In order to further diversify the search and to make use of the ever increasing parallelism of modern CPUs, we developed a heuristic approach based on the idea of *Greedy Randomized Adaptive Search Procedures* (GRASP) of Feo and Resende (1995). Our heuristic creates many different starting solutions by randomizing the construction heuristic. This is done by perturbing the sorted list of jobs used in the construction heuristic as follows. A tuning parameter $P_i \geq 1$ is introduced that steers the perturbation intensity. At each iteration of the construction heuristic, the next job to be inserted is determined by randomly selecting one of the first P_i elements in the sorted list of remaining jobs. The operations of the job are then greedily inserted as described earlier and the job is then removed from the list. Each solution is independently improved using the Simulated Annealing metaheuristic. The GRASP based approach is parallelized as follows. Each solution is constructed and improved independently and thus can be run in its own thread. The communication between threads is only needed to update the best overall solution once a thread has completed its computation. A fixed number of threads is used, and each thread restarts with a new initial solution once its Simulated Annealing metaheuristic has met the stopping criterion. We expect the parallel speedup to grow linearly with the number of threads (i.e., parallel efficiency $\approx$ 1). In other words, there should be the same outcome (except minor differences due to the stopping condition) for a parallel run and a sequential run with a computational time prolonged proportionally to the number of threads used in the parallel run.

7. Numerical Results

The algorithms presented in this paper were implemented in C++14 and compiled using the GCC MinGW-W64 compiler in version 4.9.1. All numerical experiments are conducted on an Intel Xeon E5-2620 2.1 GHz machine (6 cores) running Microsoft Windows 7. Extensive numerical experiments on different types of industrial and academic instances were performed. The generality of our approach allows to assess its performance using instances of different scheduling problems. Section 7.1 evaluates our algorithms on new Complex Job-Shop benchmark instances that stem from a real-world semiconductor manufacturing facility. Section 7.2 compares our methods with results for the Complex Job-Shop instances of Mason et al. (2005). In order to show that our approach remains competitive on problems of reduced complexity, Section 7.3 compares our methods with results for the instances for parallel batch machines of Mönch et al. (2005), and Section 7.4 provides results for the Flexible Job-Shop benchmark instances of Hurink et al. (1994).

The sampling strategy of our Simulated Annealing implementation avoids the need to adapt parameters for individual instances. For all numerical experiments, we used the following identical parameter settings: A cooling factor of $P_c = 0.99999$, a number of samples $P_s = 100$, a maximum number of iterations $P_m = 100\,000$, a temperature percentile of $P_t = 5\,\%$, and a perturbation intensity of $P_i = 5$. All given computational times refer to wall-clock-time and 6 parallel threads are used in all runs of the GRASP based approach. All heuristics are run only once since, within the GRASP based approach, many independent runs are performed.

7.1. Complex Job-Shop Instances from the Diffusion and Cleaning Area

This section presents results for new benchmark instances from the diffusion and cleaning area of a semiconductor manufacturing facility. We provide two types of instances. First, 15 industrial instances were provided by STMicroelectronics and modified to anonymize confidential data. Second, 15 random instances that are close to the industrial instances were generated. The random instances include due dates which are not present in the industrial instances. All instances are published under github.com/sebastian-knopp/cjs-instances and its details are described in the following.

Industrial Instances. We perform experiments on 15 industrial instances that were extracted from the Manufacturing Execution System (MES) of a semiconductor manufacturing facility over a period of one year. These instances represent various situations that actually appeared in production. Smaller instances with around 25 machines represent a subset of the actual area while larger instances with around 100 machines correspond to the full area. The number of jobs per instance is between 119 and 346. For each job, between one and seven operations have to be performed. Only some of the machines are capable to process multiple operations in the same batch. Sequence-dependent setup times are required only for some of the non-batching machines. Since no due dates are provided, the total weighted completion time is minimized.

Random Instances. We perform experiments on 15 random instances that are close to the industrial instances for the diffusion and cleaning area. The instance generation method described below and its parameters are chosen to serve this purpose. As in the industrial instances, machines are partitioned into batching machines and non-batching machines. We assume that all batching machines have the same capacity. A family is processable either on a random subset of the batching machines or on a random subset of the regular machines. We generated 15 instances using all possible combinations for the numbers of jobs $|J| \in \{20, 40, 60, 100, 200\}$ and batching capacities $b \in \{2, 4, 6\}$. We consider $\frac{|J|}{20}$ batching machines, $\frac{|J|}{10}$ non-batching machines, $\frac{|J|}{10}$ batching families, and $\frac{|J|}{5}$ non-batching families. We denote a discrete uniform distribution over $[a, b]$ by $DU[a, b]$. For each job $j \in J$, a random number $|O_j| \sim DU[1, 7]$ of operations is chosen. Each operation is randomly assigned to a family. Sequence-dependent setup times $\sim DU[1, 10]$ are generated between all non-batching families. We use $w_j \sim DU[1, 10]$, $r_j \sim DU[0,\ 2 \cdot |J|]$, $d_j \sim r_j + DU[p_j,\ \frac{3}{2}p_j]$ for job weights, release dates and due dates, respectively (p_j denotes the minimum sum of the processing durations of all operations of the job). The number of recipes per family is selected according to $DU[1, 5]$. For the processing time of operations $o_{i,j} \in O$ we use $p_{i,j} \sim b \cdot DU[10, 20]$ with $b = 1$ for non-batching machines. The total weighted tardiness is minimized.

We performed numerical experiments for the described industrial (I) and random (R) instances allowing a maximum computation time of 5 minutes per instance. Table 1 provides the obtained objective function values for the Simulated Annealing and GRASP based approaches. Table 3 provides results in terms of the relative deviation from the best objective function value that has been found. We provide average ($\overline{I}$, $\overline{R}$) and median ($\widetilde{I}$, $\widetilde{R}$) values of these relative deviations over all instances. The column initial refers to the solution that is computed using the non-randomized version of the construction heuristic. We clearly see that the GRASP based approach outperforms all other approaches. The static selection strategy is outperformed by the resequencing and the reassigning strategies with a slight advantage for the resequencing strategy. One reason that could explain the performance of the resequencing strategy is that a larger number of moves can be performed in the same amount of time compared to the reassigning strategy, for which additional time is spent to search for nodes that can be settled.

The number of moves performed per second is analyzed in Table 2, as well as the different selection strategies and the impact of the parallel imple-

		|J|	|M|	**Simulated Annealing**			**GRASP**		
				static	*reseq*	*reass*	*static*	*reseq*	*reass*
Industrial (total weighted completion time)	01	119	24	92973	93208	93189	92899	92803	**92532**
	02	148	22	250240	249400	243372	245939	242890	**239892**
	03	195	25	217863	203678	203485	208286	202801	**201790**
	04	209	24	271548	265801	268121	267669	**260286**	260931
	05	186	88	171229	169174	170373	167902	163345	**162884**
	06	268	26	341448	333429	333323	336012	331276	**329918**
	07	210	94	150271	150984	158954	**149680**	150115	150332
	08	310	17	465701	461574	460594	454252	451612	**447305**
	09	231	95	167754	167973	168011	167271	166798	**166643**
	10	245	94	202722	199565	204280	198112	197369	**195789**
	11	302	24	561202	562295	561767	554883	555655	**554670**
	12	302	24	350461	349444	371985	345590	**344109**	345673
	13	324	94	349147	337979	340014	346464	**334409**	334416
	14	315	101	475725	470249	505656	469239	**450909**	465354
	15	346	94	777426	726829	749775	736514	**698666**	702559
Random (total weighted tardiness)	01	20	3	10618	10613	10613	10598	**10011**	**10011**
	02	20	3	6030	6098	6354	5939	**5883**	**5883**
	03	20	3	7063	7074	7074	7036	**7006**	**7006**
	04	40	6	9201	9302	9256	**8801**	9083	9015
	05	40	6	**14208**	14246	14842	14573	14904	14674
	06	40	6	37152	33318	32629	32406	**32048**	32321
	07	60	9	17832	17522	17427	16126	**15165**	15668
	08	60	9	41609	**40514**	41537	42560	41094	42508
	09	60	9	35888	37155	37068	31984	32354	**30890**
	10	100	15	28503	30051	30209	28015	**27954**	28342
	11	100	15	34501	33315	36518	**32738**	33370	33161
	12	100	15	50505	44284	49300	**39565**	40851	43102
	13	200	30	28580	**27030**	47916	28259	27685	32242
	14	200	30	55075	53193	67950	52867	**51444**	58399
	15	200	30	66589	63042	66672	60993	**60494**	62464

Table 1: Detailed results for industrial and random instances

			reass							reseq	static	
	#Threads:		1	1	2	3	4	5	6	1	1	1
	I	$\|O\|$	$\frac{\#\text{m}}{\text{s}}$	r	r	r	r	r	r	r	r	$\frac{\text{ns}}{\|O\|}$
Industrial	01	193	5810	1.0	2.3	3.4	4.5	5.7	6.7	1.4	1.4	637
	02	293	3809	1.0	2.4	3.5	4.7	5.8	6.9	1.4	1.4	647
	03	305	3215	1.0	2.3	3.5	4.6	5.8	6.7	1.5	1.5	664
	04	370	2768	1.0	2.3	3.4	4.6	5.7	6.8	1.4	1.5	649
	05	452	1740	1.0	2.3	3.4	4.5	5.6	6.6	1.8	1.8	700
	06	461	1960	1.0	2.3	3.4	4.4	5.7	6.6	1.6	1.6	698
	07	472	1300	1.0	2.1	3.1	4.4	5.1	6.2	2.2	2.2	731
	08	480	2134	1.0	2.0	3.0	3.7	4.9	5.9	1.3	1.4	705
	09	511	1502	1.0	1.9	2.8	3.8	4.7	5.7	1.8	1.8	736
	10	539	1441	1.0	1.9	3.0	4.0	4.9	5.9	1.7	1.8	733
	11	569	2070	1.0	2.0	2.9	3.9	4.8	5.7	1.2	1.2	718
	12	720	963	1.0	1.6	2.2	3.1	3.8	4.6	1.8	1.8	787
	13	725	654	1.0	2.1	3.1	4.0	5.1	6.1	2.6	2.7	776
	14	752	748	1.0	2.0	3.1	3.7	5.2	6.1	2.3	2.3	759
	15	835	609	1.0	2.2	3.2	3.9	5.0	6.0	2.4	2.4	804
Random	01	82	18605	1.0	1.9	2.9	3.8	4.8	5.5	1.0	1.0	640
	02	80	17831	1.0	2.0	3.0	3.8	4.9	5.7	1.0	1.1	644
	03	75	18635	1.0	2.0	3.0	4.0	5.0	5.8	1.1	1.1	645
	04	147	10053	1.0	2.0	3.0	3.9	4.8	5.7	1.0	1.0	652
	05	165	7732	1.0	2.1	3.1	3.6	5.2	6.1	1.2	1.2	641
	06	159	8749	1.0	2.0	2.9	3.8	4.7	5.2	1.1	1.2	618
	07	222	5589	1.0	2.0	3.0	4.0	5.0	5.9	1.2	1.3	640
	08	269	3959	1.0	2.1	3.1	3.5	5.1	6.1	1.4	1.5	638
	09	220	5682	1.0	2.0	2.9	3.9	4.8	5.6	1.2	1.2	649
	10	398	2485	1.0	2.1	3.0	4.0	5.1	6.0	1.4	1.5	668
	11	406	2167	1.0	2.0	3.0	3.6	4.9	5.8	1.6	1.7	670
	12	387	2507	1.0	1.8	2.7	3.5	4.5	5.3	1.4	1.5	692
	13	796	885	1.0	2.0	2.9	3.9	4.8	5.6	1.9	1.9	730
	14	796	743	1.0	2.0	2.9	3.8	4.8	6.0	2.1	2.2	758
	15	768	820	1.0	1.9	2.9	3.5	4.8	5.6	2.0	2.1	765

Table 2: Analysis of the number of moves performed per second

	Initial	Local Search			Sim. Annealing			GRASP		
		static	*reseq*	*reass*	*static*	*reseq*	*reass*	*static*	*reseq*	*reass*
$\overline{I}$	12.2%	7.4%	6.4%	7.6%	3.9%	2.1%	3.8%	2.0%	**0.3%**	**0.3%**
$\widetilde{I}$	10.8%	6.7%	5.7%	6.7%	4.1%	1.6%	3.0%	1.8%	0.3%	**0.0%**
$\overline{R}$	52.9%	41.2%	40.2%	41.2%	8.3%	5.7%	15.2%	2.3%	**1.5%**	4.2%
$\widetilde{R}$	49.6%	39.9%	39.6%	36.5%	5.7%	4.0%	8.1%	1.1%	**0.0%**	2.4%

Table 3: Aggregated results for industrial and random instances.

mentation. In column $|O|$, the total number of operations for each instance is provided, since we assume it is correlated to the number of moves which are performed per second. Different variants of our algorithms have been tested on the industrial and random instances using a computational time of one minute for each variant and instance. In order to evaluate the number of moves performed per second, the time for constructing solutions is not taken into account. The GRASP based heuristic is run using a large value for the maximum number of non-improving iterations P_m in order to avoid triggering runs of the construction algorithm. This setting means there is one parallel run of the Simulated Annealing algorithm for each thread that is used. The second row of Table 2 provides the number of used threads. For runs using the reassigning strategy with one thread, the absolute number of moves performed per second is given in column $\frac{\#\mathrm{m}}{\mathrm{s}}$. All columns entitled by "r" provide the relative number of moves per second. The three right-most columns provide results for the resequencing and static node selection strategies. The results show that the reassigning strategy requires more time per move than the other strategies. Column $\frac{\mathrm{ns}}{|O|}$ provides the average number of nanoseconds that is spent per node while performing a single move. Being almost constant, these values show that the static selection strategy yields an algorithm that is linear in the number of nodes. The slight increase that can be observed might be due to secondary reasons such as an increase of cache misses coming along with the increased memory consumption of larger instances; also, additional time might be needed for a larger number of jobs. The numbers of moves performed per second increases linearly with the numbers of threads, which shows that the parallel implementation of our algorithm scales with the number of threads.

7.2. Complex Job-Shop Instances of Mason et al. (2005)

Mason et al. (2005) consider a Complex Job-Shop scheduling problem from semiconductor manufacturing and provide results for instances based on the mini-fab model of El Adl et al. (1996). Total weighted tardiness is minimized in all these instances. There is a difference to our problem definition that concerns sequence-dependent setup times. Mason et al. (2005) do not allow the setup between operations o_a and o_b to begin before the route predecessor operation of o_b is completed. In our definition, this setup can begin as soon as o_a is completed. We consider this difference by modifying the instances as follows: We consider all setup durations to be zero and prolong operation processing durations instead. We extend each processing duration by adding the longest possible setup duration that might precede the operation. It is important to note that, in case setup durations are crucial, this modification might increase (but never decrease) the optimal total weighted tardiness.

Table 4 shows results for a batching capacity of $b = 3$. Results for $b = 2$ and $b = 4$ are similar and omitted here for a shortage of space. Columns

	SB	Dispatching			MIP	Batch-Oblivious	
$\|J\|$	best	ATCS	CR	EDD	6 hrs	Initial	GRASP reass
3	1.484	3.082	2.974	2.996	**1.000**	2.024	1.268
4	2.028	2.401	3.495	3.371	**1.000**	2.257	1.313
5	1.885	2.267	3.604	3.207	**1.000**	1.575	1.098
6	1.283	1.534	1.860	1.929	1.003	1.440	**0.876**
7	1.131	1.608	1.446	1.515	1.198	1.349	**0.735**
8	1.164	1.325	1.120	1.193	2.860	1.180	**0.713**
9	1.240	1.299	1.087	1.207	N/A	1.497	**0.806**
10	1.266	1.453	1.032	1.066	N/A	1.292	**0.768**

Table 4: Results for instances of Mason et al. (2005) with b = 3

"Batch-Oblivious" show our results, all others are taken from Mason et al. (2005) for comparison. We allowed a computational time of 5 seconds per instance. Values represent normalized average total weighted tardiness and 1.000 represents the best solution found by Mason et al. (2005). For smaller instances, setup durations are crucial and our results are worse due to the assumptions for setups in the modified instances. For instances with more than 5 jobs, setup durations are negligible and our method clearly outperforms the results of Mason et al. (2005). Initial solutions obtained by our construction heuristic are strongly improved by our GRASP based approach.

7.3. Instances for Parallel Batch Machines of Mönch et al. (2005)

Mönch et al. (2005) consider a scheduling problem for the diffusion and cleaning area that models the machines in that area as parallel batch processors. This modeling does not include a Job-Shop environment, so the problem is less general than ours. From the perspective of this paper, their instances consist of jobs with exactly one operation without any sequence-dependent setup times. The instances cover a range of variations regarding instance sizes, batch sizes, processing times, release dates, and due dates. We compare our algorithms with results that are obtained by methods dedicated to this less general problem. Table 5 provides average values for total weighted tardiness. The best results for such instances that we are aware of are reported by Chiang et al. (2010). We put their results in brackets since they used a parameter-identical reimplementation instead of the original instances of Mönch et al. (2005). For this comparison, we scaled their reported results using the relative values given in their paper.

Our best method (*GRASP reass*) outperforms the results of Mönch et al. (2005) and Yugma et al. (2012). It reaches a quality that is comparable to the dedicated method of Chiang et al. (2010). In contrast to the previous section, we see stronger differences between the different selection strategies.

	Time	#Machines			Batch size	
	(s)	m=3	m=4	m=5	b=4	b=8
Mönch et al. (2005)	58	412	300	231	389	240
Chiang et al. (2010)	4	(370)	(272)	(209)	(347)	(220)
Yugma et al. (2012)	178	411	278	206	367	229
Initial	<1	630	455	356	577	383
Local Search static	14	607	434	334	553	363
Local Search reseq	30	539	399	316	487	348
Local Search reass	60	486	361	269	452	292
Sim. Annealing static	120	548	383	294	503	314
Sim. Annealing reseq	120	457	344	261	418	290
Sim. Annealing reass	120	411	303	210	382	234
GRASP static	120	502	356	266	469	280
GRASP reseq	120	429	318	244	399	262
GRASP reass	120	**382**	**274**	**197**	**356**	**212**

Table 5: Results for instances of Mönch et al. (2005) with total weighted tardiness objective

	α			β		
	0.25	0.5	0.75	0.25	0.5	0.75
Mönch et al. (2005)	592	282	68	658	232	53
Yugma et al. (2012)	**561**	269	65	647	215	**33**
GRASP reass	564	**245**	**44**	**610**	**208**	35

Table 6: Release date (α) / due date (β) dependent results for instances of Mönch et al. (2005)

		20s	30s	60s	120s	300s	1800s	3600s
edata	*avg*	2.55%	1.60%	1.02%	0.78%	0.58%	0.34%	0.27%
	max	12.22%	5.98%	4.12%	4.03%	3.85%	2.34%	2.34%
rdata	*avg*	3.13%	1.43%	0.75%	0.58%	0.35%	0.21%	0.15%
	max	15.90%	5.83%	3.83%	3.83%	1.46%	1.35%	1.35%
vdata	*avg*	4.66%	2.40%	0.35%	0.21%	0.18%	0.10%	0.09%
	max	25.32%	25.32%	2.36%	1.37%	1.37%	0.87%	0.65%

Table 7: Results for instances of Hurink et al. (1994) (gap to best known solution)

We assume that this is due to the fact that, in non-Job-Shop instances, a larger number of operations is available to be settled. We observe that the reassigning strategy strongly outperforms both the static and the resequencing selection strategies. Again, the GRASP metaheuristic approach yields a clear improvement over Simulated Annealing alone. In addition, Table 6 provides average total weighted tardiness values for varying release date distributions α and due date tightness distributions β as described in Mönch et al. (2005). We observe good results for *GRASP reass* when due dates are tight.

7.4. Flexible Job-Shop Instances of Hurink et al. (1994)

Finally, we present results for the flexible Job-Shop instances of Hurink et al. (1994) for which the makespan is minimized. These instances do not incorporate batching machines and have been widely used to assess the performance of several highly efficient dedicated methods. The instances are partitioned into edata, rdata, and vdata instances that include low, medium and high flexibility levels, respectively. The comparison with best known results from literature refers to the best known solution that is obtained by combining the results of Jurisch (1992), Dauzère-Pérès and Paulli (1997), Mastrolilli and Gambardella (2000), Pacino and Van Hentenryck (2011), Behnke and Geiger (2012) and Schutt et al. (2013). Table 7 reports aggregated results for a range of computational times ranging from 20 seconds to one hour. Rows *avg* and *max* refer to average and maximum gaps to best known solutions in percent, respectively. The results show that the GRASP based approach obtains good results even for this much less general problem. If the computational time is large enough, the GRASP based approach obtains results that are very close to the best known solutions from the academic literature. Actually, in four cases, best known solutions were improved after 1 hour of computational time (rdata-la28 (1079), rdata-la37 (1076), rdata-la40 (969), vdata-la23 (814)).

8. Conclusion

In this paper, we considered a Complex Job-Shop scheduling problem with a focus on the integration of batching machines. We reduced the structural complexity of disjunctive graphs by introducing a novel batch-oblivious representation. This representation allows to take batching decisions during a traversal of the graph and enables the implementation of resequencing and reassignment strategies that adaptively "fill up" underutilized batches. Together with an integrated batch-oblivious move, we obtain a neighborhood that is applied in a GRASP based heuristic approach. The scheduling of parallel batching machines is often considered to be important because it is a subproblem solution procedure of the shifting bottleneck heuristic. Our holistic way to modify schedules outperforms such approaches for the instances considered in our numerical experiments. Our batch-oblivious approach improves both solution quality and implementation complexity in comparison to decomposition based approaches.

Avoiding the complexity of additional batching nodes simplifies the inclusion of further constraints. Regarding the diffusion and cleaning area, we want to include the complex routing structures presented in Knopp et al. (2014) to model machines in more detail. Also in this context, we want to include additional time constraints that limit the time between certain operations (see Klemmt and Mönch (2012); Sadeghi et al. (2015)). Though we already observe good numerical results, we still see opportunities for further improvements. Enhancing the node selection strategies proposed in Section 5.4 seems promising. In addition, how to apply the conditions for testing the feasibility of moves and the move evaluation functions proposed in Mati et al. (2011) and García-León et al. (2015) for regular criteria should be investigated. Evaluating the implementation of more advanced features of GRASP (see Resende and Ribeiro (2010) and Armentano and de Franca Filho (2007)) could be relevant. The batch-oblivious approach could also be applied using different metaheuristic methods: GRASP has strong diversification and intensification mechanisms but lacks elements of mutual learning that can be found in path-relinking approaches or genetic algorithms. Speeding up individual moves by only partially updating the graph as proposed by Katriel et al. (2005), Pearce and Kelly (2007) and Sobeyko and Mönch (2015) seems applicable and promising as well.

Acknowledgements

This work is supported by the ENIAC European Project INTEGRATE. The authors would like to thank Scott Mason and Lars Mönch for providing test instances and detailed numerical results. We also would like to thank all contributors from STMicroelectronics who supported our work with valuable information and discussions.

References

Adams J, Balas E, Zawack D. The shifting bottleneck procedure for job shop scheduling. Management Science 1988;34(3):391–401.

Armentano VA, de Franca Filho MF. Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based grasp approach. European Journal of Operational Research 2007;183(1):100–14.

Behnke D, Geiger MJ. Test Instances for the Flexible Job Shop Scheduling Problem with Work Centers. Technical Report; Helmut-Schmidt-Universität, Universität der Bundeswehr Hamburg; 2012.

Bilyk A, Mönch L, Almeder C. Scheduling jobs with ready times and precedence constraints on parallel batch machines using metaheuristics. Computers & Industrial Engineering 2014;23(5):1621–35.

Brucker P. Scheduling algorithms. Springer, 2007.

Chiang TC, Cheng HC, Fu LC. A memetic algorithm for minimizing total weighted tardiness on parallel batch machines with incompatible job families and dynamic job arrival. Computers & Operations Research 2010;37(12):2257–69.

Conway RW, Maxwell WL, Miller LW. Theory of Scheduling. Addison-Wesley, Reading, MA, 1967.

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

El Adl M, Rodriguez AA, Tsakalis KS. Hierarchical modeling and control of re-entrant semiconductor manufacturing facilities. In: Proceedings of the 35th IEEE Conference on Decision and Control. volume 2; 1996. p. 1736 –42.

Feo TA, Resende MG. Greedy randomized adaptive search procedures. Journal of Global Optimization 1995;6(2):109–33.

García-León A, Dauzère-Pérès S, Mati Y. Minimizing regular criteria in the flexible job-shop scheduling problem. In: 7th Multidisciplinary International Conference on Scheduling: Theory & Applications, Prague, Czech Republic. 2015. p. 443–56.

Garey MR, Johnson DS, Sethi R. The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1976;1(2):117–29.

González MA, Vela CR, González-Rodríguez I, Varela R. Lateness minimization with tabu search for job shop scheduling problem with sequence dependent setup times. Journal of Intelligent Manufacturing 2013;24(4):741–54.

Guo C, Zhibin J, Zhang H, Li N. Decomposition-based classified ant colony optimization algorithm for scheduling semiconductor wafer fabrication system. Computers & Industrial Engineering 2012;62(1):141–51.

Hurink J, Jurisch B, Thole M. Tabu search for the job-shop scheduling problem with multi-purpose machines. Operations-Research-Spektrum 1994;15(4):205–15.

Johnzén C, Dauzère-Pérès S, Vialletelle P. Flexibility measures for qualification management in wafer fabs. Production Planning and Control 2011;22(1):81–90.

Jung C, Pabst D, Ham M, Stehli M, Rothe M. An effective problem decomposition method for scheduling of diffusion processes based on mixed integer linear programming. In: Advanced Semiconductor Manufacturing Conference (ASMC), 2013 24th Annual SEMI. IEEE; 2013. p. 35–40.

Jurisch B. Scheduling jobs in shops with multi-purpose machines. Ph.D. thesis; Fachbereich Mathematik/Informatik, Universität Osnabrück; 1992.

Kahn AB. Topological sorting of large networks. Communications of the ACM 1962;5(11):558–62.

Katriel I, Michel L, Van Hentenryck P. Maintaining longest paths incrementally. Constraints 2005;10(2):159–83.

Kim YD, Joo BJ, Choi SY. Scheduling wafer lots on diffusion machines in a semiconductor wafer fabrication facility. Semiconductor Manufacturing, IEEE Transactions on 2010;23(2):246–54.

Klemmt A, Mönch L. Scheduling jobs with time constraints between consecutive process steps in semiconductor manufacturing. In: Proceedings of the Winter Simulation Conference. Winter Simulation Conference; WSC '12; 2012. p. 194:1–194:10.

Knopp S, Dauzère-Pérès S, Yugma C. Flexible Job-Shop Scheduling with Extended Route Flexibility for Semiconductor Manufacturing. In: Proceedings of the 2014 Winter Simulation Conference (WSC). IEEE Press; 2014. p. 2478–89.

Mason S, Fowler J, Carlyle W, Montgomery D. Heuristics for minimizing total weighted tardiness in complex job shops. International Journal of Production Research 2005;43(10):1943–63.

Mastrolilli M, Gambardella LM. Effective neighbourhood functions for the flexible job shop problem. Journal of Scheduling 2000;3(1):3–20.

Mathirajan M, Sivakumar A. A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. The International Journal of Advanced Manufacturing Technology 2006;29(9-10):990–1001.

Mati Y, Dauzère-Pérès S, Lahlou C. A general approach for optimizing regular criteria in the job-shop scheduling problem. European Journal of Operational Research 2011;212(1):33–42.

Mönch L, Balasubramanian H, Fowler JW, Pfund ME. Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. Computers & Operations Research 2005;32(11):2731 –50.

Mönch L, Fowler JW, Dauzère-Pérès S, Mason SJ, Rose O. A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. Journal of Scheduling 2011;14(6):583–99.

Mönch L, Fowler JW, Mason SJ. Production Planning and Control for Semiconductor Wafer Fabrication Facilities. volume 52. Springer New York, 2013.

Mönch L, Rose O. Shifting-Bottleneck-Heuristik für komplexe Produktionssysteme: Softwaretechnische Realisierung und Leistungsbewertung. Quantitative Methoden in ERP und SCM, DSOR Beiträge zur Wirtschaftsinformatik 2004;2:145–59.

Mönch L, Rose O, Sturm R. A simulation framework for the performance assessment of shop-floor control systems. Simulation 2003;79(3):163–70.

Mönch L, Schabacker R, Pabst D, Fowler JW. Genetic algorithm-based subproblem solution procedures for a modified shifting bottleneck heuristic for complex job shops. European Journal of Operational Research 2007;177(3):2100–18.

Ovacik IM, Uzsoy R. Decomposition methods for complex factory scheduling problems. Kluwer Academic Publishers Boston, 1997.

Pacino D, Van Hentenryck P. Large neighborhood search and adaptive randomized decompositions for flexible jobshop scheduling. In: Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three. AAAI Press; 2011. p. 1997–2002.

Pearce DJ, Kelly PH. A dynamic topological sort algorithm for directed acyclic graphs. Journal of Experimental Algorithmics (JEA) 2007;11:1–7.

Potts CN, Kovalyov MY. Scheduling with batching: a review. European Journal of Operational Research 2000;120(2):228–49.

Resende MG, Ribeiro CC. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In: Handbook of metaheuristics. Springer; 2010. p. 283–319.

Roy B, Sussmann B. Les problemes d'ordonnancement avec contraintes disjonctives. Note ds 1964;9.

Sadeghi R, Dauzère-Pérès S, Yugma C, Lepelletier G. Production control in semiconductor manufacturing with time constraints. In: Advanced Semiconductor Manufacturing Conference (ASMC), 2015 26th Annual SEMI. IEEE; 2015. p. 29–33.

Schutt A, Feydy T, Stuckey PJ. Scheduling optional tasks with explanation. In: Principles and Practice of Constraint Programming. Springer; 2013. p. 628–44.

Sobeyko O, Mönch L. Heuristic approaches for scheduling jobs in large-scale flexible job shops. Computers & Operations Research 2015;.

Sourirajan K, Uzsoy R. Hybrid decomposition heuristics for solving large-scale scheduling problems in semiconductor wafer fabrication. Journal of Scheduling 2007;10(1):41–65.

Upasani AA, Uzsoy R, Sourirajan K. A problem reduction approach for scheduling semiconductor wafer fabrication facilities. Semiconductor Manufacturing, IEEE Transactions on 2006;19(2):216–25.

Yugma C, Dauzère-Pérès S, Artigues C, Derreumaux A, Sibille O. A batching and scheduling algorithm for the diffusion area in semiconductor manufacturing. International Journal of Production Research 2012;50(8):2118–32.

Yurtsever T, Kutanoglu E, Johns J. Heuristic based scheduling system for diffusion in semiconductor manufacturing. In: Winter Simulation Conference. Winter Simulation Conference; 2009. p. 1677–85.