Swarm Intelligence-Driven Dispatching Rules for Large-Scale Semiconductor Production: Integrating Simulation and Optimization to Enhance Operational Efficiency

Ramsha Ali[1 [0000−0002−4794−6560]], Peyman Eftekhari[1 [0009−0007−4198−392X]], Martin Gebser[1 [0000−0002−8010−4752]], Stephan Leitner[2 [0000−0001−6790−4651]], and Gerhard Friedrich[1 [0000−0002−1992−4049]]

[1] Department of Artificial Intelligence and Cybersecurity,
[2] Department of Management Control and Strategic Management,
University of Klagenfurt, Austria
{ramsha.ali,peyman.eftekhari,martin.gebser,
stephan.leitner,gerhard.friedrich}@aau.at
https://www.aau.at

Abstract. The abstract should briefly summarize the contents of the paper in 150–250 words.

Keywords: Swarm intelligence · Ant colony optimization · Greedy search · Semiconductor production · Scheduling · Dispatching · Simulation

1 Introduction

As the digital transformation deepens across various sectors, the demand for more powerful, energy-efficient, and smaller semiconductors continues to escalate. This surge in demand places immense pressure on semiconductor manufacturers to scale up production without compromising quality. Semiconductor production involves hundreds of sophisticated steps, each of which must be precisely controlled to ensure the functionality and yield of the final product [1]. The complexity is exacerbated by the rapid pace of innovation in semiconductor design, which frequently shifts production parameters and process requirements.

Dispatching in semiconductor production is a particularly challenging aspect of factory operations. It involves determining the sequence and timing of various production tasks, from wafer fabrication to assembly and testing, to maximize throughput [2]. In large-scale operations, the complexity is magnified by the sheer number of variables involved, including machine availability, maintenance schedules, workforce shifts, and rapidly changing order priorities. Traditional dispatching solutions often fall short in such an environment, as they cannot adequately adapt to the dynamic and complex nature of semiconductor production. This inadequacy can lead to sub-optimal decisions that compromise operational efficiency and product quality [3].

On the other hand, scheduling in large-scale semiconductor production is a cornerstone of operational management that directly influences the effectiveness and efficiency of the entire manufacturing process [4]. It involves planning and organizing production activities to ensure that resources are utilized optimally and product flows are synchronized across various stages of the manufacturing process. Effective scheduling is critical not only for maintaining high throughput but also for minimizing production delays and reducing inventory and holding costs.

The scheduling of semiconductor manufacturing is notably complex due to the high variability in production processes, the sensitive nature of the materials involved, and the stringent quality requirements [5]. Each semiconductor product may pass through hundreds of processing steps, requiring precise timing and coordination. Moreover, the high mix of product types, each with different processing needs and priorities, adds another layer of complexity [6]. This complexity is further exacerbated by the need to integrate new product introductions seamlessly into the production schedule without disrupting ongoing operations.

Scheduling production operations within the intricate landscape of semiconductor manufacturing represents one of the most daunting challenges in resource allocation. The dynamic nature of manufacturing environments—characterized by unpredictable fluctuations in process and demand, supply chain delays, and frequent machine breakdowns—significantly intensifies this complexity. Consequently, production scheduling must not only strive for near-optimal outcomes but also maintain robustness and flexibility to adapt swiftly to frequent changes in the production landscape.

In this paper, we address a large-scale Semiconductor Manufacturing Scheduling Problem (SMSP) [1], called fab for short, is a complex production environment characterized by customized job flows and high-tech machines.

Modern semiconductor fabs process tens of thousands of operations per day using more than 1000 machines [1]. These machines are organized in tool groups with specific functionalities, e.g., diffusion, etching, or metrology, and each manufacturing step can be flexibly allocated to a machine providing the required functionality. Hence, the scheduling of semiconductor production operations consists of the sub-tasks of assigning operations to machines and sequencing the operations on each machine. However, in case of machine breakdowns or process deviations, a schedule must be revised to adapt to the changed environment.

To this end, we proposed a Greedy Search based Ant Colony Optimization (GSACO) algorithm for (re-)scheduling semiconductor production operations [7]. Our algorithm harnesses Ant Colony Optimization (ACO) [8] for exploration, while Greedy Search (GS) [9] enables responses in short time. In this way, GSACO overcomes limitations of a state-of-the-art Constraint Programming approach [10] on large-scale SMSP instances.

In our previous research [7], we introduced a novel approach that synergistically combines probabilistic operation sequencing with a greedy machine assignment strategy, targeting up to five operations per lot with the primary objective of minimizing makespan. Building on this foundational work, the current paper extends these initial concepts by proposing an enhanced algorithm, GSACO-1, specifically designed to opti-

mize operational throughput. This development marks a significant advancement in our methodology, aiming to refine the dispatching rules further though simulation.

The paper is organized as follows. Section 2 provides literature reviews on FJSSP and ACO. In Section 4, we formulate SMSP in terms of the well-known FJSSP. Our GSACO-1 algorithm is presented in Section 5. In Section 6 we present the customized simulation adopted from [11]. Section 7 provides and discusses experimental results. Finally, Section 8 concludes the paper.

2 literature Review

3 Literature Review

This section represents a brief relevant literature on the Flexible Job Shop Scheduling Problem (FJSSP), which was initially used as a general scheduling model for semiconductor production. Alternatively, this moves on with a discussion around the Semiconductor Manufacturing Scheduling Problem (SMSP), and how it can proceed with Ant Colony Optimization (ACO) and the solution.

3.1 Semiconductor Manufacturing Scheduling Problem (SMSP)

The SMSP is a complex and critical challenge to optimize scheduling tasks during the manufacturing process of a semiconductor. Different methods have been adopted for a long time for varied types of machines [15]. The core aspects include job scheduling, batch processing, priority constraints, setup times of the machine, machine availability, and reentrant flow. The key challenges stay with complexity, dynamic job arrivals, and reducing downtime while maximizing throughput [**?**]. Many researchers and developers have been working on the topic and suggested their methods to solve the problem.

Mixed-integer linear programming (MILP) is a great option to solve large mathematical problems by optimizing the solution. As suggested by [**?**] in their study to identify important research problems with semiconductor manufacturing operations (SMOs), MILP models are extensively used in deterministic SMO scheduling problems. While MILP models are great for optimal solutions for small-scale instances, these models can also be used to provide upper bounds for large-scale instances. The only thing is, there is no optimal solution for such situations, but it is done by relaxing several constraints.

Another paper [**?**] was written for research of an SMSP to solve all the constraints of the semiconductor manufacturing industry such as machine status, setup time, limited waiting time, different process times on varied machines, and more. The researchers suggested a hybrid estimation of a distribution algorithm with multiple subpopulations (HEDA-MS) to solve SMSP and to make the total exceed the limited waiting time to zero.

3.2 Flexible Job Shop Scheduling

Considering the manufacturing industry, FJSSP is a common problem, especially for small batch and custom productions. The mathematical models allow us to solve the optimality issue for small-scale instances. As [**?**] explains in their paper, it is important to assign the FJSSP on a machine in a particular sequence. As the optimization criteria need the start time of all the operations, it is important to optimize the completion time also. The most suitable approach to solve this time problem depends on the function and the mathematical properties associated with it. Researchers also explained that many studies are optimizing non-regular criteria that require equal efforts for timing decisions to get the best sequencing decisions in solution approaches.

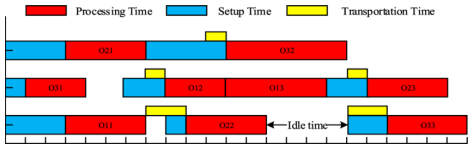

Fig. 1. Example of the flexible job-shop scheduling problem [**?**]

The study [**?**] explained that the existing methods to solve NP-hard combinatorial optimization problems are either exact or approximate. The exact methods are challenging to solve FJSSP when problems need large-size scheduling. Considering their NP-hardness, it is difficult to allocate reasonable time. [**?**] have also explained that FJSSP instances intractability is in constant need of more approximate methods. So many solutions like machine learning techniques, heuristics, meta-heuristics methods, and more are continuously being developed to tackle real-world problems more effectively.

In their recent work, [**?**] used generic programming to evolve scheduling heuristics in dynamic FJSS. They explained that Genetic programming hyperheuristics (GPHH) is a great option for heuristics scheduling, and a proper selection of the terminal makes it successful. They concluded that a two-stage GPHH with selected features for DFJSS can help in interpretable scheduling heuristics while creating a much shorter training time.

3.3 Ant Colony Optimization

ACO follows the foraging methods of ant species to find a favorable path and leaves the trails (pheromone) to let others find the path. ACO algorithms are one of the MPPT methods to search for GMPP. Sometimes it is also used as a hybrid MPPT method [**?**].

ACO is a well-suited metaheuristics algorithm to solve SMSP as it is highly appropriate to handle the dynamic and complex nature of semiconductor scheduling with its

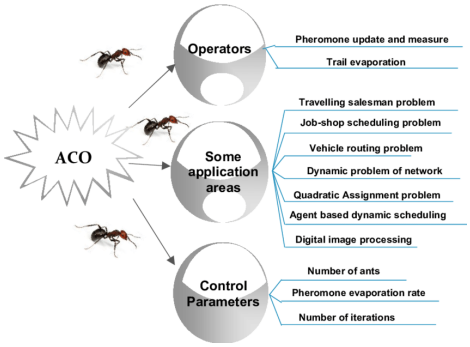

Fig. 2. Ant Colony Optimization Algorithm working for scheduling [**?**]

multi-objective nature [**?**]. ACO has a history of application to be applied to SMSP for wafer scheduling to balance the load and minimize the makespan. The dynamic nature of the ACO algorithm continuously updates the "pheromone" based on the completed jobs, and guides others to follow the optimal scheduling decisions, ultimately making the system reduce their decision-making time [**?**]. In their work, [**?**] proposed an Ant

Colony Algorithm (ACA) to solve constraints like holding times and time lags. They created this setup in two stages where they located pheromones in the first stage while using genetic algorithms to initialize. [**?**] used the ACO algorithm to form batches. They batched using a DP algorithm and combined it with the job sequences generated by the ACO algorithm that released time to update pheromone trails. ACO has been a promis-

ing algorithm for FJSSP solving. Researchers didn't stop there but created extended versions of ACOs to improve time-span minimization. [**?**] proposed Dynamic Impact, an extended method for ACO that improved convergence and optimized problems between the resources having non-linear relationships. [**?**] concluded a 33.2 percent improved optimization over ACO with the Dynamic Impact algorithm. [**?**] has explained

the importance of improved ACO to ensure real-time determination as a time-sensitive network (TSN). This improved ACO (IACO) focuses on convergence speed and schedules the time-triggered flows in TSN.

We briefly survey the relevant literature on FJSSP, which we use as general scheduling model to represent semiconductor production processes and SMSP, as well as ACO and its application to FJSSP solving.

3.4 Flexible Job Shop Scheduling

Exact mathematical models of the NP-hard FJSSP allow for solving small-scale instances to optimality. For example, [**?**] devise Mixed-Integer Linear Programming (MILP) and Constraint Programming (CP) models for a distributed FJSSP setting. They observe that CP performs well at exploring feasible solutions, leading to high-quality

Table 1. Basic notations

Symbol	Description
J	Total number of *jobs*
T	Total number of *tool groups*
M	Total number of *machines*
N	Total number of *operations*
t_m	Tool group t of machine m
$O_{i,j,t}$	Operation i of job j on tool group t
$d_{i,j,t}$	*Duration* of operation $O_{i,j,t}$ on tool group t
$O_{i,j,t,m}$	Operation i of job j on machine m of tool group t
$s_{i,j,t,m}$	*Start time* of operation $O_{i,j,t}$ on machine m of tool group t

solutions in relatively short computing time. Similarly, [**?**] make use of a CP model to effectively solve small to medium-scale FJSSP instances.

The scalability limits of exact optimization approaches motivate the design of approximation methods, where a large variety of heuristics and meta-heuristics have been proposed for FJSSP solving. Respective techniques include tabu search [**?**], simulated annealing [**?**], particle swarm optimization [**?**], genetic algorithms [**?**], artificial bee colony algorithms [**?**], and ACO [**?**] as well as hybrid approaches based on local search with meta-heuristics [**?**,**?**,**?**,**?**].

In their recent work, [**?**] train deep neural networks on historical FJSSP data to identify patterns and predict effective scheduling strategies. Related approaches based on deep reinforcement learning [**?**,**?**] or genetic algorithms [**?**] aim at optimizing the decision making and control within simulation models of semiconductor fabs. While these methods allocate production lots one by one based on their features, we address the task of scheduling the operations on given lots in advance.

3.5 Ant Colony Optimization

ACO is a meta-heuristic algorithm that mimics the foraging behavior of ants [**?**].

It was originally devised to solve the traveling salesperson problem [**?**], and meanwhile ACO has been adopted to various optimization problems, including routing [**?**], scheduling [**?**], task allocation [**?**], project planning [**?**], and network optimization [**?**].

The common idea is that artificial ants construct paths through a graph, making probabilistic decisions based on problem-specific heuristic information as well as temporary pheromone trails that indicate promising search directions. Particular strengths of ACO lie in the high potential for parallelization, given that ants can be simulated in parallel, and a certain robustness against getting stuck in local optima, as the probabilistic decision rules of ants promote exploration. However, a specific difficulty in the ACO algorithm design concerns the tuning of hyperparameters, such as the number of ants to consider, the trade-off between heuristic information and pheromone trails, and the pheromone evaporation rate.

Among meta-heuristic FJSSP solving techniques, ACO has been shown to be a particularly promising approach [**?**]. The practical difficulty remains to escape local optima and reliably converge to high-quality solutions within a short computing time limit. This

challenge has brought about a variety of extended ACO algorithms as well as hybrid approaches that combine ACO with local search methods [**?**,**?**,**?**,**?**,**?**,**?**]. While these methods have been designed and evaluated on small to medium-scale FJSSP benchmarks [**?**], our work addresses the large-scale SMSP instances encountered in the domain of semiconductor production scheduling [1]. Beyond FJSSP and SMSP investigated here, we note that hybrid optimization algorithms integrating meta-heuristics and local search have also been adopted in a variety of other application settings [**?**,**?**,**?**,**?**,**?**].

4 Problem Formulation

We formulate SMSP in terms of the general FJSSP model, using the basic notations listed in Table 1. In detail, our setting for scheduling the production of a semiconductor fab is characterized as follows:

- The fab consists of M machines, which are partitioned into T tool groups, where $t_m \in \{1, \ldots, T\}$ denotes the tool group to which a machine $m \in \{1, \ldots, M\}$ belongs.
- There are J jobs, where each $j \in \{1, \ldots, J\}$ represents a sequence of operations $O_{1,j,t_1}, \ldots, O_{n_j,j,t_n}$, to be performed on a production lot. Note that $t_i \in \{1, \ldots, T\}$ specifies the tool group responsible for processing an operation O_{i,j,t_i}, but not a specific machine of t_i, which reflects flexibility in assigning operations to machines. The total number of operations is denoted by $N = \sum_{j \in \{1,\ldots,J\}} n_j$.
- For each operation $O_{i,j,t}$, the duration $d_{i,j,t}$ is required for processing $O_{i,j,t}$ on some machine of the tool group t.

Our SMSP model reflects several features originating from the semiconductor production scenarios of the SMT2020 dataset [1]. In the SMT2020 scenarios, jobs are associated with particular products, and their operation sequences, also called production routes, coincide for the same product. Since such production routes include hundreds of operations that are performed over several months in the physical fab, distinct lots of the same product are frequently at different steps of their routes when they get (re-)scheduled. Hence, we do not explicitly distinguish production routes by products but consider operation sequences of length n_j relative to a job j, which allows for separating the operations on lots that are at different manufacturing steps of the same route.

Moreover, the machines belonging to a tool group are assumed to be uniform, i.e., an operation requiring the tool group can be processed by any of its machines. This simplifying assumption ignores specific machine setups, which may be needed for some operations and take additional equipping time, as well as unavailabilities due to maintenance procedures or breakdowns. However, the greedy machine assignment performed by our GSACO algorithm in Section 5 can take such conditions into account for allocating an operation to the earliest available machine. In addition, some transportation time is required to move a lot from one machine to another between operations, which is not explicitly given but taken as part of the operation duration in the SMT2020 scenarios.

A schedule allocates each operation $O_{i,j,t}$ to some machine $m \in \{1, \ldots, M\}$ such that $t_m = t$, and we denote the machine assignment by $O_{i,j,t,m}$. Each machine performs its assigned operations in sequence without preemption, i.e., $s_{i,j,t,m} + d_{i,j,t} \leq$

Table 2. GSACO parameters

Parameter	Description
l	Cycles/time limit
n	Operations per lot
d	Planning horizon
k	Number of ants
τ_y	Initial pheromone level
τ_z	Minimum pheromone level
τ_e	Pheromone level on edge e
ρ	Evaporation rate
c	Contribution of best schedules

$s_{i',j',t,m}$ or $s_{i',j',t,m} + d_{i',j',t} \leq s_{i,j,t,m}$ must hold for the start times $s_{i,j,t,m}$ and $s_{i',j',t,m}$ of operations $O_{i,j,t,m} \neq O_{i',j',t,m}$ allocated to the same machine m. The precedence between operations of a job $j \in \{1, \ldots, J\}$ needs to be respected as well, necessitating that $s_{i,j,t,m} + d_{i,j,t} \leq s_{i+1,j,t',m'}$ when $i < n_j$. Assuming that $0 \leq s_{1,j,t,m}$ for each job $j \in \{1, \ldots, J\}$, the makespan to complete all jobs is given by $\max\{s_{n_j,j,t,m} + d_{n_j,j,t} \mid j \in \{1, \ldots, J\}\}$. We take makespan minimization as the optimization objective for scheduling, since it reflects efficient machine utilization and maximization of fab throughput.

An example schedule (generated by GSACO) for three jobs with three operations each is displayed in Figure **??**. The start times for the operations of job 1 illustrate the scheduling constraints. That is, we have $s_{1,1,3,5} + d_{1,1,3} = 0 + 10 = s_{2,1,4,7}$ due to the precedence between the first and second operation, while the machine 7 performing the second operation is already available at time 9. Regarding the start of the third operation, $s_{2,1,4,7} + d_{2,1,4} = 10 + 2 < 13 = s_{3,1,1,1}$, i.e., the third operation needs to wait for machine 1 to complete the execution of another operation (of job 2). The completion time $s_{3,3,1,2} + d_{3,3,1} = 12 + 13$ of the third operation of job 3 constitutes the makespan 25 of the example schedule in Figure **??**.

5 GSACO Algorithm

The framework of our GSACO algorithm, whose (input and internal) parameters are summarized in Table 2, is displayed in Figure **??**. Its four submodules, indicated in bold, are described in separate subsections below.

Moreover, Algorithm 1 provides a pseudo-code representation of GSACO. For a configurable cycle number or time limit l, each of the k ants applies greedy search using the GS procedure (detailed in Algorithm **??**). That is, the first GS phase constructs an operation sequence, which is then taken as basis for greedily assigning the operations to machines in the second phase. Note that the ants run independently, so that their GS trials can be performed in parallel. As a result, k schedules along with edges between operations (described in Subsection 5.2) that have been selected for their construction are obtained. If some of these schedules improves the makespan over the best schedule found in previous iterations (if any), the best schedule gets updated. As common for

Input: dataset, l
Output: best schedule found by ants
Parameters: d, n, k, τ_y, τ_z, ρ, c
Initialize adjacency, pheromone, and machine matrix;
$operations \leftarrow \infty$;
while *cycle or time limit* l *is not reached* **do**
  foreach *ant* **from** 1 **to** k **do**
    Run GS procedure to find a schedule;
  end
  $new \leftarrow$ maximum operations of ants' schedules;
  if $new > operations$ **then**
    $operations \leftarrow new$;
    $best \leftarrow$ an ant's schedule of $operations$;
  end
  foreach *edge* e *in pheromone matrix* **do**
    $\tau_e \leftarrow \max\{\rho \cdot \tau_e, \tau_z\}$; *// evaporation*
  end
  foreach *edge* e *selected by best ant* **do**
    $\tau_e \leftarrow \tau_e + c$; *// deposit pheromones*
  end
end
return $best$;

Algorithm 1: Greedy Search based ACO (GSACO)

ACO algorithms, pheromones τ_e on edges e are subject to evaporation, according to the formula $\rho \cdot \tau_e$, while edges selected to construct the best schedule obtained so far also receive a pheromone contribution, calculated as $\tau_e + c$. Such pheromone deposition increases the chance for edges contributing to the current best schedule to get re-selected in forthcoming iterations.

5.1 Input Module

This module reads in an SMSP instance, e.g., obtained from the SMT2020 dataset [1], specifying production routes, the tool groups with their machines, and the jobs to be performed. Moreover, a limit l on the number of cycles and/or the time to spend on optimization by GSACO is given as input.

5.2 Initialization Module

In view of long production routes with hundreds of operations in the SMT2020 dataset, we introduce a configurable planning horizon n as upper bound on the length n_j of the operation sequence for a job j. The planning horizon thus constitutes a scaling factor for the size and the resulting complexity of SMSP instances. In practice, unpredictable stochastic events make long-term schedules obsolete and necessitate frequent re-scheduling, where limiting the planning horizon upfront provides a means to control the search and enable short response times.

Table 3. Example operations

No.	Operation	Tool group name	Duration
1	$O_{1,1,3}$	Diffusion_FE_125	10
2	$O_{2,1,4}$	WE_FE_84	2
3	$O_{3,1,1}$	DefMet_FE_118	6
4	$O_{1,2,2}$	Diffusion_FE_120	8
5	$O_{2,2,4}$	WE_FE_84	1
6	$O_{3,2,1}$	DefMet_FE_118	4
7	$O_{1,3,2}$	Diffusion_FE_120	9
8	$O_{2,3,4}$	WE_FE_84	3
9	$O_{3,3,1}$	DefMet_FE_118	13

To express SMSP as a search problem on graphs, we identify an instance with the disjunctive graph

whose vertices V contain the operations $O_{i,j,t}$ plus a dummy start node 0, conjunctive edges

$$E_c = \{(0, O_{1,j,t_1}) \mid O_{1,j,t_1} \in V\} \\ \cup \{(O_{i-1,j,t_{i-1}}, O_{i,j,t_i}) \mid O_{i,j,t_i} \in V, i > 1\} \tag{1}$$

connect the dummy start node 0 to the first operation and each operation on to its successor (if any) in the sequence for a job, and disjunctive edges

$$E_d = \{(O_{i,j,t}, O_{i',j',t}) \mid O_{i,j,t} \in V, O_{i',j',t} \in V, j \neq j'\} \tag{2}$$

link operations (of distinct jobs) sharing a common tool group, as such operations may be allocated to the same machine.

Any feasible schedule induces an acyclic subgraph (V, E) of the disjunctive graph G such that $E_c \subseteq E$, and $(O_{i,j,t}, O_{i',j',t}) \in E_d \cap E$ iff $s_{i,j,t,m} + d_{i,j,t} < s_{i',j',t,m}$ for distinct jobs $j \neq j'$, i.e., the operation $O_{i,j,t}$ is processed before $O_{i',j',t}$ by the same machine m of tool group $t_m = t$. Conversely, the search for a high-quality solution can be accomplished by determining an acyclic subgraph (V, E) of G that represents a schedule of short makespan.

For example, Table 3 shows nine operations belonging to the operation sequences for three jobs, as they can be obtained with the parameter $n = 3$ for the planning horizon. Conjunctive edges connect the dummy start node 0 to the operations numbered 1, 4, and 7, which come first in their jobs, then operation 1 is connected on to 2 as well as 2 to 3, and similarly for the other two jobs. In addition, mutual disjunctive edges link operations to be processed on the same tool group, e.g., those numbered 4 and 7 have the tool group *Diffusion_FE_120* in common. The resulting $(N + 1) \times (N + 1)$ adjacency matrix, where $N = 9$ is the total number of operations, 0 entries indicate the absence, and 1 entries the existence of edges, is given in Figure 3.

As initial pheromone level on edges $e \in E_c \cup E_d$, we take $\tau_y = 1$ by default. In general, representing pheromone levels by an $(N + 1) \times (N + 1)$ matrix similar to the adjacency matrix, the entries τ_e are initialized according to the following condition:

$$\tau_e = \tag{3}$$

$$
\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array}
\begin{array}{c}
\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\
\left[\begin{array}{cccccccccc}
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}\right]
\end{array}
$$

Fig. 3. Adjacency matrix for the operations in Table 3

With $\tau_y = 1$, this reproduces the adjacency matrix in Figure 3 as initial pheromone matrix for our example.

We additionally represent the possible machine assignments by an $(N + 1) \times M$ machine matrix, where M is the total number of machines. For example, with two machines per tool group and the mapping $t_m = \lceil \frac{m}{2} \rceil$ from machine identifiers $m \in \{1, \ldots, 8\}$ to the tool groups $t \in \{1, \ldots, 4\}$, responsible for processing the operations in Table 3, we obtain the machine matrix shown in Figure 4. While the dummy start node 0 needs no machine to process it, the operation $O_{1,1,3}$ numbered 1 can be allocated to either the machine 5 or 6 of tool group 3, and corresponding 1 entries indicate the machines available to perform the other operations as well.

$$
\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\left[\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right]
\end{array}
$$

Fig. 4. Machine matrix for the operations in Table 3

5.3 GS Module

The general goal of greedy search methods consists of using heuristic decisions to find high-quality, but not necessarily optimal solutions in short time. Within GSACO, each ant applies greedy search to efficiently construct some feasible schedule for a given

SMSP instance. The respective GS procedure, outlined by the pseudo-code in Algorithm **??**, includes two phases: operation sequencing and machine assignment.

The first phase constructs a sequence comprising all operations of an SMSP instance. To this end, a probabilistic decision rule based on the pheromone matrix selects edges $(O', O_{i,j,t})$ and adds their target operations $O_{i,j,t}$ to the sequence one by one. As invariant ensuring the feasibility of a resulting schedule, the predecessor operation $O_{i-1,j,t'}$ of the same job j must already belong to the sequence in case $i > 1$. This is accomplished by maintaining a set $next$ of selectable conjunctive and disjunctive edges $(O', O_{i,j,t})$ such that O' is the dummy start node 0 or already in sequence, while $O_{i,j,t}$ is the first yet unsequenced operation of its job j.

For example, starting with an empty sequence for the operations in Table 3, the initially selectable edges are $(0, O_{1,1,3})$, $(0, O_{1,2,2})$, and $(0, O_{1,3,2})$. Assuming that $(0, O_{1,2,2})$ gets selected, the first operation $O_{1,2,2}$ of job 2 is added to the sequence, and the conjunctive edge $(O_{1,2,2}, O_{2,2,4})$ as well as the disjunctive edge $(O_{1,2,2}, O_{1,3,2})$ replace $(0, O_{1,2,2})$ among the selectable edges. The process of selecting some edge to a yet unsequenced operation is repeated until each operation along with an edge targeting it has been processed. Note that selected disjunctive edges link operations based on tool groups, i.e., they do not reflect a machine assignment to be made in the second phase.

With a sequence of operations at hand, the second phase allocates operations one by one to an earliest available machine. Reconsidering the operations $O_{1,2,2}$ and $O_{1,3,2}$ as well as the machine matrix in Figure 4, where these operations are numbered 4 and 7, we can first allocate $O_{1,2,2}$ to the free machine 3 with the start time $s_{1,2,2,3} = 0$. This means that machine 3 becomes available again at time $a_3 = s_{1,2,2,3} + d_{1,2,2} = 8$. However, the other machine 4 of tool group 2 is still free, so that $O_{1,3,2}$ is greedily assigned to machine 4 with the start time $s_{1,3,2,4} = 0$ and the updated availability $a_4 = s_{1,3,2,4} + d_{1,3,2} = 9$. As we have that $a_3 = 8 < 9 = a_4$, if a third operation were to be allocated to either the machine 3 or 4 of tool group 2, our greedy assignment strategy would decide for machine 3. The described machine assignment process allocates operations according to the sequence from the first phase, yielding a feasible schedule that assigns the machines and start times for all operations.

Figure **??** displays a sequence for the operations in Table 3, where sequence numbers indicate the order in which edges to the operations are selected, as it can be generated by the GS procedure. The edges running vertically are conjunctive and connect the dummy start node 0 to the first operations of the three jobs as well as predecessors to their successor operations. In addition, two disjunctive edges are selected between the second or third operation, respectively, of job 2 and the corresponding operation of job 1, considering that these operations have the tool group 4 or 1 in common.

The resulting greedy machine assignment, computed in the second phase of the GS procedure, is denoted by m within the node labels $O_{i,j,t,m}$ in Figure **??**. In view of two machines available per tool group and three operations sharing each of the tool groups 1 and 4, the first two operations according to the sequence from the first phase, i.e., $O_{2,2,4}$ and $O_{2,3,4}$ or $O_{3,2,1}$ and $O_{3,3,1}$, respectively, get distributed among the two machines of each tool group, which leads to the machine assignments $O_{2,2,4,7}$ and $O_{2,3,4,8}$ as well as $O_{3,2,1,1}$ and $O_{3,3,1,2}$. Which machines of the tool groups 4 and 1 then become available first to allocate the operations $O_{2,1,4}$ and $O_{3,1,1}$ can be read off

	0	1	2	3	4	5	6	7	8	9
0	0	1.3	0	0	1.3	0	0	1.3	0	0
1	0	0	1.3	0	0	0	0	0	0	0
2	0	0	0	1.3	0	4.9	0	0	4.9	0
3	0	0	0	0	0	0	4.9	0	0	4.9
4	0	0	0	0	0	1.3	0	4.9	0	0
5	0	0	4.9	0	0	0	1.3	0	4.9	0
6	0	0	0	4.9	0	0	0	0	0	4.9
7	0	0	0	0	4.9	0	0	0	1.3	0
8	0	0	4.9	0	0	4.9	0	0	0	1.3
9	0	0	0	4.9	0	0	4.9	0	0	0

Fig. 5. Updated pheromone matrix for the operations in Table 3 after a few GSACO iterations

the corresponding schedule shown in Figure **??**. That is, the machine 7 of tool group 4 is available from time 9, yielding the greedy assignment $O_{2,1,4,7}$. Similarly, $O_{3,1,1,1}$ is obtained because the machine 1 gets free at time 13, while the other machine 2 of tool group 1 processes the third operation of job 3 until time 25, which is the makespan of the constructed schedule. Given that this makespan coincides with the cumulative duration of job 3 and its operations must be processed in sequence, the feasible schedule in Figure **??** happens to be optimal for the operations in Table 3.

5.4 Evaluation Module

While each ant independently constructs a feasible schedule by means of the GS procedure, the evaluation module collects the results obtained by the ants in a GSACO iteration. Among them, a schedule of shortest makespan is determined as outcome of the iteration and stored as new best solution in case it improves over the schedules found in previous iterations (if any). After evaporating pheromones τ_e by $\rho \cdot \tau_e$, where τ_z remains as minimum pheromone level if the obtained value would be smaller, the edges e that have been selected by the GS procedure for constructing the current best schedule receive a pheromone contribution and are updated to $\tau_e + c$.

Note that our contribution parameter c is a constant, while approaches in the literature often take the inverse of an objective value [**?**], i.e., of the makespan in our case. The latter requires careful scaling to obtain non-marginal pheromone contributions, in particular, when makespans get as large as for SMSP instances. We instead opt for pheromone contributions such that the edges selected to construct best schedules are certain to have an increased chance of getting re-selected in forthcoming iterations. For our example in Table 3, Figure 5 provides the updated pheromone matrix after a few iterations of GSACO with the parameter values $\rho = 0.7$ and $c = 0.5$, showing a clear separation between the more frequently selected and futile edges.

Table 4. GSACO input parameter values

Parameter	Value
l	10
n	1–5
k	10
τ_y	1
τ_z	0.00001
ρ	0.7
c	0.5

6 Simulation

7 EXPERIMENTAL EVALUATION

We implemented our GSACO algorithm in Python using PyTorch, as it handles tensor operations efficiently and provides a multiprocessing library for parallelization, thus significantly speeding up the ants' execution of the GS procedure in each GSACO iteration.[3] The first challenge consists of determining suitable values for the input parameters listed in Table 2, i.e., all parameters but the internally calculated pheromone level τ_e on edges e. We commit to the values given in Table 4, where a time limit l of 10 minutes and a planning horizon n of up to 5 operations per job are plausible for SMSP instances based on the SMT2020 dataset, whose stochastic events necessitate frequent re-scheduling in practice. For the initial pheromone level τ_y, we start from value 1, and take 0.00001 as the minimum τ_z to avoid going down to 0, considering that the GS procedure can only select edges with non-zero entries in the pheromone matrix. The values for the number k of ants, the evaporation rate ρ, and the pheromone contribution c are more sophisticated to pick. That is, we tuned these parameters in a trial-and-error process that, starting from a baseline, inspects deviations of the final makespan and convergence speed obtained with iterative modifications. Certainly, an automated approach would be desirable to perform this task efficiently for new instance sets.

To compare GSACO with the state of the art in FJSSP solving, we also run the CP solver OR-Tools (version 9.5) [**?**], while heuristic and meta-heuristic methods from the literature could not be reproduced due to the inaccessibility of source code or tuned hyperparameters. Note that CP approaches have been shown to be particularly effective among exact optimization techniques for FJSSP [**?**], where the free and open-source solver OR-Tools excels as serial winner of the MiniZinc Challenge.[4]

We performed our experiments on a TUXEDO Pulse 14 Gen1 machine equipped with an 8-core AMD Ryzen 7 4800H processor at 2.9GHz and onboard Radeon graphics card.

Table 5 illustrates the strengths of CP models on a benchmark set of small to medium-scale FJSSP instances [**?**], where the instances MK1–MK4 are due to [**?**] and

[3] The source code is publicly available in our GitHub repository: https://github.com/prosysscience/GSACO

[4] https://www.minizinc.org/challenge

Table 5. FJSSP results obtained with CP and GSACO

Instance	J	M	CP	GSACO
MK1	10	6	40	44
MK2	10	6	27	40
MK3	15	8	204	239
MK4	15	8	60	83
SFJSSP1	2	2	66	66
SFJSSP2	2	2	107	107
SFJSSP3	3	2	221	221
SFJSSP4	3	2	355	355
MFJSSP1	5	6	468	498
MFJSSP2	5	7	446	470
MFJSSP3	6	7	466	523
MFJSSP4	7	7	554	664

Table 6. Number of jobs, machines, and operations for SMSP instances

Scenario	J	M	O
LV/HM	2156	1313	up to 10747
HV/LM	2256	1443	up to 11218

the remaining eight instances have been introduced in [**?**]. In view of the small numbers J and M of jobs or machines, respectively, in these classical FJSSP instances, OR-Tools solves all of them to optimality within a few seconds, so that the CP column shows the minimal makespan for each instance. Since GSACO is an approximation method, its best schedules are not necessarily optimal, as it can be observed on instances other than SFJSSP1–SFJSSP4. This reconfirms that exact models, particularly those based on CP [**?**], are the first choice for FJSSP instances of moderate size and complexity.

To evaluate large-scale SMSP instances, we consider two semiconductor production scenarios of the SMT2020 dataset: Low-Volume/High-Mix (LV/HM) and High-Volume/Low-Mix (HV/LM). As indicated in Table 6, both scenarios include more than 2000 jobs and more than 1300 machines, modeling the production processes of modern semiconductor fabs. The main difference is given by the number of products and associated production routes for jobs, where LV/HM considers 10 production routes varying between 200–600 steps in total, while HV/LM comprises 2 production routes with about 300 or 600 steps, respectively. Originally, the LV/HM and HV/LM scenarios have been designed to represent fab load at the start of simulation runs, so that the jobs are at different steps of their production routes. We here instead focus on scheduling for planning horizons n from 1 to 5, standing for the up to 5 next operations to be performed per job. Hence, the operations O to schedule gradually increase from the number J of jobs, in case of the planning horizon $n = 1$, to more than 10000 operations for the longest horizon $n = 5$.

While running with a time limit l of 10 minutes, Table 7 reports the makespans of current best schedules found by the CP solver OR-Tools and our GSACO implementation at 1, 3, 5, 7, and 9 minutes of computing time. Considering that the SMSP instances are large, OR-Tools now takes 3 minutes to come up with the first solution(s) for the

Table 7. SMSP results obtained with CP and GSACO

		1 min		3 min		5 min		7 min		9 min	
n	Scenario	CP	GSACO	CP	GSACO	CP	GSACO	CP	GSACO	CP	GSACO
1	LV/HM	-	3735	18572	3725	3746	3725	3723	3725	3723	3725
	HV/LM	-	1405	-	1405	2242	1405	1609	1405	1600	1405
2	LV/HM	-	3773	-	3751	-	3750	-	3739	4398	3739
	HV/LM	-	1653	-	1644	-	1611	-	1611	-	1611
3	LV/HM	-	3880	-	3867	-	3836	-	3834	-	3834
	HV/LM	-	1902	-	1889	-	1889	-	1876	-	1876
4	LV/HM	-	4578	-	4540	-	4540	-	4540	-	4540
	HV/LM	-	2207	-	2113	-	2113	-	2093	-	2093
5	LV/HM	-	4680	-	4680	-	4553	-	4553	-	4553
	HV/LM	-	2667	-	2566	-	2566	-	2518	-	2518

shortest planning horizon $n = 1$ on the LV/HM scenario. Within the same fraction of computing time, GSACO already converges to a makespan of 3725 and then proceeds with iterations that do not yield further improvements. That the best schedule found by GSACO is not optimal is witnessed by a marginally better solution obtained by OR-Tools after 7 minutes. However, for the other SMSP instances, OR-Tools is unable to improve over GSACO within 9 minutes, and it cannot even provide feasible schedules for planning horizons from $n = 2$ on the HV/LM scenario or $n = 3$ on LV/HM.

The quick convergence of our GSACO implementation is also outlined by the makespan improvements plotted in Figure **??**. On the LV/HM scenario displayed in Figure **??**, GSACO obtains its best schedules within 7 minutes for all planning horizons. Only for the longest planning horizon $n = 5$ on the HV/LM scenario in Figure **??**, an improvement occurs after more than the 9 minutes of computing time listed in the rightmost column of Table 7. Comparing the SMSP instances for which OR-Tools manages to provide feasible schedules, the convergence to makespans in roughly the range of GSACO's results takes significantly more computing time. Hence, the time limit that would be necessary to break even with GSACO, as accomplished within 7 minutes for the shortest planning horizon $n = 1$ on the LV/HM scenario, cannot be predicted for the SMSP instances with longer planning horizons. Moreover, we observe that the initial schedules obtained in the first GSACO iteration by some of the ants running in parallel are of relatively high quality, while OR-Tools sometimes finds outliers as its first solutions. This phenomenon occurs for the planning horizons $n = 1$ and $n = 2$ on the LV/HM or HV/LM scenario, respectively, where the latter schedule of makespan 17664 is found after more than 9 minutes and thus not listed in Table 7.

8 CONCLUSION

Modern semiconductor fabs are highly complex and dynamic production environments, whose efficient operation by means of automated scheduling and control systems constitutes a pressing research challenge. In this work, we model the production processes of large-scale semiconductor fabs in terms of the well-known FJSSP. In contrast to classical FJSSP benchmarks, this leads to large-scale scheduling problems, even if short

planning horizons cover only a fraction of the long production routes with hundreds of operations. The resulting size and complexity of large-scale SMSP instances exceed the capabilities of common FJSSP solving methods. We thus propose the GSACO algorithm combining probabilistic operation sequencing with greedy machine assignment. Its efficient implementation utilizing tensor operations and parallel ant simulations enables GSACO's convergence to high-quality solutions in short time. In this way, GSACO overcomes the scalability limits of exact optimization by means of a state-of-the-art CP approach and achieves the performance required for frequent re-scheduling in reaction to process deviations.

While simplified FJSSP models of semiconductor manufacturing processes provide insights regarding the scalability of scheduling methods, in future work, we aim at extending GSACO to incorporate the specific conditions and functionalities of tools performing the production operations. Such features include, e.g., machine setups to be equipped before performing operations, preventive maintenance procedures for which a machine needs to stay idle, and batch processing capacities to handle several production lots in one pass. In practice, these specifics matter for machine assignment strategies and should also be reflected in the respective greedy phase of GSACO.

Our second target of future work concerns the utilization of schedules found by GSACO for decision making and control within simulation models of semiconductor fabs. This goes along with the extension of optimization objectives to practically relevant performance indicators, such as deadlines for the completion of jobs and minimizing the tardiness. In view of the dynamic nature of real-world production operations and unpredictable stochastic events, quantifying the improvements by optimized scheduling methods requires their integration and evaluation in simulation.

ACKNOWLEDGEMENTS

This work has been funded by the FFG project 894072 (SwarmIn). We are grateful to the anonymous reviewers for constructive comments that helped to improve the presentation of this paper.

9 Add-on

Dispatching and scheduling are critical in semiconductor manufacturing because they directly impact the throughput and utilization of resources. Dispatching refers to the process of assigning work to specific machines in real-time, while scheduling involves pre-planning the sequence and timing of operations to optimize certain objectives, such as minimizing the total completion time or balancing the load across different machines [4].

For smaller volumes at workcenters in the fab, which are modeled as flexible job shops, optimal solutions can be achieved using mathematical optimization [12]. However, in larger and more dynamic environments, the complexity and computational time constraints limit the feasibility of applying mathematical optimization. Consequently,

optimization is typically implemented on a local scale, isolated to individual workcenters. In complex job shop scenarios, this localized approach to production scheduling may lead to solutions that are globally sub-optimal.

The complexity of scheduling is intensified by the unpredictable nature of processing times and machine availability. Variability in processing times may arise from differences in equipment performance, material handling durations, or the unique characteristics of each set of wafers. Furthermore, frequent machine breakdowns can lead to substantial disruptions, underscoring the need for strong and adaptable scheduling strategies [13].

The objectives of this paper are to:

1. Compare the dispatcher and scheduler over the planning horizon
2. Scheduling under different machine availabilities

9.1 A Subsection Sample

Please note that the first paragraph of a section or subsection is not indented. The first paragraph that follows a table, figure, equation etc. does not need an indent, either.

Subsequent paragraphs, however, are indented.

Sample Heading (Third Level) Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 8 gives a summary of all heading levels.

Table 8. Table captions should be placed above the tables.

Heading level	Example	Font size and style
Title (centered)	**Lecture Notes**	14 point, bold
1st-level heading	**1 Introduction**	12 point, bold
2nd-level heading	**2.1 Printing Area**	10 point, bold
3rd-level heading	**Run-in Heading in Bold.** Text follows	10 point, bold
4th-level heading	*Lowest Level Heading.* Text follows	10 point, italic

Displayed equations are centered and set on a separate line.

$$x + y = z \tag{4}$$

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead (see Fig. 6).

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

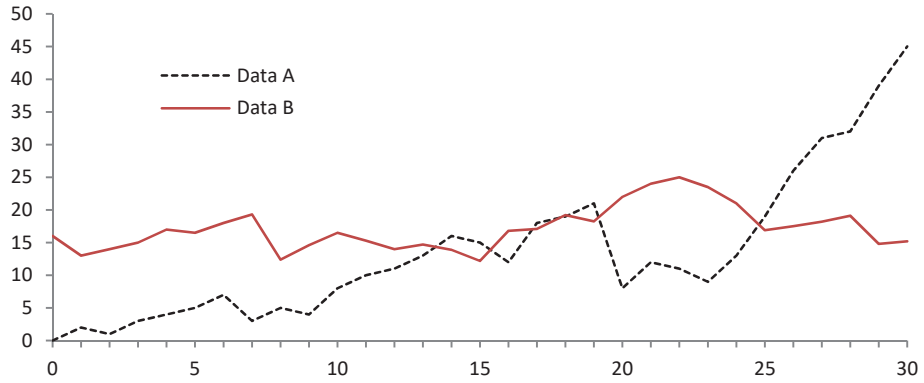

Fig. 6. A figure caption is always placed below the illustration. Please note that short captions are centered, while long ones are justified by the macro package automatically.

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, we prefer the use of square brackets and consecutive numbers. Citations using labels or the author/year convention are also acceptable. The following bibliography provides a sample reference list with entries for journal articles [16], an LNCS chapter [17], a book [18], proceedings without editors [19], and a homepage [20]. Multiple citations are grouped [16–18], [16, 18–20].

Acknowledgments. A bold run-in heading in small font size at the end of the paper is used for general acknowledgments, for example: This study was funded by X (grant number Y).

Disclosure of Interests. It is now necessary to declare any competing interests or to specifically state that the authors have no competing interests. Please place the statement with a bold run-in heading in small font size beneath the (optional) acknowledgments[5], for example: The authors have no competing interests to declare that are relevant to the content of this article. Or: Author A has received research grants from Company W. Author B has received a speaker honorarium from Company X and owns stock in Company Y. Author C is a member of committee Z.

References

1. Kopp, D., Hassoun, M., Kalir, A., Mönch, L.: SMT2020—a semiconductor manufacturing testbed. IEEE Transactions on Semiconductor Manufacturing 33(4):522–531 (2020)
2. Hopp, W. J., Spearman, M. L.: Factory physics. Waveland Press. (2011)
3. Uzsoy, R., Lee, C. Y., Martin-Vega, L. A.: A review of production planning and scheduling models in the semiconductor industry part I: system characteristics, performance evaluation and production planning. IIE transactions, 24(4), 47-60 (1992)

[5] If EquinOCS, our proceedings submission system, is used, then the disclaimer can be provided directly in the system.

4. Pinedo, M. L.: Scheduling: theory, algorithms, and systems. 6th edn. Springer (2022)
5. May, G. S., Spanos, C. J.: Fundamentals of semiconductor manufacturing and process control. John Wiley & Sons. (2006)
6. Mönch, L., Fowler, J. W., Dauzère-Pérès, S., Mason, S. J., Rose, O.: A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. Journal of scheduling, 14, 583-599. (2011)
7. Ali, R., Qaiser, S., El-Kholany, M. M., Eftekhari, P., Gebser, M., Leitner, S., Friedrich, G. A Greedy Search Based Ant Colony Optimization Algorithm for Large-Scale Semiconductor Production. In Proceedings of the 14th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2024), pages 138-149. (2024)
8. Dorigo, M., Stützle, T.: Ant colony optimization: overview and recent advances. In Handbook of Metaheuristics, pages 311–351. Springer (2019)
9. Papadimitriou, C. H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall. (1982)
10. Perron, L., Didier, F., Gay, S.: The CP-SATLP solver (invited talk). In Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming, pages 3:1–3:2. Leibniz International Proceedings in Informatics, (2023)
11. Kovács, B., Tassel, P., Ali, R., El-Kholany, M., Gebser, M., Seidel, G.: A customizable simulator for artificial intelligence research to schedule semiconductor fabs. In 2022 33rd Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC) (pp. 1-6). IEEE, (2022, May)
12. Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., Kyek, A.: Deep reinforcement learning for semiconductor production scheduling. In 2018 29th annual SEMI advanced semiconductor manufacturing conference (ASMC) pp. 301-306. IEEE, (2018, April)
13. Leachman, R. C., Hodges, D. A.: Benchmarking semiconductor manufacturing. IEEE transactions on semiconductor manufacturing 9(2), 158-169 (1996)
14. McKay, K. N., Wiers, V. C.: Planning, scheduling and dispatching tasks in production control. Cognition, Technology & Work, 5, 82-93 (2003)
15. Chan, C. W.: Situation aware dispatching system for semiconductor manufacturing. (2024)
16. Author, F.: Article title. Journal **2**(5), 99–110 (2016)
17. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016). https://doi.org/10.10007/1234567890
18. Author, F., Author, S., Author, T.: Book title. 2nd edn. Publisher, Location (1999)
19. Author, A.-B.: Contribution title. In: 9th International Proceedings on Proceedings, pp. 1–2. Publisher, Location (2010)
20. LNCS Homepage, http://www.springer.com/lncs, last accessed 2023/10/25