Simulation of Swarm Intelligence for Flexible Job-Shop Scheduling with SwarmFabSim: Case Studies with Artificial Hormones and an Ant Algorithm

Martina Umlauft[1], Melanie Schranz[2], and Wilfried Elmenreich[3]

[1] Lakeside Labs GmbH, Klagenfurt, Austria umlauft@lakeside-labs.com
[2] Lakeside Labs GmbH, Klagenfurt, Austria schranz@lakeside-labs.com
[3] Institute of Networked and Embedded Systems, University of Klagenfurt, Klagenfurt, Austria wilfried.elmenreich@aau.at

Abstract. This book chapter presents SwarmFabSim, an open-source simulation framework using NetLogo for agent-based simulation of a semiconductor production plant organized by the job-shop principle. It models the plant as a self-organizing system using swarm intelligence algorithms for bottom-up optimization. The model is composed of a set of agents, which includes machines, workcenters, lots, and processes. While the general model is not restricted to a particular programming language, SwarmFabSim and the presented case studies are implemented in NetLogo, one of the most widely used agent-based simulation platforms. SwarmFabSim consists of a structured system of code modules using a callback architecture, allowing to easily modify or exchange the swarm algorithm under investigation. Users can configure their fab model and simulations via the user interface and configuration files. The results are presented to the user via log files that include detailed results as well as overall key performance indicators: makespan, average flow factor, and lot tardiness. SwarmFabSim (running on NetLogo version 6.1 or later), including sample swarm algorithms and configuration files is available as an open-source project on GitHub. Additionally to the framework SwarmFabSim, we present a detailed case study of two algorithms for fab optimization. Together with the algorithms, we describe three different fab setups in SwarmFabSim and demonstrate how to use the framework for their evaluation. A detailed comparison of the performance results of the algorithms complements this chapter.

1 Introduction

The production processes of integrated circuits (ICs) in the semiconductor industry are organized according to the flexible job-shop principle [15]. As an NP-hard problem [14], such factory-wide process scheduling tasks establish a challenging problem. In this chapter, we particularly focus on the so-called front-end processing, where IC structures are created on silicon wafers. The factory model is inspired by the actual requirements of the semiconductor manufacturer Infineon Technologies[4].

In such an exemplary fab, it is necessary to coordinate product manufacturing of over 400 to 1200 different stations. Typically, these fabs manufacture over

[4]https://www.infineon.com

1500 different products in about 300 process steps, including lithography, doping, oxidation, etching, and measurement [15]. Thus, the great diversity of products and the historical growth of the industrial plant further increase the complexity. In addition to logistical boundary conditions, such as equipment tooling, time coupling, and batch processing, the chain of steps required in the semiconductor manufacturing process contains loops, making the problem even more difficult to solve.
An industrial context where traditional optimization methods like linear optimization reach their limits due to excessive computation time establishes a new field of application for swarm intelligence. Swarm algorithms consist of agents following local rules and interactions and can be simulated via agent-based models. In nature, swarms regularly solve complex problems using relatively simple rules for their agents. An example is the foraging strategy of ants, which solves the problem of coordination between exploration, i.e., the search for new, unknown food sources, and exploitation, i.e., the establishment of transport routes to bring food to the nest [8]. Other swarming animals solve complex movement coordination in three dimensions (e.g., flocks of birds, schools of fish) or self-organized building of complex structures (e.g., , termites), see e.g., [4, 13]. Previous work indicates that such bio-inspired algorithms can be successfully adapted to technical applications where a problem needs to be solved in a technical environment of similar complexity like the natural setting; for a detailed review on such applications, see e.g., [34]. Swarm-based solutions are of great interest due to their simple rules and properties of self-organizing systems, such as scalability, robustness, and adaptability [28]. While swarm algorithms, once implemented, come with a small computational footprint, finding and defining such a self-organizing algorithm for a given problem is a non-trivial task [11]. Trial and error or heuristic optimization approaches usually involve extensive evaluations requiring a proper simulation of the target system [10] . In many cases, the target system contains dynamic and networked processes that cannot be described by analytical methods. In addition to technological and logistical boundary conditions, such as equipment tooling, time coupling, need for secondary resources and batch processing, the chain of necessary process steps in semiconductor manufacturing contains loops, making the problem even harder to solve. The extreme miniaturization of these components leads to high demands being put on the production environment (clean room) and the manufacturing process. Due to computational complexity, existing dispatching rules and linear optimization methods can only be used on a subset of the plant. They can only consider part of the system behavior, leaving room for optimization potential.
In previous work, we have presented an earlier version of SwarmFabSim in [39]. This chapter presents a significant extension of this work with the following content: Section 2 discusses the related work on agent-based modeling environments and other job-shop simulation environments. The agent-based modeling and simulation approach is detailed in Section 3 in the context of the exemplary industrial setting. In Section 3.1, the usage of the agent-based simulation tool NetLogo is discussed. Section 3.2 introduces a model consisting of different types of agents, including workcenters, machines, and lots. The main contribution of this chapter is the implementation of SwarmFabSim framework using NetLogo (Section 4) as a system consisting of several structured code modules that interact using a callback architecture. The user interface, depicted in Section 4.1, and configuration files, explained in Section 4.2, allow the interaction and configuration of

the SwarmFabSim framework according to the requirements of a particular fab model. Despite taking inspiration from semiconductor manufacturing, the presented approach can be also applied to other settings where flexible job-shop scheduling is used. Thus, adaptation is supported by changing the configuration files describing the concrete setting. The implementation of the architecture is reviewed in Section 4.3, followed by a description of the base algorithms included in the framework in Section 4.4. Section 4.5 explains the log files with the implemented key performance indicators (KPIs) used for evaluation. Examples of specific swarm algorithms implemented in the SwarmFabSim framework are given in Section 5.1 by an algorithm based on artificial hormones and an algorithm modeled after the foraging behavior of ants. Section 6 summarizes the main contributions of this chapter and gives a reference to the open-source repository of SwarmFabSim.

2 Related Work

The Job-Shop Scheduling Problem (JSSP) is one of the most widely studied optimization problems. It has been investigated for a long time with the first competitive analysis having been published by Graham in 1966 [18]. The aim of the problem is to find the optimal schedule that minimizes makespan to produce all ordered products given a limited machine park where products compete for machine allocations [42]. The problem is NP-hard [14]. An overview of the classical techniques used to attack the problem is given by Jain and Meeran [24]. Gromico et al. apply dynamic programming to solving the JSSP optimally [19].
Ghasemi [17] presents a mixed-integer linear programming formulation to determine product sequence and sublot sizes for lot streaming in multi-stage hybrid flow shops with non-identical machines for hybrid flow shop scheduling.
Georgiadis [16] provides a comprehensive work of optimal production planning and mixed batch scheduling. In particular, the work shows the application of mixed integer linear programming modeling frameworks to optimize high-complexity production planning and scheduling problems.
Bagheri et al. address the flexible job-shop scheduling problem with an artificial immune algorithm. The algorithm is implemented in C++ and evaluated with reference data sets. The flexible job-shop scheduling problem is also part of the problem described in this chapter but the model used by Bagheri et al. does not have batch machines, which are typical for semiconductor fabs [2].
In [26], Lee et al. propose a scheduling method for a printed circuit board (PCB) manufacturing system. The approach uses new dispatching rules that are reported to perform better than existing dispatching rules and heuristic algorithms for lot sizing in terms of the total tardiness of orders.
Petrovic et al. [27] use a fuzzy rule-based system to determine the lot sizes in a job shop-scheduling problem, which was then integrated into a genetic algorithm for job shop scheduling.
Qin et al. use a two-stage ant colony algorithm for hybrid flow shop scheduling with lot sizing and calendar constraints in printed circuit board assembly [30]. Unlike in our application, they apply the ant colony algorithm to the solution space, though, while we implement the ant algorithm from the bottom up.
An extensive overview of scheduling and lot sizing with sequence-dependent setups can be found in [45].

In many instances, factory processes are simulated using discrete event process flow models with simulators specific to the respective application area. For semiconductor manufacturing, examples include AutoMod / AutoSched AP[5] or D-Simlab/D-Simcon[6]. For our research, the level of detail supported in these simulators is generally too high. Also, because they do not support agent-based modeling, they are not suitable for modeling swarm algorithms.

GPSS (General Purpose Simulation System) is a discrete-time simulation language initially developed by IBM. It is primarily used as a process flow-oriented simulation language, particularly well-suited to problems such as modeling factory operations. There exist different GPSS implementations, for example, GPSS World by Minuteman[7] or JGPSS[8] by the Polytechnic University of Catalonia. A ranked list of the most popular discrete simulation software platforms for commercial use is given in [7].

Zhang et al. address how agent-based modeling can be conceptually applied to job-shop production simulation [44].

Shukla examines multi-agent systems for production planning problems in manufacturing systems. They show how multi-agent systems can offer advantages over traditional approaches and present a conceptual framework for implementing production planning systems using multi-agent systems [35].

Pulikottil et al. review the different frameworks and technologies that support the use of multi-agent systems in manufacturing, the different applications of multi-agent systems in this area, and the current challenges in developing multi-agent systems in smart manufacturing [29].

Gwiazda et al. show how NetLogo can be used to model a production process. The work involves an agent-based modeling simulator and addresses the job-shop problem in the simulated production system using a multiple ant colony approach [21].

Zahmani and Atmani apply a genetic algorithm to solve the scheduling in a job-shop setting. The simulation has been implemented in NetLogo. The genetic algorithm is applied to discover well-functioning allocation rules [22].

Alves et al. address the job-shop scheduling problem with a hybrid approach, including optimization and dynamic agent negotiation. The agent-based model implemented in NetLogo is connected to Matlab to exchange optimized scheduling solutions [1].

Besides NetLogo, there exist several other ABM simulation platforms which are suitable for implementing a job-shop production simulation. For a comprehensive overview, see e.g., CoMSES Net / OpenABM[9]. The most well-known candidates are:

Mason is a discrete-event multi-agent simulator core written in Java, allowing the implementation of custom simulations. MASON was developed as a joint effort between George Mason University's Evolutionary Computation Laboratory and the GMU Center for Social Complexity since 2003. At the time of writing, MASON is actively supported with regular updates[10].

[5] `https://automod.de/autosched-ap/`
[6] `http://www.d-simlab.com/`
[7] `http://www.minutemansoftware.com/simulation.htm`
[8] `https://jgpss.liam.upc.edu/en/about`
[9] `https://www.comses.net/resources/modeling-frameworks/`
[10] `https://cs.gmu.edu/~eclab/projects/mason/`

Repast The Repast Suite is a family of advanced, free, and open-source agent-based modeling toolkits that have been under development for over 15 years. There are different versions of the toolkit supporting Java, C++, and Python[11].

Mesa is an open-source, Python implemented ABM framework. Its goal is to be the Python 3-based alternative to other tools like NetLogo, Repast, or MASON. An advantage is the possible integration with Python's data analysis tools. The tool is under active development since 2015[12].

Anylogic is a commercial platform that supports system dynamics, process-centric discrete event modeling, and agent-based modeling. It is used in several industrial settings, including manufacturing, supply chain, transportation, and logistics[13].

MARS (Multi-Agent Research and Simulation) is an ABM simulator running on .NET. It supports models with spatial reference (Grid-based, GPS-based, Cartesian-based) and without spatial relations (social-based, ordinary numeric). MARS is developed under a research project at the Department of Computer Science at Hamburg University of Applied Sciences[14].

3 Agent-Based Modeling for Flexible Job-Shop Scheduling

Combinatorial problems, specifically scheduling, are mostly addressed with linear optimization. Although these methods are studied intensively, they can only cope with a subset and not with the entire plant. Thus, they do not exploit their full optimization potential [25]. Consequently, applying linear optimization methods do not lead to an optimal solution for job-shop scheduling that can be computed in polynomial time [43].

In this work, we apply swarm intelligence as a novel approach in job-shop scheduling. In swarm intelligence, the behavior of the algorithm is inspired by fish, birds, or ants' natural swarm behavior. Therefore, we model the plant as self-organizing system using agent-based modeling (ABM). Thus, the swarm consists of homogeneous or heterogeneous agents. Each agent follows a set of local rules and makes own decisions due to local information leading to an optimization of the production plant form the bottom-up. Contrary to linear optimization where we calculate a global solution for the optimization, this approach has the advantages of low calculatory overhead, high adaptability, and robustness to local environmental changes [23]. ABM remodels the job-shop scheduling problem from a global solution to a solution that is generated from the bottom-up in a distributed fashion.

Using ABM for swarm algorithms is better suited than system-dynamics simulation (stock and flow) or continuous simulation using differential equations. In ABM, a swarm consists of swarm members that can be modeled as agents. They follow local rules, interact with the environment, communicate with other agents, and react on local information.

Wilensky and Rand [41] give the following guidelines when to use ABM:

[11] `https://repast.github.io/`

[12] `https://mesa.readthedocs.io/en/latest/`

[13] `https://www.anylogic.com/`

[14] `https://mars-group.org/`

- Medium number of agents: several dozen up to about 100000 agents. In our use case, we model up to several thousand agents, typically products and machines.
- Heterogeneity: in ABM, agents can be as heterogeneous as necessary. Thus, in ABM, we can model different product and machine types as agents.
- Local and complex interactions: as used in swarm intelligence can be depicted in ABM.
- Rich environments with agent-like local rules: This can be used to, e.g., model complex machine queue manipulations in our case.
- Time: ABM is a model of process that fits to our job-shop scheduling problem.
- Adaptation: almost no other method can model adaptivity of individual entities well. In ABM, agents' actions and decisions depend on past actions and current information, i.e., agents can learn. This fits the swarm model very well.

3.1 NetLogo for ABM Simulation

One of the most widely used free ABM simulation platforms is NetLogo [40]. It has a good documentation, a mature code base that is actively maintained, and thus, many extensions appear on a regular basis. NetLogo is very well known in the education of ABM and complex systems. Besides education, NetLogo has also been shown to be a sophisticated platform that can perform simulations involving several thousand of agents in feasible computation time [31, 32]. For our own performance results, see Section 4 below. The NetLogo homepage lists more than 3000 research publications from the last 10 years that have used NetLogo as an ABM simulation platform.
NetLogo offers an interactive user interface including an easy possibility for visualization to allow rapid prototyping. To perform mass simulations, it comes with the so-called BehaviorSpace, an easily configurable batch mode to configure any desired number of simulation runs with multiple parameter settings. The simulation results are logged to files and can then be post-processed with a tool of choice (R, Excel, etc.) for statistical evaluation. Additionally, NetLogo supports co-simulation offering interfaces to other programming languages such as Python [20] or R [37].
NetLogo uses a discrete scale for simulation time called ticks. Using so-called breeds NetLogo allows to implement different types of agents. They can interact either directly (based on proximity or their connection via a network topology), or indirectly (based on stigmergic information in the environment).

3.2 Modeling Job-Shop Scheduling

From the exemplary fab described in Section 1, we derived different possibilities for agents, e.g., machines, workcenters, lots, processes, and, recipes. For more details on the process of identifying agents and the corresponding challenges we refer the reader to Schranz et al. [33].
In general, the semiconductor factory can be represented with a directed graph $G = (V, E)$, where the nodes V consist of all machines M_i^m, with m as the machine type. The edges E are defined as a connection between two machines M_i^m and M_j^p if there exists a lot l_n^t of type t that uses a recipe R^t that must be executed

first at machine M_i^m with process P^m and second at machine M_j^p with process P^p. Consequently, the edges E define the neighborhood of each machine M_i. Furthermore, the **machines** M_i^m can be of two different types: single-lot-oriented which process one lot after the other, or batch-oriented which process several lots at once, e.g., such as a furnace. Each machine knows which process or processes they can perform and are aware of their current utilization. Furthermore, they can make local decisions, e.g., re-ordering their queues.
A **workcenter** $W \subset M$ defines a set of machines that can run the same or similar processes P. The exemplary fab model contains several workcenters $W^m = \{M_1^m, M_2^m, \dots\}$.
The standard unit in the semiconductor industry in production is defined as a **lot** consisting of 25 wafers. Each lot is equipped with a transponder that includes a unique identifier. Therefore, in our model we also do *not* consider single wafers[15].
The product type t is defined by the recipe R^t. The recipe contains the set of ordered processing steps that are necessary in order to manufacture the product. As there are typically multiple machines that can perform a processes, a lot l_n^t chooses one out of the suitable machines M_i^m for the current process step P^m.
In semiconductor manufacturing, historically dispatching rules assign lots to machines. In dispatching, there is one queue Q^m for all the lots that need a certain process step P^m performed by machines M_x^m of the same type. The lots are assigned to machines dependent on their priority levels that allow them to change their position in the queue. Thus, the next machine available will take the lot with the highest urgency.
In modern fabs, producers switch from dispatching to scheduling, where each machine M_i^m uses its own queue Q_i^m. Additionally, schedulers are used to optimize the assignments of lots to queues typically for a whole workcenter. Priority levels are still a widely used concept: lots typically still get priority levels assigned which can cause them to change their position in the queue.
In our model each lot is produced step by step according to the process steps in its recipe. Therefore, we need to make two decisions at each step:

1. The position of the lot in the queue. In our implementation of SwarmFabSim, every time a machine is free and wants to take a lot from the queue, the queue can be re-ordered.
2. The model can be run in one of two modes, dispatching or scheduling mode. If in scheduling mode, the lot that needs to move to the next machine, chooses the respective queue Q_i^m of machine M_i^m from all machines M_x^m that can perform the next process step P^m.

These two decisions are reflected in the callback API (application programming interface) in our simulator implementation, see Section 4.3) for more details.

4 Simulation of an ABM for a Smart Factory

The SwarmFabSim simulation framework is an implementation in NetLogo, compatible from version 6.1 up. Our framework supports single-lot-oriented and batch machines, in either, dispatching or scheduling mode (all machines in the fab have

[15]For other domains, different industry-specific units of production need to be used in the fab model.

to use the same mode). The number of machines, lots, machine types and lot types are only limited by the available memory and CPU power.
For prototyping and demonstrations, SwarmFabSim can be started from the user interface (see Section 4.1). The actual scenario to be simulated is defined in a set of configuration files (Section 4.2). To run mass simulations for simulation studies, users can either use the built-in BehaviorSpace configuration tool or the so-called headless mode. BehaviorSpace is suitable for smaller simulation studies where a number of runs are to be performed for not too many different scenarios. It lets users set up multiple parameter settings and the desired number of replications easily via a dialog window. Headless mode is suitable when a simulation study with a very large number of different scenarios has to be performed. In this mode, NetLogo is started from the command line and runs without the user interface. Simulation configurations can be either set up beforehand in BehaviorSpace or provided by text files. The latter case lends itself to automation where multiple simulations controlled by script-generated configuration files are executed by a command line script. For details, refer to NetLogo's BehaviourSpace guide[16]. The results are output to log files (Section 4.5) which can then be post-processed and statistically analyzed with any tool of choice, like R or Excel.
Unlike earlier versions, NetLogo shows adequate performance to simulate quite large systems. We tested our SwarmFabSim implementation with a configuration of 10 000 lots on a Windows 11 system with an AMD Ryzen Threadripper 3960X 24-Core Processor with 128 GB RAM running NetLogo 6.2.0. This configuration took 24 hours and 52 minutes to compute 30 simulation runs of the baseline algorithm (described in Section 4.4) in scheduling mode.
The NetLogo source code of SwarmFabSim plus several configuration files are available as open-source in our GitHub repository at: `https://swarmfabsim.github.io`.

4.1 The User Interface

SwarmFabSim simulations can be run in interactive mode controlled via the user interface shown in Figure 1. The user must first choose several settings (described below), before starting the simulation by first pressing "Setup" and then either running the simulation with "Go" or single-stepping through the simulation with the "Step" button. The following settings exist:

swarm_algorithm which algorithm the machines in the fab use. All machines use the same algorithm. The framework provides several algorithms as baseline and demo for algorithm implementers (see Section 4.4).

allocation_strategy whether to use dispatching or scheduling queuing mode. All machines use the same mode.

Config_File? whether a configuration file should be used (default: TRUE). The user interface also offers a manual setup where the constraints for the number of machine types, the total number of machines, production time, number of product types, recipe length, maximum number of lots, etc. can be set, and these are then created randomly. Since this is non-repeatable, the use of configuration files is strongly encouraged.

config_fname the name of the META configuration file (see Section 4.2 below).

[16] `https://ccl.northwestern.edu/netlogo/docs/behaviorspace.html`

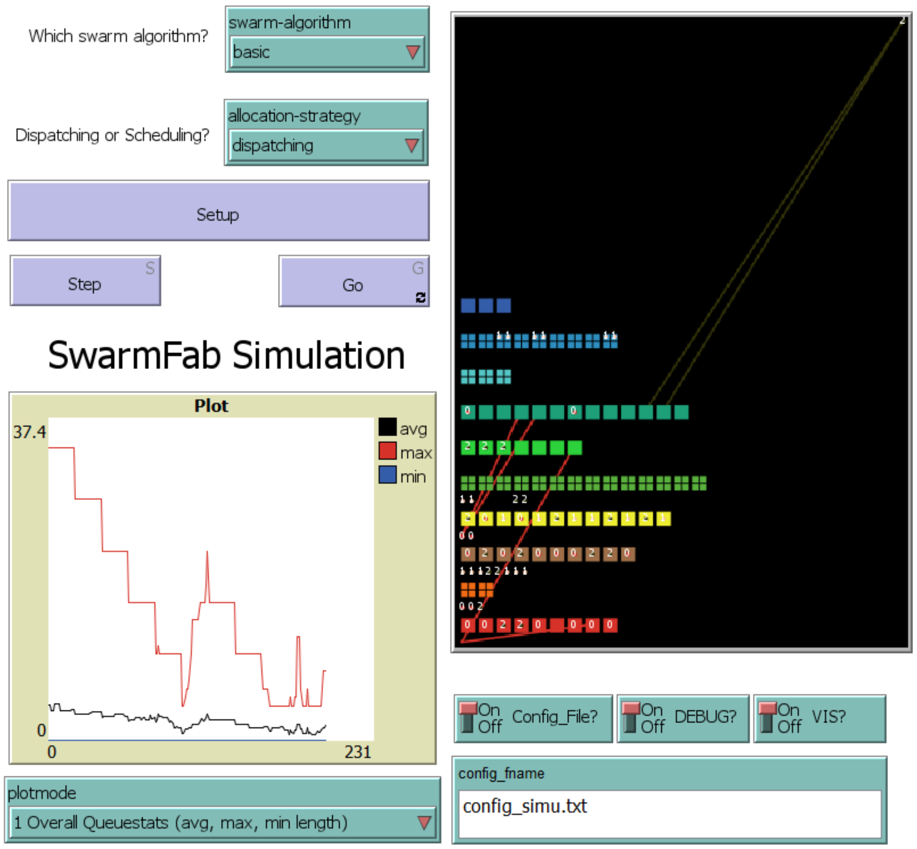

Fig. 1: SwarmFabSim User Interface. Figure reproduced from [39].

DEBUG? whether debug messages should be printed to the console (default: TRUE).

VIS? whether the machines and lot movements should be visualized (default: TRUE).

plotmode which mode to use for plotting. The plot window can show one of the following:

- **1 Overall Queuestats** the average, max, and min length of all queues in the system plotted over time.
- **2 Algorithm** a freely definable plot generated by the algorithm. The provided demo algorithm shows an example of how to plot machine occupancy (how many machines in the fab are occupied at any point in time).
- **3 Separate Queues** the current length of each queue in the system. This setting only makes sense in dispatching mode (where machines of the same type share a queue) with a moderate number of queues.

The simulated fab model on the right side of Figure 1 visualizes the lots as they move through the machines of the fab. Squares indicate single-lot-oriented machines, while the symbols consisting of four smaller squares indicate batch machines. Small red dots indicate the lots, the accompanying number indicates the lot's product type. Lines indicate each lot's last movement at every tick.

4.2 Configuration Files

The simulation scenario is defined via a set of plain text configuration files:

META this file bundles the associated configuration files and contains the file names of the MFILE, RFILE, and LFILE configuration files. The name of the META configuration file is set via the "config_fname" input field on the UI or via BehaviorSpace settings.

MFILE contains the machine definitions. For each machine type m, it defines the following parameters: a) the process id of the process P^m the machine can perform, b) the number of machines of this type, c) the processing time, d) the batch size (where a batch size of one defines a single-lot-oriented machine), and e) a maximum waiting time WT that a batch machine will wait for a batch to fill up before it starts processing (for single-lot-oriented machines, this parameter is ignored).

RFILE contains all recipes R^t used for production of each lot type t. The recipes are simple lists of the necessary process steps P^m in the required order.

LFILE defines how many lots should be produced for each respective recipe R^t (for each lot type t).

4.3 Architecture and Implementation Details

Figure 2 shows the general architecture of our simulation framework. The code entry point and main simulation loop are contained in the file SwarmFab-Simulation.nlogo. Besides that, this file also contains general declarations, the UI description, the Info tab (for documentation) and the BehaviorSpace settings. Auxiliary framework code is contained in the following .nls files:

config-reader.nls the configuration file parser.

lots+products.nls initialization code for the lots to be produced.

machines.nls initialization code for the machines in the fab.

vizualizations.nls helper code for vizualizations.

We use a callback architecture and define the API for this in the file hooks.nls. Whenever an algorithm can possibly take an action, the main code calls a hook function contained in hooks.nls from which the respective algorithm function is then called. In this way we decouple the algorithm code from the main framework code so that an algorithm implementer never has to touch or even read the actual framework code.

To implement an algorithm, a file with the name algorithm-*name*.nls that contains the algorithm code has to be provided. The algorithm must conform to the API defined in hooks.nls. Then, the algorithm has to be added to the chooser "algorithm" on the UI and calls to the algorithm functions have to be added to the respective API functions in hooks.nls. The hook API contains the following functions:

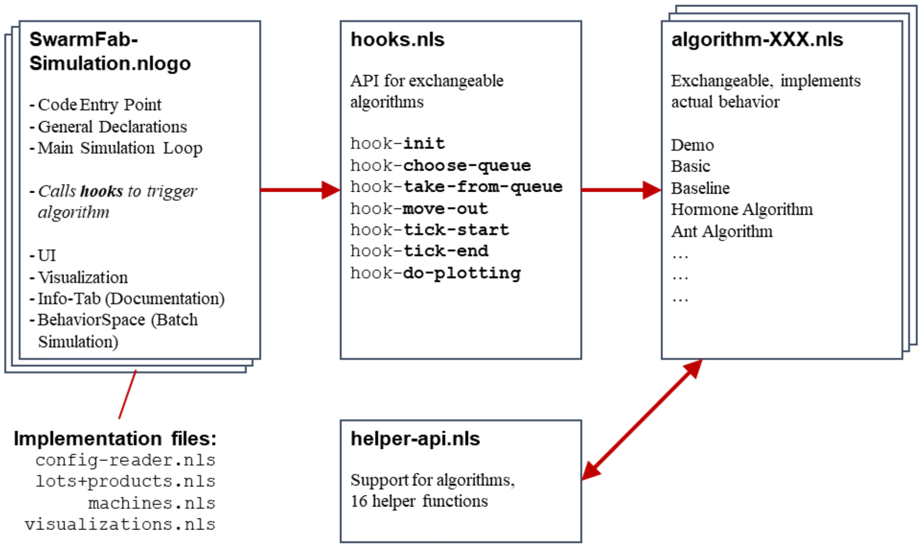

Fig. 2: Simulation Framework Architecture.

init called upon startup (mandatory).

choose-queue(lot) is only called when the simulation is in scheduling mode. In dispatching mode this function is not called, and the lot is automatically put into the common queue that serves all machines capable of the necessary next process step prescribed by the recipe. In scheduling mode, a choice has to be made by the algorithm to which queue of the set of possible next machines the lot is to be assigned. Therefore, this callback is mandatory for algorithms that support scheduling mode.

take-from-queue(machine) after a machine has finished processing a lot and has become free again, it needs to take a new lot from its queue. This callback provides the algorithm a chance to re-prioritize lots in the queue and potentially choose a lot further down the queue for processing instead of the first lot in the queue. This callback is mandatory for all algorithms even if they do not manipulate the queue.

move-out this function is called when processing has finished and a lot is about to leave a machine. At this point, an algorithm can collect data or update data at the machine. This callback is mandatory but can be empty.

tick-start, tick-end are called at the start and end of every tick respectively. This callback can be used to update data, such as evaporating pheromones. These callbacks are mandatory but can be empty.

do-plotting allows the algorithm to update the plot window on the UI. This callback is optional.

To facilitate easy algorithm development, we provide the file helper-api.nls which contains 16 commonly used convenience functions that provide, e.g., easy access to the next step in the recipe of the current lot, or to the contents of a machine's queue. We also provide example code for Demo and Basic algorithms and code to establish a performance Baseline described below (Section 4.4).

4.4 Example Code

The framework provides the following three algorithms as example code for algorithm implementers and as a baseline for performance comparisons.

Demo, Basic these algorithms are provided mainly as demonstration for algorithm implementers. They showcase different approaches to selecting a queue (shortest queue, random), taking lots from the queue, how to make algorithm data persistent between callbacks, and how to use the plot window from an algorithm.

Baseline is a memoryless/stateless algorithm provided as a reference for performance comparisons. The main idea of Baseline is to optimize lot assignment at batch machines with a simple heuristic that only takes the current workcenter of batch machines into account. As batches can only consist of lots of the same product type t, allocation of lots to batches is extremely important, especially as batch machines will wait up to the maximum waiting time WT for batches in their queue to fill up. Therefore, machines with semi-filled batches in their queue would incur a lot of idle time. Therefore, for the *choose-queue callback*, when a lot of a certain product type t arrives at the workcenter, Baseline looks for the queue where the least amount of lots of this type is missing to fill an already waiting, semi-filled batch. If there are no partially filled batches of this type in any of the potential queues, it chooses the queue with the least overall queue length (considering all batches of all other lot types) to start a new batch of the current type. For single-lot-oriented machines, it uses a simple shortest queue approach. When a batch has to be chosen from the queue during the *take-from-queue callback* if a full batch is available, that batch is chosen. In case several full batches exist, one is chosen at random. If there are only semi-filled batches, the machine waits up to the maximum waiting time WT (as specified in the MFILE configuration file, see Section 4.2). After the WT timeout, the largest batch is chosen (if there are several of the same size, one of them is chosen at random). If a batch fills up during WT, it is chosen immediately. For single-lot-oriented machines, a simple FIFO approach is used.

4.5 Log File Output

We have configured the included BehaviorSpace experiments (to run multiple simulations for a simulation study) to produce two types of result log files: a *-kpi.csv file and a *-table.csv file. Both files use a plain text, comma-separated value format which can be post-processed with any tool of choice for handling tabular data, such as R or Excel. Further values of interest can be logged by adding them to the settings in BehaviorSpace. The KPI (Key Performance Indicators) file contains the average flow factor, average lot tardiness, and average machine utilization calculated at the end of each run. It contains a line with result values for every run. The table file is a standard NetLogo log file. It traces the following values for every step of every run of the simulation:

run number the number of the run this line refers to.
swarm-algorithm which algorithm was used.
allocation-strategy whether dispatching or scheduling mode was used.
Config_File? whether a configuration file (default: TRUE) or UI settings were used to define the fab and product parameters.
config_fname the file name of the META configuration file
DEBUG? whether debug messages should be printed to the console (default: FALSE).
VIS? whether the fab should be visualized on the UI (default: FALSE).
step the current time tick value in the simulation.
avg_queue_length the mean queue length calculated over all machines at this time tick.
max_queue_length the maximum queue length calculated over all machines at this time tick.
min_queue_length the minimum queue length calculated over all machines at this time tick.

5 Case Study: Hormone and Ant Algorithms for Fab Optimization

In the following, we describe a case study using two algorithms - a hormone algorithm (Section 5.1) and an ant algorithm (Section 5.2) on our simulated fab. Section 5.3 describes the details of the three different fab setups used and Section 5.4 shows the performance results of these algorithms in comparison with the baseline algorithm.

5.1 The Hormone Algorithm

Artificial hormone algorithms are inspired by the biological endocrine system in our bodies that adjust the metabolism of tissue cells [36, 38]. The hormone algorithm is designed to work in dispatching mode where all machines of the same type share a queue. It uses artificial hormones to express how urgently machines need lots and as a mechanism to attract them to the relevant machines. The algorithm is controlled by the following seven rules and the five parameters given in Table 1.

Hormone model: every process P^x is associated with a type of hormone h^x. Hormones evaporate over time; at each simulation tick hormones are degraded according to Equation 1.

$$hormone\_amount = hormone\_amount \cdot (1 - \alpha) \qquad (1)$$

Hormones can be at any machine, and different hormone types can be at the same machine.

Machines produce hormone to attract lots. Each machine M^x produces hormone h^x of the type associated with the process P^x it can perform. Machines have the goal of minimizing their idle time and try to attract the lots they need to their queue. The amount of hormone produced by a machine is calculated by Equation 2.

$$hormone\_output = \frac{1}{lots\_in\_queue + \beta} \tag{2}$$

Machines are linked by the recipes of the lots in the system. The recipes impose a network of links between machines. Every transition from process step $P^o \rightarrow P^p$ in a recipe implies a link with strength 1 between machines $M^o \rightarrow M^p$.

Hormone diffuses upstream along the recipes in the system (iow. in the opposite direction than the recipe is processed) according to Equations 3 and 4:

$$upstream\_hormone = hormone\_amount \cdot \gamma \tag{3}$$

$$hormone\_amount = hormone\_amount - upstream\_hormone \tag{4}$$

For each upstream machine a proportional part of the upstream hormone is added according to Equation 5:

$$added\_hormone = upstream\_hormone \frac{link\_strength}{\sum link\_strengths} \tag{5}$$

Incoming lots diffuse hormone. Apart from natural diffusion, incoming lots also cause hormone to diffuse; this is calculated according to Equations 6 and 7:

$$upstream\_hormone = hormone\_amount \cdot \delta \tag{6}$$

$$hormone\_amount = hormone\_amount - upstream\_hormone \tag{7}$$

The machine where the lot came from receives the upstream hormone; this way a flow of lots can self-stabilize (Equation 8).

$$added\_hormone = upstream\_hormone \tag{8}$$

Lots are prioritized by their timing. We assume a planned cycle time (PCT) for each lot based on its raw processing time (RPT). A lot's base priority is then calculated according to Equation 9.

$$base\_priority = remaining\_RPT / remaining\_PCT \tag{9}$$

Lots are attracted by hormone. Lots are reordered in the queue according to their priority. The priority of a lot is calculated from its base priority and the attraction of the hormones present at the current machine (Equation 10):

$$priority = base\_priority \cdot attraction \tag{10}$$

where attraction is calculated according to Equation 11:

$$attraction = \sum_{i=0} h_i \cdot \varepsilon^i, \tag{11}$$

where h_i is the hormone of the process that is i steps ahead in the lot's recipe. Therefore, h_0 is the hormone of the current process. ε is a factor indicating the strength of a hormone's influence. Lots are then processed based on their priority. On batch machines, when one lot of a certain product type becomes eligible for processing, the batch is filled with as many lots of the same product type as will fit in the batch size and are available in the machine's queue.

Table 1: Parameter settings for the artificial hormone algorithm [12].

Parameter	Value
Evaporation α	.3
Pull factor smoothing β	1.0
Upstream diffusion γ	.5
Downstream diffusion δ	.2
Attraction ε	.8

A more detailed description of the algorithm and its parameters can be found in Elmenreich et al. [12].

5.2 The Ant Algorithm

Ant algorithms are inspired by the foraging behavior of certain species of ants, e.g., the Black Garden ant (*Lasius niger*) that lay pheromone trails on the ground to mark paths and recruit other ants to the exploitation of a food source [3].
Even though single ants are not very intelligent, overall, an intelligent solution with a near-optimal path emerges when the whole swarm works together. This behavior has been adapted to create technical solutions to the Travelling Salesman problem and network routing problems [5,9].
Ant algorithms can be calculated centrally (e.g., when a logistics company calculates the best routes for its fleet of trucks for the next day) or distributed (e.g., in packet network routing, where the algorithm runs independently at each node in the network). In our fab simulation, we use a distributed version of ant algorithm where we map lots to ants and calculate local decisions whenever a lot needs to choose a machine at the next workcenter. The algorithm is designed to work in scheduling mode and chooses the queue of a machine to use at each workcenter. The order of lots in the queue is not changed. The ant algorithm uses a FIFO regime to process the queues of single-lot-oriented machines. For batch machines, queue management is slightly more sophisticated: if a full batch exists, it is immediately processed (if several full batches exist, one of these is chosen at random). If no full batch exists, the machine waits for the timer *WT* to expire, after which it takes the fullest available batch in the queue.

As lots / ants traverse each workcenter W^m of machines, they need to choose one of the available machines $\{M_1^m, M_2^m, \dots\}$. To make this decision, they facilitate the pheromones stored by previous ants / lots.
To determine the amount of pheromone to deposit, lots measure the total time spent at a machine (Equation 12):

$$\Delta t = t_q + t_p \tag{12}$$

where t_q is the queueing time and t_p is the production time a lot takes at a machine. Based on this, the pheromone level τ at the respective machine is updated when the lot leaves the machine (during the move-out callback). The value is calculated according to Equation 13 as follows:

$$\tau := \tau + r \cdot (1 - \Delta t/\overline{\text{wlist}}) \tag{13}$$

where *wlist* is a sliding window list of the last three Δt values.
Unlike ant algorithms used in packet routing networks (see e.g., [5,6]), where the pheromones at all routers along the path are updated only *after* an ant packet has reached its intended final destination node by sending a so-called backward ant packet, in our case, we update the pheromone levels immediately when a lot / ant leaves the current machine at the current workcenter. This is due to the fact that the complete production of a lot typically takes so much time that updating the pheromone levels only after production is finished would not give a meaningful result as the information about machine status would already be outdated.
Subsequent lots then use the pheromones stored at machines to choose the best machine out of the set of machines capable of performing the next required process step in their recipe as follows:

- with a probability of $\phi = 5\%$, a random machine is chosen (this is the so-called exploration rate, which is necessary to discover better machines when the load situation changes).
- with $(1 - \phi) = 95\%$ probability, the machine with the highest score value is chosen (or, if several machines with the exact same highest value exist, a random machine amongst these). The score value is determined by a weighted sum of the current pheromone value and the current queue length according to Equation 14:

$$\tau + \alpha \cdot (1/\text{qlen}) \tag{14}$$

Pheromones evaporate over time at every simulation tick according to Equation 15:

$$\tau = \tau \cdot \rho \tag{15}$$

where ρ is the evaporation factor.
In the current implementation, we use the following values for the parameter settings as given in Table 2.

5.3 Fab Scenario Setup

We used three scenarios which are modeled after a typical semiconductor fab to evaluate our algorithms: SFAB, MFAB, and LFAB, with SFAB being the smallest and LFAB being the largest setup in terms of machines. The LFAB scenario also contains a higher number of product types and lots per type than the SFAB and MFAB scenarios. The parameters for the three scenarios are shown in Table 3.

Table 2: Parameter settings for the ant algorithm.

Parameter	Value
Exploration factor ϕ	5%
Weight for local heuristic α	.3
Evaporation factor ρ	.9

Table 3: Parameters used to create the three evaluation scenarios. Table from [39].

Parameter	SFAB	MFAB	LFAB
Number of machine types	25	50	100
Number of machines per type	$U(2,5)$	$U(2,10)$	$U(2,10)$
Number of products	50	50	100
Recipe length	$U(90,110)$	$U(90,110)$	$U(90,110)$
Number of lots per type	$U(1,10)$	$U(1,10)$	$U(2,10)$

Table 4 shows the parameters used to create the machine types for all simulations, where $N(\mu,\sigma^2)$ denotes the Normal Distribution and $U(a,b)$ the uniform distribution. Negative values from the normal distribution were capped for parameters that cannot be negative, like process time.

Table 4: Machine parameters used in the simulation. Table from [39].

Machine Parameter	Value
Raw process time	$N(\mu,\sigma^2)$ with $\mu = 1.16$, $\sigma^2 = 0.32$
Probability batch machine	50%
Batch size batch machines	$U(2,8)$
Waiting time batch machines	$U(1,2)$

5.4 Performance Results

Each scenario setting (SFAB, MFAB, LFAB) was run 30 times for each algorithm, and the KPI values (flow factor, tardiness, and uptime utilization) were averaged for comparison to the performance of the baseline algorithm. Flow factor describes the relation between actual and minimum production time: $t_{\text{prod}}/t_{\text{min}}$.
Tardiness describes how much additional time (due to lots waiting in a queue) has been accumulated until production of the lot: $t_{\text{prod}} - t_{\text{min}}$.
Uptime utilization represents how much time a machine has been in operation on average.
Tables 5, 6, and 7 compare the performance of the reference algorithm Baseline with the artificial hormone and ant algorithms for the SFAB, MFAB, and LFAB scenarios respectively. For the hormone algorithm, all three scenarios show promising improvements for the key performance indicators average flow factor,

Table 5: Evaluation of Artificial Hormone and Ant Algorithms in Small Scenario (SFAB). KPIs on average, changes in %, positive values denote improvement over Baseline.

	Baseline	Hormone	Change	Ant	Change
Flow factor	6.51	5.84	10.12%	6.67	-2.31%
Tardiness	6971.1	6077.2	12.56%	7157.6	-2.68%
Uptime Utilization	35.90	34.49	4.40%	36.114	-0.59%

Table 6: Evaluation of Artificial Hormone and Ant Algorithms in Medium Scenario (MFAB). KPIs on average, changes in %, positive values denote improvement over Baseline.

	Baseline	Hormone	Change	Ant	Change
Flow factor	3.21	2.87	9.64%	3.55	-10.56%
Tardiness	2481.8	2081.4	14.52%	2862.3	-15.33%
Uptime utilization	23.87	23.53	1.60%	21.62	9.43%

Table 7: Evaluation of Artificial Hormone and Ant Algorithms in Large Scenario (LFAB). KPIs on average, changes in %, positive values denote improvement over Baseline.

	Baseline	Hormone	Change	Ant	Change
Flow factor	3.47	3.03	12.29%	3.68	-5.98%
Tardiness	3126.8	2566.4	17.05%	3391.0	-8.45%
Uptime utilization	22.71	21.91	3.28%	24.48	-7.81%

average tardiness, and average uptime utilization, while the ant algorithm performs slightly worse than the baseline algorithm. Unlike the hormone algorithm, where information in the form of hormone levels travels upstream several workcenters, in the presented version of the ant algorithm, the information horizon is limited to only the respective current workcenter.

To measure simulation time, we used a Windows 11 system with an AMD Ryzen Threadripper 3960X 24-Core Processor with 128 GB RAM running NetLogo 6.2.0 processing 30 runs of each algorithm for the scenarios SFAB and LFAB to measure simulation performance and got the following results:

Baseline algorithm simulation takes approx. 1.75 mins for the SFAB scenario and 2.75 mins for the LFAB scenario.

Hormone algorithm simulation takes 0.35 hours for the SFAB scenario and 424 hours for the LFAB scenario.

Ant algorithm simulation takes approx. 1.75 mins for the SFAB scenario and 2.75 mins for the LFAB scenario (indistinguishable from Baseline).

As can be seen, the better performing hormone algorithm takes significantly more simulation time due to the higher computational complexity of dispersing the artificial hormones to previous machines according to the lots' recipes. We assume that a performance improvement could be achieved by limiting the distance hormones will be propagated to a smaller neighborhood of machines.

6 Conclusion

This chapter has depicted the use of NetLogo to model and simulate a fab producing according to the job-shop manufacturing principle. We introduced SwarmFabSim, a simulation framework implemented in NetLogo that can perform agent-based simulation of a fab modeled as self-organizing system. The fab model was adopted from the semiconductor manufacturer Infineon Technologies. Their great product diversity, the historical growth of the fab together with a broad set of technological and logistical boundary conditions lead to a complex optimization problem where swarm intelligence can be of big advantage. As swarms consist of agents that follow local rules and can be simulated as agent-based models, these methods can optimize the fab model from the bottom-up. Their optimization potential and effectiveness was evaluated in the framework by comparing an artificial hormone algorithm and an ant algorithm to a baseline algorithm that works according to simple heuristics, like FIFO queueing and filling the least empty batch first. The evaluation was based on three KPIs (flow factor, tardiness, and uptime utilization) using three different types of fab scenarios in terms of number of machines. For all three scenarios, the simulations show promising results for the hormone algorithm over the baseline algorithm. Unlike the hormon algorithm, the ant algorithm implementation performs slighty worse than the baseline algorithm. We assume that this is a limit on the performance of the algorithm, therefore, future versions will track time and update pheromone values across a (small) number of machines. Additionally to performance, we evaluated the simulation time for the scenarios of the algorithms. From this output we can see that the hormone algorithm takes significantly longer than the baseline and the ant algorithm. For a future version, we will reduce the radius to neighboring machines where the hormones are propagated to. The implementation of SwarmFabSim together with various configuration files and exemplary code is published as open-source in our GitHub repository at `https://swarmfabsim.github.io`. Adhering to the open-source concept, readers are welcome to collaborate, offering their ideas, feedback, and contributions to further enhance this free software project.

Acknowledgements This work was performed in the course of the project ML&Swarms supported by KWF-React EU under contract number KWF-20214|34789|50819, and SwarmIn supported by FFG under contract number 894072.

References

1. Alves, F., Varela, M.L.R., Rocha, A.M.A., Pereira, A.I., Barbosa, J., Leitão, P.: Hybrid system for simultaneous job shop scheduling and layout optimization based on multi-agents and genetic algorithm. In: International Conference on Hybrid Intelligent Systems. pp. 387–397. Springer (2018)
2. Bagheri, A., Zandieh, M., Mahdavi, I., Yazdani, M.: An artificial immune algorithm for the flexible job-shop scheduling problem. Future Generation Computer Systems **26**(4), 533–541 (2010)
3. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence – From Natural to Artificial Systems. Oxford University Press (1999)

4. Brabazon, A., O'Neill, M., McGarraghy, S.: Natural computing algorithms, vol. 554. Springer (2015)
5. Caro, G.D., Dorigo, M.: Antnet: Distributed stigmergy control for communications networks. Artificial Intelligence Research **9**, 317–365 (1998)
6. Di Caro, G., Ducatelle, F., Gambardella, L.M.: Anthocnet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks. European Transactions on Telecommunications **16**(5), 443–455 (2005)
7. Dias, L.M., Vieira, A.A., Pereira, G.A., Oliveira, J.A.: Discrete simulation software ranking—a top list of the worldwide most popular and used tools. In: 2016 Winter Simulation Conference. pp. 1060–1071. IEEE (2016)
8. Dorigo, M., Stützle, T.: Ant Colony Optimization. A Bradford Book, The MIT Press (2004)
9. Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation **16**(5), 53–66 (1997)
10. Elmenreich, W., D'Souza, R., Bettstetter, C., de Meer, H.: A survey of models and design methods for self-organizing networked systems. In: Proceedings of the Fourth International Workshop on Self-Organizing Systems. vol. LNCS 5918, pp. 37–49. Springer Verlag (2009)
11. Elmenreich, W., de Meer, H.: Self-organizing networked systems for technical applications: A discussion on open issues. In: K.A. Hummel, J.S. (ed.) Proceedings of the Third International Workshop on Self-Organizing Systems. pp. 1–9. Springer Verlag (2008)
12. Elmenreich, W., Schnabl, A., Schranz, M.: An artificial hormone-based algorithm for production scheduling from the bottom-up. In: Proceedings of the 13th International Conference on Agents and Artificial Intelligence. SciTePress (2021)
13. Floreano, D., Mattiussi, C.: Bio-inspired artificial intelligence: theories, methods, and technologies. MIT press (2008)
14. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research **1**(2), 117–129 (1976)
15. Geng, H. (ed.): Semiconductor Manufacturing Handbook. McGraw-Hill Education (2018)
16. Georgiadis, G.P.: Optimal Production Planning and Scheduling of Mixed Batch and Continuous Industrial Processes. Ph.D. thesis, Aristotle University of Thessaloniki (2021)
17. Ghasemi, M.: Lot streaming in hybrid flow shop scheduling. Tech. rep., Concordia University Spectrum Research Repository (2008)
18. Graham, R.L.: Bounds for certain multiprocessing anomalies. The Bell System Technical Journal **45**(9), 1563–1581 (1966). `https://doi.org/10.1002/j.1538-7305.1966.tb01709.x`
19. Gromicho, J.A., van Hoorn, J.J., da Gama, F.S., Timmer, G.T.: Solving the job-shop scheduling problem optimally by dynamic programming. Computers & Operations Research **39**(12), 2968–2977 (2012). `https://doi.org/https://doi.org/10.1016/j.cor.2012.02.024`
20. Gunaratne, C., Garibay, I.: NL4Py: Agent-based modeling in Python with parallelizable NetLogo workspaces. SoftwareX **16**, 100801 (2021)
21. Gwiazda, A., Banaś, W., Sękala, A., Topolska, S., Hryniewicz, P.: Modelling of production process using multiple ant colony approach. International Journal of Modern Manufacturing Technologies **XII**(1), 201–213 (2020)

22. Habib Zahmani, M., Atmani, B.: Multiple dispatching rules allocation in real time using data mining, genetic algorithms, and simulation. Journal of Scheduling **24**(2), 175–196 (2021)
23. Heylighen, F.: The science of self-organization and adaptivity. The Encyclopedia of Life Support Systems **5**(3), 253–280 (2001)
24. Jain, A., Meeran, S.: Deterministic job-shop scheduling: Past, present and future. European Journal of Operational Research **113**(2), 390–434 (1999). `https://doi.org/https://doi.org/10.1016/S0377-2217(98)00113-1`
25. Lawler, E.L., Lenstra, J.K., Kan, A.H.R., Shmoys, D.B.: Sequencing and scheduling: Algorithms and complexity. Handbooks in Operations Research and Management Science **4**, 445–522 (1993)
26. Lee, G.C., Kim, Y.D., Kim, J.G., Choi, S.H.: A dispatching rule-based approach to production scheduling in a printed circuit board manufacturing system. Journal of the Operational Research Society **54**, 1038–1049 (10 2003). `https://doi.org/10.1057/palgrave.jors.2601601`, `https://www.tandfonline.com/doi/full/10.1057/palgrave.jors.2601601`
27. Petrovic, S., Fayad, C., Petrovic, D., Burke, E., Kendall, G.: Fuzzy job shop scheduling with lot-sizing. Ann Oper Res **159**, 275–292 (2008). `https://doi.org/10.1007/s10479-007-0287-9`
28. Prehofer, C., Bettstetter, C.: Self-organization in communication networks: Principles and design paradigms. IEEE Communications Magazine pp. 78–85 (Jul 2005)
29. Pulikottil, T., Estrada-Jimenez, L.A., Rehman, H.U., Barata, J., Nikghadam-Hojjati, S., Zarzycki, L.: Multi-agent based manufacturing: current trends and challenges. In: 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). pp. 1–7. IEEE (2021)
30. Qin, W., Zhuang, Z., Liu, Y., Tang, O.: A two-stage ant colony algorithm for hybrid flow shop scheduling with lot sizing and calendar constraints in printed circuit board assembly. Computers & Industrial Engineering **138**, 106115 (2019). `https://doi.org/https://doi.org/10.1016/j.cie.2019.106115`
31. Railsback, S., Ayllón, D., Berger, U., Grimm, V., Lytinen, S., Sheppard, C., Thiele, J.C.: Improving execution speed of models implemented in netlogo. Journal of Artificial Societies and Social Simulation (2017)
32. Railsback, S.F., Grimm, V.: Agent-based and individual-based modeling: a practical introduction. Princeton university press, "2nd" edn. (2019)
33. Schranz, M., Umlauft, M., Elmenreich, W.: Bottom-up job shop scheduling with swarm intelligence in large production plants. In: Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications. pp. 327–334 (2021)
34. Schranz, M., Umlauft, M., Sende, M., Elmenreich, W.: Swarm robotic behaviors and current applications. Frontiers in Robotics and AI **7**, 36 (2020)
35. Shukla, O.J.: Agent Based Production Scheduling in Job Shop Manufacturing System. Ph.D. thesis, MNIT Jaipur (2018)
36. Sobe, A., Elmenreich, W., Szkaliczki, T., Böszörmenyi, L.: SEAHORSE: Generalizing an artificial hormone system algorithm to a middleware for search and delivery of information units. Computer Networks **80**, 124–142 (2015)
37. Thiele, J.C.: R marries NetLogo: introduction to the RNetLogo package. Journal of Statistical Software **58**, 1–41 (2014)

38. Turing, A.M.: The chemical basis of morphogenesis. Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences **237**(641), 37–72 (1952)
39. Umlauft, M., Schranz, M., Elmenreich, W.: SwarmFabSim: A simulation framework for bottom-up optimization in flexible job-shop scheduling using netlogo. In: Proceedings of the 12th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - SIMULTECH. p. 271–279. SciTePress (Jul 2022). https://doi.org/doi:10.5220/0011274700003274
40. Wilensky, U.: Netlogo (1999), http://ccl.northwestern.edu/netlogo/
41. Wilensky, U., Rand, W.: An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo. MIT Press (2015)
42. Yamada, T., Nakano, R.: Job shop scheduling. IEE control Engineering series pp. 134–134 (1997)
43. Zhang, G., Shao, X., Li, P., Gao, L.: An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. Computers & Industrial Engineering **56**(4), 1309–1318 (2009)
44. Zhang, T., Xie, S., Rose, O.: Agent-based simulation of job shop production. Simul. Notes Eur. **29**(3), 141–148 (2019)
45. Zhu, X., Wilhelm, W.E.: Scheduling and lot sizing with sequence-dependent setup: A literature review. IIE Transactions **38**, 987–1007 (11 2006). https://doi.org/10.1080/07408170600559706