SwarmFabSim: A Simulation Framework for Bottom-up Optimization in Flexible Job-Shop Scheduling using NetLogo

M. Umlauft[1][a], M. Schranz[1][b] and W. Elmenreich[2][c]
[1]*Lakeside Labs GmbH, Klagenfurt, Austria*
[2]*Institute of Networked and Embedded Systems, University of Klagenfurt, Klagenfurt, Austria*

Keywords: Swarm Intelligence, Flexible Job-Shop Scheduling, Agent-based Modeling.

Abstract: This paper models and simulates a semiconductor production plant organized by the job-shop principle as a self-organizing system using swarm intelligence algorithms in an agent-based simulation tool. We model a set of agents, including machines, workcenters, lots and processes. To simulate our model, we use NetLogo, one of the most widely used agent-based simulation platforms. The framework for the simulation was built as a structured system of code modules using a callback architecture that allows to exchange the used swarm algorithm easily. The user can configure their own fab model and simulations via the user interface and configuration files. The resulting log files include several key performance indicators: makespan, average flow factor, and lot tardiness. We offer the framework including sample swarm algorithms running on NetLogo version 6.1 and later as open source on GitHub.

1 INTRODUCTION

The factory-wide scheduling of a production plant organized by the flexible job-shop principle is an NP-hard challenging problem (Garey et al., 1976). A typical example for such a highly dynamic process organized by the job-shop principle is the production of integrated circuits (ICs) in the semiconductor industry (Geng, 2018). For our use case, we consider the so-called front end of line processing, where the structures of the ICs are created on silicon wafers. Our implementation is inspired by the real-world requirements of the semiconductor manufacturer Infineon Technologies[1]. In a fab, they need to schedule between 400 and 1200 different stations in their production plant. Typically, they produce more than 1500 different products in around 300 different process steps, including lithography, doping, oxidation, etching, and measuring (Geng, 2018).

As traditional optimization methods like linear optimization reach their limits in these settings due to excessive computation time (Garey et al., 1976), we propose this industrial setting as a novel field of application for swarm intelligence. Swarm intelligence algorithms use local rules and interactions and can be simulated with agent-based modeling. Swarms in nature typically solve complex problems using comparably simple rules for their swarm agents. One example is the foraging strategies of ants, which need to solve the problem of coordinating exploration, i.e., searching for new, unknown food sources and exploitation, i.e., establishing transport routes to carry the food into the nest (Dorigo and Stützle, 2004). Previous work has shown that such swarm algorithms can be successfully adapted to technical applications where a problem has to be solved in a complex (technical) environment; for an overview, see e.g., (Schranz et al., 2020). Such swarm-based solutions are of great interest because of their simple implementation and features of scalability, robustness, and adaptivity (Prehofer and Bettstetter, 2005). However, while a final swarm algorithm has a small computational footprint, defining a swarm algorithm for a given problem is a non-trivial task (Elmenreich and de Meer, 2008), which typically involves extensive evaluations using a simulation of the target system (Elmenreich et al., 2009). In many cases, the target system contains a highly dynamic process that cannot be optimized using analytical methods.

Therefore, in Section 2 we apply an agent-based modeling and simulation approach for this industrial

[a] https://orcid.org/0000-0002-0118-2817
[b] https://orcid.org/0000-0002-0714-6569
[c] https://orcid.org/0000-0001-6401-2658
[1] https://www.infineon.com

Umlauft, M., Schranz, M. and Elmenreich, W.
SwarmFabSim: A Simulation Framework for Bottom-up Optimization in Flexible Job-Shop Scheduling using NetLogo.
DOI: 10.5220/0011274700003274
In *Proceedings of the 12th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2022)*, pages 271-279
ISBN: 978-989-758-578-4; ISSN: 2184-2841

setting using the agent-based simulation tool NetLogo (Section 2.1). In Section 2.2 we introduce a modeling concept using different types of agents, including workcenters, machines, and lots. These agents are then implemented in the main contribution of this paper, the SwarmFabSim framework using NetLogo (Section 3), a system consisting of several structured code modules that interact using a callback architecture. The user interface, Section 3.1, and configuration files, Section 3.2, allow the user to interact and configure the SwarmFabSim framework according to the requirements of their own fab model. Note that despite taking inspiration from semiconductor manufacturing, our approach is applicable to other industries using flexible job-shop scheduling by simply changing the configuration files describing the concrete setting. We offer implementation insights on the architecture in Section 3.3, describe the base algorithms included with the framework in Section 3.4, and explain the log files with the implemented key performance indicators used for evaluation in Section 3.5. After describing the functionality, we present an example of a specific swarm algorithm implemented on the basis of the SwarmFabSim framework in Section 3.6, namely a hormone algorithm and the results from applying it to several simulation scenarios. Finally, Section 4 shows the related work on agent-based modeling environments and other job-shop simulation environments, and Section 5 concludes the paper.

2 AGENT-BASED MODELING FOR FLEXIBLE JOB-SHOP SCHEDULING

Scheduling is one of the most studied combinatorial problems typically addressed with linear optimization. Nevertheless, these methods can only cope with a subset of the plant and do not exploit the full optimization potential (Lawler et al., 1993). So far, no optimal solution for job-shop scheduling has been developed using linear optimization that can be computed in polynomial time (Zhang et al., 2009).

Therefore, we apply a novel approach, namely swarm intelligence algorithms, which we simulate using agent-based modeling (ABM). We model the production plant as a self-organizing system of homogeneous or heterogeneous agents inspired by fish, ants, or birds' natural swarm behavior. This approach allows us to optimize the production plant from the bottom-up instead of calculating a global optimization solution. The benefits are low calculatory overhead and high adaptability to local changing environmental conditions (Heylighen, 2001), such as starvation or machine downs. This is because the agents follow their own rules and make decisions based on local information instead of trying to calculate an overall solution. Applying ABM transforms the problem from finding an overall solution to defining a distributed algorithm that finds the solution from the bottom up.

ABM simulation is better suited to implementing swarm algorithms than system-dynamics (stock and flow) simulation or continuous simulation using differential equations. In agent-based simulation, swarm members can be modeled as agents who follow local rules and typically interact/communicate with other, nearby agents, the local environment, or information.

(Wilensky and Rand, 2015) give the following guidelines when to use ABM:

- Medium numbers: several dozen up to about 100000 agents. Our modeling problem consists of up to several thousand agents (products and machines).
- Heterogeneity: in ABM, agents can be as heterogeneous as necessary. ABM, therefore, lends itself well to modeling different product and machine types.
- Complex but local interactions, as used in swarm algorithms, are typical for ABM.
- ABM allows for rich environments, which can have agent-like rules. This can be used to, e.g., model complex machine queue manipulations in our case.
- Time: ABM is a model of process, which is a perfect fit to our problem domain.
- Adaptation: almost no other simulation method can model adaptation well. In ABM, agents' actions are typically contingent on past history; i.e., agents can learn. This fits the swarm model very well.

2.1 NetLogo for ABM Simulation

NetLogo (Wilensky, 1999) is one of the most widely used agent-based simulation platforms worldwide, is free, well documented, actively maintained with a mature code base, and offers many extensions. While NetLogo is well known for its usage in education (esp. about agent-based modeling and complex systems), it has also been shown to be a mature platform able to perform simulations of several thousand agents in feasible computing time (Railsback and Grimm, 2019; Railsback et al., 2017). For our own performance results, see Section 3 below. The NetLogo homepage lists 3000+ research papers that have used NetLogo as the simulation platform in the last 10 years.

NetLogo offers an interactive user interface and easy visualization for rapid prototyping and an easily configurable batch mode (called BehaviorSpace) to perform mass simulations with multiple parameter settings and the desired number of replications where it logs results to files. These resulting log files can then be post-processed with any tool of choice (R, Excel, etc.) to be statistically evaluated. In addition, NetLogo can be directly interfaced with other programming languages such as Python (Gunaratne and Garibay, 2021) or R (Thiele, 2014), supporting co-simulation approaches.

Simulation in NetLogo is time-based on a discrete scale using ticks. Different types of agents can be implemented using so-called "breeds" and can interact with each other either directly, based on proximity on a plane of patches, based on their connection via a network topology, or indirectly by leaving stigmergic information in the environment.

2.2 Modeling Job-Shop Scheduling

From the considered production plant described in Section 1, we identified machines, workcenters, processes, lots and recipes as possible agents (find more details on challenges in agent identification in (Schranz et al., 2021)).

The semiconductor production plant can be represented with a directed graph $G = (V, E)$, where the nodes V consist of all machines M_i^m, where m is the machine type, and the edges E are defined between two machines M_i^m and M_j^p if there exists a lot l_n^t of type t with a recipe R^t that contains the processes P^m and P^p that can be executed at machine M_i^m and machine M_j^p, respectively, in direct succession. Additionally to this consideration, the edges E define the neighborhood of each machine M_i. The **machines** M_i^m can be single lot oriented (processing one lot after the other), or batch-oriented (processing several lots at once, such as a furnace). They know which processes they can perform and what their current utilization is. Furthermore, they can make local decisions, e.g., re-ordering their queues.

A set of machines that can run the same or similar processes P form a **workcenter** $W \subset M$. The modeled production plant contains several sets or workcenters of machines $W^m = \{M_1^m, M_2^m, \dots\}$.

In the semiconductor industry, the standard unit of production is a **lot** that consists of 25 wafers, each equipped with a transponder with a unique identifier. In our model, we do therefore *not* consider single wafers, but only lots[2]. Each product type t is defined by a recipe R^t which prescribes the processing steps in the order necessary to manufacture this product. As there typically are multiple machines that can perform the same process, a lot l_n^t can choose which of the suitable machines M_i^m to use for each necessary process step P^m.

Historically, semiconductor manufacturing used dispatching to assign lots to machines. In dispatching, all machines M_x^m that can perform a certain process step P^m share one queue Q^m and lots are assigned to machines by changing their position in the queue via priority levels so that the next free machine will take the lot with the highest urgency.

As fabs are being modernized, producers switch to scheduling, where each machine M_i^m has its own queue Q_i^m and use schedulers to optimize the assignments of lots to queues for a whole workcenter of machines. Lots still get assigned priority levels that can change their position in their assigned queue.

Therefore, in our model, as each lot is produced step by step according to its recipe, there are two decisions to be made at each step:

1. The position of the lot in the queue. In our implementation, the queue can be re-ordered every time a machine becomes free and wants to take a lot from the queue.
2. If the model is run in scheduling mode, whenever a lot needs to move to the next machine, it has to choose 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 decision points are reflected in the callback API in our simulator implementation (see Section 3.3).

3 SIMULATION OF AN ABM FOR A SMART FACTORY

We implemented the SwarmFabSim simulation framework in NetLogo (compatible from version 6.1 up). Our framework supports dispatching and scheduling modes, single lot oriented and batch machines, and an arbitrary number of machine and lot types (only limited by available memory and CPU power). In a performance test of a configuration of 10 000 lots being processed in SwarmFabSim, a Windows 11 system with an AMD Ryzen Threadripper 3960X 24-Core Processor with 128 GB RAM running NetLogo 6.2.0 takes 24 hours and 52 minutes

[2]If applied to other industries, the industry-specific unit of production would be used in the model.

to compute 30 simulation runs being scheduled with the baseline algorithm (described in Section 3.4).

For prototyping and demonstrations, SwarmFab-Sim can be started from the user interface (see Section 3.1). The actual scenario to be simulated is defined in a set of config files (Section 3.2). To run mass simulations for simulation studies, we use the built-in BehaviorSpace configuration tool where you can set multiple parameter settings and the desired number of replications. The resulting log files (Section 3.5) can then be post-processed with any tool of choice (R, Excel, etc.) to be statistically evaluated.

The NetLogo source code of SwarmFabSim plus several configuration files are available as open-source in a GitHub repository: https://swarmfabsim.github.io.

3.1 The User Interface

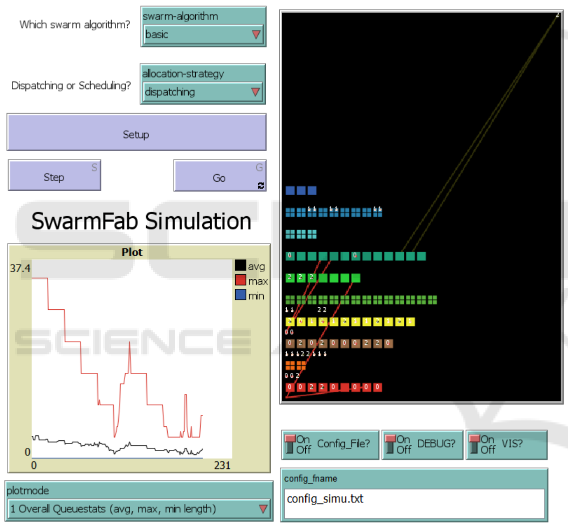
Which swarm algorithm? swarm-algorithm: basic
Dispatching or Scheduling? allocation-strategy: dispatching
plotmode: 1 Overall Queuestats (avg, max, min length)
config_fname: config_simu.txt

Figure 1: SwarmFab Simulation User Interface.

The simulation can be run in interactive mode from the user interface shown in Figure 1. The user must first choose several settings, such as which config file, algorithm, and scheduling mode to use and can then start the simulation by first pressing “Setup” and then either run the simulation with “Go” or single-step through the simulation with the “Step” button. The “world” on the right then shows the machines and the lots as they move through the fab.

3.2 Config Files

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

META: the meta config file contains the file names of the MFILE, RFILE, and LFILE config files.

MFILE: contains the machine definitions. For each machine type m, it defines the process id of the process P^m the machine can perform, the number of machines, the processing time, the batch size (a batch size of one defines a single lot oriented machine), and, in case of a batch machine, a maximum waiting time WT that this machine will wait for a batch to fill up before it starts processing.

RFILE: contains all recipes R^t used for production of the different lot types 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 lot of type t according to the respective recipe R^t.

3.3 Architecture and Implementation Details

As shown in Figure 2, the main simulation loop is contained in the file SwarmFab-Simulation.nlogo. Besides the main code entry point and the general declarations, this file also describes the UI, the Info tab (for documentation) and the BehaviorSpace settings. Auxiliary framework code is contained in the following files: config-reader.nls, lots+products.nls, machines.nls, vizualizations.nls.

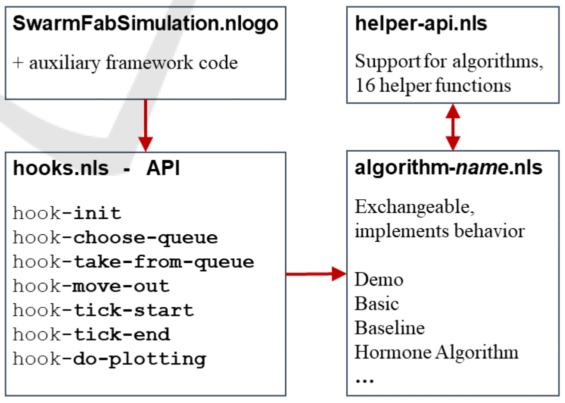

Figure 2: Simulation Framework Architecture.

The callback architecture is realized via the API (application programming interface) defined in 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. To install an algorithm, a file with the name algorithm-*name*.nls that implements the algorithm according to the hook API has to be provided, the algorithm has to be added to the UI chooser "algorithm" and the algorithm functions have to be added to the respective hook functions in hooks.nls for callback. Algorithms can use helper functions that are provided in the file helper-api.nls. Due to this decoupling, the algorithm implementer never has to touch or even read the actual framework code. The hook API contains the following functions:

init called upon startup (mandatory).

choose-queue(lot): is called for the algorithm to choose which machine a given lot shall go to next. When the simulation is 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.

take-from-queue(machine): after a machine has finished a process and has become free again, it takes a new lot from its queue. This callback provides the algorithm a chance to re-prioritize lots in the queue (mandatory).

move-out: this function is called just as a lot is about to leave a machine at the end of processing. At this point, an algorithm can collect data or update data at the machine (mandatory).

tick-start, tick-end: are called at the start and end of every tick respectively. Can be used to update data, such as evaporating pheromones (mandatory, but can be empty).

do-plotting: allows the algorithm to update the plot window on the UI (optional).

3.4 Base Algorithms

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

Demo, Basic these algorithms are provided mainly as demonstration for algorithm implementers. They demonstrate different approaches to selecting a 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. For the *choose-queue callback* it differentiates between single lot oriented and batch machines. For the former, it chooses the shortest queue, while for the latter it looks for the queue where the least amount of lots of the current type is missing to fill an already waiting, semi-filled batch. If there are no partially filled batches 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 the *take-from-queue callback*, it differentiates between single lot oriented machines where it uses a FIFO approach and batch machines, where, if a full batch is available, that batch is chosen. If several full batches exist, one is chosen at random. If no full batch exists, the machine waits up to a maximum waiting time WT (as specified in the MFILE config file, see Section 3.2). After the WT timeout, the largest batch is chosen (in case of contention, one is chosen at random). If a batch fills up during WT, it is chosen immediately.

3.5 Log File Output

When the SwarmFab simulation is run from BehaviorSpace (the standard method to run multiple simulations for a study in NetLogo), it produces 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, Excel, etc. Further values of interest can be logged by adding them to the settings in BehaviorSpace.

3.6 Case Study: The Hormone Algorithm for Fab Optimization

Artificial hormone algorithms are inspired by the biological endocrine system adjusting the metabolism of tissue cells in our body (Turing, 1952; Sobe et al., 2015). In our case study, we use artificial hormones to express the urgency of a lot and the need for attracting incoming lots by the machines. Artificial hormone is produced by the machines in the production process and can diffuse through the production system along with the lot processing steps. Lots, being swarm members, are attracted by the hormone level of a machine. For each processing step, a corresponding hormone type exists. Hormones can be at any machine, and different hormone types can be at the same machine. Every simulation tick hormone degrades exponentially at a given rate α. Machines produce hormone to attract lots. The hormone production is guided by the queue length of the incoming lots. The more lots are already waiting at a machine, the less hormone is produced. In contrast, a machine that is about to run out of lots to be processed will produce more hormone. Machines are linked via recipes. If a recipe has a process of two machines in subsequent steps, the downstream machine (that will process the lot later) is linked to the upstream machine (that processes the machine before).

Table 1: Parameter settings for the artificial hormone algorithm (Elmenreich et al., 2021).

Parameter	Value
Evaporation α	.3
Pull factor smoothing β	1
Upstream diffusion γ	.5
Downstream diffusion δ	.2
Attraction ε	.8

The algorithm is based on five parameters that allow fine-tuning the respective mechanisms; the used parameters are depicted in Table 1. A more detailed description of the algorithm and its parameters can be found in (Elmenreich et al., 2021).

We evaluated the artificial hormone algorithm in three scenarios modeled after processes in a typical semiconductor fab. Each scenario differs significantly by the number of machines, with scenario SFAB being the smallest and scenario LFAB being the largest setup. The LFAB scenario also has a higher number of product types and lots per type than the SFAB and MEDIUM scenarios. The parameters for the three scenarios are depicted in Table 2.

Table 3 depicts the distribution parameters used to create machines for a simulation, where $N(\mu,\sigma^2)$ depicts a Normal Distribution and $U(a,b)$ a uniform distribution. Negative values from the normal distribution have been capped for parameters that cannot be negative, like process time.

For the evaluation, each scenario setting has been run 30 times, and the average value is compared to the performance of the baseline and the basic algorithms. Processing 30 runs on a Windows 11 system with an AMD Ryzen Threadripper 3960X 24-Core Processor with 128 GB RAM running NetLogo 6.2.0 takes 0.35 hours for the SFAB scenario and 424 hours for the LFAB scenario. The algorithm is evaluated according to three performance metrics: Average flow factor, average tardiness, and average uptime utilization. Tardiness describes how much additional time (due to lots waiting in a queue) has been accumulated until production of the lot. Flow factor describes the relation between the actual production time and the minimum production time. Uptime utilization represents how much time a machine has been in operation.

Tables 4, 5, and 6 depict a comparison between the performance of the reference algorithms basic and baseline and the artificial hormone algorithm. All three scenarios depict promising improvements for average flow factor (FF), average tardiness (TARD), and average uptime utilization (UTIL).

4 RELATED WORK

In most cases, factory simulation is done using discrete event process flow simulation with simulators specific to the respective application domain. For semiconductor manufacturing, examples include AutoMod / AutoSched AP[3] or D-Simlab/D-Simcon[4]. For our research, the level of detail supported in these types of simulators is usually too high. Also, as they do not support agent-based modeling, they do not lend themselves to modeling swarm algorithms.

A simulation language that is not specific to a certain application domain is GPSS (General Purpose Simulation System), a discrete time simulation language originally developed by IBM. It is used primarily as a process flow-oriented simulation language which is particularly well-suited for problems such as modeling factory operations. While its popularity has declined over the years, several implementations still exist, such as GPSS World by Minuteman[5] or JGPSS[6] by the Polytechnic University of Catalonia. A ranked list of the most popular discrete simulation software platforms in commercial use is given in (Dias et al., 2016).

An example of using NetLogo to simulate a production process with a multiple ant colony approach is shown in (Gwiazda et al., 2020). The authors show how NetLogo can be used to model the production process and how a swarm intelligence algorithm – such as an ant algorithm – can be implemented in an agent-based modelling simulator and used to solve the job shop problem.

Zahmani and Atmani use a Netlogo simulation to create job shop problems with a given value for jobs and machines. The NetLogo simuation is coupled with a genetic algorithm for discovering well-working allocations of dispatching rules. (Habib Zahmani and Atmani, 2021) Pulikottil et al. review the various frameworks and different technologies that support multi-agent system use in manufacturing, the various applications of multi-agent systems in this domain, and the current challenges in the development of multi-agent systems in smart manufacturing. (Pulikottil et al., 2021) Alves et al. address the job shop scheduling problem with a hybrid approach including optimization and agents negotiating dynamically. The agent-based model implemented in NetLogo is connected with Matlab to exchange optimized scheduling solutions. (Alves et al., 2018) Bagheri et al. describe the promising application of an artificial im-

[3]https://automod.de/autosched-ap/
[4]http://www.d-simlab.com/
[5]http://www.minutemansoftware.com/simulation.htm
[6]https://jgpss.liam.upc.edu/en/about

Table 2: Parameters used to create the three evaluation scenarios.

Parameter	SFAB	MEDIUM	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 3: Machine parameters used in the simulation.

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	$U(1,2)$

Table 4: Evaluation of Artificial Hormone Algorithm in large scenario (LFAB).

	Basic	Baseline	Hormone	Improvement over Baseline
FF	3.60	3.47	3.03	12.29%
TARD	3287.3	3126.8	2566.4	17.05%
UTIL	24.33	22.71	21.91	3.28%

Table 5: Evaluation of Artificial Hormone Algorithm in medium scenario (MEDIUM).

	Basic	Baseline	Hormone	Improvement over Baseline
FF	3.46	3.21	2.87	9.64%
TARD	2757.8	2481.8	2081.4	14.52%
UTIL	21.42	23.87	23.53	1.60%

Table 6: Evaluation of Artificial Hormone Algorithm in small scenario (SFAB).

	Basic	Baseline	Hormone	Improvement over Baseline
FF	6.64	6.51	5.84	10.12%
TARD	7118.3	6971.1	6077.2	12.56%
UTIL	34.95	35.90	34.49	4.40%

mune algorithm for the flexible job-shop scheduling problem. The flexible job-shop scheduling problem has high relevance to the problem described in this paper but does not have batch machines, which are typical for semiconductor fabs. The artificial immune algorithm is implemented in C++ and evaluated with a set of tests based on reference data sets. (Bagheri et al., 2010) Zhang et al. show how agent-based modeling can be applied to job-shop production simulation conceptually. (Zhang et al., 2019) Shukla reviews multi-agent systems for production scheduling problems in manufacturing systems. They show how multi-agent systems can provide advantages over traditional approaches and present a conceptual framework for implementing production scheduling systems using multi-agent systems. (Shukla, 2018)

There are many ABM simulation platforms that can be used to implement a job-shop production simulation. For a comprehensive overview see e.g., CoMSES Net / OpenABM[7]. In the following we list the most well-known ones besides NetLogo:

Mason is a joint effort between George Mason University's Evolutionary Computation Laboratory and the GMU Center for Social Complexity. It is a discrete-event multi-agent simulator core written in Java as a basis for custom-written simulations. It has been in development since 2003. At the time of writing, it is actively supported[8].

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 based on Java, C++, and Python[9].

Mesa is an open source ABM framework written in Python. Its goal is to be the Python 3-based alternative to NetLogo, Repast, or MASON allowing for easy integration with Python's data analysis tools. The tool has been developed since 2015[10].

Anylogic is a commercial platform that supports System Dynamics, Process-centric discrete event modeling, and Agent Based Modeling. It is used in several industries, including manufacturing, supply chain, transportation and logistics, etc[11].

MARS (Multi-Agent Research and Simulation) runs on .NET Core. It is developed at the Department of Computer Science at Hamburg University of Applied Sciences[12].

[7]https://www.comses.net/resources/modeling-frameworks/

[8]https://cs.gmu.edu/~eclab/projects/mason/

[9]https://repast.github.io/

[10]https://mesa.readthedocs.io/en/latest/

[11]https://www.anylogic.com/

[12]https://mars-group.org/

5 CONCLUSION

This paper has depicted the use of NetLogo to model and simulate a factory producing according to the job-shop manufacturing principle. We contributed SwarmFabSim, a modular simulation framework written in NetLogo that can apply various algorithms to optimize a make-to-order manufacturing system and supports multiple configurable scenarios. The evaluation framework was used to assess the effectiveness of an artificial hormone algorithm compared to a naïve basic implementation and a reference baseline algorithm. The evaluation was based on three key performance indicators: Flow factor, delay, and utilization. The simulations show promising results of the artificial hormone algorithm in three reference scenarios with significant improvements over the reference algorithms. The implementation of the simulation environment is published as open source in a Git repository[13]. Readers are welcome to contribute with their ideas and developments.

ACKNOWLEDGEMENT

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

REFERENCES

Alves, F., Varela, M. L. R., Rocha, A. M. A., Pereira, A. I., Barbosa, J., and Leitão, P. (2018). Hybrid system for simultaneous job shop scheduling and layout optimization based on multi-agents and genetic algorithm. In *International Conference on Hybrid Intelligent Systems*, pages 387–397. Springer.

Bagheri, A., Zandieh, M., Mahdavi, I., and Yazdani, M. (2010). An artificial immune algorithm for the flexible job-shop scheduling problem. *Future Generation Computer Systems*, 26(4):533–541.

Dias, L. M., Vieira, A. A., Pereira, G. A., and Oliveira, J. A. (2016). Discrete simulation software ranking—a top list of the worldwide most popular and used tools. In *2016 Winter Simulation Conference*, pages 1060–1071. IEEE.

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. A Bradford Book, The MIT Press.

Elmenreich, W. and de Meer, H. (2008). Self-organizing networked systems for technical applications: A discussion on open issues. In K.A. Hummel, J. S., editor, *Proceedings of the Third International Workshop on Self-Organizing Systems*, pages 1–9. Springer Verlag.

[13]https://swarmfabsim.github.io

Elmenreich, W., D'Souza, R., Bettstetter, C., and de Meer, H. (2009). A survey of models and design methods for self-organizing networked systems. In *Proceedings of the Fourth International Workshop on Self-Organizing Systems*, volume LNCS 5918, pages 37–49. Springer Verlag.

Elmenreich, W., Schnabl, A., and Schranz, M. (2021). 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.

Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129.

Geng, H., editor (2018). *Semiconductor Manufacturing Handbook*. McGraw-Hill Education.

Gunaratne, C. and Garibay, I. (2021). NL4Py: Agent-based modeling in Python with parallelizable NetLogo workspaces. *SoftwareX*, 16:100801.

Gwiazda, A., Banaś, W., Sękala, A., Topolska, S., and Hryniewicz, P. (2020). Modelling of production process using multiple ant colony approach. *International Journal of Modern Manufacturing Technologies*, XII(1):201–213.

Habib Zahmani, M. and Atmani, B. (2021). Multiple dispatching rules allocation in real time using data mining, genetic algorithms, and simulation. *Journal of Scheduling*, 24(2):175–196.

Heylighen, F. (2001). The science of self-organization and adaptivity. *The Encyclopedia of Life Support Systems*, 5(3):253–280.

Lawler, E. L., Lenstra, J. K., Kan, A. H. R., and Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. *Handbooks in Operations Research and Management Science*, 4:445–522.

Prehofer, C. and Bettstetter, C. (2005). Self-organization in communication networks: Principles and design paradigms. *IEEE Communications Magazine*, pages 78–85.

Pulikottil, T., Estrada-Jimenez, L. A., Rehman, H. U., Barata, J., Nikghadam-Hojjati, S., and Zarzycki, L. (2021). Multi-agent based manufacturing: current trends and challenges. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–7. IEEE.

Railsback, S., Ayllón, D., Berger, U., Grimm, V., Lytinen, S., Sheppard, C., and Thiele, J. C. (2017). Improving execution speed of models implemented in netlogo. *Journal of Artificial Societies and Social Simulation*.

Railsback, S. F. and Grimm, V. (2019). *Agent-based and individual-based modeling: a practical introduction*. Princeton university press, "2nd" edition.

Schranz, M., Umlauft, M., and Elmenreich, W. (2021). 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*, pages 327–334.

Schranz, M., Umlauft, M., Sende, M., and Elmenreich, W. (2020). Swarm robotic behaviors and current applications. *Frontiers in Robotics and AI*, 7:36.

Shukla, O. J. (2018). *Agent Based Production Scheduling in Job Shop Manufacturing System....* PhD thesis, MNIT Jaipur.

Sobe, A., Elmenreich, W., Szkaliczki, T., and Böszörmenyi, L. (2015). SEAHORSE: Generalizing an artificial hormone system algorithm to a middleware for search and delivery of information units. *Computer Networks*, 80:124–142.

Thiele, J. C. (2014). R marries NetLogo: introduction to the RNetLogo package. *Journal of Statistical Software*, 58:1–41.

Turing, A. M. (1952). The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237(641):37–72.

Wilensky, U. (1999). Netlogo. http://ccl.northwestern.edu/netlogo/.

Wilensky, U. and Rand, W. (2015). *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo*. MIT Press.

Zhang, G., Shao, X., Li, P., and Gao, L. (2009). An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Computers & Industrial Engineering*, 56(4):1309–1318.

Zhang, T., Xie, S., and Rose, O. (2019). Agent-based simulation of job shop production. *Simul. Notes Eur.*, 29(3):141–148.