Using NetLogo to Simulate Large Production Plants: Simulation Performance: A Case Study

M. Umlauft[a] and M. Schranz[b]
Lakeside Labs GmbH, Klagenfurt, Austria

Keywords: Netlogo, Simulation Performance, Agent-Based Modeling, Flexible Job-Shop Scheduling.

Abstract: NetLogo is a well-known agent-based modeling and simulation platform. While it is very popular in education, it is still often perceived as having bad performance for large models, which is due to performance related issues in early implementations. We show that over time, a quite large number of scientific papers have been published using NetLogo and measure its performance on a common laptop a researcher or student might have as a personal machine. We use a NetLogo model with about 2500 lines of code and up to 10000 agents to perform our measurements and show that even with such an underpowered machine, with current versions of NetLogo it is quite possible to run simulations of larger models in reasonable simulation time.

1 INTRODUCTION

The agent-based simulation platform NetLogo (Wilensky, 1999) is one of the most widely used tools for the simulation of self-organized systems. It is open source with a good documentation and regular updates with on average two versions per year since 2002[1]. The mature code base is actively maintained, offers many extensions including a Python interface and comes with a vast collection of models in a library. NetLogo is well known for its usage in education, but also got popular as a research tool for agent-based modeling and complex systems especially because of its ease of use that made it a low-entry tool for researchers who "are not professional programmers" (Tisue and Wilensky, 2004) – this was one of the explicitly stated goals of the NetLogo developers. While highly accessible, older versions had performance problems for large models, which often required re-implementation in more high-performing computer languages (eg. C++). In the last decade, though, performance has been greatly improved; in 2017 and 2019 respectively, (Railsback et al., 2017; Railsback and Grimm, 2019) showed that newer versions of NetLogo have the ability to perform simulations of several thousand agents in large spaces within feasible computation time.

Execution speed, sophisticated and highly optimized algorithms and having most of the code compiled instead of interpreted in current versions, makes the platform now feasible to execute thousands of model runs (Railsback et al., 2017). The increased usage of NetLogo as the simulation platform of choice for agent-based simulation is reflected in the number of research papers which exceed 3,000 in the last 10 years[2].

The results of Railsback et al. (Railsback et al., 2017) motivated us to the test the performance of current versions of NetLogo using our implemented framework SwarmFabSim (Umlauft et al., 2022) as a test case. SwarmFabSim is designed for simulations of bottom-up swarm intelligence algorithms in large production plant environments. The original use-case of SwarmFabSim is the simulation of a Semiconductor production plant, based on the requirements of a leading semiconductor manufacturer, namely, Infineon Technologies[3]. In particular, we consider the so-called front end of line processing where the wafers are processed to create ICs. Typical process steps include lithography, doping, oxidation, etching, and measuring (Geng, 2018). There are between 400 and 1200 different stations that need to be scheduled in such a production plant (fab). They typically produce

[a] https://orcid.org/0000-0002-0118-2817
[b] https://orcid.org/0000-0002-0714-6569

[1] The versions of NetLogo: https://ccl.northwestern.edu/netlogo/oldversions.shtml

[2] https://ccl.northwestern.edu/netlogo/references.shtml

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

Umlauft, M. and Schranz, M.
Using NetLogo to Simulate Large Production Plants: Simulation Performance: A Case Study.
DOI: 10.5220/0012768800003758

In *Proceedings of the 14th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2024)*, pages 297-304
ISBN: 978-989-758-708-5; ISSN: 2184-2841

more than 1500 different products in around 300 different process steps.

Our paper does *not* focus on general techniques of speeding up simulation time, such as using parallel runs, instead, we showcase the performance of current Netlogo implementations by using the example of a large framework simulating Semiconductor fabs. We measure the simulation runtime without visualizations for several large models comprising of about 2500 lines of code, using a conceptual, but realistically sized fab scenario. The simulations are run on a typical laptop a researcher or student might have access to. The paper is organized as follows:

- Related work is shown in Section 2.
- We then describe why we deem Netlogo a suitable platform for our ABM simulation and describe our framework, the architecture, the used algorithm and log file output for our performance test.
- We then show our test setup and the performance analysis of simulation time on a common laptop in Section 4.
- The paper is concluded in Section 5.

2 RELATED WORK

In research and in industry, there exist typical tools for discrete event process flow simulation of a fab specific to the application type. The semiconductor industry uses, e.g., AutoMod / AutoSched AP[4] or D-Simlab/D-Simcon[5]. The level of detail to simulate flexible job-shop scheduling is very high, and they do not support ABM. Beside semiconductor-specific simulation tools, there exist tools developed not for a certain application domain. A ranked list of discrete simulation software platforms for commercial use can be found in Dias et al. (Dias et al., 2016).

Employing Agent-Based Modeling (ABM) for swarm algorithms is more suitable than utilizing system dynamics simulation (stock and flow) or continuous simulation using differential equations (Umlauft et al., 2022). In ABM, a swarm is composed of individual members that can be represented as agents. These agents follow local rules, interact with other agents and their environment. There is a set of specific guidelines given by Wilensky and Rand, when to use ABM. The reader is referred to their publication in (Wilensky and Rand, 2015) as this topic is out of the scope of this paper. Numerous authors have already demonstrated the conceptual application of ABM to job-shop production simulation. For instance (Zhang et al., 2019) illustrate this concept in their work. Furthermore, specific implementations utilizing multi-agent systems in manufacturing systems have been explored, as exemplified in the study of (Shukla, 2018).

For ABM simulation many tools and platforms exist for the simulation of the job-shop scheduling problem. A comprehensive overview is given in e.g., CoMSES Net / OpenABM[6] and (Abar et al., 2017) for various application domains. We selected a list of most commonly used ABM simulation tools:

Anylogic is a commercially available platform that supports various modeling modes including system dynamics, process-centric discrete event modeling, and ABM. Besides manufacturing, other industries like supply chain, transportation and logistics have it commonly in use[7].

MARS is a C# framework and stands for Multi-Agent Research and Simulation. Developed as an academic project at the Department of Computer Science at Hamburg University of Applied Sciences in Germany at [8] MARS provides tools for multi-agent research and simulation.

Mason is a collaboration between George Mason University (USA) and Università degli Studi di Salerno (Italy). It serves as an ABM simulation core developed in Java, providing a framework for creating custom simulations. Mason includes support for both, 2D and 3D visualization tools. As of the time of writing this paper, it continuous to receive active support and development[9].

Mesa is a is an open-source modular framework for ABM simulation developed in Python. Its primary objective is to serve as a Python 3-based alternative to NetLogo, Repast, or Mason. The Python-based architecture enables seamless integration with the corresponding data analysis tools. Mesa has been under development since 2015, and as of the time of writing this paper, continuous to receive active support[10].

Repast is a family of three distinct open source ABM toolkits built on Java, C++, and Python. With over 15 years of development, these toolkits offer robust capabilities for ABM simulation[11].

[4]https://automod.de/autosched-ap/

[5]http://www.d-simlab.com/

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

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

[8]https://www.mars-group.org/

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

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

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

In this publication we decided to use NetLogo (Wilensky, 1999) for ABM simulation, an open-source tool based on the language Logo. As already mentioned in the introduction of this paper, the code base has been consistently maintained for more than 20 years, the platform is well documented and offers many extensions to other libraries and tools that were extensively tested.

In the literature we can find several examples where researchers used NetLogo for production processes. One example is shown in (Gwiazda et al., 2020) where the authors implement a multiple ant colony approach that solves the job shop problem. The authors in (Habib Zahmani and Atmani, 2021) offer a specialized framework, demonstrating the creation of a NetLogo simulation tailored for job shop problems with predefined job and machine values. In addressing the algorithmic aspect, they employ a genetic algorithm to uncover optimal allocations of dispatching rules. The connection of ABM in NetLogo with Matlab was performed in (Alves et al., 2018)to exchange scheduling solutions. Their hybrid algorithmic approach addresses the job shop scheduling problem by combining optimization techniques with dynamically negotiating agents.

Specifically concerning the performance of NetLogo, the literature lacks comprehensive tests and simulation runs to conduct evaluations in this regard. The research conducted by Railsback et al. (Railsback et al., 2017; Railsback and Grimm, 2019) demonstrated that newer versions of NetLogo possess the capability to performs simulations involving several thousand agents within large spaces, all within reasonable computation times. Additionally to that, (Railsback et al., 2017) offers strategies to optimize the coding strategy in NetLogo, e.g., how to enhance the efficiency of agent filtering statements by utilizing global agentsets, and how to use the BehaviorSpace for experiments to achieve an improved performance in simulation experiments. Beside the most cited work of (Railsback et al., 2017), in (Antelmi et al., 2022) NetLogo exceeds the expectations in all evaluated models (incl. Flocking and ForestFire), demonstrating strong efficiency and scalability. According to their results and discussion, NetLogo achieves moderate to low execution times and effectively manages intensive workloads, despite not being explicitly designed for simulating complex models. The study compares NetLogo with several ABM simulation tools including Repast and MASON.

3 SIMULATING WITH NetLogo

NetLogo (Wilensky, 1999) is a free simulation and modeling platform for agent-based simulation maintained by Northwestern University, Chicago. It is well documented, actively maintained with a mature code base, and offers many extensions. It supports mobile agents, an environment consisting of intelligent patches (basically stationary agents), links to connect agents, and communications between agents and/or patches. It offers an interactive user interface which makes it easy to rapidly prototype and visualize agent-based models.

NetLogo is a time-based simulator using a discrete scale of time ticks. Simulation is typically started with a procedure traditionally called “setup”, which is run once at the beginning to initialize the simulation. The main simulation loop is implemented in a procedure traditionally called “go” which is then called once per tick. From there, all other procedures are then called to perform the necessary computations to simulate the model. Therefore, simulation performance is dominated by the code in “go” and all procedures called by it. Since NetLogo is a time-based simulator, unlike in event-based simulation, ticks – and therefore the “go” procedure – are also executed if there is no actual change in simulation status, eg. if agents just wait and “do nothing” for a certain number of ticks.

To investigate a model, at least a subset of its parameter-space needs to be simulated, and since swarm intelligence algorithms are typically stochastic, the researcher has to perform multiple replications for each chosen input parameter combination. To facilitate this type of mass simulation, NetLogo offers a tool called “BehaviorSpace” which lets the researcher configure the parameters and the number of desired replications in an easy fashion. The results are written into log files which can later be statistically evaluated with other tools, like R or Excel. Depending on the setup, one or multiple lines can be written into a log file at every time tick or at the end of every simulation run. Logging method and writing speed greatly influences simulation performance; logging several lines per tick onto a traditional magnetic harddrive is much more time-costly than logging one line per simulation run onto an SSD (solid-state-drive).

Also, computational effort can easily explode when many different parameter combinations are to be evaluated; in general, if you want to simulate an experiment with x parameters of y different values each at r replications, you need to perform $n = r * x^y$ simulation runs.

NetLogo BehaviorSpace supports the use of multiple CPUs and CPU cores to parallelize simulation runs (one run per core), so large multiprocessor machines can be used to tackle mass simulations. In cases where the number of runs would still be too large, NetLogo also supports finding the interesting sections of the parameter space with a tool called "BehaviorSearch", which searches the parameter space with genetic algorithms (Wilensky and Rand, 2015).

3.1 SwarmFabSim: A Framework Created in NetLogo

SwarmFabSim is a simulation framework implemented in NetLogo (compatible with versions 6.1 and up) intended to simulate the fabrication of products in a large factory. In particular, SwarmFabSim is intended to simulate bottom-up swarm intelligence algorithms optimizing front-end-of-line (FEOL) production of semiconductors, but the implementation is abstract enough to be applied to other production facilities using the job-shop principle as well.

SwarmFabSim supports an arbitrary number of machines M_i^m of machine type m which can perform process P^m and lots l_n^t of product type t to be produced. The product type t is defined by a recipe R^t which contains all necessary process steps $P^m, P^n, P^o, \ldots$ to produce a lot of this type in order. Machines and lots are modelled as agents; the number of which are only limited by the available memory and CPU power of the simulation server. The framework supports running the machines either in dispatching or scheduling mode. In dispatching mode, all machines of type m in a workcenter $W^m = \{M_i^m, M_j^m, \ldots\}$ share a common queue Q^m while in scheduling mode each machine M_i^m has its own queue Q_i^m.

Currently, we support two classes of machines - single lot oriented machines, which process one lot after the other and batch machines, which (can) process several lots of the same product type t at once. Of each class of machine, there can be several different machine types m; iow. there can be several different types of single step and several different types of batch machines to model all the different types of machines in a fab – each with their own parameters, like processing time, or batch size. A further important constraint for batch machines is that batch machines will wait for a time WT for a batch in the queue to fill up. E.g., if a batch machine M_i^m has a batch size of 4, and there is already a lot l_0^t in its queue Q^m, it will wait for more lots $l_1^t ... l_3^t$ of product type t to arrive to fill the batch. If the machine is idle, it will either wait until enough lots have arrived in the queue for the batch to be filled completely or for the waiting timer WT to expire. At which point it will immediately start processing, even if the batch is not completely filled.

The fab and load configuration, such as the number of machines and their parameters and the number of lots and their recipes are specified in config files. SwarmFabSim comes with several swarm intelligence algorithms and the framework architecture (for details see Section 3.2) supports easy extension with additional algorithms. All machines in the simulation will use the same algorithm and scheduling mode during one simulation run.

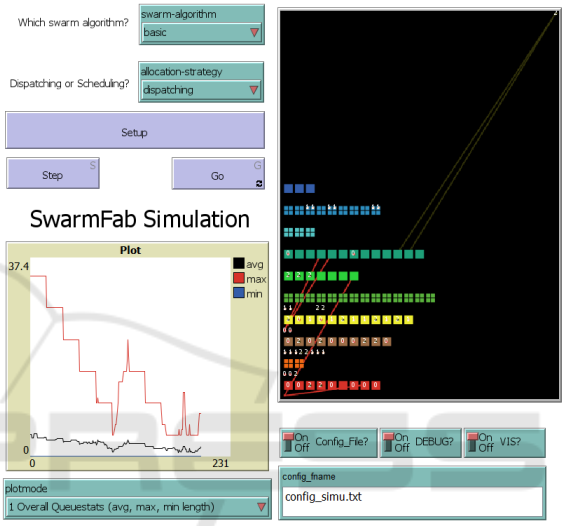

Figure 1: The SwarmFabSim User Interface.

Typically, SwarmFabSim will be used in "headless mode" (without live visualisation) via BehaviorSpace (see Section 3 above), but it can be started in interactive mode from the user interface shown in Figure 1 for demonstration purposes and prototyping / debugging.

The source code of SwarmFabSim is available as open-source from our GitHub repository[12].

3.2 Architecture

Figure 2 shows the architecture of the SwarmFabSim framework. The main software entry point and user interface are contained in the file `SwarmFabSimulation.nlogo`. This file contains the `setup` procedure that initializes the simulation and the `go` procedure that contains the main simulation loop and is run once per tick.

The `setup` procedure calls other procedures to read and parse the config files, set up the parame-

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

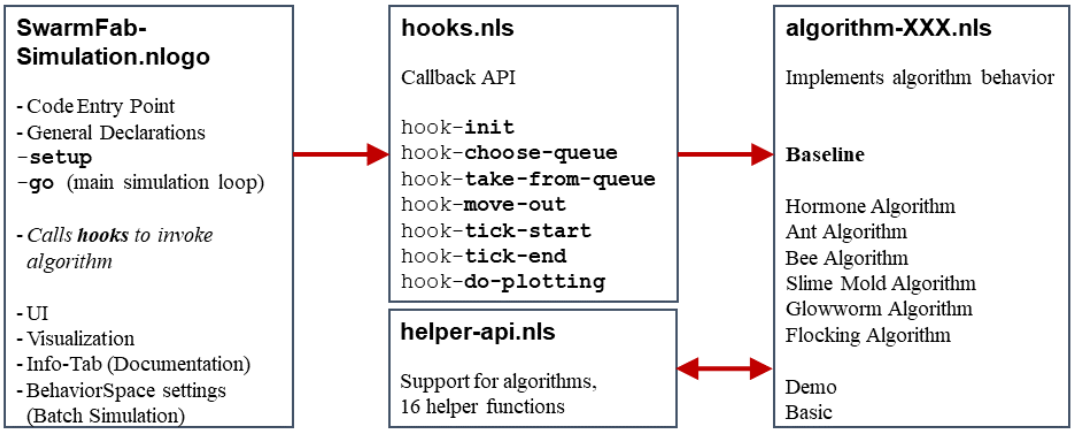

Figure 2: Simulation Framework Architecture.

ters of the simulation, and initializes the swarm intelligence algorithm via the `init` callback of the API (application programming interface) contained in `hooks.nls`. This action is only performed once at the beginning of a simulation run.

The `go` procedure calls the selected swarm intelligence algorithm via the other API functions, using the callback pattern. For each tick, the `go` procedure will

- first call the **tick-start** callback. With this procedure, the algorithm can do regular, time-based updates, like, e.g., propagate hormone values or decrease pheromone values.
- Then, any "free" lots – that is any lots not currently assigned to a queue or being processed in a machine – will be moved to an appropriate queue. In case of the dispatching strategy, where all machines of the same type share a queue, the lot is simply moved to the shared queue of the workcenter of machines of the type implied by the next step in the lot's recipe. In case of scheduling, where every machine has its own, separate queue, this decision is taken by the swarm intelligence algorithm. Therefore, the `go` procedure will call the **choose-queue** hook which will in turn call the `choose-queue` procedure of the swarm intelligence algorithm.
- Any idle machines are then called via the **take-from-queue** hook which will in turn call the algorithm to make the decision which lot(s) to take from the queue
- before all non-empty machines are then called to process their respective lot(s) for one tick. For any lot(s) that get finished during this tick, the **move-out** hook will be called. In this procedure, the algorithm can take actions like, e.g., updating a pheromone value. At this point, the lot becomes "free" again.
- Next, the **tick-end** callback will be invoked. Like with `tick-start`, the algorithm can do regular, time-based updates. Typically, algorithms use only either the `tick-start` or the `tick-end` callback.
- Global **queue length statistics** will be updated next via the `calculate-queue-length-stats` procedure and
- finally, **visualization** updates will be plotted if visualization is enabled (typically only in interactive mode).

3.3 The Baseline Algorithm

SwarmFabSim provides this algorithm as a baseline for performance comparisons. Baseline is a memoryless/stateless algorithm that uses only the `choose-queue` and the `take-from-queue` callbacks. In general, the Baseline algorithm works differently depending on the class of machine – single lot oriented or batch:

choose-queue on single lot oriented machines shortest queue.

choose-queue on batch machines the objective is to quickly fill any existing semi-filled batch in a queue. Therefore, it will assign a lot of type t to the queue with the least amount of lots of this type missing to fill a batch of type t. If the respective machine is currently idle and waiting for the WT timer to expire, it will start immediately when the batch is filled. If the machine is currently processing, this approach increases the chance of this batch being filled by the time the machine becomes idle. Even if the batch is never

completely filled and the machine has to wait for the WT timeout to expire, it will still run at a higher percentage of fill. If there are no semi-filled batches of the current lot type, Baseline chooses the queue with the least overall length (regardless of lot type) to start the new batch.

take-from-queue on single lot oriented machines FIFO.

take-from-queue on batch machines if there is a full batch of a type t (or fills up during the WT timeout), this batch is taken. If there are several full batches in the queue, one is chosen at random.

3.4 Log File Output

When SwarmFabSim is run from BehaviorSpace, it produces four different types of result log files, which are all in a plain text, comma-separated value format:

***-table.csv** this is the standard Netlogo log file. It logs a line for every tick; in our configuration it writes the following values: the run number, the chosen swarm algorithm, the chosen allocation strategy, whether a config file was used and its file name, whether debugging and visualization mode were turned on, the tick number, and the average, max, and min queue length in the system at this tick. As this file logs one line for each tick, its influence on simulation performance can be noticeable, especially if the medium on which the log files reside is slow.

***-lmvs.csv** this is a custom log file that tracks lot movements between machines in detail and logs a line for every tick and lot movement. It logs the run number, the current tick, the mode of movement (lot got put into a queue, moved into a machine, moved out from a a machine), the lot type, the unique lot number, the position of the recipe pointer (how far advanced along the recipe the lot is), the machine type, the unique machine id and a slot number in case of a batch machine. Due to the high level of detail in this log file it can become quite large. Since it logs a line for every lot movement that occurred at every tick, so potentially many lines for each tick, writing this log file has the biggest influence on simulation performance.

***-kpi.csv** this is a custom log file that is written once at the end of every simulation run. It logs the run number, the average flow factor, tardiness, and machine utilization, plus the total makespan (the simulation duration in ticks). Since this file logs only one line at the end of each simulation run, its influence on simulation performance is low.

***-lots.csv** this is a custom log file that is written once per run and lot at the end of every simulation run. It logs the run number, the lot type, the unique lot number, the total queuing time, raw processing time, and total processing time for the lot, plus the overall flow factor and tardiness for this lot. While this file is written only once at the end of the simulation run, it can potentially become quite big if the number of lots is very large. Since this file is only logged to once at the end of each simulation run, its influence on simulation performance is not very large despite the potentially large file size.

Since the *-table.csv and *-lmvs.csv log files have the most impact on simulation performance, simulations should be performed on machines with fast drives to store these files so as not to slow down simulation unnecessarily. A lot of performance can also be gained by not logging to files whose information you are not going to analyze; eg. if you do not need the detailed lot movement information in *-lmvs.csv, disabling this log file will speed up simulation.

4 EVALUATION

Our paper does *not* focus on general techniques of speeding up simulations, such as turning off visualization, using a large multi-processor server with a fast SSD disk, sampling the parameter space before starting detailed simulations, or making sure that no unnecessary calculations are repeatedly performed in the "go" procedure. Instead, we showcase the performance of current Netlogo implementations by using the example of our implemented SwarmFabSim framework, using typical models and one of our algorithms. We measure the runtime of one run without visualizations for several typical large models on a typical laptop a researcher would have access to.

4.1 Scenario Configuration

We designed our test scenario inspired by both, the aforementioned real-world data of between 400 and 1200 different stations in a fab which produces products in around 300 different process steps and on the SMT2020 testbed (Kopp et al., 2020). The SMT2020 datasets contain up to 105 workcenters (machine types) and recipes for up to 10 products that are between 242 and 583 steps in length.

Our test scenario uses a fab of 100 workcenters (machine types) of between 4 and 12 machines each and 10 product types that have recipes of 250 to 350 steps in length. We then load this fab with 100, 1000, 5000, and 10K lots (divided among the

10 product types). Monthly production is often a keenly guarded secret of a fab; the best overview of production across semiconductor fabs to our knowledge is given in (Wikipedia, 2024), where most fabs listed have monthly production capacities of less than 100000 wafers, which – using the industry standard of 25 wafers per lot – amounts to 4000 lots.

For each lot type, we generate an average of $1/10$ of the total lot number $\pm 10\%$ each; iow. for the scenario with 100 lots in total, each of the 10 lot types will have between 9 and 11 lots, for the scenario with 1000 lots in total, each lot type will have between 90 and 110 lots, etc.

We use the parameters shown in Table 1 to create the machines for the fab. $N(\mu,\sigma^2)$ for the raw process time denotes a Normal Distribution from which negative values have been capped as process time cannot be negative. $U(a,b)$ for the batch sizes denotes a uniform distribution.

Table 1: Parameters used to create the machines.

Machine Parameter	Value
Raw process time (RPT)	$N(\mu,\sigma^2)$ with μ = 1.16, σ^2 = 0.32
Probability batch machine	50%
Batch size batch machines	$U(2,8)$
Waiting time batch machines	$15\%(RPT)$

4.2 Results

We intentionally used a common laptop that might be available to any researcher for our measurements to show that, despite the common perception that NetLogo is supposedly too slow to do larger simulatin studies, current versions of NetLogo work well even on weaker hardware. This type of setup might be used by a single scientist or student to do quite large studies on their own, personal machine, without immediately requiring the use of an institute's simulation server (to which access might be limited).

We tested using NetLogo in headless mode the `reset-timer` and `timer` commands on an ASUS UX305LA Laptop from 2016, running Windows 10 with an Intel CORE i7 CPU with 4 cores, running at 2.4 GHz, 8 GB of RAM, and an SSD harddrive. We used only one of the available cores; the results are averaged over 3 runs each.

We compare the performance for the following major releases of Netlogo: 6.2.0 from December 2020, 6.3.0 from September 2022, and 6.4.0 from November 2023 as shown in Table 2.

As can be seen, there is a is an increase in simulation time of almost 20% from version 6.2.0 to version 6.3.0. Analysis of the release notes of these respective NetLogo versions reveals that in version 6.3.0 the bundled version of Java was upgraded from version 8 to version 17, which is the current long-term-support release of the Java runtime. As no other changes that would cause such a large loss in performance are listed in the release notes (eg. no large changes to the architecture or core language), the authors assume that the measured performance loss is due to the upgrade of the Java version. Experimenters who do not need the new features of versions 6.3.0 and 6.4.0 (see release notes[13] whose simulations are very large and time-intensive might want to consider staying with version 6.2.0 instead.

Table 2: Simulation Performance in seconds, over NetLogo versions and number of lots.

Netlogo Version / Number of lots	6.2.0	6.3.0	6.4.0
100	51	61	60
1000	122	144	142
5000	690	816	782

We also tested a very large scenario with 10000 lots (equivalent to 250000 wafers) on NetLogo version 6.4.0 which finished on average in 2101 seconds; iow. a good half of an hour. Tests with logging turned off (apart from recording the runtime at the end of each run) for the 1000 lot scenario on NetLogo version 6.2.0 finished on average in 95 seconds versus 122 seconds with full logging, giving a speedup of 22% on an SSD drive.

5 CONCLUSION

While NetLogo is popular in education and has gained popularity as a simulation tool in research, it is still often conceived as not well performing for large models. This is due to performance related issues in early implementations. We measured simulation performance of a quite large NetLogo model on a common laptop a researcher or student might have as a personal machine and show that even with such an underpowered machine, with current versions of NetLogo it is quite possible to run simulations of larger models in reasonable simulation time. Our measurements did show an approximately 20% increase in simulation time from version 6.2.0 to version 6.3.0 and beyond which is probably due to the upgrade of the underlying Java distribution from version 8 to version 17 mentioned in the NetLogo release notes. For

[13] http://ccl.northwestern.edu/netlogo/docs/versions.html

time-intensive simulations which do not require the newly introduced features of or depend on any bug-fixes in NetLogo version 6.3.0 and up, staying with version 6.2.0 might be worth considering.

ACKNOWLEDGEMENT

This work was performed in the course of project SwarmIn supported by FFG under contract number 894072.

REFERENCES

Abar, S., Theodoropoulos, G. K., Lemarinier, P., and O'Hare, G. M. (2017). Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, 24:13–33.

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.

Antelmi, A., Cordasco, G., D'Ambrosio, G., De Vinco, D., and Spagnuolo, C. (2022). Experimenting with agent-based model simulation tools. *Applied Sciences*, 13(1):13.

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.

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

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.

Kopp, D., Hassoun, M., Kalir, A., and Mönch, L. (2020). SMT2020 – A semiconductor manufacturing testbed. *IEEE Transactions on Semiconductor Manufacturing*, 33(4):522–531.

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 (JASSS)*.

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

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

Tisue, S. and Wilensky, U. (2004). Netlogo: Design and implementation of a multi-agent modeling environment. In *Proceedings of the Agent 2004 Conference on Social Dynamics: Interaction, Reflexivity and Emergence*, volume 2004, pages 7–9, Chicago, IL.

Umlauft, M., Schranz, M., and Elmenreich, W. (2022). Simulation of swarm intelligence for flexible job-shop scheduling with swarmfabsim: Case studies with artificial hormones and an ant algorithm. In *International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pages 133–155. Springer.

Wikipedia (2024). List of semiconductor fabrication plants. https://en.wikipedia.org/wiki/List_of_semiconductor_fabrication_plants. last visited: March 1st, 2024.

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