Bottom-Up Bio-Inspired Algorithms for Optimizing Industrial Plants

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

Keywords: Swarm Intelligence, Bio-Inspired Algorithm, Bee Algorithm, Bat Algorithm, Flexible Job-Shop Scheduling, Agent-Based Modeling.

Abstract: Scheduling in a production plant with a high product diversity is an NP-hard problem. In large plants, traditional optimization methods reach their limits in terms of computational time. In this paper, we use inspiration from two bio-inspired optimization algorithms, namely, the artificial bee colony (ABC) algorithm and the bat algorithm and apply them to the job shop scheduling problem. Unlike previous work using these algorithms for global optimization, we do not apply them to solutions in the solution space, though, but rather choose a bottom-up approach and apply them as literal swarm intelligence algorithms. We use the example of a semiconductor production plant and map the bees and bats to actual entities in the plant (lots, machines) using agent-based modeling using the NetLogo simulation platform. These agents then interact with each other and the environment using local rules from which the global behavior – the optimization of the industrial plant – emerges. We measure performance in comparison to a baseline algorithm using an engineered heuristics (FIFO, fill fullest batches first). Our results show that these types of algorithms, employed in a bottom-up manner, show promise of performance improvements using only low-effort local calculations.

1 INTRODUCTION

In today's production plants organized by the job shop principle we face an increased complexity in scheduling due to the dynamics of customized, flexible, on-demand production combined with a high product diversity. Throughout this paper, we consider the semiconductor manufacturer Infineon Technologies[1] that additionally deals with typical low-volume batches of integrated circuits in the logic and power sector. Exemplary, they produce 1500 products in around 300 process steps using up to 1200 stations (Schranz et al., 2021; Khatmi et al., 2019). These production plant conditions lead to an NP-hard problem where linear optimization methods reach their limits for a global plant optimization due to the excessive computation time (Lawler et al., 1993). Centrally precomputed swarm intelligence algorithms have already been used for the optimization of industrial production plants. They show good performances and thus, are used as an alternative or extension for linear optimization methods (for a comprehensive review, the reader is referred to Gao et al. (Gao et al., 2019)). Nevertheless, they face the same problems in terms of calculation time and complexity (Khatmi et al., 2019). As proposed in (Schranz et al., 2021), we use the novel approach to model the production plant as a self-organizing system of agents, using local rules, local knowledge and local interactions. This transfers the problem of computing an overall solution to engineering a distributed algorithm that produces a solution from the bottom-up. An optimization from the bottom-up is able to dynamically react on changing environmental conditions (e.g., tool downs, product priorities) and to produce near-optimal solutions for NP-hard problems.

In this paper, honeybees and microbats serve as inspiration to derive two distributed swarm intelligence algorithms. Honeybees live in a colony, search for pollen, and transport it back to their hive. To attract other bees for the same food source, they perform a waggle dance that shows the direction and distance to the food source. They also use pheromones to communicate a possible attack. Microbats use echolocation signals to search for nearby prey. They update the loudness, rate of pulse and the frequency of the signal based on the distance of the bats to the prey. Our contribution is related to exactly this natural bee and bat behaviour originally designed as the artificial bee colony algorithm (ABC) (Karaboga and Basturk, 2007) and the bat algorithm (Yang, 2010), but engi-

[a] https://orcid.org/0000-0002-0118-2817
[b] https://orcid.org/0000-0002-0714-6569
[1] Infineon Technologies, https://www.infineon.com/

Umlauft, M., Gojkovic, M., Harshina, K. and Schranz, M.
Bottom-Up Bio-Inspired Algorithms for Optimizing Industrial Plants.
DOI: 10.5220/0011693400003393
In *Proceedings of the 15th International Conference on Agents and Artificial Intelligence (ICAART 2023) - Volume 1*, pages 59-70
ISBN: 978-989-758-623-1; ISSN: 2184-433X

neered onto the problem of the semiconductor production plant. For the first time, the ABC and the bat algorithm are not used as a centrally computed optimization approach, but rather the rules are adapted and used as local rules to perform the optimization from the bottom-up. In this paper we introduce a model of the production plant and the problem to optimize in Section 2. In Section 3, we describe the general, global optimizer version of the ABC and bat algorithms and their main characteristics. Then, we use the novel approach of bottom-up production plant optimization by engineering the ABC and bat algorithms to the problem statement (Section 4), and evaluate the resulting algorithms in a NetLogo framework to show their performance compared to an engineered baseline algorithm using FIFO queues and fill-least-empty-batch first principles (Section 5). The paper is closed out with sections on related work (Section 6) and the conclusion (Section 7).

2 MODEL

We model the semiconductor production plant as follows: there are a number of so-called **lots** $L^t = \{l_1^t, l_2^t, \dots\}$ (comprised of 25 wafers each), which need to be produced. The lots follow a specific recipe R^t related to their product type t. This recipe prescribes which **process steps** $P = \{p_1^m, p_2^m, \dots\}$ to take in which order. The plant has a set of **machines** M^m, where m indicates the machine type that is related to the process step P^m. Each machine M_i^m has a queue Q_i^m. In conclusion, the recipes imply a directed graph $G = (V, E)$ of possible movements between the machines of the plant, where the nodes V consist of all machines M_i^m and the edges E are defined between two machines M_i^m and M_j^p if there exists a lot l_n^t with a recipe R^t containing processes P^m and P^p in direct succession. A set of machines M^m that can run the same process P^m form a **workcenter** $W^m \subset M$. The modeled production plant contains several workcenters of machines $W^m = \{M_1^m, M_2^m, \dots\}$. As there typically are multiple machines per workcenter W^m, a lot l_n^t must choose which (or be assigned to one) of the suitable machines $M_i^m \in W^m$ to use for each necessary process step $P^m \in R^t$. The machines M_i^m can be one of two kinds: either **single-step** (processing one lot after the other), or **batch**-oriented (processing a batch of several lots at once, such as a furnace). In a production cycle, single-step and batch machines follow each other. This makes optimization especially hard because in the optimal case batch machines would like to accumulate a full batch of lots of a product type before starting their process. If these lots are all in the same queue of one of the preceeding single-step machines, the batch machine would have to either wait a long time to fill the batch or to run with a partially filled batch to avoid idling. Conversely, for single-step machines theory suggests that high utilization is only possible when arrival times are uniform (Stidham Jr, 2002). I.o.w. single-step machines would optimally be "fed" by a stream of lots with an evenly spaced arrival rate instead of waves of lots from preceeding batch machines. Therefore, these switches between batch and single-step processing introduces so-called WIP (work in progress) waves between machines which are a major obstacle to production optimization.

3 ORIGINAL ALGORITHMS

3.1 The ABC Algorithm

The artificial bee colony (ABC) algorithm is inspired by the foraging behavior of honeybees introduced in (Karaboga and Basturk, 2007; Karaboga, 2010) for the optimization of numerical problems. It uses typical swarm concepts: recruitment of foragers to rich food sources resulting in positive feedback and abandonment of poor sources by foragers causing negative feedback. For the ABC you consider an optimization problem that is first converted to the problem of finding the best parameter vector, which minimizes an objective function. It uses different kinds of agents:

- Food source: is described with several parameters, including distance and orientation to the nest, quality of the food and concentration, and ease of extracting the food.
- Employed foragers: are bees that are associated with the food source they are "employed" at. They hold the information about the food source they are associated with (distance and orientation from the nest). They share the information of the food source with a certain probability by performing so-called waggle dances. For each food source there is only one employed bee (number of employed bees equals the number of food sources).
- Unemployed foragers: are bees that continuously look for a food source. We differ between (1) scouts that search for a new food source in the surrounding of the nest, and (2) onlookers that wait in the nest and get information about the food source from the employed foragers. They watch several waggle dances by employed foragers and decide on the most profitable source.

Generally, the ABC algorithm as global optimizer has following steps (for more formal details, the reader is referred to (Karaboga, 2010)):

Algorithm 1: ABC as global optimizer.

1: Initialization Phase (population of the food source)
2: **repeat**
3: Employed Bees Phase
4: Onlooker Bees Phase
5: Scout Bees Phase
6: Memorize the best solution
7: **until** Cycle = Maximum Cycle Number or a Maximum CPU time

Several research works already used the ABC algorithm in job scheduling applications (see Section 6). Summarized, they typically use population of bees that present a solution space, and make calculations centrally. Instead, in this paper, we do not create a solution space, but rather choose the bottom-up approach, where bees represent agents (instead of solutions) that work with local rules from which a global behavior (optimization of the industrial plant) emerges. Thus, we follow a completely new approach of the ABC algorithm application.

3.2 The Bat Algorithm

The bat algorithm was first introduced in (Yang, 2010) and was inspired by the behavior of microbats that use echolocation to detect and hunt their prey. Like other metaheuristic nature-inspired algorithms, it is primarily used for optimization problems. The algorithm is based on three idealized rules of the behavior of microbats:

1. Bats use echolocation to measure the distance to the prey.
2. Bats move with a velocity v_i at a certain position x_i with a fixed frequency f_{min} and they can adjust the loudness A_0 and the rate of pulse $r \in [0,1]$ of their signal depending on their proximity to the prey.
3. The loudness of the emitted signal varies from maximum A_0 to minimum A_{min}, depending on the proximity of the bat to the prey.

The movement and frequency of the bats are described by equations 1 – 3. The first equation denotes the pulse frequency f_i of the signal, where $\beta \in [0,1]$ is a vector drawn from a uniform distribution. The update rules for the positions x_i and velocities v_i of bats are defined by Equations 2 and 3. The positions and velocities are updated at every time step t. In Equation 2, x_* denotes to the best current global solution.

$$f_i = f_{min} + (f_{max} - f_{min}) \cdot \beta \tag{1}$$

$$v_i^t = v_i^{t-1} + (x_i^t - x_*) \cdot f_i \tag{2}$$

$$x_i^t = x_i^{t-1} + v_i^t \tag{3}$$

Once the current best global solution is found, a new solution for each bat is generated by a random walk. Equation 4 describes how the new solution is achieved, where $\varepsilon \in [-1,1]$ is a random number and A^t is the average loudness of bats at the current time step.

$$x_{new} = x_{old} + \varepsilon A^t \tag{4}$$

The loudness A_i of the emitted signal and the pulse rate r_i are updated according to Equations 5 and 6, where α and γ are constant values. The loudness decreases the closer the bat is to its prey. On the contrary, the pulse rate increases when the bat homes in on the target. The pseudo code for the original bat algorithm is described in Algorithm 2.

$$A_i^{t+1} = A_i^t \cdot \alpha \tag{5}$$

$$r_i^{t+1} = r_i^0 \cdot [1 - exp(-\gamma t)] \tag{6}$$

Just as with the ABC algorithm, the original bat algorithm is applied to the problems within the solution space and is used as a global optimizer. For the objective of our problem adjustments have been made to adapt the algorithm to the bottom-up approach.

Algorithm 2: Bat algorithm as global optimizer.

1: Initialization Phase (initialization of the bat population x_i, objective function f(x), frequency f_i at x_i, pulse rate r_i, velocity v_i and loudness A_i)
2: **repeat**
3: Generate new solutions by adjusting position x_i, frequency f_i and velocity v_i according to Equations 1 – 3
4: Evaluate solutions and select the best one
5: Generate new local solution around the best solution
6: Generate new random solution (random fly)
7: Adjust loudness A_i and pulse rate r_i
8: Select the best solution x_*
9: **until** Cycle = Maximum Cycle Number or a Maximum CPU time

4 THE ALGORITHMS IN A BOTTOM-UP APPROACH

4.1 The ABC Algorithm

For the needs of our use case of production scheduling, the following mapping has been chosen:

- food source = machine, $M_i^m \in W^m, i = 1,2,\ldots,I$
- bees = lot from one product, $l_n^t \in L^t, n = 1,2,\ldots,N$

where machine M_i^m belongs to a workcenter $W^m \subset M$ and M represents all machines in the fab. Each lot $l_n^t \in L^t$ has access to its R^t (sequence of process steps P^m) and thus, a list of the machines that are used by the lot.

Therefore lots, which we modeled as bees, need to choose the next machine M_i^m out of the set W^m, to perform their next process $P^m \in R^t$. For this, it is important to distinguish between l_n^t modeled as scout bee l_{SB} and onlooker bee l_{OB}, due to each lot type choosing machines differently.

The scout bees are those that explore the food sources. They share their experience and knowledge of food quality with all other bees. In the beginning of our simulation, there is no a-priori quality knowledge of any machine. For this reason, our simulation starts with the Scout Bees Phase, due to scout bees selecting their food source randomly. This leads to l_n^t modeled as l_{SB} starting the search for their machines first, to collect quality information. The collected information will be later used to attract l_{OB}. Upon finishing the process $P^m \in R^t$ at the chosen machine $M_i^m \in W^m$, l_{SB} will evaluate the quality of the machine M_i^m following Equation 7:

$$Q(M_i^m) = \frac{1}{w_{SB}} \tag{7}$$

where w_{SB} represents queuing time and processing time of a previous l_{SB} that has chosen machine $M_i^m \in W^m$ to perform its next process $P^m \in R^t$. Therefore, the longer it took machine M_i^m to process the previous l_{SB}, the lower the quality $Q(M_i^m)$ will be.

Unlike the original ABC algorithm, in our model of the industrial use case, there is no employed bee concept as such, due to all l_n^t, only moving forward and not coming back to the hive to bring the information about machines (i.e., to perform the waggle dance). Instead, we model this function of an employed bee as the machine's own memory of its quality communicated by the l_{SB}. Lots modeled as onlooker bees, i.e. l_{OB}, and interested in having their process performed on one of the machines $M_i^m \in W^m$, can access the machines' memory and read the quality information left by the l_{SB}. Therefore, our model uses stigmergy as communication media, i.e., no messages are exchanged among the swarm members.

Once l_{SB} provides quality information of possible next machines, l_{OB} will probabilistically choose the best-quality machine. The probability $P_r(M_i^m)$ with which l_{SB} choose the best solution, which is one M_i^m out of the set W^m, is described in (Karaboga, 2010) and given as follows:

$$P_r(M_i^m) = \frac{fit(M_i^m)}{\sum_{i=1}^{I} fit(M_i^m)} \tag{8}$$

where $fit(M_i^m)$ denotes the fitness value of machine M_i^m over the sum of fitness values of all I machines in workcenter W^m.

In the original ABC algorithm, bees in the Employed Bees Phase try to find a new better solution in the neighborhood of an existing solution. The new solution is generated randomly and then compared to the existing one. If the newly selected food source has better quality, greedy selection will be applied.

In our adaptation, in the Employed Bees Phase the neighborhood consists of lots waiting in Q_i^m of a chosen M_i^m. Namely, when lots choose their preferred $M_i^m \in W^m$ they are put in Q_i^m to wait for their processing turn. By default, the l_n^t at the beginning of the Q_i^m will be processed first, following a simple First-In First-Out algorithm. In our model, when queued and in case their next step $P^m \in R^t$ corresponds to a batch machine, all such lots enter the Employed Bees Phase. The reasoning for this is the different mode of processing of single-step and batch machines. Since batch machines wait for a certain time period for lots to accumulate, it may happen that there are some lots that could have filled the batch machine if they had been processed first at the previous single-step machine.

Lastly, in the original ABC algorithm, employed bees abandon their food sources due to the food source having poor quality or becoming poor quality due to excessive exploitation. As already stated before, in our model there is no concept of an employed bee, therefore we use a predefined limit value l that serves as a machine's quality value timeout. After a certain number of onlooker lots l_{OB} have visited and exploited the chosen $M_i^m \in W^m$, its quality $Q(M_i^m)$ will have to be re-evaluated by another l_{SB}.

In contrast to the original ABC algorithm, our variant implements different phases wrt. the different process steps in R^t of l_n^t. For each processing step in R^t, the l_n^t moves from M_i^m to M_j^p, and go through the following steps:

Algorithm 3: Bottom-up ABC.

```
1: Initialization Phase (population of lots and machines)
2: repeat
3:     switch m_prev → m_next do
4:         case 0 → SingleStep:
               Scout Bees Phase;
               Onlooker Bees Phase;
5:         case 0 → Batch:
               Scout Bees Phase;
               Onlooker Bees Phase;
6:         case SingleStep → SingleStep:
               Scout Bees Phase;
               Onlooker Bees Phase;
7:         case SingleStep → Batch:
               Employed Bees Phase;
8:         case Batch → Batch:
               Scout Bees Phase;
               Onlooker Bees Phase;
9:         case Batch → SingleStep:
               Scout Bees Phase;
10: until all lots have found their last machine
```

4.1.1 Initialization Phase

Firstly, a number of lots and machines in a fab is set, as well as the limit l after which a machine's quality will have to be updated. The machine's memory gets initialized as well.

4.1.2 Case 0 to SingleStep

At the beginning of our simulation, there is no a-priori information on machines, as already explained. Therefore lots that are first to choose a machine $M_i^m \in W^m$, must do it without comparing their quality value given that it does not exist yet. For this reason, firstly l_{SB} are allocated. After the machine M_i^m processes them, it will be evaluated by the l_{SB}. Once all machines in W^m had been evaluated, the Onlooker Bees Phase will begin. Namely, l_{OB} will choose probabilistically, the best one from the W^m.

4.1.3 Case 0 to Batch

Given the difference in the mode of operation for both single-step and batch machines, the implementation of the concept described above is slightly changed. Namely, instead of calculating the machine's quality as in Equation 7, after a random l_{SB} allocation, l_{OB} lots will choose $M_i^m \in W^m$ as follows: Firstly, all machines in W^m that already have some other lots in their queues of the same product type (i.e., belonging to the same group L^t), will be inspected. Then a number of free places n_{fs} (until the full batch) in such machines, will be counted. If there is a batch-oriented machine $M_i^m \in W^m$ with $n_{fs} = 1$, the lot l_{OB} will choose this machine. If there is more than one machine with $n_{fs} = 1$, then the best machine is considered the one with the shortest remaining wait time $t_{M_i^m}$ for lots to accumulate, before M_i^m processes a semi-filled batch. If there is not one $M_i^m \in W^m$ with $n_{fs} = 1$, then the l_{OB} lot will search for machines having $n_{fs} = 2$ places missing, etc.

4.1.4 Case SingleStep to SingleStep

Assuming lots have progressed with their production following R^t, some of the machines with good quality $Q(M_i^m)$ will be chosen more frequently than others in the same workcenter W^m. This will eventually lead to accumulated lots in the queues of such machines, therefore the total waiting time w_{SB} will increase. To maintain w_{SB} up to date, the limit value l of a machine M_i^m gets decreased each time M_i^m gets chosen by a lot. When the limit value l becomes equal to 0, this M_i^m will lose its "evaluated" status. This will mean that some of the incoming l_{SB} will choose this M_i^m randomly, and upon finishing the process will re-evaluate it. This will provide a new, updated $Q(M_i^m)$ value for the incoming l_{OB}. Additionally, since l_{OB} chooses the best $M_i^m \in W^m$ probabilistically, there is a $1 - P_r(M_i^m)$ chance, that this l_{OB} will change its status to an l_{SB}. Namely, instead of exploiting the best machine, it will explore the W^m and select one machine randomly.

4.1.5 Case SingleStep to Batch

Since the influence of batch machines on the production of industrial plants is severe, it is important to optimize this change in each lot's recipe. As already mentioned earlier in this section, it may happen that some lots waiting in a queue of a single-step machine could have filled out a batch machine if they had been processed first. In the original ABC algorithm, in the Employed Bees Phase, a new solution gets generated and compared to the existing one and then a better one gets selected following the greedy selection approach. Therefore, in our algorithm, we model a better solution generated in the Employed Bees Phase and greedy selection as follows:

All lots in queue Q_i^m of a single-step machine

M_i^m that have next machine M_j^p batch-oriented, will be compared. Instead of processing the first lot in the Q_i^m, single-step machine M_i^m will process the lot which gives the minimum values from the set $[\, n_{fs}, t_{M_i^m}]$, as described in **case** $0 \rightarrow Batch$.

4.1.6 Case Batch to Batch

The concept here is similar to the one described in **case** $0 \rightarrow Batch$. Instead of a single l_{OB}, here we consider the whole batch of lots processed by a batch-oriented M_i^m. Since the next process step P^p of these lots corresponds to another batch-oriented M_j^p, the group of lots will choose one $M_j^p \in W^p$ as given above.

4.1.7 Case Batch to SingleStep

Finally, since a batch machine waits and then processes several lots at once, this results in waves of lots being introduced to the following machines. In order to evenly distribute these lots and not create overwhelming queues of lots for single-step machines, we modeled this case by randomly distributing lots over the whole set W^m of the next possible machines.

4.2 The Bat Algorithm

As with the ABC algorithm, we propose a mapping for the bottom-up bat algorithm to the production scheduling problem as follows:

- bats = machine, $M_i^m \in W^m, i = 1,2,\ldots,I$
- prey = lot from one product, $l_n^t \in L^t, n = 1,2,\ldots,N$

where machine M_i^m belongs to a workcenter $W^m \subset M$ and M represents all machines in the fab. Each lot $l_n^t \in L^t$ has access to its R^t (sequence of process steps P^m) and thus, the machine types that have to be used to produce the lot.

In the original bat algorithm, the population of bats is initialized with a certain position, velocity and frequency, as well as loudness and pulse rate of the echolocation, which varies depending on the distance of the bat to the prey. With each iteration new solutions are generated by adjusting these parameters, then the bats are evaluated and the best solution is chosen.

When mapping the original algorithm to this particular production scheduling problem parts of the algorithm have been adapted and simplified. First of all, since the bats or the machines in the factory are static, as opposed to the original algorithm where the bats fly and change their position and velocity, there is no need to use the before mentioned equations 1 – 3. The bats are static in said environment due to the fact that the machines in the production are themselves static, and the lots are the ones that are dynamic and move throughout the factory from one work center to the next. Thus, the problem is reconfigured from bats having to search and hunt their prey, to prey being "lured" by the bat that is closest to them. In other words, the lot to be processed is being chosen by the most optimal machine in the work center. The mechanism behind the choice is further described in the following section.

The distance from the bat to the prey is calculated based on the Equation 9.

$$Dist = min \sum (\alpha \cdot P_{distr} + \beta \cdot Q_{distr}) \tag{9}$$

In terms of production, the equation corresponds to the minimum value of the sums of the product (P_{distr}) and queue (Q_{distr}) distributions of a specific work center. The equation is calculated before the lot chooses a queue of a current work center, depending on the lot's recipe.

Product distribution is calculated for each machine in the work center. In Equation 10 the sum in the nominator corresponds to the number of the lots with the same product type as the lot choosing the queue for each machine. The sum in the denominator corresponds to the number of the lots with the corresponding product type for the whole work center.

$$P_{distr} = \frac{\sum_{j=1}^{num(M_i^m)} l_j^t}{\sum_{k=1}^{I} l_k^t} \tag{10}$$

Queue distribution is also calculated for each machine in the work center. As shown in Equation 11 it is achieved by dividing the queue length of each of the machines in the work center by the sum of all the queue lengths of the work center.

$$Q_{distr} = \frac{qlen(M_i^m)}{\sum_{j=1}^{I} qlen_j(M_i^m)} \tag{11}$$

The product and queue distributions for each machine in the work center are saved in the memory of their respective machines. Once both distributions are calculated for each machine, as described in Equation 9, they are multiplied by their respective weights, added up and saved in the memory again, thus creating a list of values for each machine. The machine with the minimum distribution value then is chosen to process the lot.

Parameters α and β are the weights that are applied to the product and queue distributions respectively. For this version of the algorithm α and β are constant values (α = 0.7, β = 0.3). The values were chosen based on testing.

The before mentioned solution is proposed for the case of a work center with single-step machines. To choose the optimal machine in a work center with batch machines we use the same mechanism as the cases "0 to Batch" and "Batch to Batch" in the ABC bottom-up algorithm described in Sections 4.1.3 and 4.1.6.

The pseudo code for the bottom-up bat algorithm is shown in Algorithm 4.

Algorithm 4: Bottom-up bat algorithm.

1: Initialization Phase (initialization of the alpha, beta parameters, memory)
2: **repeat**
3: **switch** $m_{singleStep} \rightarrow m_{batch}$ **do**
4: **case** *SingleStep*:
Calculate Q_{distr} for the WC;
Calculate P_{distr} for the WC;
Save Q_{distr} and P_{distr} to the memory of each machine of the WC;
Calculate the weighted sum of the queue and product distributions for each machine in the WC;
Save the weighted sum of the distributions (the distance) to memory of each machine of the WC;
Choose the machine with the minimum weighted sum value (Choose the bat with the minimum distance to the prey);
5: **case** *Batch*:
Analogous to bottom-up ABC cases "case 0 → Batch" and "case Batch → Batch";
6: **until** all lots have found their last machine

5 EVALUATION AND RESULTS

5.1 Simulation Environment

We use NetLogo(Wilensky, 1999) which is one of the most-used agent-based simulation platforms worldwide and free to use. On top of this we implemented a simulation framework which models the machines, queues, and products in the fab and supports different plug-in algorithms to optimize and control product movement, queue management and process scheduling. Machines, their queues, and the lots to be produced are each modelled as agent types (NetLogo "breeds") with their own, respective attributes. The main simulation loop is run once per time tick and drives the movement of lots through the factory: first, each lot not currently being processed or in a queue gets to choose the next queue. This is facilitated by the *choose-queue* callback to the currently active algorithm. Then, every machine that is not currently processing chooses the next lot (or set of lots, in case of a batch machine) from its respective queue. This decision is made via the *take-from-queue* callback to the currently active algorithm. After the lot(s) has/have finished processing, the algorithm is called again (*move-out* callback) to facilitate communications or other updates before the lot(s) move to the next queue. At the beginning and end of every tick, two more algorithm callbacks (*tick-start, tick-end*) enable time-based updates (such as, e.g., the degradation of pheromones). We use NetLogo BehaviorSpace to control the simulation runs and result log file creation and the R statistics package to post-process the results. A detailed description of our simulation framework can be found in (Umlauft et al., 2022) and the source code plus necessary configuration files can be obtained at our GitHub repository at https://swarmfabsim.github.io.

5.2 The Baseline Algorithm

Baseline is a memoryless/stateless algorithm provided as reference for performance comparisons. The idea of the baseline algorithm is to use simple heuristics which use only local information to calculate its decisions like the swarm algorithms that it is compared to. We choose this model of only using local information/computation because we want to consider problem sizes that would be too large to calculate an optimal global solution in feasible time.

For single-step oriented machines, assigning a lot to the machine with the shortest queue out of the possible machines for the next step and to run these queues in FIFO mode fulfills these requirements. In real-world production plants, such as Infineon Technologies, of course more sophisticated approaches are used, typically involving several priority levels to reshuffle lots in a queue. The underlying principle on which those approaches are based, can in its most simple form be abstracted to "shortest queue/FIFO".

For batch machines, the most important idea is to use the machine at maximum capacity and neither run it with semi-filled batches nor have it sit idle. Therefore, in the real world, there is a waiting time WT that a batch machine will wait for enough lots of the same type to arrive to fill a whole batch. When only using information local to the current machine (the smallest information horizon), the best a simple algorithm can do is to start the machine immediately once enough lots of a type have arrived to fill a batch or to wait for

the waiting time WT and to then take the largest semi-filled batch available. In real life, this basic principle would be enhanced by corporate secret algorithms that additionally take other information into account, like deadlines that lots have to conform to.

In our version of baseline, we try to keep the information horizon as small as possible and therefore do not implement such communications. We also deliberatey do not consider the interplay between single-step machines and batch machines as described in Section 2, because we want to investigate the potential for performance improvement inherent in addressing these alternating processing modes.

In detail, the baseline algorithm works as follows: for the *choose-queue* callback it differentiates between single-step and batch machines. For machines that process lot-by-lot, it simply chooses the shortest queue. For batch machines 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 of this type in any of the potential queues, it chooses the queue with the least overall queue length.

For the *take-from-queue* callback, it uses a FIFO approach on single-step machines, while on batch machines, 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. After 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 and processed immediately.

5.3 Simulation Settings

We evaluated the algorithms in three scenarios, a small (SFAB), a medium-sized (MEDIUM), and a large (LFAB) model of a fab with different numbers of machines and machine types. The LFAB scenario also has a higher number of product types and lots per type than the SFAB and MEDIUM scenarios. The parameters for these scenarios are shown in Table 1.

Table 1: Parameters used to create the three evaluation scenarios.

Parameter	SFAB	MEDIUM	LFAB
Mach. types	25	50	100
Mach. / type	$U(2,5)$	$U(2,10)$	$U(2,10)$
Product types	50	50	100
Recipe length	$U(90,110)$	$U(90,110)$	$U(90,110)$
Lots per type	$U(1,10)$	$U(1,10)$	$U(2,10)$

Table 2 depicts the distribution parameters used to create machines, where $N(\mu,\sigma^2)$ denotes 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.

Table 2: Machine parameters used in the simulation.

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

5.4 Results and Discussion

For the evaluation, each scenario setting has been run 30 times, and averaged.The algorithms are evaluated according to four performance metrics: makespan, average flow factor, average tardiness, and average machine utilization. Tables 3, 4, and 5 depict a comparison between the performance of the reference algorithm "Baseline" and the ABC and bat inspired algorithms.

Makespan (MS): represents the total time for the production plant to produce all lots from start to finish (in simulation ticks). All lots are introduced into the fab at the same time at the start of the simulation.

textbfFlow factor (FF): describes the relation between the actual production time (including queue waiting times) and the theoretical minimum production time (sum of raw processing times of all required steps in the recipe). The result is averaged over all lots.

Tardiness (TRD): describes how much additional time (due to lots waiting in a queue) has been accumulated until production of the lot compared to the theoretical minimum production time. The result (in simulation ticks) is averaged over all lots.

Machine utilization (UTL): represents how much percent of time an average machine has been in operation. For this, the total sum of working time of all machines is divided by the simulation time and normalized by the number of machines.

Table 3: Small Scenario (SFAB). Changes in %, positive values denote improvement over Baseline.

	Baseline	ABC	Change	Bat	Change
MS	10398	10838	-4.2%	9851	5.3%
FF	6.51	6.55	-0.6%	5.24	19.6%
TRD	6971.1	7031.5	-0.9%	5312.5	23.8%
UTL	35.90	33.74	6.0%	32.58	9.2%

Although the ABC and the bat algorithm are not new alogrithms, we engineer them as literal swarm intelligence algorithms: instead of performing a global

Table 4: Medium Scenario (MEDIUM). Changes in %, positive values denote improvement over Baseline.

	Baseline	ABC	Change	Bat	Change
MS	4536	5403	-19.1%	5604	-23.5%
FF	3.21	3.36	-4.9%	3.30	-2.8%
TRD	2481.8	2656.0	-7.0%	2576.9	-3.8%
UTL	23.87	19.58	18.0%	19.34	19.0%

Table 5: Large Scenario (LFAB). Changes in %, positive values denote improvement over Baseline.

	Baseline	ABC	Change	Bat	Change
MS	6207	6528	-6.0%	5790	6.7%
FF	3.47	3.54	-1.9%	2.78	20.0%
TRD	3126.8	3210.38	-2.7%	2244.0	28.2%
UTL	22.71	20.37	10.3%	23.94	-5.4%

optimization by applying them to solutions in the solution space, we chose a bottom-up approach where a global behavior emerges from local rules in an agent-based modeled plant. The performance evaluation shows that the ABC algorithm performs slightly worse than Baseline for the SFAB and LFAB scenarios and worse in the MEDIUM scenario. For the bat algorithm, SFAB and LFAB scenarios depict very promising improvements, while the MEDIUM scenario performs worse.

As described in Section 2, our understanding is that the job shop problem is exacerbated by the switch between single-step oriented and batch machines. Therefore, an algorithm like Baseline, that pursues a strategy to best fill semi-filled batches (and therefore reduces idling time and inefficient use of batch machines) is hard to beat by algorithms that only employ local calculations to decide queue assignment. This has also been observed in our previous work (Umlauft et al., 2022), where we show how a hormone algorithm that spreads information (in the form of artificial hormones) back several machines is able to consistently beat Baseline while an ant algorithm that uses only local pheromones (at the current workcenter) is not. In general, an optimal algorithm for this problem should take these switches from single-step to batch machines and vice versa explicitly into account. We are therefore planning to improve on the proposed algorithms in the future by increasing the size of the local neighborhood by looking several machines "ahead" and anticipating switches between machine kinds.

6 RELATED WORK

The following paragraphs gives a summary of the currently available research activities on the ABC algorithm. Multiple considerations and variants of the ABC algorithm for job-shop scheduling problem (JSSP) exist: (Karaboga and Basturk, 2007) uses the ABC algorithm for optimizing multivariable functions. (Yao et al., 2010) (Han et al., 2012) propose the improved artificial bee colony (IABC) algorithm that can transform roles during the search process and enhance convergence rate. By comparing with Genetic algorithm (GA) and the simple ABC, IABC is also examined to be an effective and efficient method for solving the JSSP. (Gupta and Sharma, 2012) use additional mutation and crossover operator of GA in the classical ABC algorithm. They add a crossover operator after the employed bee phase and a mutation operator after onlooker bee phase of ABC algorithm, with the criterion to decrease the maximum completion time. (Zhang et al., 2013) apply the ABC algorithm with the objective of minimizing total weighted tardiness. (Alvarado-Iniesta et al., 2013) examine how to optimize the time and effort required to supply raw material to different production lines in a manufacturing plant in Juarez, Mexico by minimizing the distance an operator must travel to distribute material from a warehouse to a set of different production lines with corresponding demand. The ABC algorithm is applied in order to find the optimal distribution of material with the aim of establishing a standard time for this duty by examining how this is applied in a local manufacturing plant. (Pan et al., 2011) aim at minimizing total weighted earliness and tardiness penalties for the lot-streaming JSSP with equal sized sublots. examine the problem under both the idling and no-idling cases and propose a novel discrete artificial bee colony (DABC) algorithm. (Li et al., 2014) proposes the DABC algorithm for solving the multi-objective flexible JSSP with maintenance activities. Due to the reasonable hybridization of the ABC search and tabu search (TS) based local search, the proposed DABC algorithm has the ability to obtain promising solutions for the problem considered. (Tasgetiren et al., 2013) they expand the DABC to solve the no-idle permutation JSSP with the total tardiness criterion by applying a novel speed-up method for the insertion neighborhood. (Liu and Liu, 2013) use the hybrid DABC algorithm for minimizing the makespan in permutation JSSP. Also (Pan et al., 2014), they use the DABC with a hybrid representation and a combination of forward decoding and backward decoding methods for solving the problem. (Wang et al., 2012) introduce an enhanced

Pareto-based artificial bee colony (EPABC) algorithm to solve the multi-objective flexible JSSP with the criteria to minimize the maximum completion time, the total workload of machines, and the workload of the critical machine simultaneously. (Kumar et al., 2014) outline a new hybrid of ABC algorithm with GA. The proposed method integrates crossover operation from GA with original ABC algorithm. The proposed method is named as Crossover based ABC (CbABC). The CbABC strengthens the exploitation phase of ABC as crossover enhances exploration of search space. (Yurtkuran and Emel, 2014) propose the modified artificial bee colony (M-ABC) algorithm to solve p-center problems. The proposed approach has two main contributions: random key-based encoding for solution representation and a new multi search strategy in which different search strategies are employed in one overall search process. (Li and Pan, 2015) present a novel hybrid algorithm (TABC) that combines the ABC and TS to solve the hybrid flow shop (HFS) scheduling problem with limited buffers. The main advantages of the proposed algorithms are as follows: the TS-based self-adaptive neighborhood strategy is embedded in TABC, which can balance the exploitation and exploration abilities of the algorithms; (2) the TS-based local search is applied to the employed bees and onlookers with different functions, which can further enhance the exploitation ability of the proposed algorithm. In (Gao et al., 2015) the TABC is compared against and outperforms eleven heuristic and meta-heuristics algorithms. Further, (Gao et al., 2016) investigate the FJSP with fuzzy processing time. (Sharma and Pant, 2017) introduce a hybrid version of ABC and shuffled frog-leaping algorithm. (Sharma et al., 2018) present a modified ABC algorithm to solve JSSP, in the onlooker bee phase of ABC, to maintain a proper harmony amid exploration and exploitation capabilities, beer froth phenomenon inspired position update is incorporated. The proposed strategy is named as Beer froth artificial bee colony algorithm. (Zhuang et al., 2019) present JSSPs with two sequence-dependent setup times. The problem is different from the traditional open JSSP, with higher complexity. The mixed integer linear programming model is proposed to solve small-scale problems and get exact solutions. Compared with other four intelligent algorithms, such as GA, PSO, ant colony optimization and cuckoo search algorithm, two experiments have been conducted and the computational results shows that the proposed artificial bee colony algorithm can achieve the best result in the large-scale problems A detailed review on ABC variants on data clustering can be found in (Kumar et al., 2017). They state that the ABC is a simple and flexible algorithm and requires less parameters to be tuned in comparison to other meta-heuristic algorithms. Furthermore, in (Khader et al., 2013) present a thorough and extensive overview of most research work focusing on the application of ABC, with the expectation that it would serve as a reference material to both old and new, incoming researchers to the field. Further applications are presented in (Karaboga et al., 2014).

Bat algorithm and various improved and modified versions of the algorithm have been applied to the job shop scheduling problem (JSSP) over the last decade since the algorithm was first introduced in (Yang, 2010). In 2012 bat algorithm was used in (Musikapun and Pongcharoen, 2012) to develop the Bat Algorithm based Scheduling Tool to solve multi-stage multi-machine multi-process scheduling problems. (Marichelvam et al., 2013) apply bat algorithm to the multistage hybrid flow shop (HFS) scheduling problems. The results show the bat algorithm outperforms both the genetic algorithm and the particle swarm optimization algorithm. (Luo et al., 2014) develop the discrete bat algorithm for optimal permutation flow shop scheduling problem. (Xu et al., 2017) developed an improved bat algorithm to solve a dual flexible job-shop scheduling problem (DFJSP). The authors use a dual flexible encoding strategy to produce the mapping between operations and bat populations. The experiment results show that the proposed model and improved algorithm could be effectively applied to DFJSP. (Zhu et al., 2017) proposed a modified bat algorithm to solve multi-objective JSSP. They were the first to successfully apply the bat algorithm to the multi-objective flexible JSSP. (Dao et al., 2018) put forward an optimization algorithm based on parallel versions of the bat algorithm, random-key encoding scheme and makespan scheme to solve JSSP. The results show an improvement in the convergence and accuracy compared to the original bat algorithm and particle swarm optimization. (Lu and Jiang, 2019) were the first to apply the bat algorithm to the low-carbon JSP. They propose a bi-population based discrete bat algorithm, more specifically they propose a parallel searching mechanism to divide the population, as well as a modified discrete updating approach to make the bat algorithm search within a discrete domain. More recently, (Chen et al., 2019) proposed another improved bat algorithm for solving the JSSP, which focuses on speeding up convirgance and determining the optimal global solution. They compare the improved bat algorithm to the original bat algorithm and particle swarm optimization algorithm and successfully outperform them in minimizing makespan and finding the optimal global solution. (Yang and

He, 2013) provide a comprehensive review of different variants of the bat algorithm and its applications, as well as various case studies. A more recent review was provided in (Jayabarathi et al., 2018). It focuses on the usage of the bat algorithm and its application to optimize various engineering problems.

7 CONCLUSION AND FUTURE WORK

This paper has proposed the use of two bio-inspired algorithms, namely the artificial bee colony (ABC) and the bat algorithm in a bottom-up manner to tackle the problem of optimization for a semiconductor fab using the job-shop manufacturing principle. Unlike previous work, where these algorithms are calculated for global optimization, we do not apply them to solutions in the solution space. Instead, we chose a bottom-up approach and applied them as literal swarm intelligence algorithms in a bottom-up manner. This means that the active agents (artificial bees, bats) were mapped to entities in the production plant and only use information from their current local neighborhood to take decisions from which the global solution then emerges automatically. We used NetLogo to simulate the fab and to measure the performance of the proposed algorithms compared to an engineered baseline algorithm. The evaluation was based on four key performance indicators: makespan, flow factor, tardiness, and machine uptime utilization. Our results show that these types of algorithms, employed in a bottom-up manner, show promise of performance improvements using only low-effort local calculations. The implementation of the simulation environment is published as open source in a Git repository[2].

Future work will expand on the details of the proposed algorithms by investigating the impact of the size of the local neighborhood used for calculations (eg. by looking ahead or back several machines) and adaptations to explicitly address the switching between single-step oriented machines and batch machines during the production process.

ACKNOWLEDGEMENT

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

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

REFERENCES

Alvarado-Iniesta, A., Garcia-Alcaraz, J. L., Rodriguez-Borbon, M. I., and Maldonado, A. (2013). Optimization of the material flow in a manufacturing plant by use of artificial bee colony algorithm. *Expert Systems with Applications*, 40(12):4785–4790.

Chen, X., Zhang, B., and Gao, D. (2019). An improved bat algorithm for job shop scheduling problem. In *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 439–443. IEEE.

Dao, T.-K., Pan, T.-S., Pan, J.-S., et al. (2018). Parallel bat algorithm for optimizing makespan in job shop scheduling problems. *Journal of Intelligent Manufacturing*, 29(2):451–462.

Gao, K., Cao, Z., Zhang, L., Chen, Z., Han, Y., and Pan, Q. (2019). A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. *IEEE/CAA Journal of Automatica Sinica*, 6(4):904–916.

Gao, K. Z., Suganthan, P. N., Chua, T. J., Chong, C. S., Cai, T. X., and Pan, Q. K. (2015). A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion. *Expert systems with applications*, 42(21):7652–7663.

Gao, K. Z., Suganthan, P. N., Pan, Q. K., Tasgetiren, M. F., and Sadollah, A. (2016). Artificial bee colony algorithm for scheduling and rescheduling fuzzy flexible job shop problem with new job insertion. *Knowledge-based systems*, 109:1–16.

Gupta, M. and Sharma, G. (2012). An efficient modified artificial bee colony algorithm for job scheduling problem. *International Journal of Soft Computing and Engineering (IJSCE)*, 1(6).

Han, Y.-Y., Pan, Q.-K., Li, J.-Q., and Sang, H.-y. (2012). An improved artificial bee colony algorithm for the blocking flowshop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 60(9-12):1149–1159.

Jayabarathi, T., Raghunathan, T., and Gandomi, A. (2018). The bat algorithm, variants and some practical engineering applications: A review. *Nature-inspired algorithms and applied optimization*, pages 313–330.

Karaboga, D. (2010). Artificial bee colony algorithm. *scholarpedia*, 5(3):6915.

Karaboga, D. and Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39(3):459–471.

Karaboga, D., Gorkemli, B., Ozturk, C., and Karaboga, N. (2014). A comprehensive survey: artificial bee colony (abc) algorithm and applications. *Artificial Intelligence Review*, 42(1):21–57.

Khader, A. T., Al-betar, M. A., and Mohammed, A. A. (2013). Artificial bee colony algorithm, its variants and applications: a survey. *Journal of Theoretical and Applied Information Technology*, 47(2):434–459.

Khatmi, E., Elmenreich, W., Wogatai, K., Schranz, M., Umlauft, M., Laure, W., and Wuttei, A. (2019). Swarm in-

telligence layer to control autonomous agents (swilt). In *STAF (Co-Located Events)*, pages 91–96.

Kumar, A., Kumar, D., and Jarial, S. (2017). A review on artificial bee colony algorithms and their applications to data clustering. *Cybernetics and Information Technologies*, 17(3):3–28.

Kumar, S., Sharma, V. K., and Kumari, R. (2014). A novel hybrid crossover based artificial bee colony algorithm for optimization problem. *arXiv preprint arXiv:1407.5574*.

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.

Li, J.-q. and Pan, Q.-k. (2015). Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm. *Information Sciences*, 316:487–502.

Li, J.-Q., Pan, Q.-K., and Tasgetiren, M. F. (2014). A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities. *Applied Mathematical Modelling*, 38(3):1111–1132.

Liu, Y.-F. and Liu, S.-Y. (2013). A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem. *Applied Soft Computing*, 13(3):1459–1463.

Lu, Y. and Jiang, T. (2019). Bi-population based discrete bat algorithm for the low-carbon job shop scheduling problem. *IEEE Access*, 7:14513–14522.

Luo, Q., Zhou, Y., Xie, J., Ma, M., and Li, L. (2014). Discrete bat algorithm for optimal problem of permutation flow shop scheduling. *The Scientific World Journal*, 2014.

Marichelvam, M., Prabaharan, T., Yang, X.-S., and Geetha, M. (2013). Solving hybrid flow shop scheduling problems using bat algorithm. *International Journal of Logistics Economics and Globalisation*, 5(1):15–29.

Musikapun, P. and Pongcharoen, P. (2012). Solving multi-stage multi-machine multi-product scheduling problem using bat algorithm. In *2nd international conference on management and artificial intelligence*, volume 35, pages 98–102. IACSIT Press Singapore.

Pan, Q.-K., Tasgetiren, M. F., Suganthan, P. N., and Chua, T. J. (2011). A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. *Information sciences*, 181(12):2455–2468.

Pan, Q.-K., Wang, L., Li, J.-Q., and Duan, J.-H. (2014). A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. *Omega*, 45:42–56.

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, Methodolgies, Technologies and Applications (SIMULTECH)*, pages 327–334.

Sharma, N., Sharma, H., and Sharma, A. (2018). Beer froth artificial bee colony algorithm for job-shop scheduling problem. *Applied Soft Computing*, 68:507–524.

Sharma, T. K. and Pant, M. (2017). Shuffled artificial bee colony algorithm. *Soft Computing*, 21(20):6085–6104.

Stidham Jr, S. (2002). Analysis, design, and control of queueing systems. *Operations Research*, 50(1):197–216.

Tasgetiren, M. F., Pan, Q.-K., Suganthan, P., and Oner, A. (2013). A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion. *Applied Mathematical Modelling*, 37(10-11):6758–6779.

Umlauft, M., Schranz, M., and Elmenreich, W. (2022). SwarmFabSim: a simulation framework for bottom-up optimization in flexible job-shop scheduling using netlogo. In *Proc. 12th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, volume 1, pages 271–279. INSTICC, SciTePress.

Wang, L., Zhou, G., Xu, Y., and Liu, M. (2012). An enhanced pareto-based artificial bee colony algorithm for the multi-objective flexible job-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 60(9-12):1111–1123.

Wilensky, U. (1999). Netlogo. Webpage. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, http://ccl.northwestern.edu/netlogo/.

Xu, H., Bao, Z., and Zhang, T. (2017). Solving dual flexible job-shop scheduling problem using a bat algorithm. *Advances in Production Engineering & Management*, 12(1).

Yang, X.-S. (2010). A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)*, pages 65–74. Springer.

Yang, X.-S. and He, X. (2013). Bat algorithm: literature review and applications. *International Journal of Bio-inspired computation*, 5(3):141–149.

Yao, B. Z., Yang, C. Y., Hu, J. J., Yin, G. D., and Yu, B. (2010). An improved artificial bee colony algorithm for job shop problem. In *Applied Mechanics and Materials*, volume 26, pages 657–660. Trans Tech Publ.

Yurtkuran, A. and Emel, E. (2014). A modified artificial bee colony algorithm for-center problems. *The Scientific World Journal*, 2014.

Zhang, R., Song, S., and Wu, C. (2013). A hybrid artificial bee colony algorithm for the job shop scheduling problem. *International Journal of Production Economics*, 141(1):167–178.

Zhu, H., He, B., and Li, H. (2017). Modified bat algorithm for the multi-objective flexible job shop scheduling problem. *International Journal of Performability Engineering*, 13(7):999.

Zhuang, Z., Huang, Z., Lu, Z., Guo, L., Cao, Q., and Qin, W. (2019). An improved artificial bee colony algorithm for solving open shop scheduling problem with two sequence-dependent setup times. *Procedia CIRP*, 83:563–568.