Production and Maintenance Scheduling Supported by Genetic Algorithms

Duarte Alemão, Mafalda Parreira-Rocha, Jose Barata

Production and Maintenance Scheduling supported by Genetic Algorithms

Duarte Alemão[1], Mafalda Parreira-Rocha[1] and José Barata[1]

[1] UNINOVA, CTS, Faculdade de Ciências e Tecnologia,
NOVA University of Lisbon, Caparica, Portugal
{d.alemao, mafalda.rocha, jab}@uninova.pt

Abstract. The market demand has changed in recent years due to increased interest in more customized and diversified products by the consumers, leading to a change in production lines, which are becoming more flexible and dynamic. At the same time, the amount of data available in the factories is growing more and more, thereby the number of errors in the production schedule may occur often. Several approaches have been used over time to plan and schedule the shop-floor production. However, some only consider static environments, where the tasks are allocated to the machines, not considering that machines may not be available and sometimes maintenance interventions are needed. The introduction of maintenance increases the scheduling complexity and makes it harder to allocate the tasks efficiently. So, new solutions have been proposed, giving manufacturing systems the ability to quickly adapt to some disturbances that may occur. Thus, Artificial Intelligence approaches have been adopted to do the task allocation for the shop-floor. Those approaches can find suitable solutions faster than traditional approaches. This article proposes an architecture, based on Genetic Algorithm, capable of generating schedules including both production and maintenance tasks.

Keywords: Dynamic Job-Shop Scheduling, Maintenance Task Allocation, Genetic Algorithms, Manufacturing Systems.

1 Introduction

Nowadays market demands for more diversified and customized products are forcing factories to change from mass production standard systems to more dynamic and agile shop-floor systems. This change grants factories the ability to reconfigure and quickly adapt to market requirements. However, factories have several types of machines, each one with their own specific technical features, which can execute different tasks, located in different places of the factory, instead of a straightforward production line system. This fact makes the task allocation process more complex since the products do not follow all the same path anymore. Furthermore, it is most of the times a prerequisite from companies that machines should be kept working as much time as possible to maximize the production and consequently the income. Nevertheless, sometimes those machines need to be subject to maintenance operations, to keep them producing high-

quality products. To create a more reliable schedule, those maintenance operations should be defined along with the production operations. To cope with the presented difficulties this article proposes a solution based on Genetic Algorithm (GA), which aims to solve the task allocation problem in a job-shop, known as Job-Shop Scheduling Problem (JSSP). It includes both production and maintenance tasks in the same schedule with the objective of minimizing the total execution time in the shop-floor. Thus, a more reliable schedule of the factory is available, which should decrease the delays and the unexpected number of fails in the shop-floor. The architecture was designed to generate schedules based on the existing tasks and stations available on the shop-floor.

The proposed paper is organized as followed. Chapter 1 introduces an overview of the document. Subsequently in chapter 2 the concept of scheduling and the importance of maintenance task allocation in Agile Manufacturing Systems is depicted, followed by the solution description in chapter 3. Chapter 4 presents the systems execution environment, with a brief explanation of how it works. Finally, chapter 5 presents the solution analysis and the further developments.

2 Related Work

The manufacturing industry is subject to market requirements, which are constantly changing. This change is imposed by consumers' demand for more customizable and unique products and their variants. So, the manufacturing processes needed to change as well. To produce this wide variety of products demanded by many consumers, the factories need to adapt quickly to market requirements [1]. Thus, to provide a fast and efficient response to the market demand, manufacturing systems need to be agile, which is the ability to survive and prosper in competitive markets and environments of change [2]. In that way, it is important to have an efficient planning of the shop-floor. Namely, it is necessary to allocate products to machines in the shop-floor, as well as maintenance interventions. They are needed to keep machines working without causing defects on products, which are reflected in cost for the company. By scheduling maintenance operations, some unpredictable failures may be avoided, such as machine breakdowns.

Reflecting this, in Agile Manufacturing Systems (AMS) a large variety of machines with specific requisites need to stay functional as many time as possible to maximize the production. However, sometimes, they need to be the subjected to maintenance operations to deliver high-quality products. There are different maintenance types, which can be mainly divided into Corrective Maintenance (CM), Time-Based Maintenance (TBM) and Predictive Maintenance (PdM). CM is applied after a failure occurs, such as a machine breakdown, and tries to recover the resource to its functional state [3], [4]. TBM is applied to prevent, in advance, potential failures and it is taken while the system is still working, at predefined time intervals [3], [4]. Finally, PdM can be applied to prevent future failures when there is a deteriorating physical parameter, such as pressure or voltage, that can be measured instead of doing it in predefined intervals [5]. It infers the current state of a machine and predicts the future progression to estimate the time before a failure occur [6]. Based on that information it is possible to define an appropriate schedule for a production system, including maintenance tasks.

Scheduling is the process of time optimization, where the time of a set of tasks (jobs) performed on a product is assigned to the available time of a group of stations [7], [8]. It defines the sequence by each job is executed on a machine during a predefined time horizon, trying to avoid overlaps and to optimize some objectives [7]. The JSSP is a Non-deterministic Polynomial-time Hard (NP-Hard), which means they are quite difficult to solve using traditional optimization techniques [9]. Although mathematical optimization methods can achieve the optimal solution for small problems, for large-scale problems those optimization techniques are very limited [10]. Besides that, manufacturing systems are very dynamic, meaning new products frequently arriving, inconstant job durations and machines that are not always available, which represented the need for more dynamic approaches. Thus, many researchers have concentrated their efforts on heuristic approaches, such as Tabu Search, Simulated Annealing, Ant Colony Optimization, Particle Swarm Optimization, Neural Network and Genetic Algorithm (GA). The interest in GA has grown in the past years due to its high flexibility to adapt to different problems, to find near-optimal solutions and because it is easy to handle.

3 Dynamic Scheduling Solution

In this section, an overview of the scheduling tool architecture is presented and described. The proposed scheduling architecture is capable of generating schedules including both production and maintenance tasks, which are allocated to different production stations.

3.1 System Overview

The scheduling tool architecture was designed to generate a schedule to the shop-floor including production and maintenance tasks in a given time horizon. Those tasks are allocated to the available stations in empty time spaces, in order to get the minimum total execution time (makespan) within the established time horizon. The scheduling tool interacts with another tools as well as the shop-floor in order to acquire the necessary data to generate the schedules. Thus, the scheduling tool can access the information about the available stations and the tasks to allocate. The scheduling architecture is represented in **Fig. 1**.

The scheduling collects the information about the available stations in the shop-floor and the tasks to be allocated. Then, using a GA, it is possible to allocate those operations to each station, i.e. to set the start and end times of each task, in the empty spaces, avoiding the overlapping between tasks in the same station or tasks of the same product. After that, is possible to determine the execution time of each station and get the station with the higher makespan value, which represents the minimum makespan found in that process. The collection of schedules, containing a schedule for each station, is then stored.

The shop-floor provides the topology of the factory, i.e. the present stations and human operators to perform the maintenance tasks. The production tasks are provided by the management unit. It provides the information about each task and the time horizon

window in which the schedule should be performed. The maintenance section is responsible for providing the maintenance tasks, based on the operator's knowledge or by analysis methods, such as data mining. After generating the necessary schedules, those are sent to the shop-floor, so the operators know the schedule of each station, to the maintenance unit and to other tools for analysis and testing purposes.

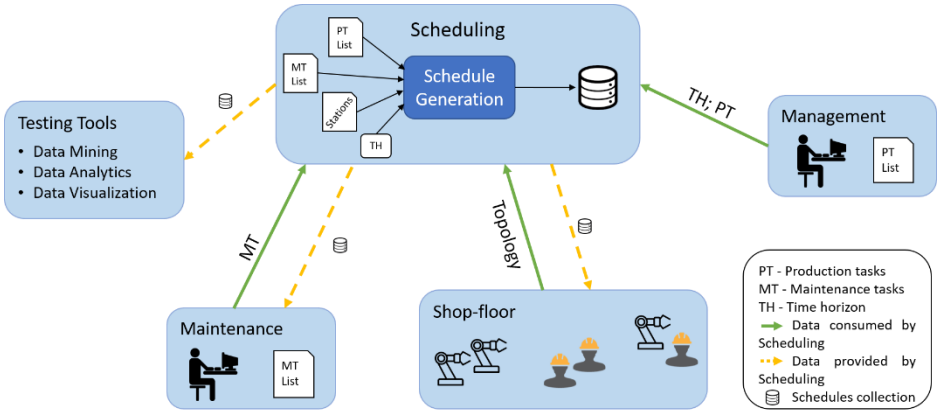

Fig. 1. System interactions overview

To generate a schedule, it is necessary to allocate the tasks efficiently to obtain a feasible and valid schedule, i.e. respecting some rules such as avoiding that a station is operating more than one task at the same time or preventing a product from being present in different machines at the same time. A GA architecture is proposed in this paper to solve this JSSP since it is a combinatorial problem.

3.2 Scheduling Algorithm

The scheduling tool was designed using GA. It is a stochastic searching adaptive approach to solve optimization problems based on natural selection mechanisms, first proposed by Holland [11]. GA allows to evolve a population of chromosomes (solutions) through a predefined number of generations and gets the best one that represents the best schedule found by the algorithm [10]. First, a random population is initialized based on the first chromosome created. Then, the fitness value of each individual is calculated. If a stop condition is reached, such as the predefined number of generations to evolve or a threshold fitness value, the process is finished, otherwise, it continues to the next generation. After that, the elite population, i.e. the best chromosomes present in the previous generation, is selected to be present in the current generation. Then, the selection process is done, and the selected individuals are matched with each other. Next, the mutation is applied in some individuals to provide some diversity in the population. Finally, the fitness value of each one is again calculated and the process is repeated till the termination criterion is reached [10], [12].

GA is known to be good solving optimization problems since it does not need to evaluate all the search space to extract a good solution. However, it not always can find an optimal solution but finds a near-optimal solution the most of the times [13].

The most important step in a GA is to define what is a chromosome. The representation of a chromosome influences the performance of the algorithm. In this problem, the chromosome represents the sequence of jobs assigned to each station, containing the times between which each task will be performed. It is a schedule solution for the entire shop-floor, where each gene represents a task to perform. Each task is composed of an ID, a duration, an associated product ID and an associated station ID. Assuming that Machine j, Mj only performs one type of tasks on a Product Pi. Each gene results from the merge of a product associated with a station, represented as PiMj, as demonstrated in the example of **Fig. 2**. The first operation of product 1 will be performed in machine 2, then the first operation of product 2 comes to be performed on machine 3, etc.

After obtaining the best solution, the chromosome is divided into several schedules, one for each station present in the chromosome.

P_1M_2	P_2M_3	P_1M_1	P_3M_1	…	P_2M_2

Fig. 2. Chromosome encoding

Then, it is necessary to define the maximum number of generations that the algorithm should evolve and the size of the population, i.e. the number of individuals present in each generation. The number of generations is the simplest way for terminating the evolution process. Low values may not be enough for the algorithm to evolve into good solutions, on the other hand high values may consume processing time unnecessarily because a good solution can be found too soon. At the same time, the population size should be large enough to have diversity among the individuals, but not too large that processing power and time are being consumed pointlessly. A small population could be very difficult to evolve into a good solution since the mating between individuals may not lead to a good offspring when the search space is too large. So, both parameters should be chosen cautiously to give the algorithm enough flexibility to evolve and do not consume time unnecessarily. These parameters depend on the size and complexity of each problem.

Another termination criterion is the steady fitness value. The steady fitness allows stopping the process if the fitness value remains the same for a given number of generations. This value needs to be smaller than the maximum number of generations. It helps when the evolution process is stagnant and does not evolve for a while.

Then, a small portion of the population, the survivor population, is selected to be present in the next generation, without being subject to crossover or mutation parameters. This is known as elitism and allows to keep the best individuals through the generations.

After that, the selection method for both the survivor and the offspring populations need to be chosen. In this case, was used the Roulette Wheel selector with size three, i.e. composed of three individuals. Firstly, is calculated the selection probability of each

of the three individuals, which is obtained by dividing the fitness value of one individual by the total sum of the fitness values of the three individuals: $(i) = \frac{f_i}{\sum_i^n f_i}$.
Then, the individual with higher fitness value has more chances to be chosen, so the new generation can be improved. This process is represented in **Fig. 3**.

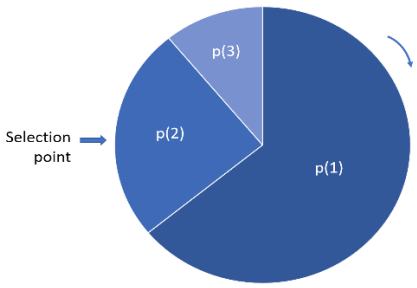

Fig. 3. Roulette wheel selection.

Next it is necessary to choose the crossover and mutation methods and their probabilities, which is the main factor to create new improved generations, since the characteristics of the parents are preserved to the children and the children can be improved. In this task allocation problem was chose a partially-matched crossover, which merges half of a parent chromosome with half of another parent and prevents duplicate genes in the child chromosome. The mutation is used to prevent the population to get stuck. The method selected was the Swap Mutation, which exchanges the positions of two genes with each other and does not duplicate genes. Both crossover and mutation probability values should be adapted to each situation since different values get better results in different situations.

Finally, it is necessary to define the fitness function. It evaluates and sets the fitness value of each chromosome. This value allows the evolution engine to select the offspring and survivor populations, as well as to know the best solution. In this task allocation problem, some rules were considered to define the fitness function:

- Only one operation of each job may be processed at a time;
- No pre-emption is allowed, which means that is not possible to interrupt a task and resume it later;
- Each job must be processed to completion;
- Jobs may be started at any time, no release times exist;
- Jobs may be finished at any time, no due dates exist (within the time window defined);
- No machine may process more than one operation at a time;
- Machine setup times are not considered;
- Machines may be idle within the schedule period;
- Machines are available at any time since they are considered available;
- An operation can only be performed once;
- Operations precedence within a product should be respected;

- Maintenance tasks must be executed during the maintenance shift.

The best fitness value is calculated by finding the minimum makespan among all generations, where the total makespan is given by the station with the maximum execution time in each generation. To obtain the fitness value of each individual, it is necessary to go through all the genes of each chromosome. The start and end times are set to the first job present in the chromosome, then to the second one and so on. At the same time, the makespan of each station is updated each time the times are assigned to a task.

The equation (1) describes the fitness function. The fitness value *f* is given by the minimum of the maximum completion time (makespan) of each generation times a weight which is increased if some task is not allocated in the right order.

$$f = Min[\, C_{max} * weight \,] \tag{1}$$

The total makespan of the schedule, C_{max}, is represented in equation (2). It is represented by the maximum total time to process all the operations, including idle time between operations, i.e. the end time of the last operation being executed.

$$C_{max} = Max[fs_j] \;\; , \;\; j = 1, \dots, n \tag{2}$$

The total execution time of each station j, fs_j, is set by the later final time of the operations allocated to that station, f_{ih}, calculated in (3) since the operation h is allocated to station j.

$$fs_j = \begin{cases} f_{ih}, \; fs_j < f_{ih} \,; h \in j \\ fs_j, \; otherwise \end{cases} \;\; , \;\; i = 1, \dots, n; \quad h = 1, \dots, m; \quad j = 1, \dots, k \tag{3}$$

The finishing time of each operation is calculated in (4), where the start time of operation *h* of product *i*, s_{ih}, is replaced by the start time of maintenance task *h*, sm_h, and the process time of operation *h* of product *i*, p_{ih}, is replaced by the process time of maintenance task h, pm_h, when the task involved is a maintenance task.

$$f_{ih} = s_{ih} + p_{ih} \;\; , \;\; i = 1, \dots, n; \quad h = 1, \dots, m \tag{4}$$

The start time of each operation depends if it is a production operation or a maintenance operation. In a production operation, constraint (5), it is calculated based on the previous operation of the corresponding product if any, otherwise starts at time zero. Though, if there is already another operation allocated in the same station, to that specific time slot, this one is altered to a new time, starting at the end of that task, *fjh*.

$$s_{ih} = \begin{cases} s_{ih-1} + p_{ih-1}, no\ overlapping\ h > 1 \\ 0, \; no\ overlapping; h = 1 \\ f_{jh-1}, \qquad\qquad overlapping \end{cases} \;\; , \;\; i = 1, \dots, n; \quad h = 1, \dots, m \tag{5}$$

On the other hand, if it is a maintenance operation, the start time of the operation is calculated based on the user intent. It could be desirable to have the maintenance tasks allocated as soon as possible, as late as possible or a third option where there are no pretensions about maintenance tasks allocation. However, the maintenance tasks should be executed only during the maintenance shift.

The maintenance task's start-time to "as soon as possible" scenario is calculated in (6). The production tasks are allocated only after the maintenance tasks. The Maintenance Shift Start time is represented by MSS, the Maintenance Shift End time is represented by MSE, DT represents the day the task is performed and 1440 minutes are the total minutes of one day. An operation start time is assigned and if a conflict occurs due to an overlay of operation that time is changed to the next slot within the maintenance shift time. If that maintenance shift is already full, it is allocated to the beginning of the maintenance shift of the day after and so on.

$$sm_h = \begin{cases} MSS + 1440 * DT, & there\ is\ overlapping;\ f_{ih} > MSE \\ s_{ih-1} + p_{ih-1}, & there\ is\ overlapping;\ f_{ih} \leq MSE \\ MSS, & there\ is\ no\ overlapping\ between\ tasks \end{cases} \quad (6)$$

To calculate the start time of a maintenance task "as late as possible", the constraint (7) is used. In this case the maintenance tasks are allocated after the production tasks. First, the task is allocated to the last shift maintenance slot of the time window. If there is overlapping with another task, it is allocated to the next slot in that shift. In the case that shift is already full, it is allocated to the previous day and so on. The time window, TW, represents the number of days reserved to the schedule.

$$sm_h = \begin{cases} MSS + 1440 * (DT - 2), & there\ is\ overlapping;\ f_{ih} > MSE \\ s_{ih-1} + p_{ih-1}, & there\ is\ overlapping;\ f_{ih} \leq MSE \\ MSS + 1440 * (TW - 1), & there\ is\ no\ overlapping\ between\ tasks \end{cases} \quad (7)$$

Finally, when there are no pretensions about the maintenance allocation, the maintenance and production tasks are randomly chosen, and the maintenance tasks are allocated as in the "as soon as possible case" (equation (6)).

To check the operations order, each product contains the information about all operations belonging to that product. If an operation is executed after another one with higher priority, then a parameter representing the weight of the errors occurred is incremented (multiplied by 10), leading to a very large fitness value when the solution is invalid. If the total makespan is, for example, 254, but there is one error, so the fitness value will be 2540. This way is possible to know the number of errors present in each generation.

Finally, it is obtained a solution containing a schedule for each station in the shop-floor, with the respective start and end times of each operation.

4 Execution Environment

Every existent station is obtained from the shop-floor and each task to allocate is obtained from the maintenance (if it is maintenance task) or the management (if it is production task). Each time a new schedule is requested, the previous data are stored in the scheduling tool, except the maintenance tasks which are stored when inserted in the system by a maintenance human operator. Each station stored in the scheduling tool contains a unique ID, the total execution time of that station (updated each time a new

task is assigned there) and a collection of skills which it can execute, each skill contains a collection of possible configurations containing the duration of that skill with that configuration. However, for simplicity, it is assumed that each skill only contains one configuration. In turn, each task to allocate is stored in the scheduling tool containing a unique ID, an associated skill ID, an associated product ID, an operation number which is used to order the operations in an ascending way per product, an associated station ID, a completion field to check if the task was already performed and the start and end times of the task which are inserted during the algorithm execution. When it comes a new task from a new product, it is also created the correspondent product, containing the ID of the product, a collection with all the tasks associated with that product and a collection with the precedent products, which need to be allocated before this one. This way, the information of each product can be accessed. The available maintenance teams are considered infinite. However, they only operate between 6 a.m. and 2 p.m. of each day, which means that maintenance tasks need to be allocated in that gap.

Then a new schedule generation can be triggered by a human operator, through a graphical user interface, or by the system itself at predefined times. When it happens, it generates a schedule which allocates the maintenance tasks as soon as possible, another that allocates the maintenance tasks as late as possible and another one which allocates them without preference. The scheduling algorithm starts by set the times for each task, then it checks if the operations order is correct, if it is not, reallocates the current task, and finally checks if there is no other task allocated to that station at the same time. At the end, the collection of schedules is sent to the other tools, to test the efficiency of the schedules. This process is represented in **Fig. 4**.

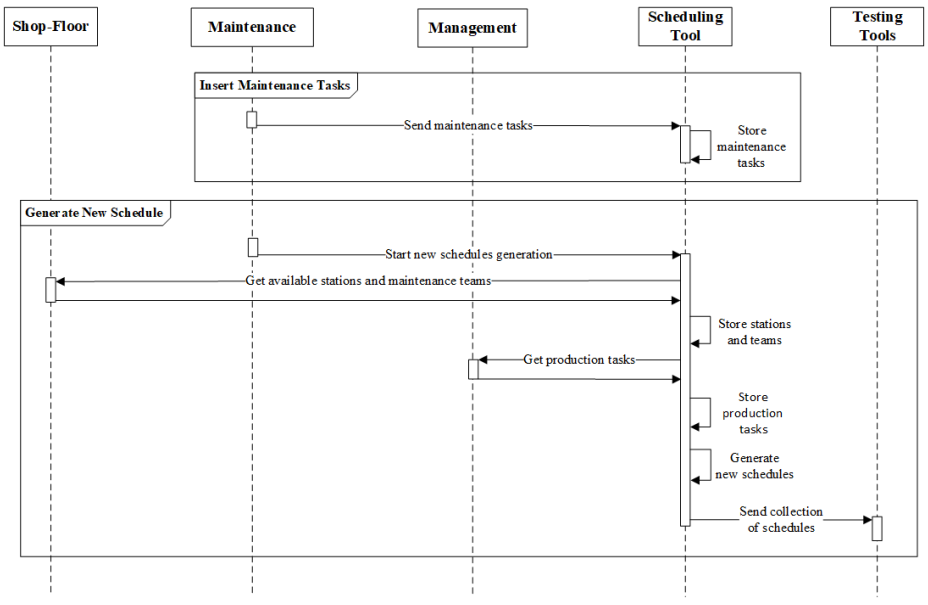

Fig. 4. Execution environment sequential diagram

5 Conclusions and Further Work

5.1 Conclusions

This paper presents a generic architecture designed to solve a JSSP including both production and maintenance tasks. Considering the maintenance operations to be performed on the production stations this architecture allows to have a more reliable schedule for the shop-floor. The presented solution is a GA based architecture, where the main goal is to perform the task allocation based on the information of the tasks and stations provided.

This way, this scheduling tool will try to allocate the tasks to the stations in order to minimize the total execution time of all the machines and respecting the order of the tasks of each product. Furthermore, the maintenance operations need to be allocated during the maintenance shifts times. Nevertheless, each operation needs to have the information about the station where it will be performed, since the developed architecture only sets the times of each one and does not choose on which machines they will be performed.

This tool was designed to avoid the overlapping of products on the same station at the same time, as well as to have the same product allocated to different machines at the same time, while the operations' sequence is preserved. After obtaining the schedule with the minimum makespan, this one will be divided into smaller schedules, one per station, and that collection can be sent to other tools, for higher level tests and analyses.

5.2 Further Work

In the future, the presented architecture will be improved in order to deal with the precedence between products, once some products may require that other ones are finished so they can start to be operated as well as date constraints, such as start and due dates of each product.

This tool will be implemented and tested in a real demonstrator using different problem sizes, i.e. with different numbers of tasks, products and stations used.

References

1. A. Azab and B. Naderi, "Modelling the Problem of Production Scheduling for Reconfigurable Manufacturing Systems," *Procedia CIRP*, vol. 33, pp. 76–80, 2015.
2. I. Chalfoun, K. Kouiss, A. L. Huyet, N. Bouton, and P. Ray, "Proposal for a generic model dedicated to Reconfigurable and Agile Manufacturing Systems (RAMS)," *Procedia CIRP*, vol. 7, pp. 485–490, 2013.
3. N. Aissani, B. Beldjilali, and D. Trentesaux, "Dynamic scheduling of maintenance tasks in the petroleum industry: A reinforcement approach," *Eng. Appl. Artif. Intell.*, vol. 22, no. 7, pp. 1089–1103, 2009.
4. R. Ruiz, J. Carlos García-Díaz, and C. Maroto, "Considering scheduling and preventive

maintenance in the flowshop sequencing problem," *Comput. Oper. Res.*, vol. 34, no. 11, pp. 3314–3330, 2007.

5. A. Raza and V. Ulansky, "Modelling of Predictive Maintenance for a Periodically Inspected System," *Procedia CIRP*, vol. 59, no. TESConf 2016, pp. 95–101, 2017.
6. A. Ladj, C. Varnier, and F. B. S. Tayeb, "IPro-GA: an integrated prognostic based GA for scheduling jobs and predictive maintenance in a single multifunctional machine," *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 1821–1826, 2016.
7. B. Do Chung and B. S. Kim, "A hybrid genetic algorithm with two-stage dispatching heuristic for a machine scheduling problem with step-deteriorating jobs and rate-modifying activities," *Comput. Ind. Eng.*, vol. 98, pp. 113–124, 2016.
8. L. Asadzadeh, "A local search genetic algorithm for the job shop scheduling problem with intelligent agents," *Comput. Ind. Eng.*, vol. 85, pp. 376–383, 2015.
9. P. Sharma and A. Jain, "Performance analysis of dispatching rules in a stochastic dynamic job shop manufacturing system with sequence-dependent setup times: Simulation approach," *CIRP J. Manuf. Sci. Technol.*, vol. 10, pp. 110–119, 2015.
10. S. Balin, "Non-identical parallel machine scheduling using genetic algorithm," *Expert Syst. Appl.*, vol. 38, no. 6, pp. 6814–6821, 2011.
11. J. Holland, *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press, 1975.
12. N. Kundakcı and O. Kulak, "Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem," 2016.
13. J. Li, Y. Huang, and X. Niu, "A branch population genetic algorithm for dual-resource constrained job shop scheduling problem," *Comput. Ind. Eng.*, vol. 103, p. 330, 2016.