JJMIE

Volume 20, Number 1, March. 2026
ISSN 1995-6665
Pages 71– 86
https://doi.org/10.59038/jjmie/200107
Editor-in-Chief: Prof. Faris M. AL-Oqla

Jordan Journal of Mechanical and Industrial Engineering

Towards Predictive and Reactive Job Shop Scheduling: A Hybrid Approach in Flexible Manufacturing Systems

Azrul Azwan Abdul Rahman[1*], Muhamad Alif Adam[1], Oluwamayowa Joshua Adeboye[1], Muhamad Arfauz A. Rahman[2]

[1]*Faculty of Industrial and Manufacturing Technology and Engineering (FTKIP), Universiti Teknikal Malaysia Melaka, Melaka, Malaysia.*
[2]*Queen's University Belfast, 123 Stranmillis Road, Ashby Building, BT9 5AH Belfast, United Kingdom*

Received 8 Jul 2025 *Accepted 5 Feb 2026*

Abstract

In the contemporary manufacturing landscape, unpredictable shop-floor disturbances and shortened product life cycles significantly challenge traditional scheduling methods, often leading to missed due dates and inefficient resource utilization. This study addresses these challenges by proposing a robust predictive–reactive scheduling framework designed for flexible manufacturing systems. The methodology utilizes a hybrid simulation–optimization approach, integrating discrete-event simulation to map real-time system states with a genetic-algorithm-based optimizer to identify high-performing scheduling policies. The framework is uniquely capable of dynamically generating and evaluating alternative dispatch sequences, enabling rapid recovery from operational disruptions such as equipment failure or order changes with minimal deviation from planned workflows. To enhance optimization efficiency, the model incorporates a permutation-based path generator and complexity filters that constrain the search space for the genetic algorithm. Case-study experiments conducted within a reconfigurable manufacturing context demonstrate that the proposed approach significantly improves performance, reducing makespan by up to 23% in standard scheduling scenarios and by up to 38% during rescheduling events, while simultaneously maintaining high system utilization. These findings underscore the effectiveness of integrating simulation with heuristic optimization to provide responsive, real-time control in complex production environments.

Keywords: Job Shop Scheduling; Simulation-Based Optimization; Flexible Production Systems; Genetic Algorithm, Predictive-Reactive Schedule.

1. Introduction

In today's dynamic market environment, businesses particularly in the manufacturing sector are under increasing pressure to enhance their flexibility. Manufacturing systems are expected to respond rapidly to demand variability, product innovations, and frequent order modifications [1]–[4]. As a result, flexible manufacturing systems (FMS) have garnered significant research attention in recent years. Notably, many of the algorithms, scheduling rules, and strategic approaches applied in this domain have their origins in earlier developments [5]. As of today, no framework or specialized technique has been established to achieve precise and rapid control of these systems. Conventional scheduling approaches often involve significant simplifications of real-world systems and tend to be complex in their mathematical formulations. Additionally, many existing algorithms struggle to deliver timely results due to the high computational load caused by numerous variables and constraints [6]–[8].

In contrast, simulation-based methods for scheduling and control offer substantial benefits. These include the ease of evaluating new layouts and schedules, the ability to facilitate direct production control, and their effectiveness as a foundational tool for scheduling and optimization tasks [9]–[12]. Thus, this work concentrates on creating a simulation model for a flexible manufacturing cell capable of accommodating variations in system layout, product mix, and adaptable routing and processing sequences. Building upon this model, a schedule optimization framework will also be developed.

Prior research has examined the integration of simulation and optimization algorithms for the scheduling and rescheduling of job shops in flexible manufacturing settings, considering various optimization restrictions and the imperative to fulfill customer requirements [13]–[16]. In this context, the predictive component is responsible for generating an initial feasible schedule tailored for a flow shop representing a flexible production system using predictive techniques as a foundation for further analysis. This schedule is produced through a hybrid approach that first employs an optimization algorithm to create a rough schedule, which is then refined through a rule-based simulation system that adjusts the plan locally to generate the final schedule. Once this predictive schedule is

* Corresponding author e-mail: azrulazwan@utem.edu.my.

deployed in a real-world manufacturing setting, the reactive component of the system takes over, adapting the schedule in response to disruptions or changes by generating alternative strategies. This reactive adaptation phase also relies on a hybrid of simulation and optimization methods [17].

All reviewed literature insists that analytical methods should always be preferred when they describe the problem sufficiently and accurately. However, because of complexity and number of constraints of FMS, it is not always possible. Contemporary researches investigate a wide range of solutions that could be applied. The current state of computing power and availability creates an opportunity for all those methods to be successfully implemented. Current work focuses on developing a framework, which should bear optimization with the use of genetic algorithm first, and later may allow other strategies to be included.

2. Simulation

Simulation is defined by the VDI (2010) as the representation of a system and its dynamic processes within a model possessing experimental capability, enabling knowledge transfer from the model to the real system. This definition highlights the fundamental strength of simulation-based scheduling approaches, particularly in complex manufacturing environments. Compared to traditional analytical and mathematical scheduling methods, simulation requires substantially less abstraction and simplification of real-world systems, allowing operational details to be modeled with greater fidelity [18]. Moreover, simulation models are inherently more adaptable; changes in system layout, control logic, or operating conditions can be implemented with relative ease through parameter and logic adjustments. In contrast, reformulating mathematical objective functions and constraints to reflect system changes is often labor-intensive and, in many cases, impractical without introducing simplifying assumptions that compromise realism [19].

The proposed framework adopts Discrete-Event Simulation (DES) using the simulation software, Tecnomatix plant simulation software, 2012 as its modeling paradigm. DES represents system evolution as a chronological sequence of discrete events such as job releases, processing completions, transportation activities, or machine failures each occurring at a specific instant and triggering a state change. This approach is particularly suitable for flexible manufacturing systems, where material flow is driven by individual work-in-progress units, machine availability, and routing decisions rather than continuous processes. By modeling these interactions explicitly, DES enables accurate representation of shop-floor behavior under both nominal and disturbed operating conditions.

Any scheduling activity inherently aims to achieve one or more operational targets, which may vary dynamically depending on production priorities and system conditions. Consequently, evaluating and comparing alternative schedules requires the definition of appropriate performance measures and objective functions. As noted in [20], top-level objectives in flexible manufacturing systems are often not explicitly stated; instead, they are decomposed into a set of interrelated sub-objectives. It therefore becomes the responsibility of decision-makers to select relevant objectives and determine suitable trade-offs to achieve overall system performance goals.

These objectives may frequently conflict with one another, leading to different scheduling outcomes. For instance, makespan is widely adopted in scheduling research due to its intuitive interpretation and ease of measurement [21]–[23]. However, strict adherence to due-date compliance can result in longer makespan values, illustrating the inherent trade-offs between competing objectives. To address such conflicts, objective functions may be reformulated for example, by incorporating penalty-based cost functions for lateness or by applying weighted combinations of multiple performance criteria [24].

A comprehensive review of the literature reveals a broad spectrum of possible scheduling objectives, many of which cannot be simultaneously implemented within a single simulation model without excessive complexity. Therefore, for model development and experimental evaluation, a focused subset of performance measures was selected. These include total completion time (makespan), average relative throughput time, average utilization of workstations, mean tardiness, maximum tardiness, and the number of tardy jobs [25]–[29]. Mean tardiness is defined as the average amount by which job completion times exceed their respective due dates, with early completions contributing a value of zero to the calculation. Maximum tardiness represents the largest single due-date violation, while the number of tardy jobs indicates how many jobs miss their due dates [30].

Within the proposed simulation–optimization framework, these performance measures are evaluated through discrete-event simulation runs, enabling realistic assessment of scheduling solutions under operational constraints and variability. This simulation-based evaluation forms the foundation for both predictive schedule generation and reactive rescheduling, ensuring that optimization decisions are grounded in accurate representations of system behavior rather than idealized analytical assumptions.

3. Genetic Algorithm

Among the various techniques explored, a significant portion of the research has been conducted using Genetic Algorithms (GAs) [31], [32]. As noted in [33], a Genetic Algorithm is a search and optimization method inspired by the principles of natural selection and genetics. It implies the evolving of population (i.e. set of solutions), composed of individuals (solutions) under specified selection rules through generations, in order to improve the fitness (i.e. minimize or maximize objective function). In current model, there is also restriction for objective function because GAwizard will not accept a negative value. Thus, in some cases, objectives should be transformed. That is, for example, [20] distinguishes mean tardiness, listed above, and mean lateness objectives. If a job finishes early, a negative value should be entered in the calculation of an average, and that may lead to a negative value of objective function. Although throughput time receives a lot of attention in literature, especially that related to lean

manufacturing, there is no clear or common formulation of function for this objective. The one presented in this work is used to widen the list of performance evaluation criteria. It is calculated as an average value for all jobs; and for every job, it is a ratio of throughput time during simulation run to a calculated value. Calculated value includes transportation and processing times and implies that there are no waiting times in front of a station or blockages of conveyors by other jobs. Thus, while calculated value is set to be 100%, the average relative throughput time will be always higher than that. In best schedules achieved with this model, this value varies from 115% to 125%.

Main parameters that GAwizard provides to user are generation size and number of generations. Important parameters like mutation and crossover are hidden in GA Optimization object. [33] provides a comprehensive review of many studies of GA. It is emphasized, that there is no single good set of parameters to use with GA and performance of set differ depending on objective function and constraints from problem to problem. Moreover, all parameters are interdependent. That is, different sets of mutation, crossover, population size and number of generations may show same results or a bug gap in

performance. Eventually, population size is recommended to be in range of 20 to 40. Parameters sets in which number of generations is higher than population size show better performance. Other settings that were not changed and evaluated in this model, however, also substantial influence on optimization results and speed.

While calculus-based methods have distinct output of optimization process, GA does not assure to find best solution, for example, global maximum or minimum of function. Thus, GA optimization process doesn't have a logical end and requires definition of certain point, at which process should be stopped. [33] suggest next criteria for termination, which is (1) Correct answer; (2) No improvement; (3) Statistics; (4) Number of iterations (After N number of generations). The best chromosome needed to check to ensure that the problem had an acceptable answer. The GA process should be terminated if the best solution (chromosome) remains unchanged over a predefined number of iterations. This may indicate that the algorithm has either found an optimal solution or has become trapped in a local minimum. However, in some cases, the optimal solution may not improve for a prolonged period until a beneficial crossover or mutation occurs. Another suitable stopping criterion involves monitoring the population's cost metrics such as the mean or standard deviation. If these values stabilize below a specific threshold, indicating minimal variation among individuals, it suggests convergence. If none of these conditions are met, a maximum number of iterations should be enforced to halt the algorithm.

Other sources also suggest a time limit for termination, which might be of great use when there is short time frame for decision-making. Number of generations is by default required by GAwizard. Correct answer (goal) is provided also by default, although it is hidden inside other object – GA Optimization, which is used by GAwizard, it is also considered as valuable criterion when upper or lower bound of the objective function is known (e.g. upper bound for stations utilization is 100%).Statistics-based termination is not implemented.

Table 1. Comparison of Optimization Techniques for Flexible Manufacturing Systems within this Framework

Optimization Method	Strengths	Weaknesses	Application in FMS
Genetic Algorithm (GA) *(Current Method)*	• Excellent at navigating large, complex search spaces (NP-hard problems) • Can be integrated with simulation to handle real-world variability[3]. • Inspired by natural selection, evolving solutions over generations[4].	• No guarantee of finding the global optimum; may get trapped in local minima • Performance is highly sensitive to parameter settings (mutation, crossover) • Requires clear termination criteria as it has no logical end	Complex scheduling where many variables (sequence, allocation, paths) change simultaneously
Priority Dispatching Rules (e.g., SPT, EDD)	• Extremely fast and easy to implement • Computationally "cheap" for immediate decision-making.	• Often result in sub-optimal schedules compared to GA • Effectiveness depends heavily on buffer availability; less impactful in systems without intermediate buffers.	Rapid, "good enough" scheduling when there is no time for optimization.
Calculus-Based / Analytical Methods	• Can provide exact, distinct optimal outputs • Highly reliable if the problem is mathematically well-defined	• Often require extreme simplification of the real system • Struggles with the high computational load of many variables/constraints • Difficult to update when the real system changes[17].	Highly structured systems with low variability where a pure mathematical formulation is possible
Simulated Annealing (SA)	• Good at escaping local optima by allowing "worse" moves early in the process. • Generally, requires less memory than GA.	• Typically, slower to converge than GA for very large populations. • Hard to tune the "cooling schedule" parameters.	Fine-tuning a single specific sequence rather than global policy optimization.
Ant Colony Optimization (ACO)	• Strong at finding optimal paths and routing in dynamic networks. • Naturally adapts to changes in the system layout.	• Can take a long time to converge in high-dimensional scheduling problems. • Risk of "stagnation" where all ants follow the same sub-optimal path.	Routing-heavy FMS where the physical path on conveyors is the primary constraint.

As an attempt to widen the range of optimization, possibilities to choose from was made. Due to time limit, however, only GA and dispatching rules were implemented. The list of implemented priority dispatching rules chosen from [20] are included (1) EDD reorder; (2) FCFS (First Come First Served); (3) SPT (Shortest Processing Time); (4) LPT (Longest Processing Time); (5) FOPR (Fewest Operations Remaining); (6) MOPR (Most Operations Remaining); (7) SRPT (Shortest Remaining Processing Time); (8) LRPT (Longest Remaining Processing Time). However, these works function in a different way. Priority dispatching rules are usually used in job shops, where job may be stored in buffer between operations. Thus, at the point of time when station becomes available next job is chosen from a waiting list. Developed model, in contrary, does not have this buffer between stations. Job has to follow process sequence and leave the conveyor after finish [34], [35]. Hence, these rules are only used to reorder release list and do not have significant impact on scheduling outcome.

While exact methods (e.g., Mixed-Integer Linear Programming) provide guaranteed optimality, they are often computationally prohibitive for the dynamic nature of the shop floor. Conversely, while Priority Dispatching Rules (PDR) offer speed, they lack the global perspective necessary for long-term efficiency.

The selection of a Genetic Algorithm for this framework is predicated on its ability to navigate the "solution landscape" of the JSSP effectively. By mimicking biological evolution utilizing crossover, mutation, and selection the GA explores diverse scheduling permutations simultaneously. Although the literature acknowledges that GA may converge on a local optimum, its integration with Discrete-Event Simulation (DES) in this study mitigates this risk.

The simulation acts as a high-fidelity fitness evaluator, ensuring that the evolved GA parameters are grounded in the physical realities and constraints of the manufacturing environment, thereby providing a robust balance between computational speed and schedule quality. For proper understanding, Table 1 tabulates the Comparison of Optimization Techniques for Flexible Manufacturing Systems within this Framework while Table 2 shows the comparative analysis of optimization methodologies for manufacturing scheduling.

To enhance clarity and reproducibility, Figure 1 illustrates the structure of the GA employed in the proposed simulation–optimization framework, including its key parameters and interaction with the discrete-event simulation model. The GA operates as an iterative search process that evolves a population of candidates scheduling solutions toward improved performance with respect to predefined objectives. Each individual in the population represents a feasible scheduling solution encoded as a chromosome. The chromosome structure comprises decision variables relevant to the flexible manufacturing environment, including the sequence of job releases, alternative process sequences (where flexibility is permitted), workstation allocation for each operation, and routing paths within the material-handling system. This encoding allows the GA to explore a wide solution space while preserving feasibility with respect to technological and routing constraints.

At the beginning of the optimization process, an initial population is generated and evaluated. Standard genetic operators' selection, crossover, and mutation are then applied to evolve the population across successive generations. Selection favors individuals with superior fitness values, while crossover and mutation introduce diversity and enable exploration of new scheduling configurations. The population size and number of generations are defined by the user, while mutation and crossover mechanisms are handled internally by the GA optimization module.

Fitness evaluation constitutes the core of the optimization loop and is performed using the discrete-event simulation model. Each candidate solution is executed within the simulation environment to compute performance indicators such as makespan, workstation utilization, throughput time, and tardiness. By embedding simulation within the fitness evaluation process, the GA assesses solutions under realistic system dynamics, resource constraints, and operational variability rather than relying on simplified analytical approximations.

The optimization process continues until one or more termination criteria are satisfied. These criteria include reaching a predefined number of generations, achieving a target fitness value, or observing no further improvement over a specified number of iterations. The final output of the GA is an optimized scheduling policy that can be directly applied within the predictive phase or used as a basis for reactive rescheduling under dynamic shop-floor conditions.

Table 2. Comparative Analysis of Optimization Methodologies for Manufacturing Scheduling.

Method	Characteristic Strength	Critical Limitation	Applicability to Proposed Framework
Genetic Algorithm (GA)	Global search capability; robust performance in multi-dimensional, non-linear search spaces.	Stochastic nature; results represent "near-optimal" rather than absolute optima.	**High:** Ideal for evolving scheduling policies that balance makespan and machine utilization.
Priority Dispatching Rules (PDR)	Low computational overhead; instantaneous decision-making.	Myopic; often yields sub-optimal sequences in high-variability environments.	**Low:** Insufficient for the "Predictive" phase of the proposed framework.
Exact Mathematical Programming (MILP)	Guarantees global optimality for defined constraints.	Intractable for large-scale industrial problems due to "combinatorial explosion."	**Low:** Computationally too slow for real-time reactive scheduling.
Simulated Annealing (SA)	Strong local search and avoidance of local minima via "cooling" schedules.	Slower convergence rates compared to population-based heuristics in complex JSSP.	**Moderate:** Useful for fine-tuning but lacks the parallel exploration of GA.

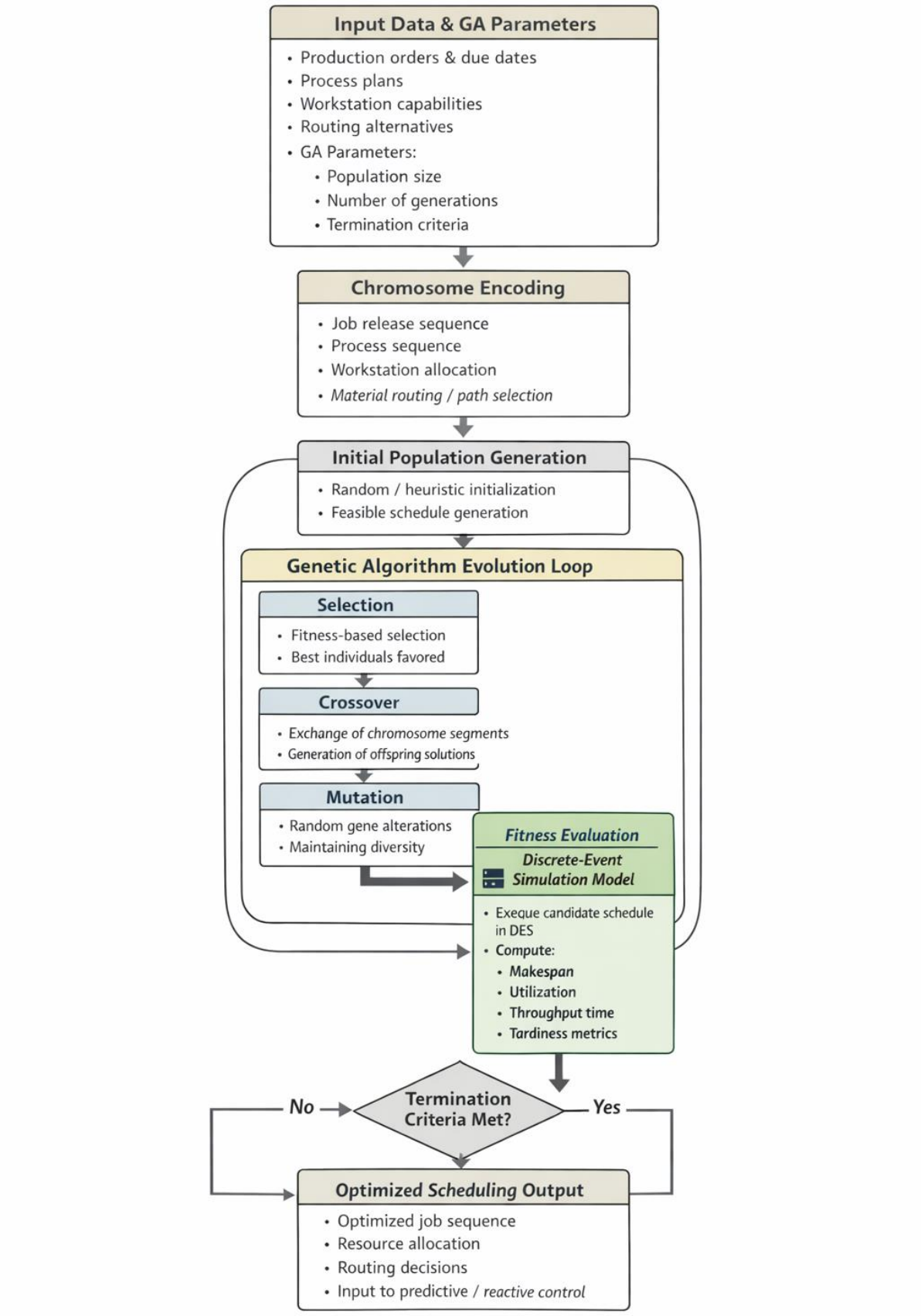

Figure 1. Structure of the genetic algorithm–based optimization framework and its interaction with the discrete-event simulation model

4. Case Study

4.1. Task Description

Real system that was modeled consists of conveyor lines and workstations as shown in Figure 2. The system was controlled by simulation model, thus introduction of optimization possibilities using the same simulation model was desired. Materials were moved using conveyors systems and placed into buffers located near processing stations for subsequent operations.

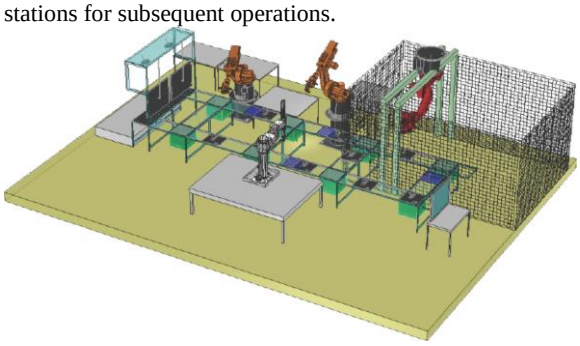

Figure 2. Real system model

After processing is done, product will be picked up by a carrier and transported further. Flexibility in process sequence was preserved. Same processes might be provided by different workplaces, and products might stay at workplace for different processes. It was required to obtain a framework that would be able to generate an optimal schedule with respect to different objectives and system settings. To improve clarity and reproducibility, Table 3 summarizes the key inputs, assumptions, constraints, and outputs of the proposed simulation–optimization model."

4.2. Input Data

The model required following data as an input for simulation:

Process plan. Sequence or list of required processes for every product type.

Workplaces (stations). List of processes that each workplace provides, with processing time for every product. If two workplaces provide the same process, processing time for a product may differ.

The order. That is, the number of products of each type and due dates if necessary.

While table 4 shows the Process plan, table 5 showed how data for stations was entered and also represented the reference input data. Processing times were provided for each product if product required that process. All available on-station processes were entered as index for columns. Process names were exactly matched those in process plan. Several stations could provide same process, and processing time might differ. Processing time entered was seconds. In reference data, proc_1 was available on two stations – H1 and H2, thus product that had this process in its process plan could be forwarded to either of them. Proc_5 was also available on two stations, H3 and H4. Number of processes on stations was not limited in any way.

Table 3: Summary of Framework Parameters and Operational Logic

Category	Description and Parameters
Model Inputs	Process Plan: The specific sequence or list of required processes for every product type. Workplace Data: A list of available stations, the processes they provide, and product-specific processing times. Order Requirements: The total quantity of each product type and their respective due dates. System Layout: Configuration of conveyor lines, stoppers, and workstation locations. Real-time Status (Reactive): Station downtime schedules and the current location/progress of Work-in-Progress (WIP) units.
Assumptions	Discrete-Event Nature: The system is modeled as a sequence of discrete events (e.g., job arrivals, process completions). WIP Location: During rescheduling, in-process products are assumed to be located in buffers immediately preceding stations. Shortest Path Default: For units already in progress during a disruption, the framework assumes they will follow the shortest path to completion and are not subject to re-optimization.
Constraints	Non-Negative Objective: The Genetic Algorithm (GA) objective function must be non-negative to be accepted by the GAwizard. Buffer Capacity: Source control mechanisms restrict job release based on the available capacity of destination buffers to prevent system blockages. Sequence Logic: Jobs must adhere to a defined process sequence and cannot bypass required operations. Optimization Complexity: The number of process steps per product is limited to nine to maintain computational efficiency.
Model Outputs	Optimized Schedule: Recommended release sequences and assigned workstation allocations. Performance Metrics: Quantitative data on makespan, station utilization, and mean/maximum tardiness. Operational Logs: Detailed timestamps for processing starts/stops and carrier transportation paths for real-system control.

Table 4. Process Plan

Name	Sequence	First Process	Second Process	Third Process	Forth Process	Fifth Process
T1	True	Pr1	Pr3	Pr4		
T2	True	Pr2	Pr5	Pr1	Pr4	
T3	True	Pr5	Pr2	Pr4	Pr3	
T4	True	Pr3	Pr4	Pr5	Pr1	Pr2
T5	True	Pr2	Pr4	Pr3	Pr1	
T6	False	Pr2	Pr3	Pr4		
T7	False	Pr1	Pr4	Pr5		
T8	False	Pr2	Pr4			
T9	True	Pr5	Pr3	Pr1		
T10	True	Pr2	Pr3	Pr5		

Table 5. Workplaces (Stations) Processing times

Workplace H1		
MU_Type\Process	Pr1	
T1	1200	
T2	1500	
T3		
T4	180	
T5	1830	
T6		
T7	600	
T8		
T9	1080	
T10		

Workplace H2		
MU_Type\Process	Pr1	Pr2
T1	1200	
T2	1500	900
T3		660
T4	180	120
T5	1830	1200
T6		660
T7	600	
T8		840
T9	1080	
T10		2100

Workplace H3		
MU_Type\Process	Pr3	Pr5
T1	1800	
T2		240
T3	510	420
T4	300	210
T5	915	
T6	720	
T7		600
T8		
T9	480	1200
T10	660	600

Workplace H4		
MU_Type\Process	Pr4	Pr5
T1	420	
T2	600	240
T3	735	420
T4	135	210
T5	495	
T6	1200	
T7	900	600
T8	1500	
T9		1200
T10		600

Above was data required for the advance scheduling when there were no in-process units located inside system and all stations were available, which is called Normal in UI. Second scenario – Rescheduling, required additional input for simulation and optimization.

Approximate downtime of stations was entered in Table 7. One row represented a one start of event, which means each station might be entered several times. Table 8 contained list of products that were already inside system. For each instance of product, name, serial number (WorkOrderNo), due date and location were provided. As a location only buffered before stations entered, possibility to use other locations was not provided. nPr was the consecutive number of next process that product requires, and Process Sequence was used to define the sequence which product follows.

For products that must follow one specified sequence, it should be 123..., and those products that required processing without strict succession may be for instance 231. For all in process products, the shortest path was chosen at the start of simulation, and these products were not affected by optimization process.

Table 6. Order

Name	Qty	DueDate
T1	5	01.05.13 13:10
T2	10	01.05.13 11:35
T3	12	02.05.13 10:55
T4	4	01.05.13 14:55
T5	8	02.05.13 16:15
T6	10	01.05.13 11:40
T7	3	01.05.13 9:40
T8	6	01.05.13 20:40
T9	15	02.05.13 11:40
T10	2	01.05.13 18:40

Table 7. Workplaces Downtime

Workplace	Start	Duration
S1	01.05.13 8:30	1:30:00.0000
S2	01.05.13 8:15	1:00:00.0000

Table 8. Work-in-progress products

Work OrderNo	Name	DueDate	Location	nPr	Process Sequence
5	T1	01.05.13 8:10	S3_Buffer	2	123
6	T2	01.05.13 8:25	S1_Buffer	3	1234
7	T3	01.05.13 8:55	S2_Buffer	2	1234
8	T4	01.05.13 8:45	S4_Buffer	2	12345
9	T5	01.05.13 8:20	S2_Buffer	1	1234
10	T2	01.05.13 8:30	S1_Buffer	3	1234

4.3. Typical Scheduling Sequence

The model had a certain degree of flexibility in regard to layout as illustrated in figure 2, that user can influence without manual change in the internal controls and methods by changing or adding line and moving or changing number of stations. The settings of model had to be entered in the dialog window before the simulation started, such as the speed of lines. Some settings had influence on scheduling, hence needed to change to achieve desired result from scheduling procedure.

Stage 2 involved identifying a schedule that met the defined requirements. When the schedule needs fast and there is no time for optimization, it might be a good idea to run several simulations with different dispatching rules and settings of the model to check if due dates or other objectives are being met. Otherwise, optimization with genetic algorithm is required. Though, depending on complexity of order, a good schedule also could be achieved very fast. Once an appropriate schedule was generated, the simulation was executed with logging enabled to collect the data necessary for both real system control and performance evaluation. These logs included detailed records of processing start and stop times at each station, along with all transportation activities critical for managing carrier movement. Additional outputs captured the product sequences and assigned paths, reflecting the processing order and the stations involved.

During the system's control phase, rescheduling could be initiated in response to events such as equipment failure, scheduled maintenance, unexpected changes in orders, or the impending completion of the current schedule. In such cases, the reactive loop served to update the model with the system's current status and reinitiate the scheduling process using the new input inputs. In addition to standard data provided to the model, the following input should be transferred, which were start and approximate downtime of stations, in-process products that are currently located in buffers before stations with their respective progress of processing plan, list of remaining products or a new products order. GA optimization searched for a good schedule by changing: a sequence of products; sequence of processes (if sequences are not fixed) for each product; allocation (if required process is available on several stations); path on lines to follow.

A case when order contained a number of products that had earliest start date would be considered for a rescheduling procedure. Only those products that were available for immediate release should be in list for scheduling to reduce the optimization complexity.

4.4. Model Description

Figure 3 showed the layout of developed model. The lines, stoppers and stations were positioned to mimic the real model. All methods within the model could be categorized into two main groups. The first group consisted of methods responsible for controlling the model during the simulation execution. The second group included methods used to set up the model and organize input data prior to the simulation run.

4.4.1. Control methods

These methods were responsible for controlling a behaviour of objects during simulation run. Main methods were included:

- Move. It was divided in several sections depending on where was it called from: from the end of each line by carrier, which was separated by carrier's status – pick-up tour, transportation and return to buffer; at stations exit by product; exit from buffer before station by product. Carrier was being moved according to own PathsTable_Carriers when empty, and used path of product when was loaded.
- Line Stopper. This source code was used to halt the line at a sensor when the subsequent section was occupied and unable to accept an incoming carrier. When there was conjunction of several lines (intersection) and carriers were located at each line,carriers were released in order they had been stopped at sensor. User-defined attribute Wait_Prio was used for this purpose. The first carrier that had been stopped would have a higher priority, and would be moved first, as soon as next line would be able to receive it.
- Load and Unload. These methods were used to load carriers at the system's entry point with units released by the source, and to unload carriers at the exit.
- Call_Carrier. Was called when unit started processing at station. It calculated time that was required to get from Carriers_Buffer to station (theoretically, there was no possibility to assess the lines current workload and possible blocks on the way), and with regard to processing time send the carrier in advance. If there won't be any blocks on lines, it will arrive exactly when processing of product is finished.

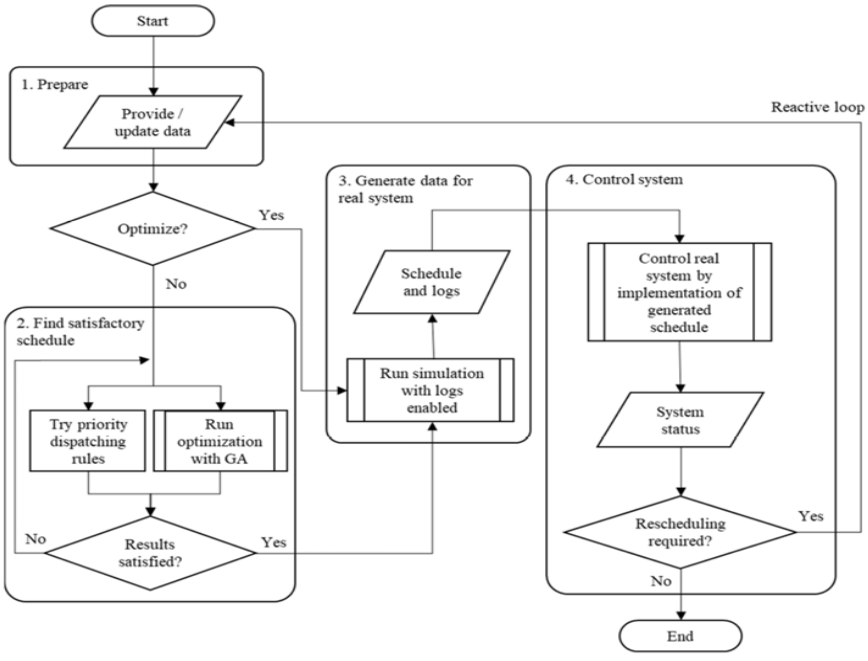

Figure 2. Scheduling problem and Control process flow

- Trace. Used to track data to calculate objectives and data that was required for source control.
- Logging. Filled two tables: TransportLog and ProcessingLog. These tables contained data related to start, duration, activity and paths that were required to control the real system.
- Pr_time. A simple function that was called by station in order to get processing time, with regard to current product that was going to be processed and the process required.

4.4.2. Model initialization methods

These methods were responsible for processing the initial input data, generating paths, and performing other setup tasks, as outlined below.

- Path_Generator. Explained in 4.5.3. Similar as Path_Generator_Carriers, which created predefined paths for empty carriers. Though it defined only shortest paths from entrance line to each station, and from each station to exit line, which made only eight paths for a reference layout.
- Create_Prod_List. This method converted the initial input data into structured tables necessary for model control during the simulation.
- FindShortestPath. This method identified and selected the shortest available path from a predefined set of possible routes.

4.4.3. Auxiliary methods

There were number of methods that were hidden inside objects and serves a specific function, for example, Callback inside UI dialog that makes the dialog work and stores all settings in hidden table. There was also method that created observers in stations, for example, GA_Termination folder contained methods that were responsible for optional termination triggers, and Reschedule, Objectives and Dispatching_Rules folders with respective methods.

4.4.4. Generations of paths

a) Permutation Method

All potential process sequences were derived using permutation. It was required when product doesn't have to strictly follow the process plan. To support a better schedule, an alternative sequence could be assigned to each instance of a product.

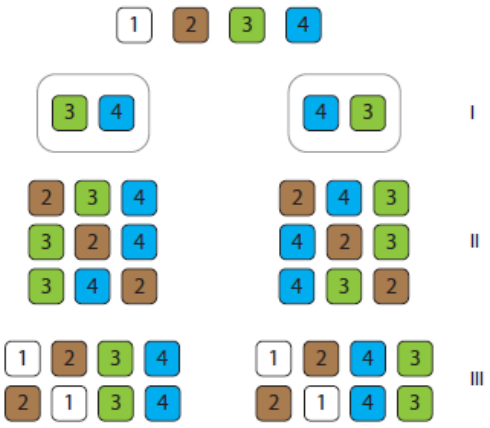

Figure 4. Permutation algorithm

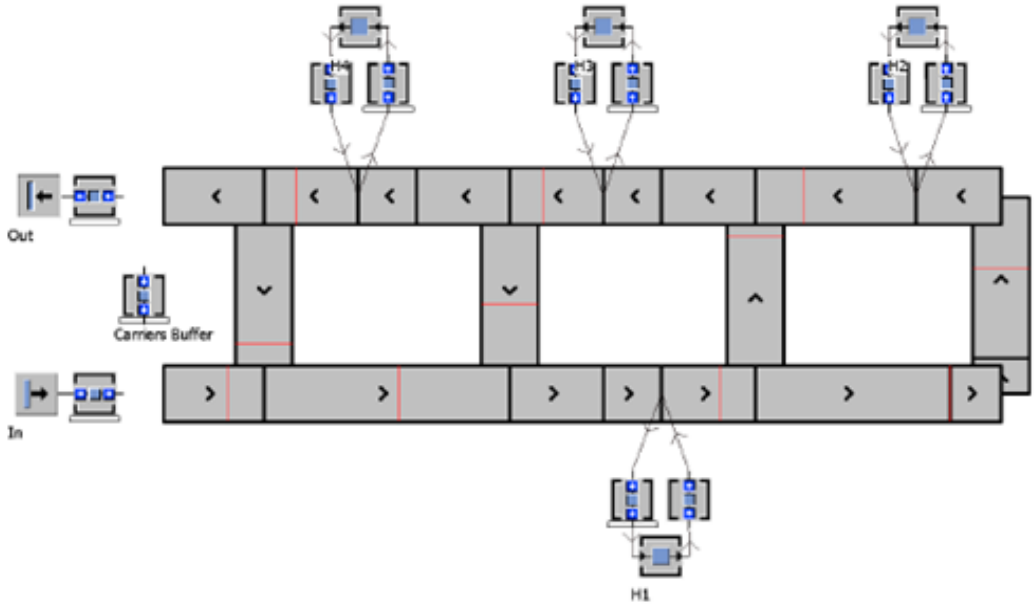

Figure 3. Simulation Model Layout

Figure 4 illustrates the algorithm used for generating permutations. The algorithm was formulated to accept a character string (e.g., "1234") and provide a comprehensive list of all potential permutations. It began by selecting the last character in this case, "4" and then inserted the preceding character "3" into every possible position relative to "4," resulting in two new strings: "34" and "43." This process continued by inserting the next character into each existing string at all valid positions. For example, inserting "2" into "34" produced the strings "234," "324," and "342." The procedure was repeated until all characters were included in each string. Ultimately, the number of generated permutations was N!, which for the example with four characters resulted in 24 unique permutations.

The method itself supports unlimited number of characters in string. Time required to perform all permutations varies from instant for 5-6 characters, to slightly more than two seconds for ten characters. The code provided in this model was written from scratch, with limited programming experience and no time to find a more optimal structure. Although it was very simple, it had two nested loops and thus became very inefficient with growing number of characters. Developed model, however, supports maximum nine process steps and therefore performance of this method should be sufficient. The input string to the method in model consisted of digits from 1 to a number of processes for product, and represented a sequence in which product followed the process plan. Each digit represented a process number in initial process plan. From reference process plan in Table 1, input string for product T6 will be 123, which corresponds to a process sequence Pr2 -> Pr3 -> Pr4. Then, one of permuted strings, for instance 213, corresponds to a process sequence Pr3 -> Pr2 -> Pr4.

b) Complexity level

These filters were used to limit the number of paths that are generated by Path_Generator. It is an attempt to force GA to find a better schedule when time available for optimization is limited. The objective was to eliminate longer paths, so constraining the range of possibilities that the GA explores. In Level 3, all possible paths were generated. A full set of paths implied all sequences of processes with respective sequences of workplaces (stations), and all possible routes to follow for each sequence of workplaces. Level 2 was an optional filter in method, which excluded long ways between workplaces, while Level 1 was an optional filter which left only one shortest path for every sequence of processes.

c) Path generator structure and method

The path generator method was the most important method in the model. Plant simulation offered convenient approach to change the layout and settings of the model manually, changing most of settings of stations and combinations of lines. It also provided different objects to control the flow of products and model behavior. For the optimization, manual input of these settings and establishing the controls would be very time-consuming. GA in particular required as set of parameters that it might change in order to improve the fitness function. In developed model, these settings included sequence of products being released, sequence of processes for every product, at which station each process be provided and the path on conveyor lines to follow. Path generator method created a set of possible paths for every product type. Every path consisted of sequence of objects such as lines, buffers and stations, and implied process plan and allocation. The GA then selected a path for each product instance, combined the products in various sequence arrangements, and executed simulations to evaluate their corresponding objective function values.

Figure 5 illustrates the structure of this method. As shown, a sequence of processes was constructed for each product type in the process plan using the permutation method. Subsequently, for each process sequence, a corresponding sequence of stations was determined. This was done by checking every station table to find out if station provided the process or not. At this point, consistency of input data was very important. If process name that was taken from process sequence did not exactly match the name of process provided by station, station would be omitted. If none of the stations provided the required process, then error would appear. Subsequent to the generation of all potential station sequences, pathways were explored. The method traced the routes from one station to the subsequent station until all potential pathways were identified. If complexity filter of level 2 was enabled, a method FindShortestPath is called, which chooses the shortest path segment. Consequently, these path segments were being combined into a complete path.

d) Source control

Source control options limited the number of units that were being released by source during simulation run to avoid blocks of the system. There were three strategies to choose from, and one default that was being evaluated always. The default one prevented release of unit if number of allowed units inside the system had been reached. The first optional control checked the number of units that were located in destination buffer plus those that were located on lines and at the moment were heading to the destination buffer, and compared this number with buffer's capacity. Destination buffer is the buffer before station, to which a unit located in source will go first.

The second optional control counted the number of units that would have to enter the destination buffer during the rest of their process plan, among units that were currently located in system. If this number reached the capacity of destination buffer, source would not release next unit. This number would be recalculated when station after destination buffer would either finish or start processing. The third control checked the rest of process plan of every unit currently located in system, and for every buffer calculated the total number of units that entered. If capacity of any buffer was exceeded, unit would not be released. Control methods would wait for changes in system and evaluate the state of the system again. These changes were programmed by wait until statements, which suspended methods until release of an empty carrier, and then until this carrier was loaded or one unit (product) would leave the system.

Each of these options limited the number of units inside the system in ascending order from default to the third one, which in turn might lead to idling of several stations and thus to not optimal schedule. However, locks of the system were very dependent from product mix, therefore control strategy was chosen by trial and error. An option to use buffers after each station significantly reduced the probability of lock and reduced the required level of source control limitation.

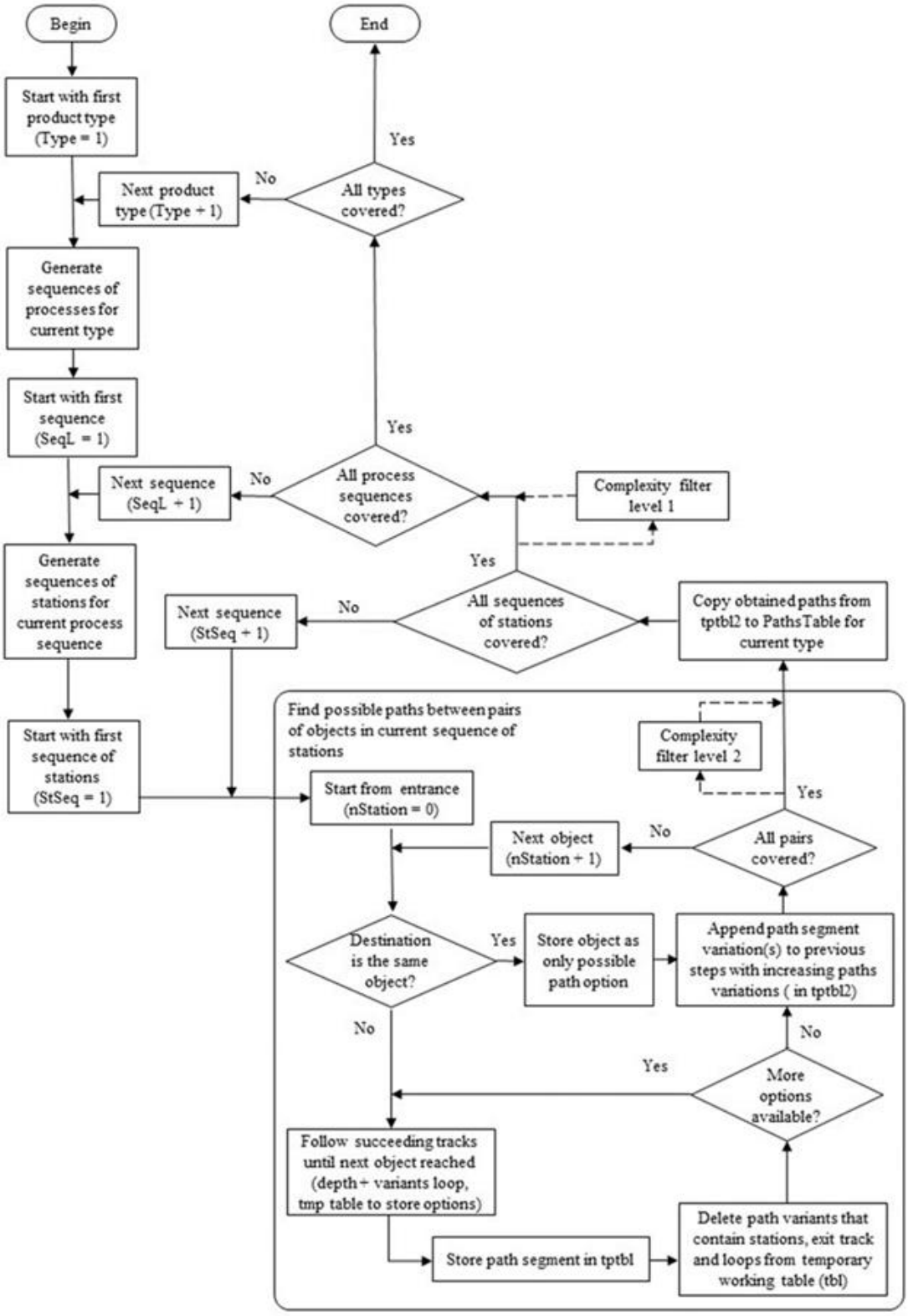

Figure 5. Path generator method structure

e) GA setup and control

Plant Simulation provided ready-to-use objects to use genetic algorithm (GA) for optimization. To facilitate the utilization of GA, a wizard known as GAwizard has been introduced. This object serves as a repository for several methods and objects employed in managing the GA optimization process. Because of this fact, there were some difficulties in transferring settings. Wizard was also being opened with internal method, which responds to OnSelectControll of wizard icon, thus command openDialog could not have a desired impact. Most of setting were transferred with method SetGA, which passes parameters that were to be optimized, number of generations and generation size. Inconvenience, however, arises after SetGA was called because GAWizard resets all settings. When wizard was opened and new settings were applied (e.g. optimization direction, generation size…) dialog did not display new settings. For some reason, whenever wizard was closed and opened again it resets internal object GAOptimization, which provided the possibility to terminate optimization as soon as goal had been reached, and this option would not be active anymore. In addition to SetGA, there was custom method GAonOpen_Custom that was stored inside UI. When new settings were applied, GAWizard was closed, setGA was called, and after that this custom method called method that opened the dialog, and reapplied option for Goal termination again. In this way wizard displayed new settings, termination option was re-enabled and distributed simulation (if was used before) activated.

Penalty value for fitness function was used to assign bad fitness for simulation run if system was blocked. Fitness was calculated whenever EventController stops. It means if system was blocked and not all units have left the system EventController stops because there were no upcoming events, in which case calculated fitness was wrong.

f) Graphical user interface - dialog

To streamline model configuration and control, a unified graphical user interface (GUI) was developed comprising four functional tabs: Settings, Data, Scheduling, and Evaluation. The Settings tab centralizes core system parameters such as maximum concurrent material units, number of carriers, intersection transfer times, and process-sequence enforcement and provides visualization controls (e.g., icon swapping for carriers and products) as well as release-control and path-complexity options. The Data tab offers direct access to key input tables (workplaces, process plans, orders, in-process units, failed stations, and rescheduling orders), enabling on-the-fly edits and rapid data review. Within the Scheduling tab, users select and tune dispatching rules and genetic-algorithm parameters (generation size, population count, termination criteria), while optional trace-table logging can be toggled to balance insight against performance. Finally, the Evaluation tab displays the results of simulation runs against all defined objectives, with custom “plausibility” and observer-driven update routines ensuring that table views remain synchronized with underlying data after user edits or external imports. Collectively, this interface empowers users to configure, execute, and assess predictive–reactive scheduling experiments without manual scripting, thereby enhancing usability and reproducibility in complex manufacturing scenarios.

5. Results and Discussion

5.1. Performance Across Multiple Processes

GAWizard framework enabled the execution of optimization across multiple parallel streams, thereby reducing the overall optimization time. This might be realized by utilization of processing power of computers on network and multi-core CPUs, which are mainstream nowadays.

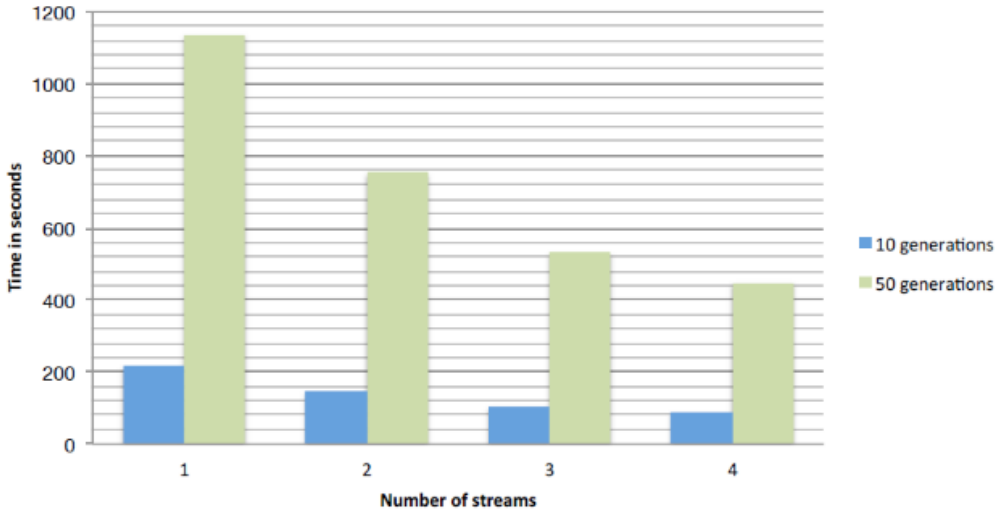

Figure 6. Multi-process performance

All the optimization runs were performed on computer with Intel i5-2400 processor, which is quad-core. All simulations and optimization runs were performed for the reference scheduling order provided.

Figure 6 illustrates the improvement in performance when running optimization using single versus multiple streams. The time required with four parallel streams was approximately 2.5 times shorter compared to the single-stream execution. Used CPU utilized turbo-boost technology, which increased frequency of cores depending on load. This might be the main cause of non-linearity of performance improvement, and the rate is considerably less than 4.

5.2. Release and Control options comparison

A series of optimization experiments were conducted to evaluate the effect of various parameter settings on the resulting schedules. Figure 7 shows the results. Names of bars contained different choices for pickup (CB11) and control option of the source. First digit represented the pickup on the way to buffer and second pick up after unload.

These options were specifically developed to avoid blocks of system. As a result, first second and the last combinations showed very similar performance. As anticipated, release control option 3 and 2 were overly restrictive, resulting in weak schedules. All other combinations of these settings caused systems blockages.

5.3. Improvement rate

A series of optimization runs was carried out with the number of generations fixed at 20, while varying the generation size to determine whether the range of 30 to 40 individuals per generation yields optimal performance. Figure 8 illustrates the observed trend: as the generation size increased, the rate of improvement decreased. This rate was measured as the ratio of fitness improvement to the time consumed, indicating diminishing returns in performance gains with larger generation sizes.

$$\text{rate} = (\text{initialfitness} - \text{optimized}) / \text{timeconsumed} \quad (1)$$

While optimization runs constrained by time or a limited number of generations demonstrated better performance with smaller generation sizes, extended runs revealed the opposite trend larger generation sizes yielded improved results over time.

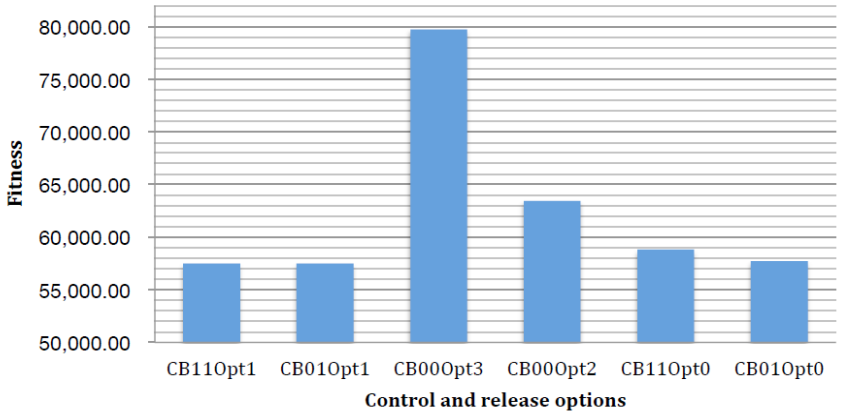

Figure 7. Best schedules comparison

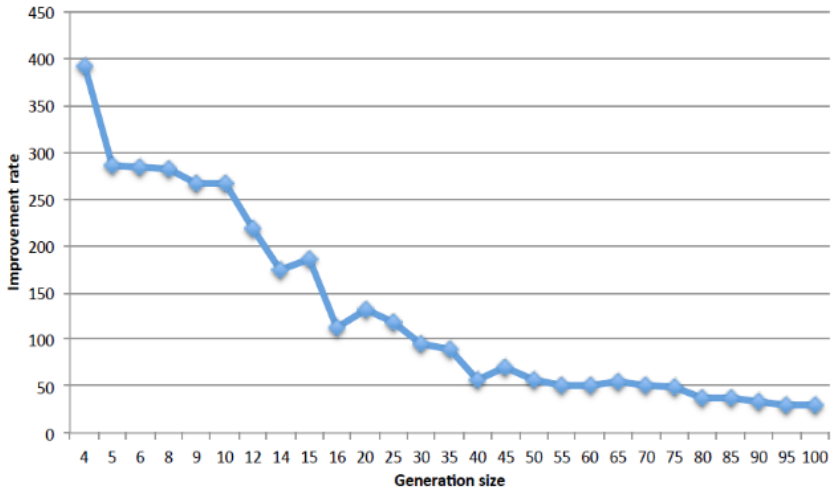

Figure 8. Improvement rate in 20 generations

  - Volume 20, Number 1 (ISSN 1995-6665)

5.4. Complexity levels

Figure 9 showed a tested layout with increased complexity. In addition to reference layout objects, there were more lines, and two more stations that provided processes: S5 – Pr2, Pr3 and Pr5; and S6 – Pr3 and Pr4. Processing times were the same as original stations had. With a lot more options and variants, model did work as it should, and optimization was working just as for original layout.

The modification of layout was simple and fast, thus proving possibility for easy system change and testing. Figure 10 showed comparison of performance, when optimization had been run for 10 minutes, with all the same settings but different complexity level filters. In extended optimization runs with 50 generations, Level 2 demonstrated slightly improved performance, achieving approximately 30% better fitness (goal: minimization) compared to Level 1. For searching of best schedule in long optimization run, however, level 3 was most likely to outperform level 2.

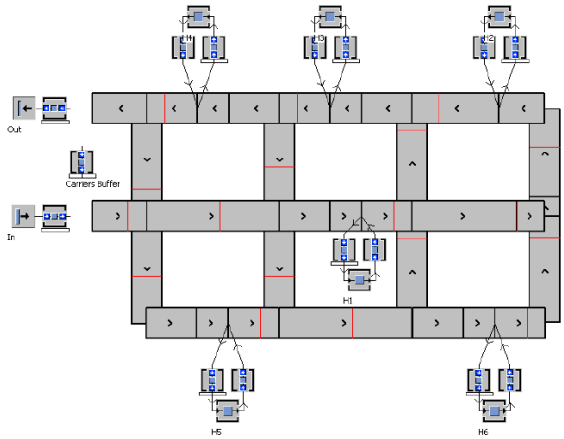

Figure 9. Increase complexity layout

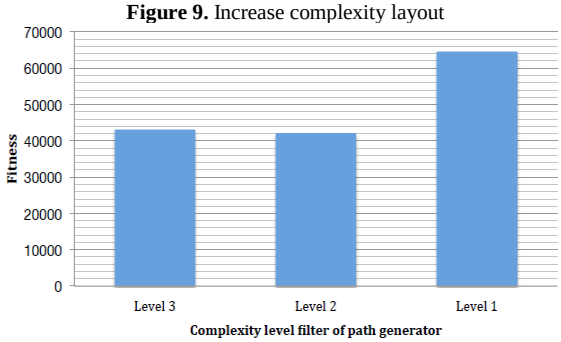

Figure 10. Performance of complexity level filters

While Path_Generator's performance was sufficient for reference layout, it turned out to be inappropriate for more complex systems. Figure 11 showed an example of path generator "fail".

With current design of method, tended to produce all possible paths, it produced huge amounts of "ridiculous" paths as well. Given that shown example was only the path segment between two stations and there were other useless paths, for a complete sequence of stations the paths collection would be filled with options that should in no way be included into optimization. Moreover, Path_Generator consumed enormous amount of time with increasing layout complexity. As a temporary solution, complexity level 2 should be used. It left only the shortest segments for every pair of stations, thus eliminating unwanted looping and roaming paths. It also consumed much less time, 9 seconds for level 2 vs. 15 minutes for levels 1 and 3.

Complete redesign was absolutely necessary. At least two levels of improvement for Path_Generator should be provided. The first was developing a criterion that would permit several paths, including those longer ones, and deny all extremely inefficient. It would not affect the way Path_Generator worked for reference layout but would be crucial for achievement of planned flexibility regarding system layout. Secondly, it was simple but still important tuning structure for performance.

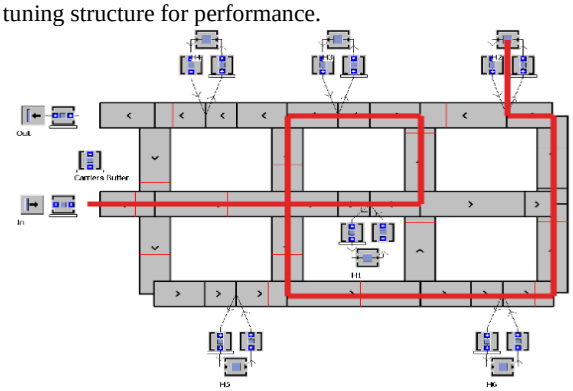

Figure 11. Path_Generator fail

That was, the core of generator, which followed the connected objects and generates path segments between stations, should be moved to the beginning of method. Paths for all possible pairs of stations and in/out lines should be generated first and stored ina data table. And then only all paths' variants should be combined from these segments. It is to avoid procedures that are repeated for every product type, sequence, etc. It should be done once.

6. Conclusion

The results obtained from the proposed simulation–optimization framework demonstrate the effectiveness of integrating discrete-event simulation with a genetic algorithm for scheduling in flexible manufacturing systems. The observed improvements across key performance measures particularly makespan, workstation utilization, and tardiness-related metrics indicate that the proposed approach is capable of generating high-quality schedules while accounting for realistic system constraints and dynamic interactions.

The reduction in makespan achieved by the optimized schedules reflects the ability of the genetic algorithm to effectively explore alternative sequencing and routing configurations under the guidance of simulation-based fitness evaluation. Unlike traditional analytical approaches that rely on simplified assumptions, the use of discrete-event simulation enables a more accurate representation of material flow, workstation interactions, and transportation delays. As a result, the optimized schedules are not only computationally efficient but also operationally feasible, which is essential for practical deployment in real manufacturing environments.

The results confirm that embedding simulation within the optimization loop allows the genetic algorithm to implicitly account for congestion effects and resource contention that are often neglected in purely mathematical scheduling models.

The best genetic algorithm solutions across generations further illustrate the stability and convergence behavior of the optimization process. The relatively low variability observed in the performance measures indicates consistent solution quality as the algorithm progresses, suggesting that the combination of genetic operators, simulation-based fitness evaluation, and complexity filtering effectively guides the search toward near-optimal regions of the solution space. This stability is particularly important for industrial applications, where predictable system behavior is often preferred over marginal performance gains.

From a practical perspective, the predictive–reactive capability of the proposed framework represents a significant advantage over static scheduling approaches. By updating the simulation model with the current shop-floor state and re-evaluating alternative scheduling strategies under disturbance scenarios, the framework supports informed rescheduling decisions that minimize performance degradation. This aligns well with the operational realities of flexible manufacturing systems, where unexpected events such as machine breakdowns or order changes are unavoidable.

Overall, the discussion confirms that the proposed DES–GA framework addresses key limitations identified in existing scheduling literature. By combining high-fidelity simulation with evolutionary optimization in a unified predictive–reactive architecture, the framework bridges the gap between theoretical scheduling models and practical shop-floor execution. The results therefore not only validate the effectiveness of the proposed approach but also highlight its potential applicability in real-world flexible and reconfigurable manufacturing environments.

7. Acknowledgement

The authors would like to thank Universiti Teknikal Malaysia Melaka (UTeM) for the support and opportunity to conduct this research study.

References

[1] Y. C. Choi, and P. Xirouchakis, “A holistic production planning approach in a reconfigurable manufacturing system with energy consumption and environmental effects”, International Journal of Computer Integrated Manufacturing, Vol. 28, No. 4, 2015, pp. 379–394. https://doi.org/10.1080/0951192X.2014.902106

[2] J. Leng, Y. Zhong, Z. Lin, K. Xu, D. Mourtzis, X. Zhou, P. Zheng, Q. Liu, J. L. Zhao, and W. Shen, “Towards resilience in Industry 5.0: A decentralized autonomous manufacturing paradigm”, Journal of Manufacturing Systems, vol. 71, 2023, pp. 95–114. https://doi.org/10.1016/j.jmsy.2023.08.023

[3] M. Khakifirooz, M. Fathi, A. Dolgui, and P. M. Pardalos, “Scheduling in industrial environment toward future: Insights from Jean-Marie Proth”, International Journal of Production Research, vol. 62, no. 1–2, 2024, pp. 291–317. https://doi.org/10.1080/00207543.2023.2245919

[4] R. K. Singh and S. Modgil, “Adapting to disruption: The impact of agility, absorptive capacity and ambidexterity on supply chain resilience”, International Journal of Productivity and Performance Management, vol. 74, no. 2, 2025, pp. 637–658. https://doi.org/10.1108/IJPPM-01-2024-0057

[5] D. Y. Lee and F. DiCesare, “Integrated scheduling of flexible manufacturing systems employing automated guided vehicles”, IEEE Transactions on Industrial Electronics, vol. 41, no. 6, 1994, pp. 602–610. https://doi.org/10.1109/41.334577

[6] T. Nehzati, “Research outline on reconfigurable manufacturing system production scheduling employing fuzzy logic”, International Journal of Information and Electronics Engineering, Jan. 2012. https://doi.org/10.7763/IJIEE.2012.V2.214

[7] M. Niehues, P. Sellmaier, T. Steinhaeusser, and G. Reinhart, “Adaptive job-shop control using resource accounts”, Procedia CIRP, vol. 57, 2016, pp. 351–356. https://doi.org/10.1016/j.procir.2016.11.061

[8] M. M. Nasiri, R. Yazdanparast, and F. Jolai, “A simulation optimisation approach for real-time scheduling in an open shop environment using a composite dispatching rule”, International Journal of Computer Integrated Manufacturing, vol. 30, no. 12, 2017, pp. 1239–1252. https://doi.org/10.1080/0951192X.2017.1307452

[9] G. Lugaresi and A. Matta, “Generation and tuning of discrete event simulation models for manufacturing applications”, 2020 Winter Simulation Conference (WSC), IEEE, Dec. 2020, pp. 2707–2718.

[10] M. Parente, G. Figueira, P. Amorim, and A. Marques, “Production scheduling in the context of Industry 4.0: Review and trends”, International Journal of Production Research, vol. 58, no. 17, 2020, pp. 5401–5431. https://doi.org/10.1080/00207543.2020.1741390

[11] E. Mahmoodi, M. Fathi, M. Tavana, M. Ghobakhloo, and A. H. Ng, “Data-driven simulation-based decision support system for resource allocation in Industry 4.0 and smart manufacturing”, Journal of Manufacturing Systems, vol. 72, 2024, pp. 287–307. https://doi.org/10.1016/j.jmsy.2023.11.019

[12] M. Schlecht, R. de Guio, and J. Köbler, “Automated generation of simulation model in context of Industry 4.0”, International Journal of Modelling and Simulation, vol. 45, no. 2, 2025, pp. 441–453. https://doi.org/10.1080/02286203.2023.2206075

[13] H. Fei, Q. Li, and D. Sun, “A Survey of Recent Research on Optimization Models and Algorithms for Operations Management from the Process View”,Scientific Programming, Vol. 2017, No. 1, 2017. https://doi.org/10.1155/2017/7219656

[14] Z. Wang, W. Liao, and Y. Zhang, “Rescheduling Optimisation of Sustainable Multi-Objective Fuzzy Flexible Job Shop under Uncertain Environment”, International Journal of Production Research, vol. 62, no. 24, 2024, pp. 8904–8920. https://doi.org/10.1080/00207543.2024.2354830

[15] Y. Fu, K. Gao, L. Wang, M. Huang, Y. C. Liang, and H. Dong, “Scheduling Stochastic Distributed Flexible Job Shops Using a Multi-Objective Evolutionary Algorithm with Simulation Evaluation”, International Journal of Production Research, vol. 63, no. 1, 2025, pp. 86–103. https://doi.org/10.1080/00207543.2024.2356628

[16] W. Javaid and S. Ullah, “Data Driven Simulation Based Optimization Model for Job-Shop Production Planning and Scheduling: An Application in a Digital Twin Shop Floor”, Journal of Simulation, 2025, pp. 1–15. https://doi.org/10.1080/17477778.2025.2469687

[17] M. Niehues, M. Blum, U. Teschemacher, and G. Reinhart, “Adaptive job shop control based on permanent order sequencing: Balancing between knowledge-based control and complete rescheduling”, Production Engineering, vol. 12, no. 1, 2018, pp. 65–71. https://doi.org/10.1016/j.procir.2015.06.024

[18] F. Klügl, Agent-Based Simulation Engineering, Habilitation thesis, TU Munich, 2010.

[19] K. E. Stecke, "Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems", Management Science, vol. 29, no. 3, 1983, pp. 273–288. https://doi.org/10.1287/mnsc.29.3.273

[20] Zeestraten MJ. Scheduling flexible manufacturing systems. Doctoral Thesis, Technische Universiteit Delft; 1989.

[21] K. C. Udaiyakumar and M. Chandrasekaran, "Application of firefly algorithm in job shop scheduling problem for minimization of makespan", Procedia Engineering, vol. 97, 2014, pp. 1798–1807. https://doi.org/10.1016/j.proeng.2014.12.333

[22] A. N. H. Zaied, M. M. Ismail, and S. S. Mohamed, "Permutation flow shop scheduling problem with makespan criterion: literature review",Journal of Theoretical and Applied Information Technology, vol. 99, no. 4, 2021, pp. 830–848.

[23] A. A. Rahman, O. J. Adeboye, J. Y. Tan, M. R. Salleh, and M. A. A. Rahman, "An optimization approach for predictive–reactive job shop scheduling of reconfigurable manufacturing systems", Jordan Journal of Mechanical and Industrial Engineering, vol. 16, no. 5, 2022, pp. 793–809.

[24] X. Hao and L. Lin, "Job shop rescheduling by using multi-objective genetic algorithm", 40th International Conference on Computers and Industrial Engineering: Soft Computing Techniques for Advanced Manufacturing and Service Systems, IEEE, 2010.

[25] H. H. Zhang, Y. N. Zeng, and L. Bian, "Simulating multi-objective spatial optimization allocation of land use based on the integration of multi-agent system and genetic algorithm", International Journal of Environmental Research, vol. 4, no. 4, 2010, pp. 765–776. 10.22059/ijer.2010.263

[26] L. Sahoo, A. K. Bhunia, and P. K. Kapur, "Genetic algorithm based multi-objective reliability optimization in interval environment", Computers and Industrial Engineering, vol. 62, no. 1, 2012, pp. 152–160. https://doi.org/10.1016/j.cie.2011.09.003

[27] I. Harjunkoski, C. T. Maravelias, P. Bongers, P. M. Castro, S. Engell, I. E. Grossmann, J. Hooker, C. Méndez, G. Sand, and J. Wassick, "Scope for industrial applications of production scheduling models and solution methods", Computers & Chemical Engineering, vol. 62, 2014, pp. 161–193. https://doi.org/10.1016/j.compchemeng.2013.12.001

[28] K. Z. Gao, P. N. Suganthan, Q. K. Pan, T. J. Chua, T. X. Cai, and C. S. Chong, "Discrete harmony search algorithm for flexible job shop scheduling problem with multiple objectives", Journal of Intelligent Manufacturing, vol. 27, 2016, pp. 363–374. https://doi.org/10.1007/s10845-014-0869-8

[29] M. H. Seyyedi, A. M. F. Saghih, and Z. N. Azimi, "A fuzzy mathematical model for multi-objective flexible job-shop scheduling problem with new job insertion and earliness/tardiness penalty", International Journal of Industrial Engineering, vol. 28, no. 3, 2021. https://doi.org/10.23055/ijietap.2021.28.3.7445

[30] C. E. Liddick, "NOAA operations scheduling", in 28th Aerospace Sciences Meeting, 1990, pp. 1–18.

[31] N. Bhatt, and N.R. Chauhan, "Genetic algorithm applications on Job Shop Scheduling Problem",International Conference on Soft Computing Techniques and Implementations, IEEE, 2015.

[32] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future", Multimedia Tools and Applications, vol. 80, 2021, pp. 8091–8126. https://doi.org/10.1007/S11042-020-10139-6

[33] R. L. Haupt and S. E. Haupt. Practical Genetic Algorithms, In: Studies in Computational Intelligence, vol. 18, Springer, 2006.

[34] A. Rajabinasab and S. Mansour, "Dynamic flexible job shop scheduling with alternative process plans: An agent-based approach", International Journal of Advanced Manufacturing Technology, vol. 54, nos. 9–12, 2011, pp. 1091–1107. https://doi.org/10.1007/S00170-010-2986-7

[35] M. Fera, F. Fruggiero, A. Lambiase, G. Martino, and M. Elena, "Production scheduling approaches for operations management", in Operations Management, InTech, 2013 https://doi.org/10.5772/56193