A SIMULATION-OPTIMIZATION APPROACH FOR THE ROBUST JOB SHOP SCHEDULING PROBLEM WITH CONDITION-BASED MAINTENANCE AND RANDOM BREAKDOWNS

by

Md Hasan Ali

Submitted in partial fulfillment of the requirements
for the degree of Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
April 2022

Table of Contents

List of Tables

List of Figures

Abstract

The integration of scheduling and maintenance activities is important in shop floor decision-making. Job shop scheduling with maintenance activities requires proper planning due to its complex nature. A simulation-optimization approach is proposed in this thesis to solve the Job Shop Scheduling Problem (JSSP) under both planned and unplanned machine unavailability, considered based on Condition-Based Maintenance (CBM) and random breakdowns, respectively. Furthermore, the proposed approach accounts for uncertainty in the machine degradation rate and the duration of CBM and Corrective Maintenance (CM) activities, and aims to generate robust schedules that have good performance both on average and under adverse scenarios.

The objective function of the original problem is first approximated using multiple surrogate functions that are independently optimized using Genetic Algorithm (GA). The first surrogate function considers the actual jobs only, the second one adds up deterministic CBM, the third one adds up deterministic breakdowns happening at the *Mean-Time-To-Failure* (MTTF), and the remaining surrogate functions consider jobs with deterministic CBM and different breakdown scenarios. These random breakdown scenarios are determined by solving a diversity maximization problem such that the breakdown time differences between the surrogate functions are maximized. The approximated and optimized schedule is then evaluated through simulation with the original objective function considering stochastic degradation of machines, random breakdowns, and uncertain CBM and CM duration. A weighted average of the expected makespan and its 90th percentile is used as the objective function to ensure schedule robustness. To prevent a premature conversion, a stopping rule motivated by Simulated Annealing (SA) is employed in the outer loop of the proposed approach.

The proposed approach performed well with a maximum average improvement of 8.68% for 15-job and 15-machine and a minimum average improvement of 5.46% for 50-job and 15-machine over the initial solution with an average run time of 1.88 hours and 6.01 hours, respectively. Numerical experimentation demonstrated that the proposed approach can effectively generate high-quality schedules while considering CBM, random breakdowns, and parameter uncertainty.

Acknowledgements

I would like to start by thanking my parents who motivated me in all the steps of this work. This work is dedicated to my parents.

I owe my heartfelt gratitude to Dr. Ahmed Saif for his invaluable instruction, mentoring, encouragement, constructive criticism, and inspiration throughout this thesis. During this hard time, your compassion, understanding, and perseverance inspired me to keep confident and move forward.

Finally, I would like to thank my family, friends, and colleagues, whose names are not included here but who kept me in their minds.

Chapter 1

Introduction

Competitiveness exists everywhere, and to stay competitive, one must accomplish tasks on schedule. To achieve trust and maintain a good relationship with clients in the business sector, a company should deliver its order within the time-frame to stay competitive. Failure to deliver an order within the time-frame or to fulfill a customer's demand within the deadline leads to a penalty which can be incurred in many forms, such as monetary loss, loss of goodwill, or both. To avoid penalties and logistics congestion, a company should adopt proper sequencing and scheduling. Proper sequencing and scheduling can deliver a product or service or a project on time.

In general concept, scheduling refers to the process of organizing, managing, and optimizing work and workloads. Typically, scheduling includes two issues: one is the allocation of resources and the other one is the determination of sequences. To be more specific on the shop floor, scheduling refers to the allocation of operations to the machines that can perform these operations, and sequencing refers to the determination of the sequences of the operations to the machines. On the shop floor, a common assumption is that machines are available all the time. But in reality, machines are prone to deterioration or degradation, which leads to random failure during the usage period. Moreover, to get the maximum availability of machines on the production floor, machines should undergo maintenance.

Maintenance encompasses a combination of the managerial, technical, and administrative decisions and the set of actions that are necessary for retaining or restoring a product or asset functionality [1]. Maintenance activities have a substantial impact on a firm's key performance indicators, including cost, reliability and product quality. In many industries, costs related to maintenance (or lack of) constitute a significant portion of the total production cost. Bevilacqua and Braglia [2] estimate that maintenance activities costs can attain 15-70% of total production costs. In other aspects,

reliability of machinery and production systems in industries have consequential effect on industries profitability as well as competitiveness in the business sector. Economic losses of an industry for the loss of production due to unexpected shutdowns and malfunctions were estimated at 7% of an industry turnover [3], which can be more subject to the industry types. This highlights the importance of maintenance strategies of machines and production systems in the industry. It is necessary to have a proper maintenance plan that can maintain the Overall Equipment Effectiveness (OEE) at its peak level [4].

Maintenance activities aim at increasing the availability of machines by decreasing the probability of failure, i.e., increasing reliability, and reducing the restoration time of failed machines. One of the planned maintenance strategies is Preventive Maintenance (PM), which has two main approaches, namely Time-based Maintenance (TBM) and Condition-Based Maintenance (CBM). In TBM, scheduled or periodic PM activities, including lubrication, calibration, repair, adjustment, refurbishment, and replacement, are carried out at predetermined times or intervals to procrastinate the degradation processes. A key assumption of TBM is that the degradation behaviour of machines or systems is predictable during normal usage based on statistics or experience, which enables maintenance activities to be planned as a function of time only [5]. However, it is difficult to estimate machine degradation solely based on time, i.e., if loading conditions vary widely, the TBM approach might become ineffective in reducing devastating machine failures. Moreover, it can lead to unnecessary maintenance actions that might increase the cost. Considering the uncertain nature, it is not too easy to properly plan maintenance actions beforehand.

To alleviate this situation, CBM works by monitoring the actual condition of the machines and recommends maintenance actions only when their behaviour seems unusual. Therefore, a CBM program, if properly planned and effectively executed, can not only increase machine availability but also reduce unnecessary PM costs. The fundamental assumption of CBM is that it is possible to detect the prognostic parameters which can lead to the determination or approximation of the possible failure of machines or equipment before it happens. In context to machines, degradation or deterioration and ageing are the common prognostic parameters.

One of the simplest presumptions in scheduling problems is the availability of machines throughout the scheduling horizon. However, in reality, machines are exposed (or prone) to planned and stochastic unavailable periods. Planned unavailability may be due to, e.g., periodic inspection, lubricating, adjustments, tool modification, refurbishment with fixed and beforehand known starting time and duration. Due to the uncertain nature of the machine degradation rates, it is challenging to activate the maintenance action. Stochastic unavailability of machines throughout the scheduling horizon may be due to unexpected breakdowns of machines or any other unpredictable issues that cause machines to stop for a period of time. The duration of maintenance action on the shop floor is also uncertain, regardless of the type of maintenance. As a holistic approach, jobs need to be completed in time, planned unavailability and stochastic unavailability are competing at the same time in a set of machines. In this thesis, planned unavailability of machines is going to be decided by the CBM, which represents the unavailable period of a machine. CBM is performed based on the uncertainty of the machine degradation rates. In addition, stochastic unavailability of machines is generated randomly as the machine operation processing time increases over time. The duration of CBM and Corrective Maintenance (CM) actions is also considered uncertain based on the operation processing time.

In reality, any decision regarding scheduling in a shop environment has to be considered in a holistic approach. Integration of job scheduling with CBM and random breakdowns of machines will provide a smart robust schedule as well as a decision-making tool for the management of an organization. This thesis attempts to solve this problem with the implementation of a simulation-optimization algorithm to generate robust schedules in a job shop setting.

The objectives of this thesis are presented in the next section, followed by a description of the thesis organization.

1.1 Thesis Objective

The main objective of this thesis is to propose a simulation-optimization algorithm that robustly optimizes the job sequence while considering CBM and random breakdowns in a job shop environment. The proposed algorithm considers the uncertainty of machine degradation rates, random breakdowns, and uncertainty in CBM and CM

duration. Furthermore, the proposed algorithm can handle realistic-size problems efficiently as well as generate near-optimal solutions that are good on average, but also with unfavourable scenarios.

1.2 Thesis Organization

The remainder of the thesis is structured as follows: a literature review of the Job Shop Scheduling Problem (JSSP), which focuses on machine scheduling and JSSP is presented in Chapter 2. Some background knowledge about scheduling, CBM and metaheuristics is provided in Chapter 3. The problem under consideration, including its assumptions and mathematical formulation, is presented in Chapter 4. The simulation-optimization approach proposed to solve the problem is described in Chapter 5. The numerical experiments conducted and the results obtained are summarized in Chapter 6. Lastly, conclusions are drawn and future research directions are recommended in Chapter 7.

Chapter 2

Literature Review

This chapter reviews the literature relevant to machine scheduling and JSSP. The literature review is segmented into four sections: Machine scheduling, JSSP and solution approach, JSSP with maintenance, and JSSP with CBM. Next, the research gaps and contributions are highlighted.

2.1 Machine Scheduling

A seminal research paper by Johnson in 1954 recognized scheduling as an independent research area [6]. Johnson considered the production model to minimize the makespan or equivalently the total time required to complete all jobs for the two-machine flow shop problem. In 1956, Smith [7] introduced the Shortest Processing Time (SPT) for single machine scheduling to minimize makespan. Later, McNaughton [8] introduced parallel machine scheduling by providing a simple algorithm that was able to find preemptive schedules. By the end of the 1950s, deterministic machine scheduling had been recognized as an independent research area with its own range of problems and solution approaches. As a result, machine scheduling problems have been classified into single stage with one machine or more machines operating in parallel, and multistage, in which machines operate in series. However, in multistage systems, each job consists of several operations, each is performed by a specific machine, and a distinction was made between flow shop and job shop problem.

Since then, JSSP has attracted considerable attention among researchers. Jackson [9] extended the work of Johnson [6] to the job shop problem, considering two machines and each job consists of at most two operations to minimize makespan. Due to the complex structure of JSSP, dominance rules did not play a significant role in restricting the search. Roy and Sussmann [10] used a disjunctive graph formulation to represent JSSP. Németi [11] proposed disjunctive arc branching to solve JSSP, and Brooks and White [12] introduced active schedule generation branching. Even though

complex Branch and Bound algorithms were available, Fisher and Thompson's [13] 10*10*10 JSSP, which had 10 jobs, 10 machines, and 10 operations per job, was still not solved optimally.

The interesting thing is that there was no clear understanding of computational complexity among researchers till the end of the 1970s. The algorithms were originally developed in the 1970s for small instances of problems (single machine, two machines, two operations per job, or two jobs in a multi-machine job shop). Later, complexity was derived when the numerical parameters, such as the number of the machine or job were more than 2 in JSSP [14–16]. During the early days, researchers realized the computational complexity of job shop problems intuitively.

Meanwhile, linear programming and constraint programming problems were solved using the Lagrangean relaxation and column generation techniques. The advent of complexity theory in 1970 showed why scheduling problems were difficult to solve and were called "NP-hard problems". It was unlikely that there would be an algorithm that could find optimal solutions to NP-hard problems in a reasonable amount of time. During the 1980s and 1990s, decision-makers realized that finding solutions to NP-hard problems in a reasonable amount of time was very important. Since then, decision-makers have sought search algorithms capable of finding a near-optimal solution in a reasonable amount of time. As a result, some local search algorithms such as Hill Climbing, Simulated Annealing (SA) [17], and Tabu Search (TS) [18] were discovered. There were also other algorithms, like the Genetic Algorithm (GA) [19], the Ant Colony Optimization (ACO) [20], or other evolutionary techniques, developed later and used to make sure that near-optimal solutions were found in a reasonable amount of time for complex problems [21].

Based on the nature of the problems, optimization algorithms for scheduling were mainly divided into two categories, namely exact optimization methods and approximate optimization methods [22]. Exact methods are further classified as efficient rule approaches, mathematical programming approaches, and enumerative methods (Branch and Bounds methods). Similarly, approximate methods are classified as constructive methods (priority dispatching rules, insert algorithms, bottleneck-based heuristics), artificial intelligence methods (neural networks, expert systems, and knowledge-based methods), local search methods (search procedure, iterative

improvement, SA), and metaheuristics methods (GA, ACO, Particle Swarm Optimization (PSO), firefly algorithm, and other algorithms).

2.2 JSSP and Solution Approach

The earliest approach used in exact methods for JSSP is efficient rule-based methods that can obtain an exact optimum solution, which started from Johnson's [6] work. This work also introduced greater influence later in JSSP research. Wagner [23] proposed mathematical programming to solve JSSP optimally. Later, it was found that mathematical programming approaches had their own limitations due to the excessive computational time and effort required to get the solution. Therefore, enumerative methods came into play to solve JSSP, especially the Branch and Bound methods. Lomnicki [24] implemented Branch and Bound methods to get exact optimal solutions for a 3-machine JSSP. A common form of mathematical formulation proposed by Manne [25] is called discrete linear programming for JSSP, which computes optimal solutions with less computational time than Wagner [23]. Garey et al. [26] showed that JSSP is an NP-complete problem when the number of machines is more than two. It has been shown in the literature that only a small instance of JSSP can be solved to proven optimality. So, for a large-scale problem, exact methods were unable to provide an optimal solution in a reasonable amount of time. On the other hand, approximate methods were able to handle large-scale realistic instances effectively. As computer technology and intelligence algorithms have advanced since the 1980s, JSSP research methodologies have gradually transitioned from exact optimization procedures to approximation methods.

Zhang et al. [27] proposed a hybrid algorithm which is a combined form of PSO and TS to solve a multi-objective Flexible Job Shop Scheduling Problem (FJSSP). PSO was used to optimize the schedules, and local optimum was avoided using TS. Another study that solved JSSP using multi-objective GA to minimize makespan and total tardiness was presented by Lei [28]. A knowledge-based Variable Neighborhood Search (VNS) algorithm was investigated for FJSSP, where the randomized neighbourhood structure process has been modified with a knowledge module [29]. The VNS part looks for good solutions, whereas the knowledge module examines the knowledge of a good solution and sends feedback to the algorithm. Rahmati et

al. [30] adapted two evolutionary algorithms, Non-dominated Sorting Genetic Algorithm (NSGA) and Non-Ranking Genetic Algorithm (NRGA), for FJSSP to minimize the makespan, critical machine workload, and total workload of machines.

The graphical method introduced by Akers [31] was combined with GA to solve JSSP [32] in the form of a heuristic. The GA was responsible for finding the initial solution, and the graphical method tried to improve the initial solution. They applied the new heuristic to 205 standard instances chosen from the literature and compared the results with other approaches. They found that the new heuristic improved the best-known values for 57 instances out of 205. Gao et al. [33] solved the remanufacturing scheduling problem as FJSSP with uncertainty in job return timing and divided it into two stages of scheduling and rescheduling. A two-stage Artificial Bee Colony (ABC) algorithm was proposed as a solution methodology to minimize makespan, and its performance was enhanced by ensemble local search. The proposed algorithm's results were compared to five existing metaheuristics for the scheduling stage and six heuristics for the rescheduling stage. It was claimed from the results that the proposed two-stage ABC is effective in both stages.

Liu and Kozan [34] solved a JSSP with buffering requirements and formulated it as a Mixed-Integer Programming (MIP) model. Considering the large size of the instances, a novel constructive heuristics was employed to obtain an initial solution within a reasonable amount of time. A hybrid metaheuristics with a combination of SA and TS was applied to find a high-quality solution schedule. Another study that considers JSSP with four buffering constraints (no-wait, no-buffer, limited-buffer, and infinite-buffer) was investigated by Liu et al. [35]. They solved the problem with insertion-based heuristics with a constructive algorithm embedded inside. Optimization of maximum earliness and tardiness for JSSP was addressed by Yazdani et al. [36]. They formulated the problem as Mixed-Integer Linear Programming (MILP) and solved it with an Imperialist Competitive Algorithm (ICA) hybridized with an efficient neighbourhood search.

Nouri et al. [37] investigated FJSSP, where they applied hybridization of two metaheuristics and claimed the proposed method was more efficacious than other approaches. In the first stage, a Neighbourhood-based Genetic Algorithm (NGA) with a scheduler agent was applied for global exploration of the search space and in the

second stage, TS was applied to improve the solution quality of the final population obtained from NGA. A two-level metaheuristics algorithm was studied for JSSP by Pongchairerks [38] which has an upper-level and lower-level algorithm. The lower level finds the best schedule as an iterated local search algorithm, and the upper level is a population-based algorithm that is used as a parameter controller for the lower-level algorithm. From the results, it was claimed that the two-level metaheuristics algorithm outperforms the results of PSO and GA from the literature. Later, the two-level metaheuristics [38] was enhanced by the same author for JSSP [39]. The upper level was enhanced by considering multiple solutions starting with every iteration, and as a result, the lower level worked as a multi-start iterated local search algorithm. It was demonstrated that an improved version of two-level metaheuristics outperforms its previous variant.

A two-level PSO algorithm was presented by Zarrouk et al. [40] for FJSSP while minimizing the makespan and robustness of solutions. The lower level handles the operations assigned to machines, whereas the upper level determines the sequence of operations on the machines. Experimental results showed that the proposed two-level PSO algorithm can provide better results than other metaheuristics in a shorter time. Mohammadi and Moaddabi [41] inquired about two metaheuristics algorithms for JSSP and compared their performance with other two algorithms from the literature. The proposed algorithms are ACO and Harmony Search Algorithm (HSA), and the comparison algorithms are GA and SA. For all of the instances, GA performance was better and the proposed ACO and HSA performed better than SA. Gohareh and Mansouri [42] evaluated and validated a simulation-optimization approach for stochastic job shop where Markov decision process was used for machines states and ACO was applied for optimization.

2.3 JSSP with Maintenance

Wang and Yu [43] developed a Filtered Beam Search (FBS) heuristic to solve FJSSP with machine availability and maintenance resource constraints. They considered PM which has a fixed start time and a flexible start time. Two types of maintenance resources were considered: unlimited maintenance resources and only one maintenance resource. Another FJSSP with PM consideration was addressed by Moradi et al. [44]

to minimize makespan for production and system unavailability for maintenance part simultaneously. Four evolutionary algorithms were proposed, namely NSGA-II (Non-dominated Sort Genetic Algorithm-II), NRGA, NSGA-II with Composite Dispatching Rule (CDRNSGA-II), and NRGA with CDR (CDRNRGA). Results showed that CDRNSGA-II and CDRNRGA outperformed NSGA-II and NRGA and proved the efficiency of CDR with evolutionary algorithms. Hasan et al. [45] applied GA with the Shifted Gap Reduction (SGR) heuristic to solve JSSP with machine unavailability and random breakdowns. It was shown that the impact on the schedule is very low if the machine's unavailability is recognized in advance. Harrath et al. [46] came up with a multi-objective GA for JSSP that takes into account both PM and CM strategies to minimize makespan and total maintenance costs.

To investigate the FJSSP with parallel machine and maintenance constraints, two metaheuristics algorithms, hybrid GA and SA, were implemented [47]. The solutions of the two metaheuristics were compared with the solutions obtained from LINGO software. Results showed that the solutions obtained from the two metaheuristics were much more efficacious than those obtained from LINGO in terms of solution time and optimality. Two surrogate measures for robust FJSSP were reinforced to minimize makespan [48], where the first surrogate measure dealt with the probability of machine breakdowns and the second one handled float times (slack) and breakdown periods. Experimental results suggested that the first surrogate measure provided a better solution for small instances, whereas the second surrogate measure performed well for both small and relatively large instances. Ye and Ma [49] investigated joint production and maintenance planning where the production part is FJSSP and the maintenance part is PM. A GA was designed to solve the FJSSP. They commented that the joint consideration of production and maintenance planning has more superiority than independently considered production maintenance planning.

Ahmadi et al. [50] addressed FJSSP with the consideration of random machine breakdowns using evolutionary algorithms. Stability of the schedule was considered based on the slight deviation between the start and completion times of jobs under uncertain conditions of prescheduling and realized schedule. They applied NSGA-II and NRGA to improve the makespan and stability, as well as determine the states and conditions of the machine breakdown using a simulation approach. Fitouri et al. [51]

proposed a decision-making approach for JSSP to minimize makespan and maintenance costs, and GA was proposed to solve the problem. Nouiri et al. [52] presented a two-stage PSO algorithm to solve FJSSP under machine breakdowns and generate robust and stable solutions. Results indicated that the solutions obtained from the two-stage PSO algorithm were more statically superior than hybrid GA. A two-stage GA framework was examined to solve FJSSP with random machine breakdowns to obtain robust and stable solutions by Sajadi et al. [53]. In the first stage, a simple GA minimizes the makespan where everything is deterministic and no disruptions are considered. In the second stage, two different multi-objective algorithms, NSGA-II and NRGA, were applied with the consideration of random machine breakdowns. Numerical results indicated that NRGA performs better than NSGA-II and can produce robust and stable schedules. Zhai et al. [54] integrated JSSP and predictive maintenance using GA, where they monitored the health condition of the machine and predicted future deterioration.

A new heuristic was proposed for JSSP with maintenance to minimize makespan, which is a combination of Modified Genetic Algorithm (MGA) and Heuristic Displacement of Genes (HDG) [55]. MGA controls the random character of general GA, such as crossover, mutation, and enhances the random character by guiding it through a logical procedure. The solutions obtained from MGA were further improved by the HDG technique with the possible displacement of the genes in the obtained solution. A novel multi-objective optimization algorithm was introduced to minimize the makespan and total energy consumption for FJSSP with PM and transport processes in the shop [56]. The proposed multi-objective optimization algorithm is a combined form of GA and a Differential Evolution (DE) algorithm (MOGA-DE) and the results concluded that the proposed algorithm is an efficient algorithm for integrated FJSSP-maintenance planning. Gupta and Jain [57] developed a discrete event stochastic simulation model for FJSSP with random breakdown and PM routing flexibility. The results obtained from the simulation model were used as an input for the statistical Response Surface Methodology (RSM) for the analysis of the results. A confirmatory experiment revealed that the deviation between experimental and predicted results is less than 5%, which confirms that the proposed simulation method can produce 95% confidence level solutions. For the first time in dynamic FJSSP, a

Greedy Randomized Adaptive Search Procedure (GRASP) with PM activities consideration was proposed to minimize mean tardiness, schedule instability, makespan, and mean flow time [58]. It was claimed that the proposed approach improves the performance of dynamic FJSSP. Souza et al. [59] developed a simulation-based optimization algorithm with the consideration of deterministic and stochastic unavailability due to PM and CM. Three surrogate measures were used to approximate the true makespan objective function and independently optimise it using GA. The solution obtained from the GA procedure is then simulated with random breakdown scenarios and SA is employed to avoid premature conversion. The results showed that the simulation-based optimization algorithm performed well in terms of average improvement and run time.

2.4 JSSP with CBM

From the literature, only a few studies considered JSSP with a CBM strategy. Zheng et al. [60] developed a framework for FJSSP with CBM to reduce the unavailability of machines. The initial solutions (preschedule) obtained by applying Integrated Genetic Algorithm (IGA) and PM was added to the initial solution with the implementation of the Insertion Algorithm (IA). The results obtained from the proposed IGA were compared with hGA (GA with the neighbourhood) and Simultaneous Scheduling Algorithm (SSA). The proposed IGA approach showed better performance than hGA and SSA in terms of execution time. Another study that used IA to insert CBM into preschedule in FJSSP was solved by Zheng et al. [61]. More specifically, during the idle time of machines, they inserted the CBM into the schedule. Then the FJSSP with CBM was solved with three metaheuristics, namely GA, ACO, and ABC. The results showed that GA performed better than ACO and ABC. Zandieh et al. [62] presented a simulation-optimization framework with a combination of Sigmoid function and Gaussian distribution to simulate the CBM in FJSSP environment. The condition of the machine was monitored by degradation based on the shocks of the machine. A hybrid ICA with SA was proposed to solve the integrated problem and was shown to be quite effective.

Rahmati et al. [63] designed a novel framework for integrating FJSSP and CBM using a simulation-optimization approach with the consideration of degradation of

machines. Two degradation levels were considered: the lower level for PM and the upper level for CM. Results obtained from the proposed framework suggests that the HSA implemented in the simulation-optimization framework performed better than GA. The latest work which dealt with FJSSP considering CBM to integrate scheduling and maintenance was investigated by Ghaleb et al. [64]. They formulated the problem as a stochastic MIP and solved it through a modified hybrid GA, and the results were compared with SA and ICA. Results showed that modified hybrid GA performs very well in the case of small and large instances and outperforms SA and ICA. Their results also insights that the total cost is 35% lower when the decisions are made based on the machine's deteriorating information.

2.4.1 Research Gaps and Contributions

Despite the necessity and role of maintenance strategies on the production floor, specifically CBM, a common and usual consideration in most production and scheduling problems is that machines are always available. This assumption not only exempts the research field from the complex area of maintenance consideration but also does not consider real-life scenarios where machines can break down for several reasons or need to be repaired and fixed. Therefore, there is a gap in the literature on how classical methods can approach or model the JSSP with maintenance strategies while considering uncertainty in machine degradation rates, random breakdowns, and stochastic maintenance duration. This study deals with this gap and attempts to cover a part of it.

In this thesis, we propose a simulation-optimization approach for a combined version of scheduling-maintenance planning that considers JSSP and a maintenance strategy that includes both CBM and CM. Besides the uncertain nature of failures, the proposed approach considers other aspects of uncertainty, namely the degradation rate of machines, random breakdowns, and the duration of CBM and CM actions. Schedule robustness is ensured by optimizing a composite objective function that consists of a weighted average of the expected makespan and the 90th percentile of makespan. In every iteration, the proposed simulation-optimization approach optimizes $(v + 3)$ surrogate makespan functions using GA. To introduce diversity into the breakdown scenarios while executing the GA procedure in the inner loop of the

algorithm, we select v scenarios by solving a diversity maximization problem. Then, it simulates the best schedules obtained from the GA procedure to estimate their true objective values and adds the best among them to an elite list that initiates the next iteration's GA populations. A stopping rule inspired by SA is applied to enhance early exploration of the search space and reduce the chance of premature convergence to a local optimum. The proposed algorithm is designed in such a way that it can handle realistic size instances efficiently and generate robust schedules.

Chapter 3

Background

In this chapter, the fundamental classification scheme of machine scheduling is provided, followed by the general concepts of job shop scheduling problems. Then, CBM and breakdowns are described. Next, the procedural steps of metaheuristics strategies for GA and SA are presented.

3.1 Preliminaries of Machine Scheduling

Hoos and Stützle [65] describe that scheduling problems appear in all situations where a well-defined set of actions needs to allocate the limited resources with time slots given that the problem is optimized and feasible. Baker and Trietsch [66] identifies that the "machine scheduling problem" is a special type of scheduling problem where the scheduling of jobs describes a complete schedule by representing the allocation of the machines to jobs and the sequences of operations. In the next subsection, we present the machine scheduling fundamental classification scheme and then the overall description of the JSSP.

3.1.1 Machine Scheduling Classification Scheme

Graham et al. [67] presented and classified scheduling problems using a 3-field classification scheme $\alpha|\beta|\gamma$ in a comprehensive work of deterministic sequencing and scheduling survey published in 1979. Schmidt [68] and Ma et al. [69] further expanded this classification scheme with the consideration of the unavailability period of machines.

Let us consider there are n independent set of jobs, that needs to be processed on m independent set of machines. Each Job i consists of the p ordered operations, each has duration P_{ip}. By using three field problem classification scheme $\alpha|\beta|\gamma$, various job, machine, and scheduling characteristics are presented.

3.1.1.1 Machine Environment

The first field $\alpha = \{\alpha_1, \alpha_2, \alpha_3\}$ represents the machine environment. Initially, Graham et al. [67] introduced $\alpha = \{\alpha_1, \alpha_2\}$ and then it further extended to $\alpha = \{\alpha_1, \alpha_2, \alpha_3\}$ by Schmidt [68] and Ma et al. [69] with the consideration of α_3 as machine unavailability parameter. Parameter α_1 assumes $\{\alpha_1 = J\}$, which represents job shop environment, where a set of jobs have to process on m set of machines. Parameter $\{\alpha_2 = \phi\}$ denotes the number of machines or stages is variable. As mentioned, parameter α_3 represents the machine's unavailability. Ma et al. [69] presented unavailable periods in machines as "holes". Parameter $\{\alpha_3 = h_{kl}\}$ signifies that each machine k has an arbitrary number of holes. Here, k denotes which machine has holes whereas l symbolizes how many holes are on the machine.

3.1.1.2 Job and Resource Characteristics

The second field $\beta = \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5\ \beta_6\}$ describes job and resource characteristics. Parameter β_1 indicates the preemption possibility. Pinedo defined [70] preemption as the ability to stop the processing of a job on a machine at any given time and substitute it by any other job. The scheduler is permitted to interrupt a job at any time on a machine and assign another job on that machine instead. Parameter β_1 assumes $\{\beta_1 = \phi\}$, when no preemption is allowed and $\{\beta_1 = r - a\}$, which denotes jobs are resumable. Parameter β_4 characterizes the release date r_j of a job that defines when a job is ready to be processed on a machine at the earliest. Parameter β_4 assumes $\{\beta_4 = \phi\}$ when each job is available at time zero, i.e., $r_j = 0$.

3.1.1.3 Optimality Criteria

The third field γ represents the optimality criteria, also referred to as performance measure. Such criteria used in the literature is related to the minimization of performance measures. Makespan (C_{max}) represents the entire time required to accomplish a set of jobs. It is the time from the commencement of the first job in the group to the end of the last job in the group. A minimum makespan usually implies a good utilization of the machine(s).

3.2 Job Shop Scheduling Problem

The two common types of scheduling problems are the Flow Shop Scheduling Problem (FSSP) and JSSP. Other varieties of scheduling problems are based on these two basic types. Baker and Trietsch [66] explained JSSP in the following manner: The main difference between JSSP and FSSP is that in JSSP there is no unidirectional flow of jobs throughout the schedule horizon, whereas in FSSP all the jobs go through with unidirectional flow and in the same sequence. Other elements that are still present in the FSSP are also characterized in the JSSP. There are a set of machines and a collection of jobs to be processed. Each job still has a precedence requirement and has its own machine sequencing. Job re-circulation, in which a job can visit the same machine more than once and can skip certain machines, is not implemented in this thesis. There is no fixed machine that starts the initial job and no terminal machine that performs the last job. Baker and Trietsch [66] illustrated a 4-job and 3-machine JSSP shows in Table 3.1 where (a) describes the processing time and (b) describes each individual job sequencing.

	(a) Processing times				**(b) Routings**		
	Operation				**Operation**		
	1	**2**	**3**		**1**	**2**	**3**
Job 1	4	3	2	**Job 1**	1	2	3
Job 2	1	4	4	**Job 2**	2	1	3
Job 3	3	2	3	**Job 3**	3	2	1
Job 4	3	3	1	**Job 4**	2	3	1

Table 3.1: An example of JSSP with 4-job and 3-machine [66].

Based on the information in Table 3.1, a Gantt chart represents the feasible schedule. To visualize the Gantt chart, a triplet formulation of ijk was used where i represents jobs, j represents operations, and k represents machines. The Gantt chart is shown in Figure 3.1.

The number of combinations to be examined to find the optimal solution in case of flow shop scheduling is $n!$, whereas the number of combinations grows exponentially for JSSP as the number of jobs and machines increases. In case of JSSP, the number of combination that needs to be analyzed is $(n!)^m$, which makes JSSP an NP-complete problem [26]. An NP-complete problem is a problem that cannot be

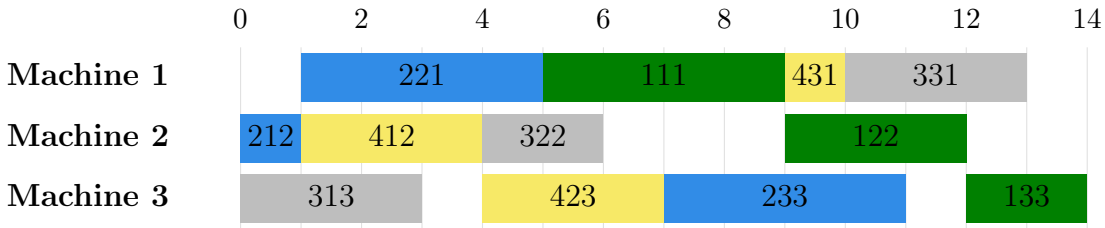


Figure 3.1: Gantt chart of JSSP with 4-job and 3-machine [66].

solved in polynomial time in any known way.

3.2.1 Characteristics and Factors in JSSP

It is very important to generate the right mix between control and flexibility when creating a job shop scheduling system. Griffin [71] stated some of the common characteristics of job shop due to the fact of a wide variety of products. The following characteristics are stated by [71]:

- There are many orders in various states of completion at any given time;
- Execution of orders competes for demands on facilities and manpower;
- Every order is unique in some way. As a result, properly predicting the time required to complete operations is difficult;
- Workflow is intermittent and orders can be interrupted or delayed;
- At each machine, there is frequently a queue of work, and it is often challenging to determine which order in the queue should have priority;
- Machine failure, scrap, rework, design changes, and rush orders result in many changes in the job shop;
- Orders are expedited via multiple departments with much effort to determine the status;
- Due to the significant clerical workload required to make the changes, schedules and shop loads are rarely changed;

Four factors serve to describe and classify a specific JSSP [71]. They are:

- Considering the job arrival pattern, when n jobs arrive simultaneously in a shop that is idle and available to process, the scheduling problem is said to be static. The scheduling problem is dynamic if jobs arrive intermittently, potentially according to a stochastic process;
- The number of machines m, that make up the job shop must be specified;
- It is necessary to specify the workflow process through the machine. The shop is a flow shop if all jobs follow the same sequence. Conversely, a job shop where the jobs do not follow the same sequence of operations;
- In the scheduling process, the criterion for evaluating the shop's performance plays a critical role.

3.3 Maintenance Preliminaries

In the next subsection, we discuss how maintenance practices have changed over time. Then, the important concepts of CBM and breakdown are described in the next two subsections.

3.3.1 Historical Perspective

Machines, or systems comprised of machines, will fail at some point during their life cycle. To bring the machines back to an operational state, maintenance has been considered since 1940 and accounted for as an unavoidable element [72]. Till 1940, only CM was carried out, and maintenance issues were neither considered during the design of the machine system nor the impact of maintenance on the system and business performance. Since the evolution of Operations Research (OR) after the Second World War, organizations have widely used PM at the component level as well as at higher levels. The OR models explored various types of maintenance policies and the selection of parameters for these policies. The impact of maintenance on business performance had not been addressed until 1970.

After 1970, a more integrated approach to maintenance was used by the government as well as private sector. This happened because of the close link between reliability and maintainability. The terms reliability and maintainability were begun

to be implemented more widely in defence equipment. The concept of reliability and maintainability also established the foundation of Reliability Centered Maintenance (RCM) which was adopted by manufacturers of civilian aircraft. RCM is concerned with the optimization of preventive maintenance actions while taking into account the consequences of failure. At the same time, Total Productive Maintenance (TPM) originated in Japan in the field of manufacturing. Here, maintenance is considered in terms of how it affects manufacturing or production process by impacting equipment availability, rate of production, and quality of output. Now, RCM and TPM are widely employed in a variety of industrial areas, and many versions have been evolved to expand and simplify their implementation. There has been a shift toward CBM since the late 1970s and early 1980s. This was made possible by advances in sensor technology, which allowed PM activities to be based on condition (or level of degradation) rather than age and/or usage. Figure 3.2 shows the maintenance function in terms of time perspective.

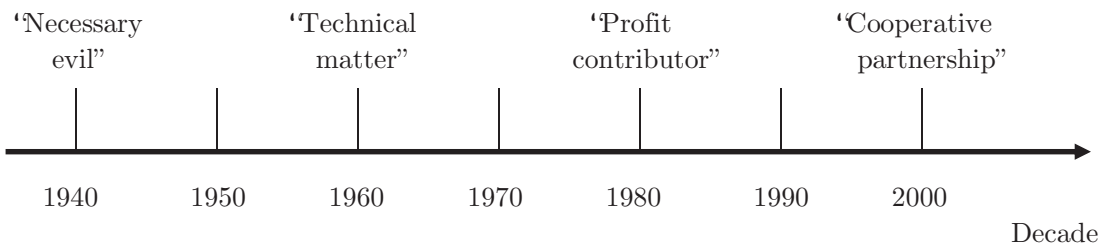


Figure 3.2: Maintenance function evolved in time perspective [73].

The integrated maintenance approach was built based on reliability and maintainability. This thesis considers CBM and random breakdowns to bring the machines back into an operational state. To get a proper understanding of the maintenance approach used in this thesis, we introduce the concepts of reliability and maintainability. Ebeling [74] defined reliability as “the probability that a system or component will be operational to perform a required function for a specific period of time when operating under stated conditions and settings”. Ebeling [74] also defined maintainability as “the probability that a failed component or system will be back or restored or repaired to a specific operational state within a period of time following prescribed procedures”. So, when a machine fails and is restored to an operational state, the interaction between reliability and maintainability provides the total operational time,

which determines the availability of the machine.

3.3.2 Condition-Based Maintenance

A CBM is a PM action that has the following characteristics [72]:

- it uses a measurable parameter that is connected with the rate of degradation over time;
- data collected using proper condition monitoring techniques are used to determine changes in the measurable parameter;
- for the collection of data, no intrusion into the item is required.

From this definition, the most important consideration is to identify a measurable parameter that correlates between measurement and level of degradation. The parameter can be monitored continuously or at discrete points in time, online or offline, and the monitoring frequency must be defined if it is observed at discrete points in time. The chosen parameter can be used to measure degradation directly or indirectly.

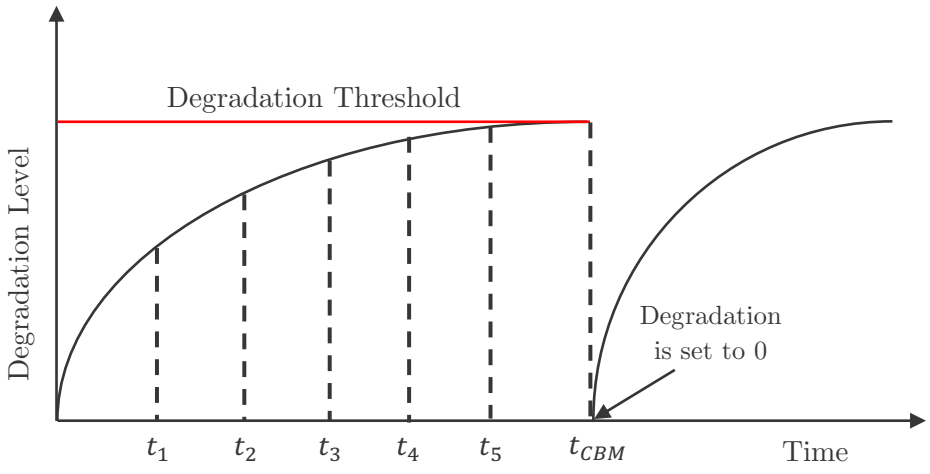


Figure 3.3: Concept of CBM.

Figure 3.3, illustrates the concept of CBM. The degradation increases over time, and it is a cumulative process. The inspection of the degradation level happens at t_1, t_2, t_3, t_4, t_5 to check the status of the machines. A possible action of CBM occurs at t_{CBM} and machine degradation level is set to 0. CBM actions can be further

classified into two categories: one is condition monitoring and inspection, and the other one is functional tests. In this thesis, the first category of condition monitoring and inspection is used.

3.3.3 Breakdown

The mean operating time until the failure of a machine is known as the "*Mean-Time-To-Failure*" (MTTF). The concept of MTTF was introduced by Barlow and Proschan [75] to study the reliability characteristics of a repairable system as well as the monotonic behaviour of MTTF. Since then, MTTF has become popular for use in failure time prediction of a system or machine deterministically. Ebeling [74] described the MTTF equation governed by the Weibull distribution is as follows as Equation (3.1) and (3.2),

$$MTTF_k = \theta_k \Gamma\Big(1 + \frac{1}{\beta_k}\Big) \tag{3.1}$$

$$\Gamma(x) = \int_0^\infty y^{x-1} e^{-y} dy \tag{3.2}$$

where k is the machine index, θ_k is the scale parameter of the failure function of machine k, and β_k is the shape parameter of the failure function of machine k. Equation (3.1) is used for the approximation of the surrogate functions while executing GA procedure.

3.4 Metaheuristics Preliminaries

The optimal solution is the ultimate goal when solving a problem, whether it is large or complex. The procedure to find an optimal solution should be within a reasonable amount of time. The solution time of the problem is increased with the increase of input parameters or as the problem size increases. Mathematical models such as linear, integer, and non-linear programming depend on algorithms to obtain optimal solutions. These algorithms might not be able to provide an optimal, near-optimal, or even feasible solution within a reasonable amount of time. To solve these types of complex and large problems, heuristics, more specifically, metaheuristics, is employed.

It is developed to get optimal or near-optimal solutions within a reasonable amount of time. Figure 3.4 shows the relationship between master heuristics and a set of subordinated heuristics.

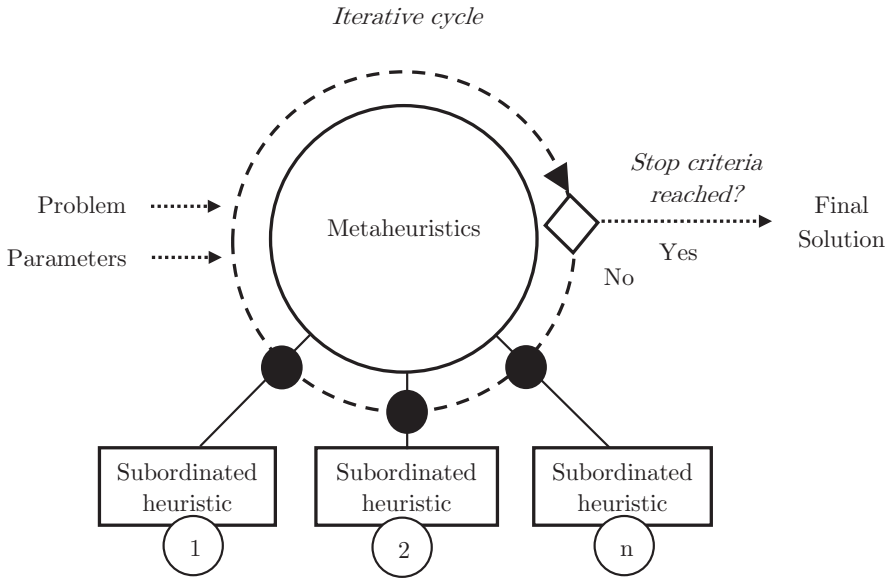


Figure 3.4: Relationship between metaheurisitcs and subordinated heuristics [76].

Generally, when it is not possible to find the solution of a problem using exact methods within a reasonable amount of time, heuristics can find a feasible solution within a reasonable amount of time with the implementation of search methods and usage principles. A higher-level heuristic is termed as metaheuristic which is a combination of a master heuristic (metaheuristic cycle) and a set of subordinated heuristics. The term "metaheuristics" was introduced by Glover [77] in 1986. The term metaheuristics is a combination of two words: meta in sense of higher-level and heuristics meaning to find or discover or search [76]. Master heuristic navigates and synchronized the set of subordinated heuristics to explore the search space iteratively whereas subordinated heuristics may be a general local search or construction methods, or a high or low-level procedures. To discover a good optimal solution, metaheuristics need to be conducted with a good balance between exploration and exploitation [78–80]. Exploration refers to the diversification of search space where promising areas are identified with high-quality solutions within search space. In contrast, exploitation characterizes the intensification of the promising areas (excellent quality solutions) explored in exploration to obtain a better solution [81–83].

As metaheuristics are iterative processes to find an optimal solution, different types of termination or stopping criteria are used. Different metaheuristics algorithms have different termination criteria. The most common termination criteria are limiting the iteration number and total run time. Limiting the iteration number refers to an algorithm that stops when a predetermined number of iterations has been reached, whereas total run time focuses on the predetermined amount of time that has passed. When other methods fail to solve and obtain a solution to a large-scale optimization problem, the ability to solve and obtain better solutions through metaheuristics is chosen to solve that problem [84]. Though metaheuristics have the characteristics to obtain better solutions, they need to be designed in such a way that they lay out the overall algorithmic structure and rules. The algorithmic structures and rules must be followed to create a good heuristic that will work for a particular problem. Two metaheuristics are used in this thesis, namely Genetic Algorithm (GA) and Simulated Annealing (SA). The next two subsections describe the GA and SA.

3.4.1 Genetic Algorithm

The Genetic Algorithm was first introduced by Holland [85] in 1975. The basis of GAs is natural selection and genetic principles with the implementation of search methods [86]. Natural selection and genetic principles are the combinations of breeding, selection, and mutation that happen over many generations to produce a population that represents good solutions. Davis [87] applied GA to JSSP, where every machine's preferred sequence of operations was formed. The algorithm attempts to follow the behaviour found in natural selection, known as survival of the fittest. In the survival of the fittest, each individual carries good genes that have a better possibility of survival in future generations. Figure 3.5 shows the representation of genes, chromosomes, and population.

The general procedure of genetic algorithm starts with a finite set of individuals that are represented by a list of chromosomes. The list of chromosomes is called population and this starting finite set of individuals is the initial solution for the genetic algorithm procedure to carry forward for the next generations. The list of chromosomes represents the characteristics of individuals. With the initial solution, the generation has started. Then, the algorithm randomly chooses two individuals to

form pairs, exchanges the parts (genes) of chromosomes between them, and creates new individuals in a procedure called crossover.

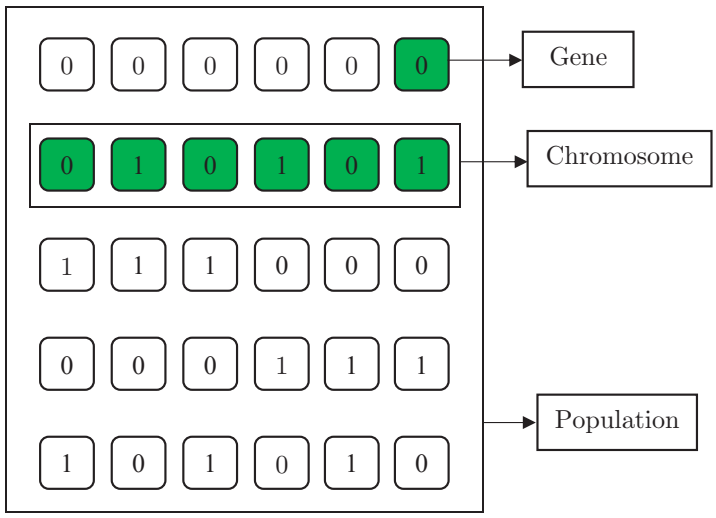


Figure 3.5: Representation of genes, chromosomes and population set.

The new individuals created in the crossover process are expected to carry characteristics from both parents that would lead to a better solution. Also, the individuals created in the crossover process are pushed through the mutation process, where some variability in their genes has been added. After the mutation, each of the individuals is evaluated based on a fitness function (cost function). A fitness function can be used to evaluate their fitness for the next generation. The fitness function determines the ability of an individual to compete with other individuals. After evaluation based on the fitness function, those who are the worst performers are killed, meaning that they do not proceed to the next generation of participation. The GA cycle creates new individuals and processes its steps until the termination criteria are met, meaning that the diversity among individuals is less or the result is good enough.

A more detailed description of GA is available in the literature, which is not addressed here. Bierwirth et al. [88] found evolutionary algorithms such as GA can produce good results for JSSP. Figure 3.6 shows the general procedure that a GA follows.

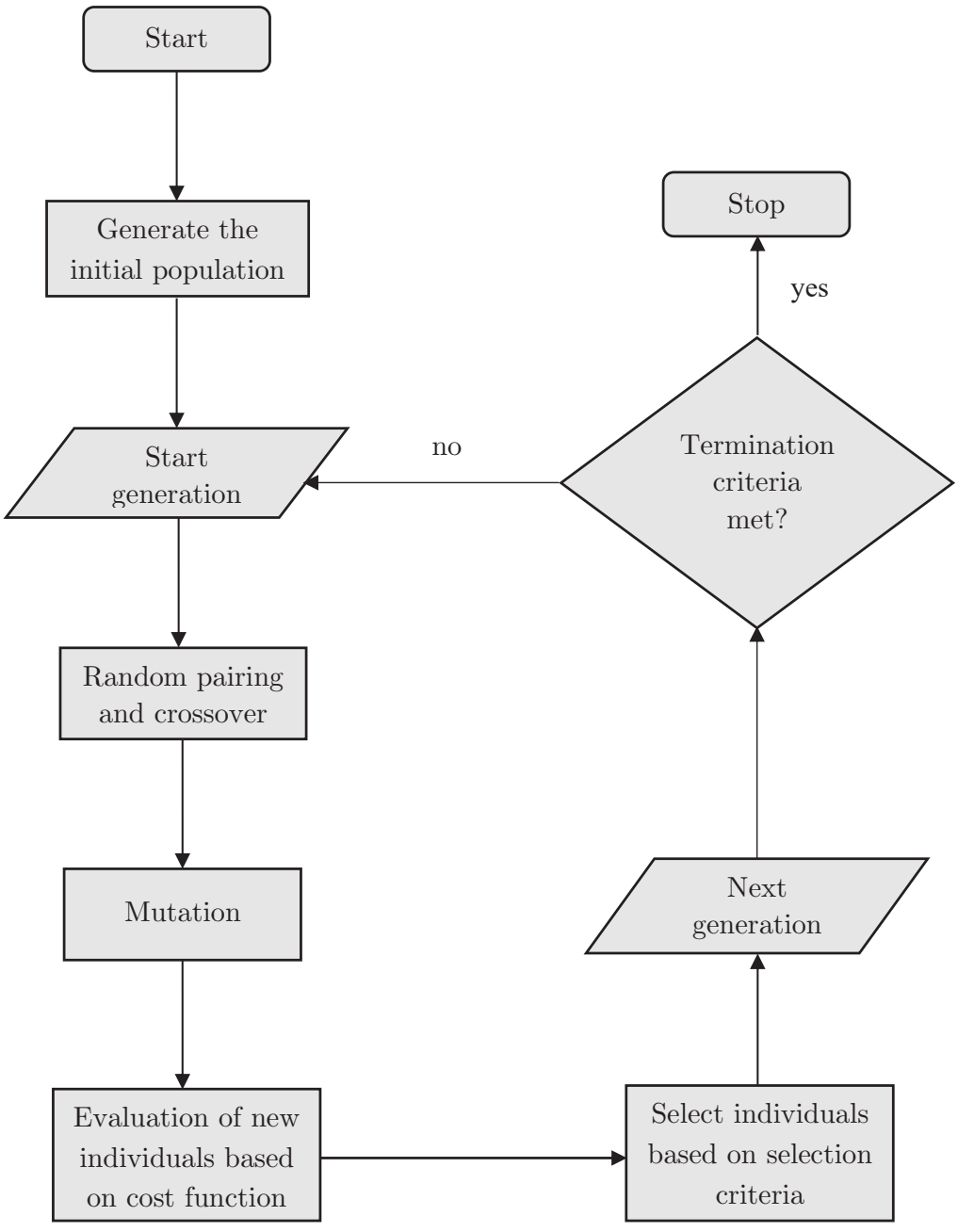


Figure 3.6: General procedure of GA.

3.4.2 Simulated Annealing

Metropolis et al. [89] first introduced SA in 1953 with the development of an algorithm to simulate physical annealing. Their objective was to observe the physical structure of the material while undergoing the annealing process. The Monte Carlo technique was the basis of that algorithm and generated a sequence of solid states. Later, SA was applied to optimization problems by Kirkpatrick et al. [90]. Since then, it has become one of the known metaheuristics applied to optimization problems. The main attribute of SA is to escape from the local optimum and broadly search the whole solution space for better solutions. In this context, local optima refers to the best solution within its immediate neighbourhood regions in a problem. The solution of a problem might fall into a local optimum, if the whole solution space is not broadly searched due to the search limitation of the solution space. By accepting a worse solution than the actual good solution, SA tries to escape from local optimum regions based on a neighbourhood algorithm. The neighborhood random search is closely tied to the temperature parameter.

The SA process starts with the setting of the temperature parameter to a high value. With the setting of the temperature parameter to a high value, it usually increases the probability of acceptance of new solutions, even if these solutions are not better than the current solutions. In each iteration or after n iterations, the temperature parameter is updated, which decreases the probability of accepting a worse solution than the current better solutions. After some iterations, the algorithmic procedure stops when the temperature is low. At that point, the best solution obtained is announced as the optimum approximation of the objective function value. One of the most important parameters in the SA procedure is the way temperature is updated and termed as "temperature schedule". The temperature schedule directly influences the execution time of the algorithm as well as the quality of the best solution. A general procedure of SA is shown in Algorithm 1.

Let us denote the initial solution, initial temperature, and temperature update factor by s_0, t_0, and α, respectively. The neighborhood of solutions s_0 is designated as $N(s_0)$. One of the main characteristics of SA is its memoryless property. The memoryless property denotes the fact that solutions are not saved as they are discovered during the search procedure. The best solution was returned as an approximation of

Input:

Initial solution s_0;

Initial temperature $t_0 > 0$;

Temperature reduction factor α;

Maximum number of repetitions *nrep*;

Objective function f;

Output:

The approximation of the optimal solution;

repeat

  repeat

    Randomly select $s \in N(s_0)$;

    $\Delta = f(s) - f(s_0)$;

    if $\Delta < 0$ **then**

      $s_0 = s$

    else

      generate random $x = U(0, 1)$;

      if $x < exp(-\Delta/t)$ **then**

        $s_0 = s$

      end

    end

  until *iteration_cont* = *nrep*;

  Set $t = \alpha(t)$;

until *Until stopping condition* = *True*;

Algorithm 1: General Simulated Annealing procedure [91].

the objective function value.

Chapter 4

Problem Description

This section provides a brief description of the problem under consideration, including the modelling assumptions of the schedule and the maintenance policy, followed by the mathematical formulation. The following notations are used to model the problem:
Indices:

i, j: Indices of jobs $\{i = 1, 2, ..., n,\ j = 1, 2, ..., n\}$;

p: Index of operations $\{p = 1, 2, ..., m\}$;

k: Index of machines $\{k = 1, 2, ..., m\}$;

l, u: Indices of positions (orders) on machines $\{l = 1, 2, ..., n,\ u = 1, 2, ..., n\}$;

Parameters:

P_{ip}: Predetermined duration of operation p in job i;

r_{ipk}: 1, if operation p of job i requires machine k (binary parameter); Otherwise, 0;

t_k^p: Average duration of CBM of machine k;

t_k^r: Average duration of CM of machine k;

β_k: Shape parameter of machine k;

θ_k: Scale parameter of machine k;

D_k^*: Recommended CBM degradation threshold;

ξ_{kl}: Realized number of breakdowns of machine k when processing the job in the l^{th} position;

M: A big number;

Decision variables:

X_{ikl}: 1, if job i is scheduled in the l^{th} position on machine k; Otherwise 0;

Y_{kl}: 1, if CBM is performed before the l^{th} job operation on machine k; Otherwise 0;

S^0_{kl}: Planned start time of the job scheduled in the l^{th} position of machine k;

Auxiliary variables:

q_{kl}: Duration of the job being processed in the l^{th} position on machine k;

S^w_{kl}: Realized start time of the job in the l^{th} position of machine k;

C^0_{kl}: Planned completion time of the job in the l^{th} position of machine k;

C^w_{kl}: Realized completion time of the job in the l^{th} position of machine k;

b_{kl}: Degradation of machine k before processing the job scheduled in the l^{th} position;

a_{kl}: Degradation of machine k after processing the job scheduled in the l^{th} position;

The JSSP is described as follows: there are n independent set of jobs, that needs to be processed on m independent set of machines. Each job i consists of the p ordered operations, each has duration P_{ip} and each can and must be performed by a single machine. Besides, the following assumptions are considered for this thesis:

- Machines and jobs are independent of each other.
- At time zero, all machines and jobs are available.
- Set up time of machines and jobs movement time between machines are negligible. Inspection time for CBM is also negligible.
- At any given time, a machine can perform only one job and a job can be processed by only one machine.
- Machines might become unavailable at times due to CBM and CM actions.

- All operations have the same priority level and are performed perfectly.
- When a machine fails, CM is performed on it with average duration of t_k^r and the operation is resumed from the point where it was interrupted. On the other hand, CBM can be performed only between operations upon inspection of the machine condition with average duration of t_k^p.

The objective function of the problem robustly minimizes the weighted sum of the average makespan and the 90th percentile of makespan.

4.1 Maintenance Policy

Two types of maintenance policies are considered: CBM and CM. CBM is a type of PM where maintenance is carried out based on the actual condition of machines. The CBM is implemented in this thesis based on the degradation of machine. At the time of schedule implementation, the machine condition is inspected after each operation and a decision is made based on the observed condition on whether a maintenance action is performed or not. However, since the schedule (i.e., sequence of operations on each machine) must be decided before machines start processing jobs, the condition of machines must be predicted using the concept of degradation. We assume that the degradation of a machine caused by a given operation p in job i is proportional to the operation duration P_{ip}, and that degradation is a cumulative process. Furthermore, to account for its uncertain nature, we model the degradation rates as independent random variables, each follows an operation-specific exponential distribution, i.e., operation p in job i causes degradation $DR_{ip}P_{ip}$, where $DR_{ip} \sim Exp(\eta_{ip})$ [63].

At the outset, all machines are assumed to have zero degradation. After each operation, the cumulative degradation is calculated and compared to a predetermined threshold. If the cumulative degradation exceeds the threshold value, a CBM is carried out. Upon performing CBM, the machine cumulative degradation is set back to zero, i.e., machine condition becomes "as good as new". CBM can be performed only between operations upon inspection of the machine condition. So, there will be no CBM in the middle of a job that is being processed, rather it is always when a job is finished based on the inspection and threshold value examination. In reality, the duration of the maintenance action is uncertain regardless of the maintenance

personnel's expertise or experience. It is not possible to precisely determine the duration of the maintenance action. Consideration of uncertainty in the duration of the maintenance action implies real-life scenarios. The CBM duration is assumed uncertain and is modelled as a uniformly distributed random number based on the average operations processing time.

CBM is carried out to reduce the frequency of breakdown, but machine breakdown is uncertain after every measures have taken. Though trying to predict the failure time deterministically, random breakdown still likely to occur in real life scenario. A breakdown can be modelled as stochastic point process, to be more specific a non-homogeneous Poisson process as explained by Cui et al. [92]. Equation (4.1) describes the τ^{th} breakdown time of machine k as follows:

$$T_{\tau k}^{BD} = {R_k}^{-1}\left[\prod_{\chi=0}^{\tau}(1-\zeta_\chi)\right], \tag{4.1}$$

where $\zeta_\chi \sim U(0,1)$ is a uniformly distributed random number, R_k^{-1} is the inverse reliability function. When the failure process is governed by a Weibull distribution, as assumed in this work, the reliability function is $R_k = e^{\left(-\frac{t_k^d}{\theta_k}\right)^{\beta_k}}$, where θ_k and β_k are the scale parameter and shape parameter of machine k, respectively, and t_k^d is the degradation of machine k at time t. The breakdown scenarios are generated randomly using Equation (4.1). Here, t_k^d is the nominal degradation, which is taken as the average degradation level between the start and end of a specific job, i.e., the degradation at the middle of the job. For instance, let us consider 3 operations (i.e., job 1 operation 2, job 2 operation 2, and job 3 operation 2) until the next CBM happens. The degradation of machine k at time t for P_{12}, P_{22}, and P_{32} is calculated based on Equations (4.2), (4.3), and (4.4), respectively. Here, P_{12} is the first operation, P_{22} is the second operation, and P_{32} is the third operation on machine k. Once a CBM is carried out, the value of t_k^d is set back to 0 and the same calculations are repeated until the next CBM.

$$t_k^d = (DR_{12} * P_{12})/2 \tag{4.2}$$

$$t_k^d = DR_{12} * P_{12} + (DR_{22} * P_{22})/2 \tag{4.3}$$

$$t_k^d = DR_{12} * P_{12} + DR_{22} * P_{22} + (DR_{32} * P_{32})/2 \tag{4.4}$$

If a breakdown occurs, the operation affected by it is immediately suspended and a CM (minimal repair) is carried out to fix the machine immediately. The operation that has been suspended is instantly resumed after CM. Following a CM, the machine's age is assumed to be restored to its age at the moment of failure, i.e., "as bad as old" rather than "as good as new". Similar to the CBM duration, the duration of CM is assumed uncertain and is modelled as a uniformly distributed random number. Figure 4.1 shows a sample 4-job and 3-machine example with CBM and CM.

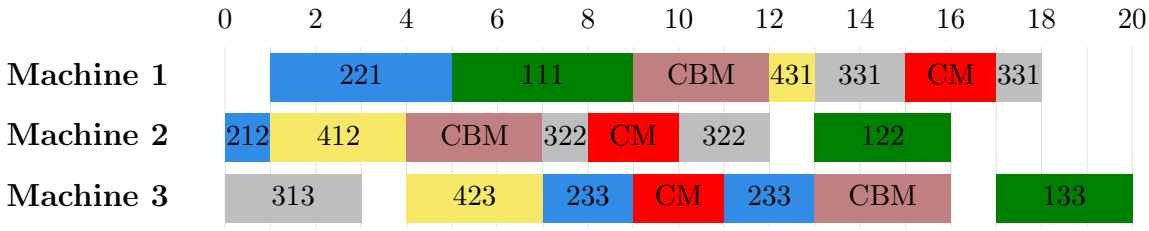


Figure 4.1: A sample schedule with CBM and CM.

4.2 Mathematical Formulation

Before going through the simulation-optimization as a solution approach considered with random degradation of machines, random failures, and uncertain CBM and CM duration, we present a mathematical formulation for the JSSP under consideration. The stochastic parameters are the random degradation of machines, random failures, and uncertain CBM and CM duration, which can be determined with certainty beforehand in each processing position. Although the optimum solution of the stochastic JSSP cannot be found using this mathematical model, it can be used to undertake an *ex post* evaluation of the solutions generated using the simulation-optimization approach to be presented in Section 5.3.2.

4.2.1 Model Constraints

Based on the problem definition and assumptions that were provided earlier, the constraints for the model can be formulated as follows:

$$\sum_{i=1}^{n} X_{ikl} = 1 \qquad k = 1, 2, ..., m;\ l = 1, 2, ..., n \tag{4.5}$$

$$\sum_{l=1}^{n} X_{ikl} = 1 \qquad i = 1, 2, ..., n;\ k = 1, 2, ..., m \tag{4.6}$$

$$q_{kl} = \sum_{i=1}^{n} X_{ikl} P_{ip} \qquad k = 1, 2, ..., m;\ l = 1, 2, ..., n;\ p = 1, 2, ..., m \tag{4.7}$$

$$DR_{kl} = \sum_{i=1}^{n} X_{ikl} DR_{ip} \qquad k = 1, 2, ..., m;\ l = 1, 2, ..., n;\ p = 1, 2, ..., m \tag{4.8}$$

$$C^0_{kl} = S^0_{kl} + q_{kl} \qquad k = 1, 2, ..., m;\ l = 1, 2, ..., n \tag{4.9}$$

$$S^0_{kl} \geq C^0_{k[l-1]} + t^p_k Y_{kl} \qquad k = 1, 2, ..., m;\ l = 2, 3, ..., n \tag{4.10}$$

$$\Big(2 - \sum_{k=1}^{m} X_{iku} r_{ipk} - \sum_{k=1}^{m} X_{ikl} r_{i(p+1)k}\Big) M + \sum_{k=1}^{m} S^0_{kl} r_{i(p+1)k} \geq \sum_{k=1}^{m} C^0_{ku} r_{ipk}$$
$$i = 1, 2, ..., n;\ l = 1, 2, ..., n;\ u = 1, 2, ..., n;\ p = 1, 2, ..., m-1 \tag{4.11}$$

$$S^0_{k1} \leq M\Big(1 - \sum_{p=1}^{n} r_{i1k} X_{ik1}\Big) \qquad i = 1, 2, ..., n;\ k = 1, 2, ..., m \tag{4.12}$$

$$b_{k[l-1]} + DR_{k[l-1]} q_{k[l-1]} - M Y_{kl} \leq D^*_k \quad k = 1, 2, ..., m;\ l = 2, 3, ..., n; \tag{4.13}$$

$$b_{k1} = 0 \qquad k = 1, 2, ..., m \tag{4.14}$$

$$b_{kl} = a_{k[l-1]}(1 - Y_{kl}) \qquad k = 1, 2, ..., m;\ l = 2, 3, ..., n \tag{4.15}$$

$$a_{kl} = b_{kl} + DR_{kl}q_{kl} \qquad k = 1, 2, ..., m;\ l = 1, 2, ..., n; \tag{4.16}$$

$$C^w_{kl} = S^w_{kl} + q_{kl} + \xi_{kl}t^r_k \qquad k = 1, 2, ..., m;\ l = 1, 2, ..., n \tag{4.17}$$

$$S^w_{kl} \geq S^0_{kl} \qquad k = 1, 2, ..., m;\ l = 1, 2, ..., n \tag{4.18}$$

$$\Big(2 - \sum_{k=1}^{m} X_{iku}r_{ipk} - \sum_{k=1}^{m} X_{ikl}r_{i(p+1)k}\Big)M + \sum_{k=1}^{m} S^w_{kl}r_{i(p+1)k} \geq \sum_{k=1}^{m} C^w_{ku}r_{ipk}$$
$$i = 1, 2, ..., n;\ l = 1, 2, ..., n;\ u = 1, 2, ..., n;\ p = 1, 2, ..., m-1 \tag{4.19}$$

$$S^w_{kl} \geq C^w_{k[l-1]} + t^p_k Y_{kl} \qquad k = 1, 2, ..., m;\ l = 2, 3, ..., n \tag{4.20}$$

$$S^0_{kl}, S^w_{kl}, C^0_{kl}, C^w_{kl}, a_{kl}, b_{kl} \geq 0 \qquad X_{ikl}, Y_{kl} \in \{0, 1\}. \tag{4.21}$$

Constraints (4.5) and (4.6), respectively, ensure that a job can be processed by only one machine and a machine can perform only one job at a time. Constraint (4.7) states the job duration that is being processed in the l^{th} position on machine k and constraint (4.8) calculates the degradation level of a job that is being processed in the l^{th} position on machine k. Constraint (4.9) calculates the job completion time that is scheduled to be processed in the l^{th} position on machine k. Constraint (4.10)

guarantees that the planned start time of a job on a machine is later than the planned completion time of the previous job on that machine plus the duration of any required CBM on the machine. Constraint (4.11) confirms that the planned start time of a job on a machine is later than the planned completion time of that job on the previous processing machine (i.e., the precedence relationship). Constraint (4.12) makes sure that the start time of the first operation on a machine is set to 0. Constraints (4.13) forces the CBM to be scheduled is not earlier than every D_k^* units of degradation on machine k. At the start of the horizon, constraint (4.14) sets all machines' degradation to zero. The machine's degradation is determined by constraints (4.15) and (4.16) before and after the operation scheduled in the l^{th} position on machine k, respectively. Constraint (4.17) calculates the actual completion time of a job by taking into account the job's start time, processing time, and needed CM time while processing the job. Constraints (4.18)–(4.20) ensure that the actual start time of a job is later than its planned time, nor the actual finish time of its process on the previous machine, nor the actual finish time of the previous job on the current machine, plus any required CBM time.

4.3 Objective Function

The main objective is to find a schedule that minimizes the makespan. Robustness is an important metric that represents the stability of the schedules under any unfavourable scenarios. The objective function is a composite form of two objectives: the average makespan and the 90th percentile of makespan. The average makespan represents the solution quality, i.e., the solution that gives the best expected value is near optimal. The first objective is formulated as,

$$OF_1 = \mathbb{E}\Big[\max_k \; C_{kn}^w\Big] \tag{4.22}$$

The second objective 90th percentile of the makespan represents the solution that captures most of the unfavourable scenarios, i.e., less sensitive to unfavourable scenarios. The overall objective function thereby leads to a more robust schedule. The second objective is formulated as,

$$OF_2 = min\ C^{0.9} \mid \mathbb{P}\left[C^{0.9} \geq \max_k C^w_{kn}\right] \geq 0.9 \tag{4.23}$$

Considering the two objective function in Equations (4.22) and (4.23), the objective function to be minimized is defined in Equation (4.24),

$$Z = \rho.OF_1 + (1-\rho).OF_2 \tag{4.24}$$

where ρ is the weight given to the average makespan is a real number between 0 and 1.

The mathematical formulation shown in Equation (4.5) - (4.24) is a MILP formulation that can be solved using off-the-shelf optimization solvers. However, commercial solvers can not solve a stochastic JSSP with random machine degradation rates, random failures, and uncertain duration of CBM and CM. This is because the stochastic parameters make it impossible for commercial solvers to solve realistic size problems in a reasonable amount of time.

Chapter 5

Methodology

In this chapter, the proposed methodology is described. First, the surrogate functions used to approximate the true makespan are defined, followed by the detailed description of the simulation-optimization algorithm.

5.1 Surrogate Functions

Surrogate functions are commonly used in the literature to alleviate computational costs [93–95]. The general concept in surrogate functions is to introduce a computationally less expensive problem called a surrogate problem and then replace the original problem with the surrogate one to approximate the objective function.

First, we note that given a schedule (sequence of operations to be performed on each machine) and a realization of the uncertain parameters (deterioration rate, time of breakdowns, maintenance duration), one can evaluate the makespan using simulation. However, simulation is computationally expensive, especially when a small confidence interval is required, and there is an exponential number of possible schedules, i.e., $(n!)^m$, where n is the number of jobs and m is the number of machines. Thus, except for "toy" problems, finding the optimal schedule using simulation is practically impossible.

Alternatively, we propose the approximation of the true makespan function by using surrogate functions that are less expensive to compute. In particular, we use surrogate functions that assume the deterioration rate, time of breakdowns, and maintenance duration are deterministic parameters, so that maintenance stops are pre-determined, and that breakdowns happen at known times. Evaluating such a function is pretty straightforward, so that it can be "optimized" using a population-based metaheuristic like GA. Once the best schedule and its respective makespan are found based on the surrogate functions, we simulate the schedule to find its true expected makespan that accounts for stochastic degradation of machines, random

breakdowns, and uncertain CBM and CM duration.

5.2 Oracles

Surrogate functions are defined as "Oracles" throughout the text. The definition of the term "Oracle" from the Merriam-Webster dictionary stipulates "2a. A person giving wise or authoritative decisions or opinions" [96]. In this thesis, Oracles are known as self-contained entities that evaluate job sequences and return a performance measure. The results obtained from oracles are later used by the simulation to explore the best job sequence based on true objective function. To analyze the job sequence, $(v+3)$ oracles are used and return the best found schedule's makespan as performance metrics. We approximate the true makespan function using $(v+3)$ oracles, described as follows:

Oracle 1, denoted as $Z_1(X)$, represents schedule X's makespan value without any consideration of maintenance or machine breakdown. The schedule's makespan is calculated directly assuming that all machines are available throughout the planning horizon.

Oracle 2, denoted as $Z_2(X)$, represents the schedule's makespan including CBM performed for every machine. All degradation rates are assumed deterministic and are set to $1/\eta_{ij}$, their expected values in the stochastic case. Hence, for a given schedule X, the accumulated degradation of a machine after each operation can be easily computed, and if it exceeds the degradation threshold, a CBM is scheduled. The CBM's duration is set equal to its mean value in the uncertain case. With that, one can easily compute the schedule's makespan by treating CBM as operations with known positions and duration.

Oracle 3, denoted as $Z_3(X)$, represents the schedule's makespan including CBM and breakdowns that occur at the machine's *Mean-Time-To-Failure* (MTTF). The CBM portion of this oracle is computed exactly as in $Z_2(X)$. For a failure process governed by a Weibull distribution, the average amount of time machine k runs until it fails is calculated as $MTTF_k = \theta_k \Gamma\left(1+\frac{1}{\beta_k}\right)$. Upon knowing the position and duration of all CBM and CM activities, one can easily compute the schedule's makespan.

Oracles 4 to v+3, denoted as $Z_4(X)$,..., $Z_{v+3}(X)$, each represents the schedule's makespan including CBM and one of v predetermined breakdown scenarios. The

CBM portion of this oracle is computed exactly as in $Z_2(X)$. The breakdown scenarios are generated randomly using Equation (4.1) and v predetermined breakdown scenarios are selected by solving a diversity maximization problem. Upon knowing the position and duration of all CBM and CM activities, one can easily compute the schedule's makespan.

Real makespan model, denoted as $Z_r(X)$, represents the objective function value considering the actual condition of machines. The stochastic parameters are machine degradation rates, time of breakdowns, and duration of maintenance actions. This function is evaluated using simulation for a large number of scenarios, each constitutes a realization of the stochastic parameters drawn at random according to the probability distributions described earlier. The average (OF_1) and the 90th percentile (OF_2) values over all scenarios are calculated, and the weighted average based on Equation (4.24) is declared as the estimated objective value for schedule X.

5.3 Solution Algorithm

The solution algorithm consists of an inner loop and an outer loop. In the inner loop of the algorithm, GA is applied to independently optimize the sequence of jobs. Then, the objective function is evaluated with the job sequence found in the GA procedure using simulation. SA is used in the algorithm's outer loop to prevent the algorithm from premature conversion. This section first describes the metaheuristics GA applied to JSSP in detail. Next, the simulation-optimization algorithm is presented with a detailed description.

5.3.1 Genetic Algorithm Applied to the JSSP

The GA applied to JSSP is described in the next subsections. Initially, GA starts with the population creation and chromosome representation, followed by the crossover process using roulette wheel procedure. Then, the gene selection procedure is described, the mutation procedure is explained with the right shifting method, and the entire genetic algorithm is revised.

5.3.1.1 Chromosome Representation and Initial Population Creation

A chromosome consists of several genes. Each chromosome is a solution that is represented by an array as described in Section 3.4.1. Binary or permutation representations are used as the standard coding technique for the illustration of solutions found in the early studies of GA. The drawback of using standard coding technique in modern optimization problem is, it will add infeasible solutions (individuals) to the population. To overcome this drawback, we used chromosome representation proposed by Bierwirth [97] to code the solutions. This technique not only saves the computer resources to check or fix infeasible solutions (individuals) but also covers all the feasible solutions considering job repetition permutation.

Let us consider we have J_i set of jobs, each has a predetermined number of operations. Each operation needs to be processed on a predetermined machine k. The number of operations of each job is the same as the number of machines k in the job shop. For example, in the case of 3-job and 4-machine, a chromosome (individual) can be stated as (J_1, J_2, J_1, J_2, J_3, J_2, J_3, J_1, J_3, J_1, J_3, J_2). The important consideration is J_i appears in the chromosome at most k times and their position in the chromosome are chosen randomly. Considering the following given machine sequence (1, 3, 4, 2; 1, 2, 4, 3; 3, 2, 1, 4) and by reading the randomly created chromosome from left to right (J_1, J_2, J_1, J_2, J_3, J_2, J_3, J_1, J_3, J_1, J_3, J_2) operations sequence can be written as (O_{11}, O_{21}, O_{13}, O_{22}, O_{33}, O_{24}, O_{32}, O_{14}, O_{31}, O_{12}, O_{34}, O_{23}). The operations sequence created can be evaluated based on any performance metric of interest by giving it to a schedule builder. To evaluate the performance metrics of the population set, the oracles explained in Section 5.2 are utilized as schedule builders. The pseudo-code for creating starting population size of $popSize$ is described in Algorithm 2. An illustration of population creation for $i = 3$ and $k = 4$ is shown in Figure 5.1 step by step.

5.3.1.2 Crossover Process

After the creation of initial population, the crossover process is an important step to create new chromosomes (individual or children) from parents. The new chromosome will be created from the two other chromosomes known as parents. Bierwirth et al. [88] proposed Precedence Preservative Crossover (PPX) is used in this thesis. The

Input:

Number of jobs i;

Number of machines k;

Population size *popSize*;

Output: An array *population* returns *popSize* individuals.

Create an empty *population* array;

while *The size of population* $<$ *popSize* **do**

Create an *individual* incremental integer array starting from 1 up to $i * k$;

Randomly permutate the genes of the *individual* array;

foreach *gene* $\in$ *individual* **do**

$gene \leftarrow gene \mod i + 1$

end

Append *individual* to *population* array;

end

Algorithm 2: Initial population creation pseudo code.

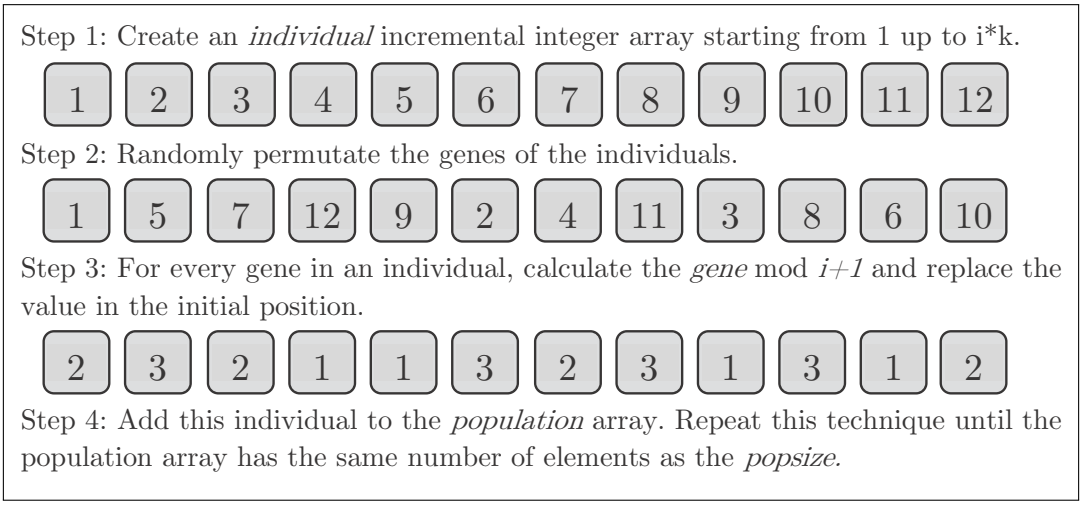


Figure 5.1: Population creation example.

procedure of PPX is described as follows: Consider the length of the parent chromosome is n. First, select two parents chromosomes and create an array of n size with a randomly filled gene set of $\{1, 2\}$. Based on the created gene sequence array, extract a gene from one of the parent chromosome and append the same gene in the new child array with the deletion of the same gene on the other parent chromosome. Follow this procedure until the child array has n elements and both parent chromosomes are

empty. An illustration of step by step crossover process is shown in Figure 5.2.

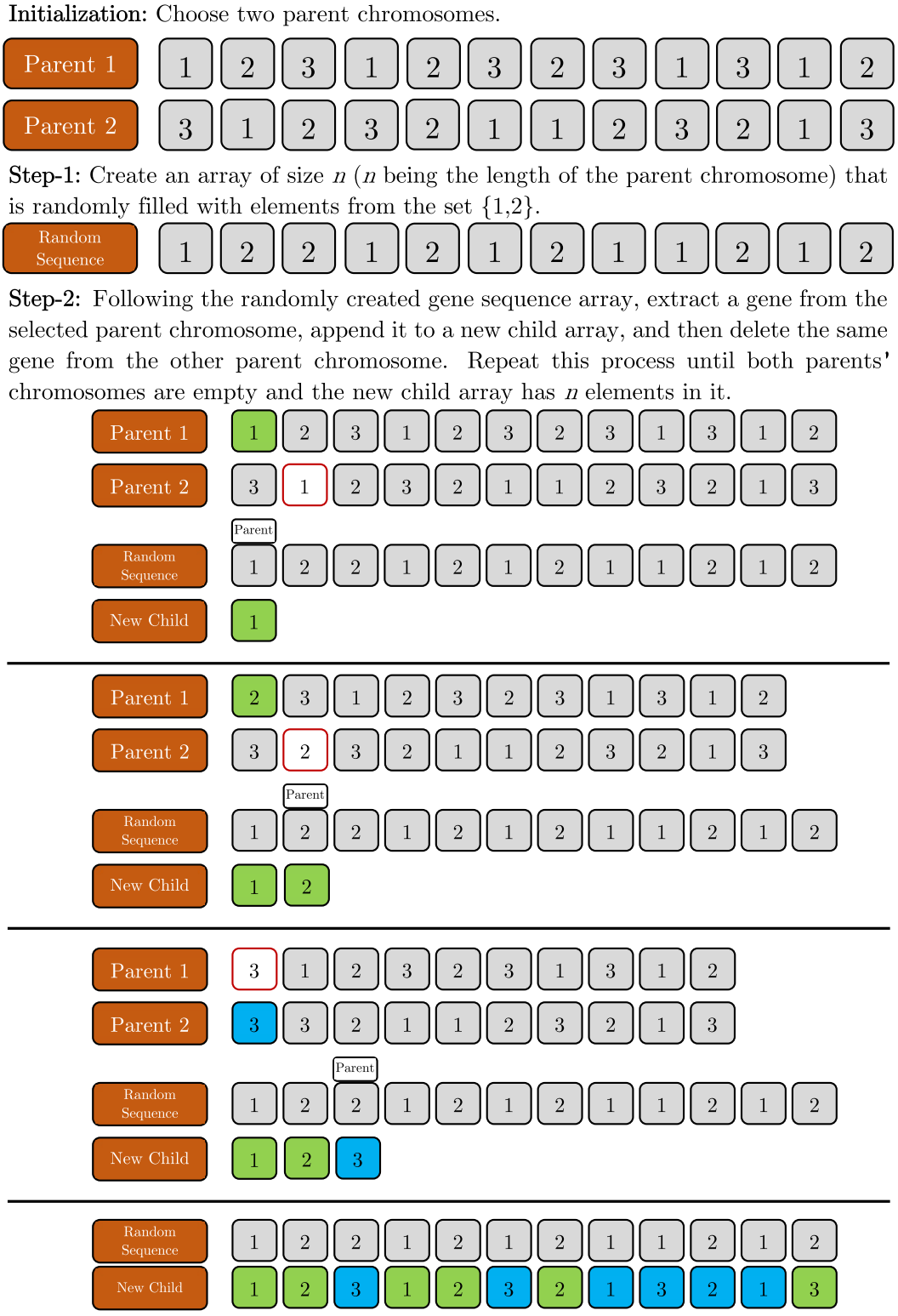


Figure 5.2: Example of PPX crossover process.

5.3.1.3 Chromosomes Selection

To select which chromosomes will participate in the future generations crossover procedure, Roulette wheel selection was used [98]. Roulette wheel selection is also called stochastic sampling with replacement [99]. A chromosome is given a survival chance for future generation crossover based on its fitness value from the roulette wheel selection process. In this context "Fitness" refers to the evaluation of the chromosomes based on a given metric of interest. The oracles described in Section 5.2 are the schedule builders as well as provide the fitness evaluation of the chromosomes, i.e., the makespan evaluation of the chromosomes. The evaluation of individual chromosome fitness is calculated by taking the reciprocal of each individual chromosome's makespan value (i.e., $f_i = (1/f_{makespan})$). The roulette wheel is designed on the relative fitness evaluation of each individual i.e., the ratio of individual chromosome fitness and total fitness. This relative fitness evaluation showed that individuals (chromosomes) with a higher relative fitness value, which is shown as a bigger slot size in the roulette wheel, also have a higher survival chance than smaller slot size individuals. So, the wheel is rotated randomly and the individuals are selected and added one by one to the array for future generation crossover process.

Roulette wheel selection based on fitness is shown in Figure 5.3. It is obvious that individuals who have a bigger slot size in the wheel also have a better survival chance. The roulette wheel is implemented based on Algorithm 3.

5.3.1.4 Mutation

To add diversity to the generated individuals, mutation is performed. Gene mutation rate and population mutation rate are two parameters that control the mutation process in individuals. Gene mutation rate is the percentage of genes in an individual that are mutated, whereas population mutation rate stipulates the percentage of individuals in a population set that is going to be mutated. These two parameters are very important to bring versatile diversity among individuals. The mutation process starts by selecting random individuals based on the population mutation rate. Then, the genes of these selected individuals are mutated based on the gene mutations rate. We used the right shift mutation procedure in the proposed algorithm. For the right shifting procedure, first tag the genes that are going to be mutated based on the

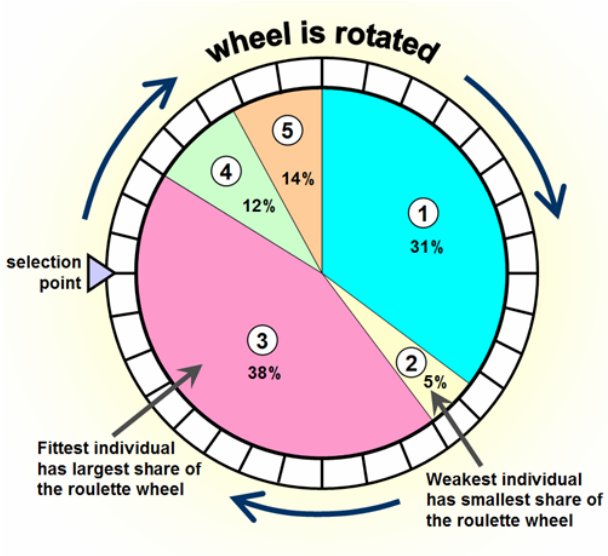


Figure 5.3: Roulette Wheel [100].

Input:

Population size n;

Maximum population size $popSize$;

Output: A new population pop

Step 1. Evaluate the fitness of each individual, $f_i = (1/f_{makespan})$, in the population;

Step 2. Compute the probability (slot size), p_i, of selecting each individual in the population: $p_i = f_i / \sum_{j=1}^{n} f_j$;

Step 3. Compute the cumulative probability, q_i, for each individual: $q_i = \sum_{j=1}^{i} p_j$;

Step 4. Generate an uniform random number, $r \in [0, 1]$;

Step 5.

if $r < q_1$ **then**
| select the first individual x_1

else
| select individual x_i such that $q_{i-1} < r < q_i$

end

Step 6. Repeat steps 4-5 until $popSize$ individuals were selected;

Algorithm 3: Roulette wheel procedure [98].

gene mutation rate in selected individuals. Finally, the tagged genes are shifted to the right and mutated individuals (chromosomes) have their genes replaced by the tagged genes.

Figure 5.4 shows the right shifting procedure of mutation in individuals. With the right shifting mutation process, the newly created individuals are feasible, so there is no waste of computer resources to check, validate, and fix the mutated individuals. Algorithm 4 describes the GA applied to the JSSP.

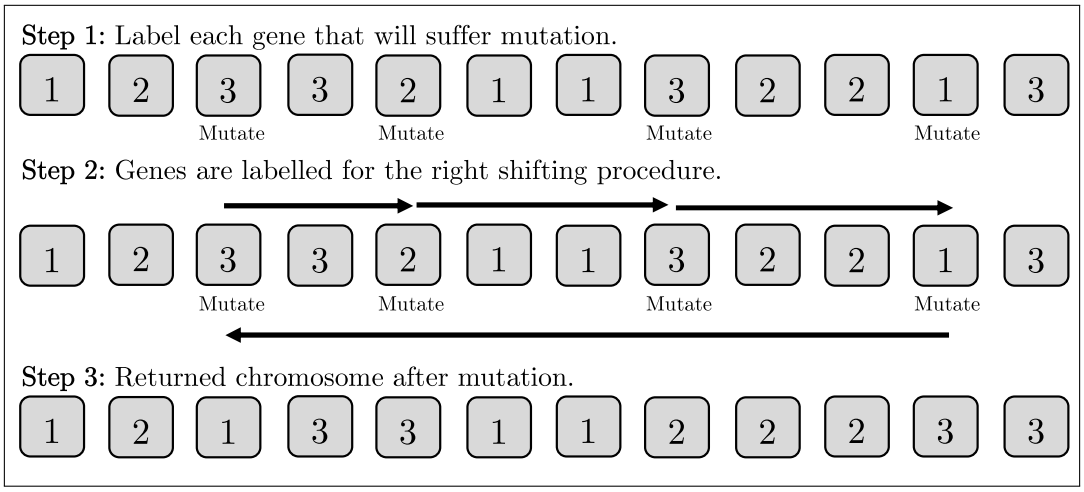


Figure 5.4: Mutation right shift.

5.3.2 The Simulation-Optimization Algorithm

The proposed simulation-optimization algorithm looks for robust schedules. It includes an inner loop in every iterations that uses parallel GAs to "optimize" oracles. The inner loop starts with population creations, some of them inherited from the previous iteration in the form of an "elite list", whereas others are randomly created. The elite list carries forward the best individuals of that current generation to the next generations, so that next generations individuals have some good genes characteristics from the previous elite list individuals. The elite list has a specified size and follows the FIFO rule. As soon as the elite list reaches its maximum number of individuals, the first individual to enter the list leaves it and creates a space for the newly found elite individual, which is inserted into the list. At the very beginning of the GA, all chromosomes are created randomly to generate the initial population.

Input:

Oracle *orac*;

Elite array *elitelist*;

Maximum number of generation *maxGen*;

Maximum size of population *popSize*;

Population mutation rate $popMutationRate$;

Gene mutation ratio $geneMutationRate$;

Output: A job sequence with repetition (x) that represents the approximation of the Oracle's optimal solution.

Initialization. Randomly create a population set (*Popold*) with *popSize* individuals;

Randomly choose and substitute individuals from *Popold* with individuals in *elitelist*;

Create an empty array of *bestScoreSelection*;

while $gen < maxGen$ **do**

Create a empty array of *Popnew*;

while *Popnew number of elements is smaller than popSize* **do**

Append to *Popnew* individuals generated by the PPX crossover procedure by randomly selecting parents in *Popold*;

end

Mutate the entire *Popnew* array following the $popMutationRate$ and $geneMutationRate$;

Stack the *Popold* and the newly created *Popnew*;

Using the biased roulette wheel procedure, select up to *popSize* individuals and set it as *Popold*;

Append to the *bestScoreSelection* array the minimum objective value found in *Popold* and it's corresponding chromosome;

end

Return x as the chromosome with the least objective function value from the *bestScoreSelection* array;

Algorithm 4: The GA applied to the JSSP.

The proposed simulation-optimization algorithm works as follows: $(v+3)$ parallel GA starts the inner loop and using one of the oracles $Z_1(X), \ldots, Z_{v+3}(X)$ as performance metric, each of the GAs tries to optimize and improve the solutions. After the GA procedures are finished, the individuals that represent the best found solution for each oracles are labeled $x_1^*, \ldots, x_{v+3}^*$, for $Z_1(X), \ldots, Z_{v+3}(X)$, respectively. Then $Z_r(x_1^*), \ldots, Z_r(x_{v+3}^*)$ are estimated through simulation, and the solution among them that minimizes Z_r is labeled x^*.

To introduce diversity in the breakdown scenarios, we consider oracle 4 to $(v+3)$ with v random breakdown scenarios. We select the v scenarios among a large number of randomly-generated scenarios (i.e., those generated for the simulation) such that the v selected scenarios have the maximum distance from each other. We select these v scenarios by solving a diversity maximization problem [101]. In a diversity maximization problem, one of the important elements is the scenario distance matrix. The following procedures show how the scenario distance matrix is calculated among a large number of scenarios. Suppose we have three breakdown matrix scenarios with three machines that are as follows:

$$Scenario\,1 = \begin{bmatrix} 4 & 7 & 9 \\ 3 & 10 & \\ 5 & & \end{bmatrix}, \quad Scenario\,2 = \begin{bmatrix} 4 & 6 & \\ 3 & 8 & \\ 7 & 9 & 3 \end{bmatrix}, \quad Scenario\,3 = \begin{bmatrix} 3 & 11 & 6 \\ 6 & & \\ 5 & 8 & \end{bmatrix}$$

The number of breakdowns in each of the scenarios is unequal. The maximum number of breakdowns present in the scenario matrix is 3. So, the first step is to make all the breakdown matrix scenarios with an equal number of breakdowns without affecting the values of the respective breakdown matrix. To do that, we added the last value of that respective row until that matrix is square and has the same number of breakdowns as the highest breakdown matrix scenario.

$$Scenario\,1 = \begin{bmatrix} 4 & 7 & 9 \\ 3 & 10 & 10 \\ 5 & 5 & 5 \end{bmatrix}, \quad Scenario\,2 = \begin{bmatrix} 4 & 6 & 6 \\ 3 & 8 & 8 \\ 7 & 9 & 3 \end{bmatrix}, \quad Scenario\,3 = \begin{bmatrix} 3 & 11 & 6 \\ 6 & 6 & 6 \\ 5 & 8 & 8 \end{bmatrix}$$

Then, the distance between breakdown time is calculated within each scenario, and the absolute value is taken.

$$Scenario\ 1 = \begin{bmatrix} 4 & 3 & 2 \\ 3 & 7 & 0 \\ 5 & 0 & 0 \end{bmatrix}, \quad Scenario\ 2 = \begin{bmatrix} 4 & 2 & 0 \\ 3 & 5 & 0 \\ 7 & 2 & 6 \end{bmatrix}, \quad Scenario\ 3 = \begin{bmatrix} 3 & 8 & 5 \\ 6 & 0 & 0 \\ 5 & 3 & 0 \end{bmatrix}$$

Next, the distance between the breakdown time from one scenario to another is calculated, and its absolute value is taken.

$$The\ distance\ in\ breakdown\ time\ between\ scenarios\ 1\ and\ 2, \Delta_{12} = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 2 & 0 \\ 2 & 2 & 6 \end{bmatrix}$$

$$The\ distance\ in\ breakdown\ time\ between\ scenarios\ 1\ and\ 3, \Delta_{13} = \begin{bmatrix} 1 & 5 & 3 \\ 3 & 7 & 0 \\ 0 & 3 & 0 \end{bmatrix}$$

$$The\ distance\ in\ breakdown\ time\ between\ scenarios\ 2\ and\ 3, \Delta_{23} = \begin{bmatrix} 1 & 6 & 5 \\ 3 & 5 & 0 \\ 2 & 1 & 6 \end{bmatrix}$$

In the last step, the values obtained from the previous steps are summed up within each breakdown time distance matrix. Finally, the scenario distance matrix is created with the summed up values.

Sum of values in Δ_{12} = 0+1+2+0+2+0+2+2+6 = 15

Sum of values in Δ_{13} = 1+5+3+3+7+0+0+3+0 = 22

Sum of values in Δ_{23} = 1+6+5+3+5+0+2+1+6 = 29

$$The\ scenario\ distance\ matrix, \Delta_{ij} = \begin{bmatrix} 0 & 15 & 22 \\ 0 & 0 & 29 \\ 0 & 0 & 0 \end{bmatrix}.$$

The diversity maximization problem of v random scenarios selection is given below:

Maximize, $D = \sum_{i=1}^{n} \sum_{j=1}^{n} \Delta_{ij} x_i x_j$

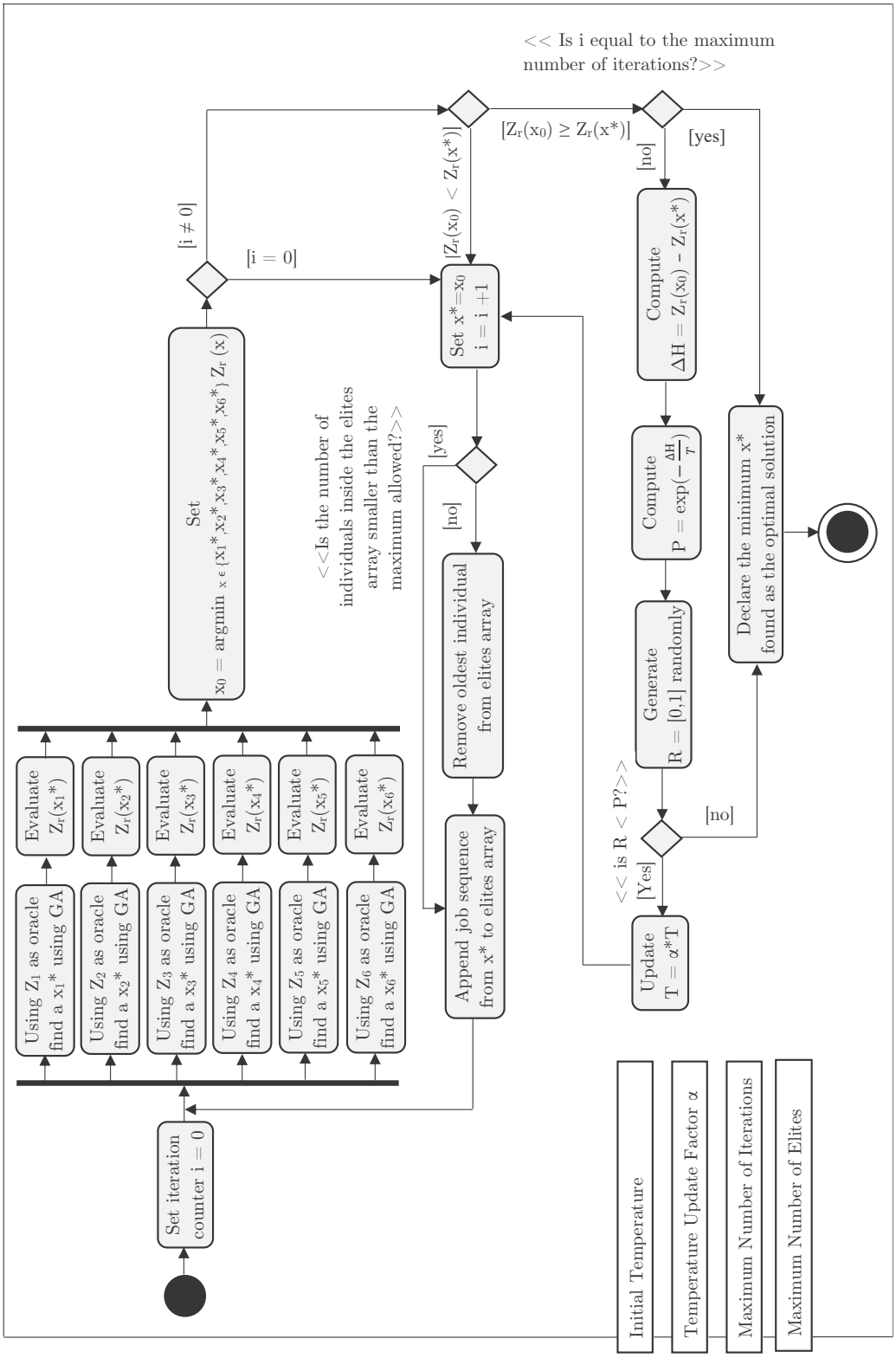
subject to,

$$\sum_{i=1}^{n} x_i = v$$

$$x_i \in \{0, 1\}, \forall i = 1, ..., n,$$

where Δ is the scenario distance matrix, n is the total number of scenarios, x_i is a binary decision variable that takes value 1 if the scenario is selected and 0 otherwise, and v is the number of scenarios to be selected (here, v=3).

The outer (main) loop updates the elite list and enacts the stopping rule. We employ a stopping rule inspired by SA, thus a suitable temperature schedule is selected with a temperature update factor α. The SA procedure in the outer loop accepts worse solutions than the current best solution to escape from the local optima and search for a better result. The algorithm is initiated with an empty elite list and a random best solution x_0. This random best solution is called the initial solution. At every iteration, if the solution x^* obtained from the inner loop is found to be better (i.e, has a lower Z_r value) than x_0, we set $x_0 \leftarrow x^*$, append x^* to the elite array, considering its maximum length, and proceed to the next iteration. If x^* does not improve on x_0, it still gets a chance for participation in the next iteration and being added to the elite list. If x^* does not improve on x_0, a uniform random number between R= [0, 1] is created. The probability of accepting a non-improving solution is calculated as $P = e^{\frac{-(x_0 - x^*)}{T}}$, where T is the temperature factor. If $R < P$ the temperature T is updated to $T * \alpha$, set $x_0 \leftarrow x^*$, and the inner loop of the algorithm is called. If $R \geq P$ or the maximum number of iterations is reached, the algorithm terminates and the best solution found throughout the algorithm's execution is declared as the approximate optimal solution. Algorithm 5 shows the simulation-optimization algorithm with GA in the inner loop and SA in the outer loop.



Algorithm 5: Simulation-optimization algorithm with GA in the inner loop and SA in the outer loop.

Chapter 6

Results and Discussion

This chapter includes all the experiments for the proposed algorithm and main discussions of the findings. There is a section for initial parameter estimation, followed by the final experiments and sensitivity analysis.

6.1 Initial Parameter Setting

To understand the behaviour of the proposed approach on makespan and run time, initial parameter setting is investigated where the GA parameters are varied. Final experiments are carried out based on the results obtained from the initial parameter setting. The algorithm was coded using Python in Anaconda Jupyter notebook. A computer with Intel Xeon Platinum 8276 (x4) with 2.20 GHz CPU and 32GB RAM was used to run the code.

The behaviour of GA is explored here by changing its parameter one by one. The parameters varied here are: Gene mutation rate, population mutation rate, population size, and the maximum number of generations. Even though a lot of effort have been made to find good values of the parameters for the proposed algorithm, these parameters must be used only as good initial starting points. The parameters found here and selected for final experiments are not optimal for all the problem instances. Many iterations and experiments are not included in this analysis since they are beyond the scope of this research.

The instance used for initial testing was created by expanding the Taillard [102] instances with the addition of parameters that are required for CBM and random breakdowns. Taillard instances include the processing times of operation in machine and the machine sequence of each job. For parameter setting phase, instance 'ta11' is chosen, which has 20-job and 15-machine. The time to perform CBM and CM are calculated based on the average processing time and multiplied by a factor between

0 to 1. The multiplication factor for CBM and CM are chosen as 0.8 and 0.6, respectively. The shape and scale parameters β_k and θ_k are set to 2.5 and 6, respectively. CBM threshold was set at 40 and the machine degradation rate was set to 0.25. The total number of runs was set to 25 and the maximum number of individuals on the elite list was set to 4. Six oracles are used, i.e., $v = 3$, to evaluate the schedule.

6.1.1 Gene Mutation Rate

The first parameter tested is the gene mutation rate. Gene mutation rate refers to the number of genes mutated in a randomly selected individual and followed the right shifting procedure discussed in Section 5.3.1.4. The gene mutation rate is varied from 0.05 to 0.45 with an increment of 0.05 to investigate the impact. In the case of gene mutation rate testing, the other parameters such as population mutation rate was fixed to 0.15, population size was fixed to 300, and the maximum number of generations was fixed to 200.

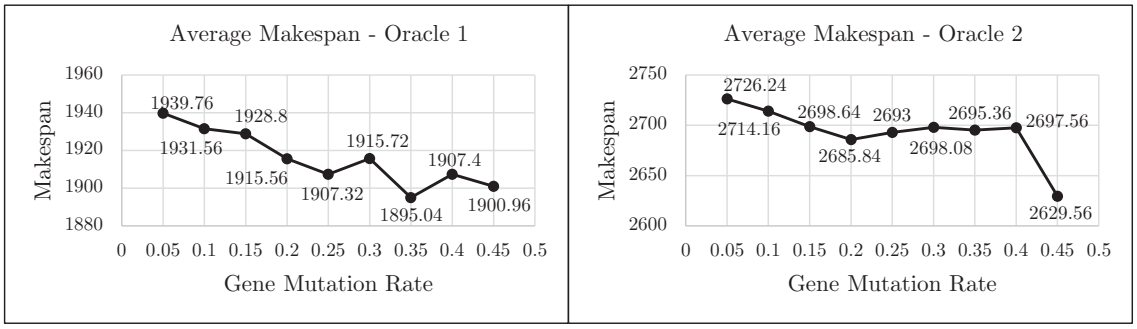


Figure 6.1: Impact of the gene mutation rate on the average makespan - Oracles Z_1 and Z_2.

The results from oracles Z_1 and Z_2, Z_3 and Z_4, Z_5 and Z_6 are shown in Figures 6.1, 6.2, and 6.3, respectively. There is no common trend found from the six oracles average makespan. The average makespan of oracle Z_1 and Z_2 decreases as the gene mutation rate increases, whereas the other oracle results are random. A gene mutation rate of 0.1 is recommended to avoid the degradation of the solution. The average run time is almost the same for all the oracles, thus not presented here.

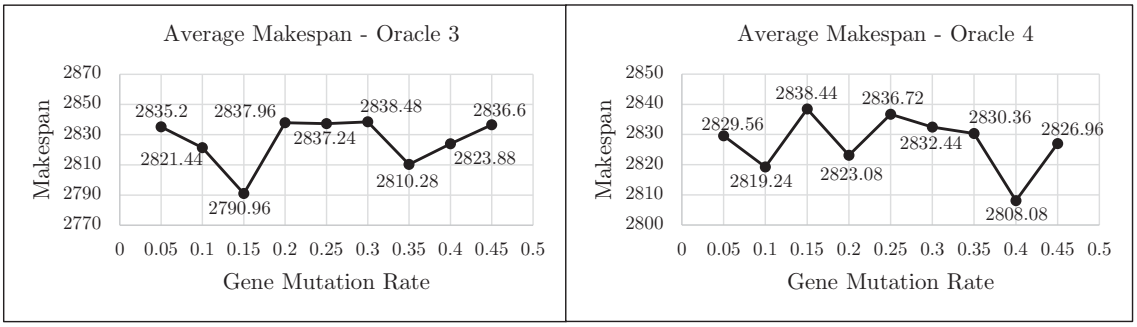


Figure 6.2: Impact of the gene mutation rate on the average makespan - Oracles Z_3 and Z_4.

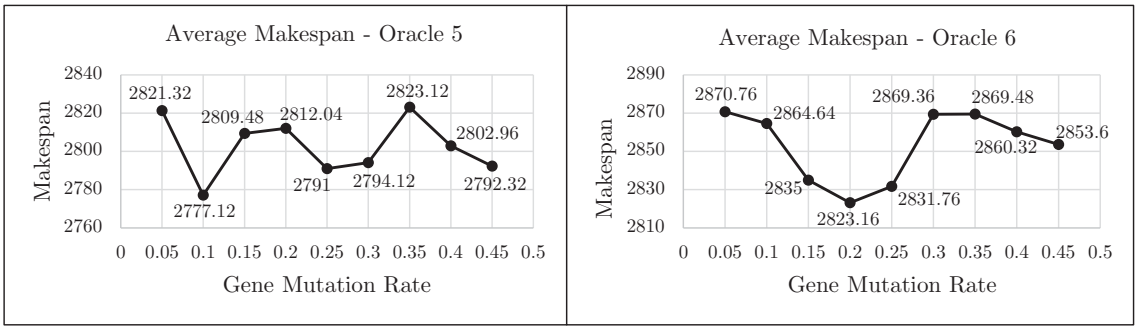


Figure 6.3: Impact of the gene mutation rate on the average makespan - Oracles Z_5 and Z_6.

6.1.2 Population Mutation Rate

The population mutation rate is another important parameter in the GA procedure of the proposed approach. The population mutation rate dictates how many individuals are randomly selected for mutation. To investigate this effect, the population mutation rate is varied from 0.05 to 0.45 with an increment of 0.05.

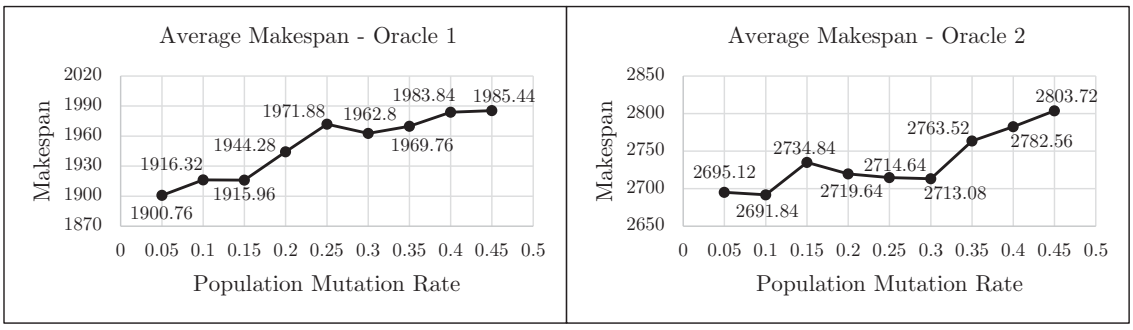


Figure 6.4: Impact of the population mutation rate on the average makespan - Oracles Z_1 and Z_2.

The population size was fixed to 300, the maximum number of generations was fixed to 200, and the gene mutation rate was fixed to 0.1. After the completion of 25 runs, the average makespan results from oracles Z_1 and Z_2, Z_3 and Z_4, Z_5 and Z_6 are shown in Figures 6.4, 6.5, and 6.6, respectively. The average makespan increases as the population mutation rate increases for all the oracles. A population mutation rate of 0.05 is recommended to obtain a good performance. The average run time is almost the same for all the oracles, thus not presented here.

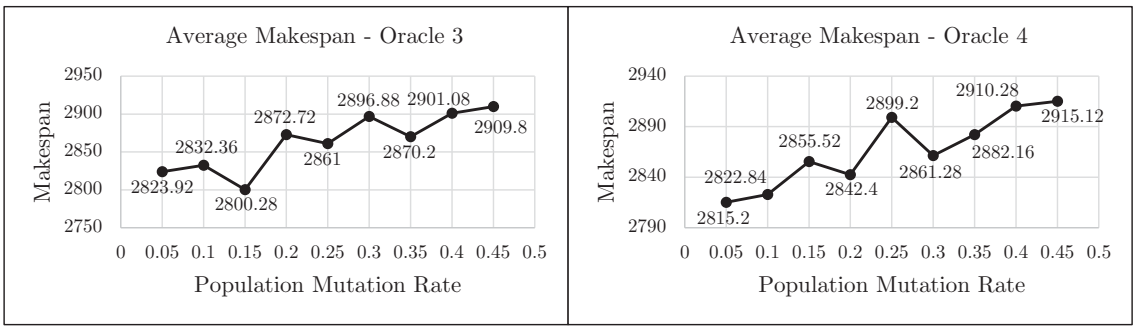


Figure 6.5: Impact of the population mutation rate on the average makespan - Oracles Z_3 and Z_4.

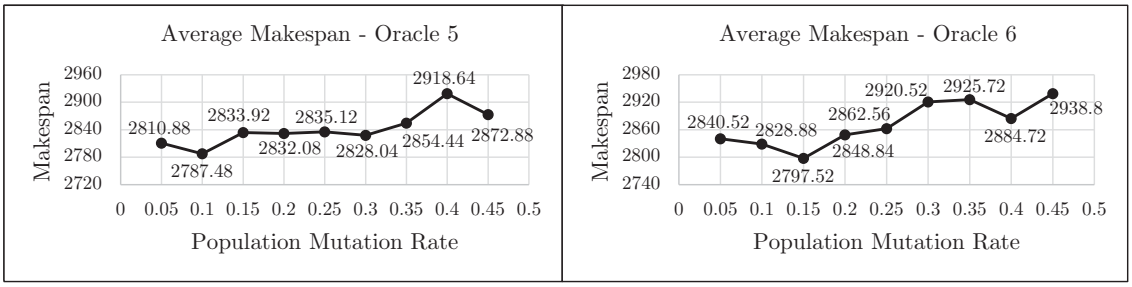


Figure 6.6: Impact of the population mutation rate on the average makespan - Oracles Z_5 and Z_6.

6.1.3 Population Size

Population size indicates how many individuals are created and how many among the created individuals is survived after each generation. Population size is directly proportional to the usage of computational resources, meaning that the more populations are created, the more computational resources are required. More usage of

computational resources directly leads to an increase in the average run time. So, population size needs to be balanced to achieve the desired level of performance with a reasonable amount of run time. In this round of testing, the population size is varied from 50 to 450 with an increment of 50. The other parameters were set as follows: gene mutation rate was fixed to 0.15, the population mutation rate was fixed to 0.1, and the maximum number of generations was fixed to 200. Figures 6.7, 6.8, 6.9, 6.10, 6.11, and 6.12 shows the impact of the population size on average makespan and average run time for oracles Z_1, Z_2, Z_3, Z_4, Z_5, and Z_6, respectively.

It is evident that with the increase in the population size, the average makespan is decreasing for all the oracles. But the overall performance of the algorithm did not improve when compared to the average run time for oracles. For example, considering oracle Z_4 and 50 population size as the base, the highest improvement of 7.66% is found for a population size 450 with an increase in average run time of 854.60%. So, considering the average improvement of makespan and average run time, a value of population size 200 is recommended.

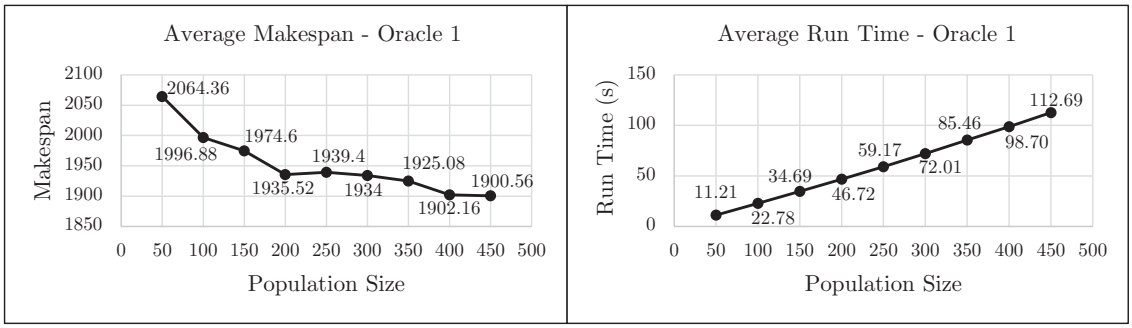


Figure 6.7: Impact of the population size on the average makespan and average run time - Oracle Z_1.

6.1.4 Maximum Number of Generations

The maximum number of generations has also an impact on the computational resources. The maximum number of generations stipulates how many crossover/mutation iterations of a population will be attempted until it stops. This parameter testing was designed to vary the maximum number of generations from 50 to 450 with an increment of 50. The gene mutation rate, population mutation rate, and population size were fixed to 0.1, 0.15, and 300, respectively. The impact of the maximum number

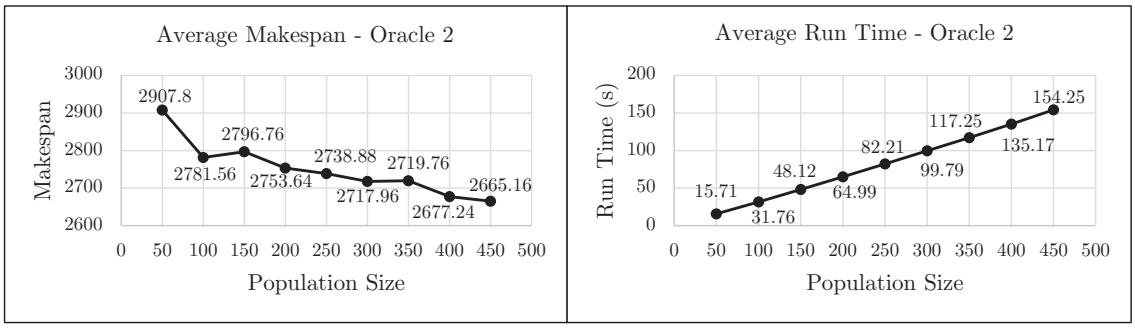


Figure 6.8: Impact of the population size on the average makespan and average run time - Oracle Z_2.

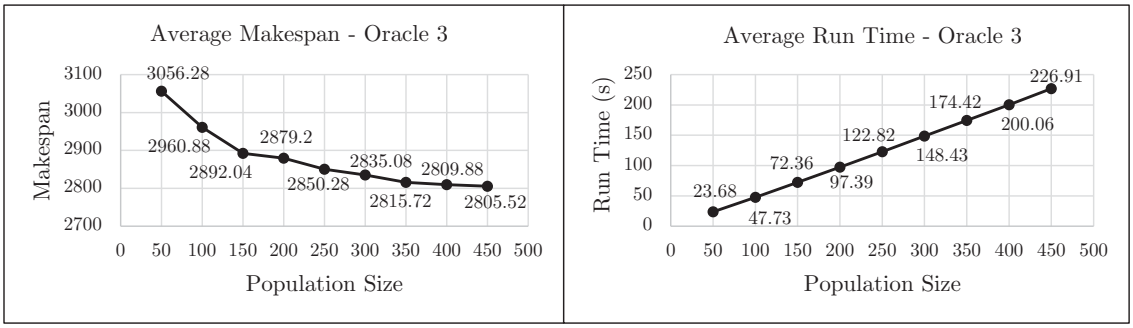


Figure 6.9: Impact of the population size on the average makespan and average run time - Oracle Z_3.

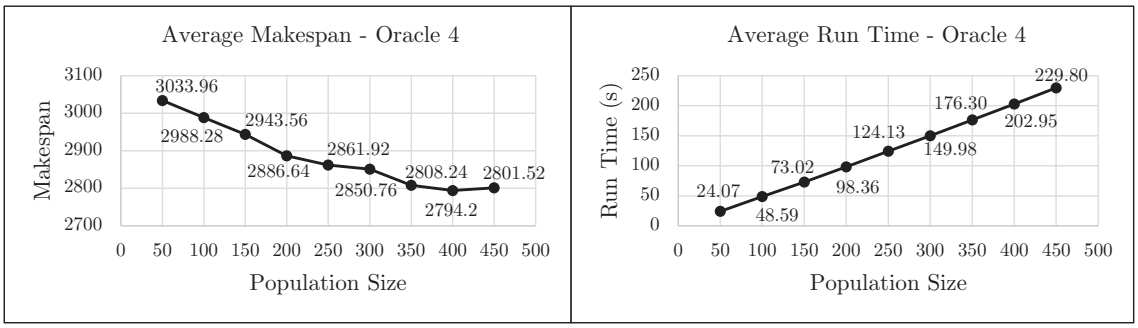


Figure 6.10: Impact of the population size on the average makespan and average run time - Oracle Z_4.

of generations on average makespan and average run time is shown in Figures 6.13, 6.14, 6.15, 6.16, 6.17, and 6.18 for oracles Z_1, Z_2, Z_3, Z_4, Z_5, and Z_6, respectively.

Similar to what was observed in the population size test, the maximum number of generations has significant impacts on average makespan and average run time. For instance, considering oracle Z_3 and 50 as the maximum number of generations as

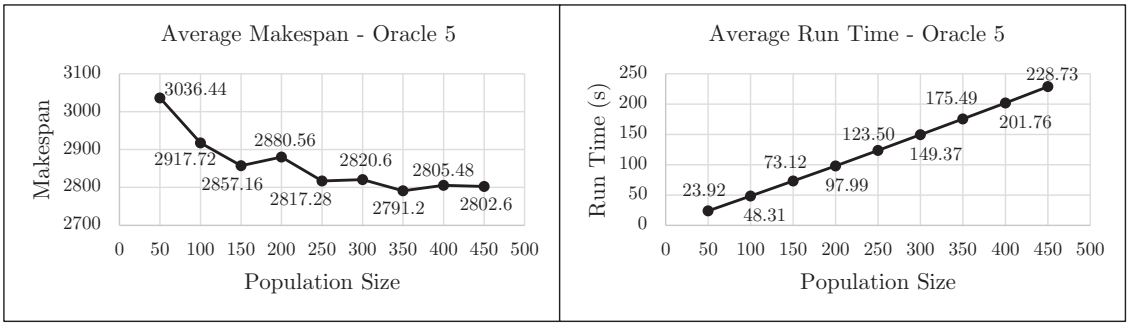


Figure 6.11: Impact of the population size on the average makespan and average run time - Oracle Z_5.

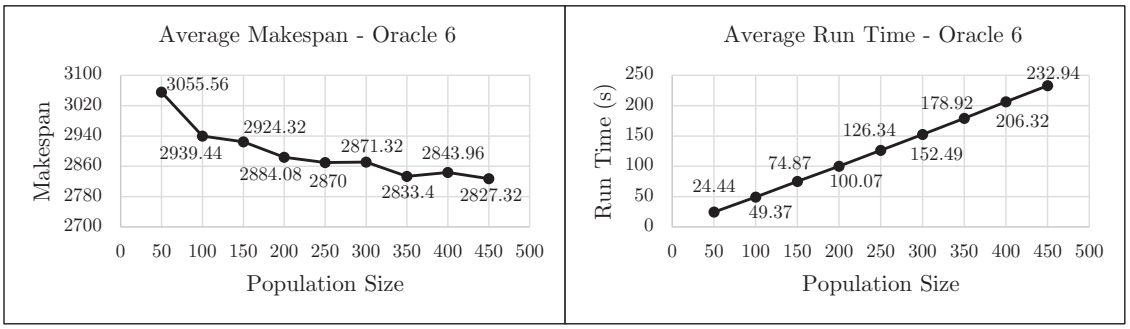


Figure 6.12: Impact of the population size on the average makespan and average run time - Oracle Z_6.

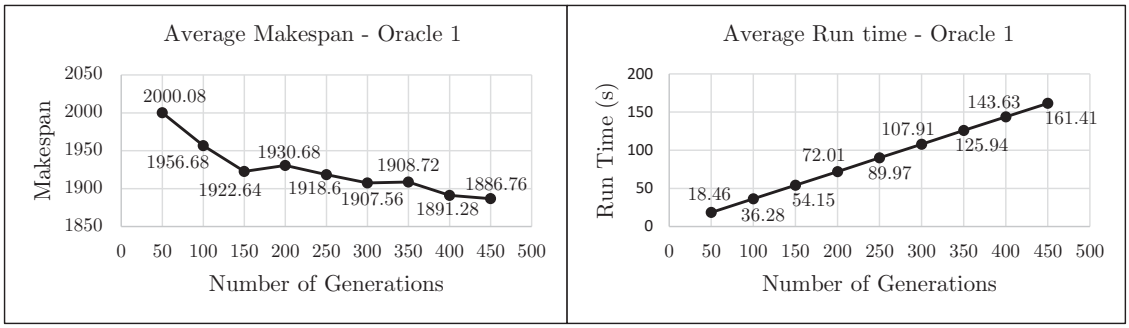


Figure 6.13: Impact of the maximum number of generations on the average makespan and average run time - Oracle Z_1.

the base, the highest improvement of 5.64% is found with an increase of 779.32% in run time. As the improvement of average makespan is slim with a steep increase of average run time, a value of the maximum number of generations 150 is recommended.

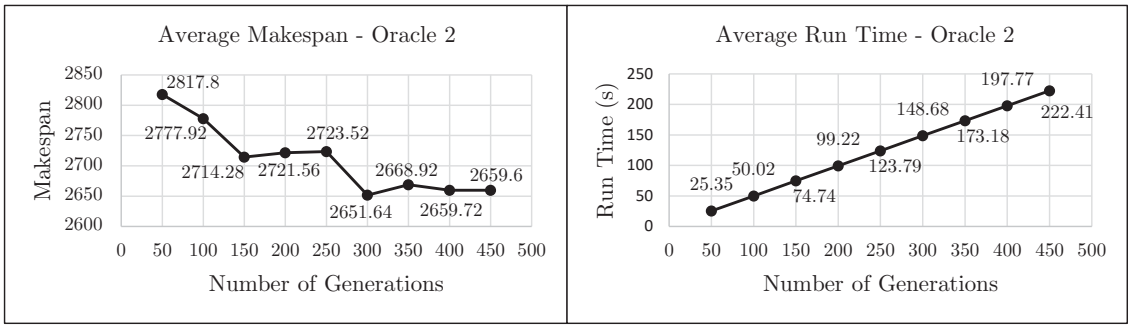


Figure 6.14: Impact of the maximum number of generations on the average makespan and average run time - Oracle Z_2.

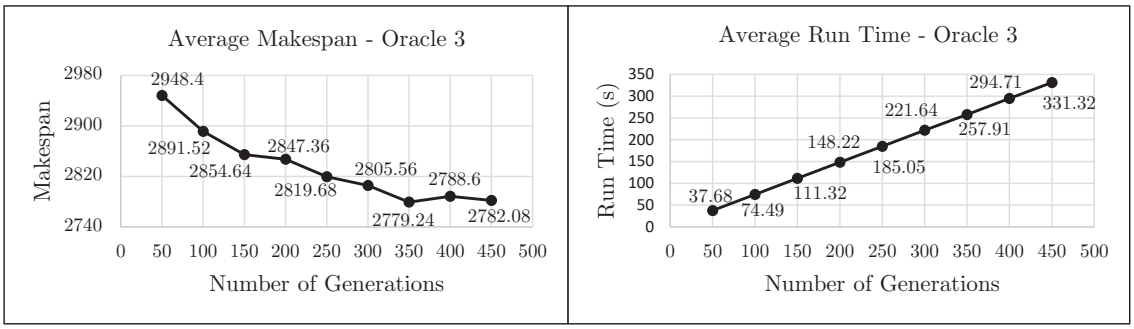


Figure 6.15: Impact of the maximum number of generations on the average makespan and average run time - Oracle Z_3.

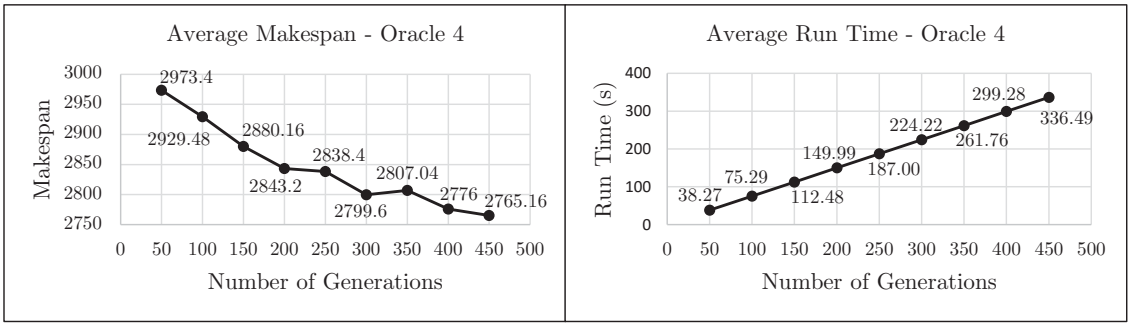


Figure 6.16: Impact of the maximum number of generations on the average makespan and average run time - Oracle Z_4.

6.2 Final Experiments

For final experiments, Six Taillard instances were considered to test the proposed approach performance. Table 6.1 depicts the Taillard instances.

There are three types of parameters for the final test: global parameters, GA parameters, and SA parameters. For the final test, the global parameters were set as

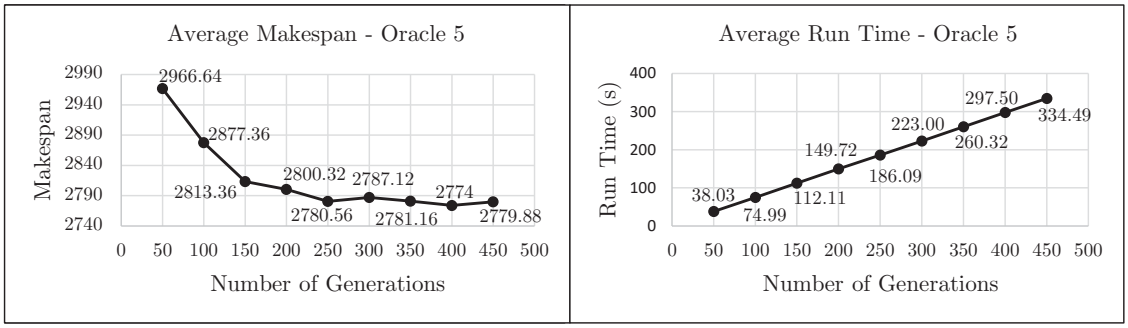


Figure 6.17: Impact of the maximum number of generations on the average makespan and average run time - Oracle Z_5.

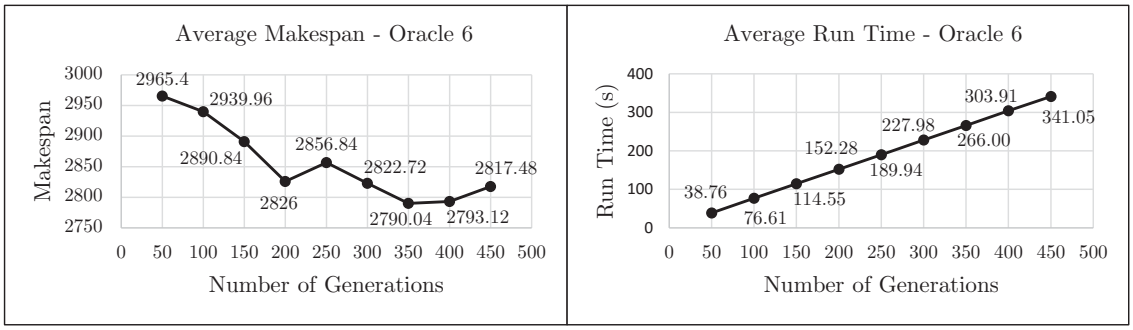


Figure 6.18: Impact of the maximum number of generations on the average makespan and average run time - Oracle Z_6.

follows: The CBM factor was set to 0.8; CM factor was set to 0.6; the scale parameter was set to 6; the shape parameter was set to 2.5; the maximum number of individuals in the elite list was set to 4; ρ was set to 0.5. For approximation of sequences using GA procedure, the CBM and CM duration are calculated as average processing time$*$ 0.8, and average processing time$*$ 0.6, respectively. As mentioned earlier, the CBM and CM duration are random when simulating the optimized schedule found from the GA procedure. During the execution of simulation procedure, the CBM duration is selected at random from a uniform distribution in the range $\in [0.6, 0.8]*$ average processing time. Similarly the CM duration is selected at random from a uniform distribution in the range $\in [0.4, 0.6]*$ average processing time. The number of scenarios for the simulation procedure was set to 500. CBM threshold was set to 40 and machine degradation rate was set to $\sim Exp(0.25)$.

The GA parameters were set as follows: the population mutation rate was set to 0.05; the gene mutation rate was set to 0.1; the population size was set to 200;

Instances	Machines	Jobs
ta01	15	15
ta11	15	20
ta21	20	20
ta31	15	30
ta41	20	30
ta51	15	50

Table 6.1: Taillard's Instance Size [102].

the maximum number of generations was set to 150. The SA parameters were set as follows: the maximum number of iterations was set to 50; temperature update factor was set to 0.8; temperature update run was set to 3. The total number of runs was set to 5 for all instances. The proposed simulation-optimization algorithm was run with all these parameter settings.

An analysis is carried out to observe how frequent each oracle provides the minimum makespan based on the simulation procedure. For that, we run the proposed simulation algorithm with the instance 'ta11' for 50 runs. Figure 6.19 shows each oracle's relative share of expected minimum makespan after simulation. From that analysis, it is clear that we cannot eliminate any oracle and every oracle is important for the objective function minimization.

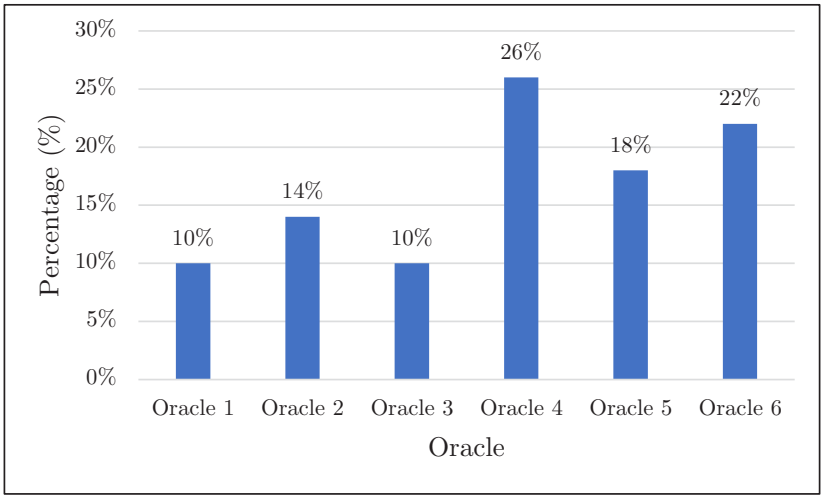


Figure 6.19: Percentage (%) of oracle share in objective function minimization.

The average behaviour of the proposed algorithm is shown in Figures 6.20 and 6.21. The initial solution was fixed to measure the average improvement. The maximum

average improvement of 8.68% over the initial solution is achieved with an average run time of 1.88 hours for 'ta01' instance, which has 15-job and 15-machine. The second-highest average improvement of 7.43% is found for instance 'ta11' with an average run time of 2.46 hours. Figure 6.20 shows that as the size of the instances are increasing, the average improvement is decreasing. For example, the average improvement of 5.46% is found for instance 'ta51' which has 50-job and 15-machine. We can also conclude that the average run time increases as the instance size increases.

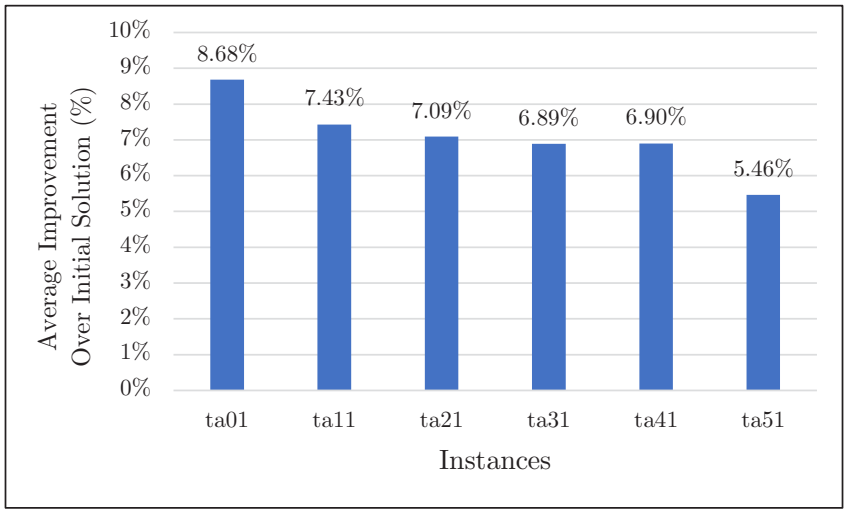


Figure 6.20: Average improvement over initial solution.

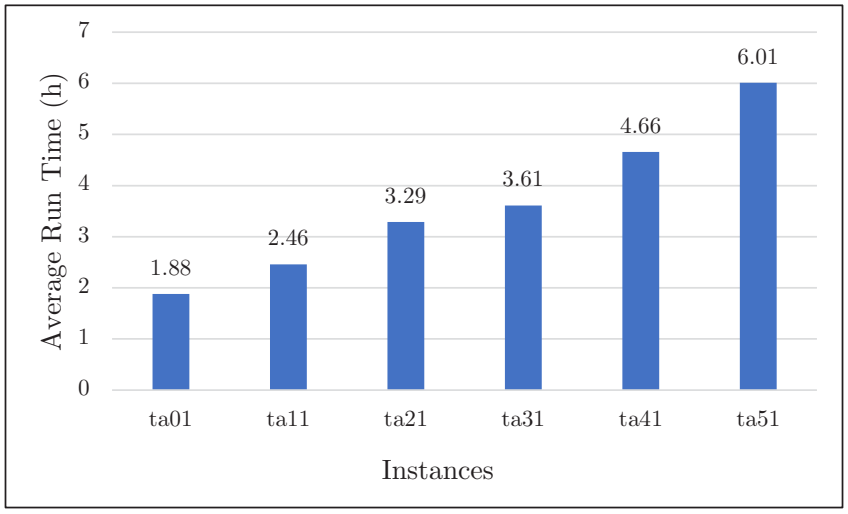


Figure 6.21: Average run time.

6.3 Sensitivity Analysis

The sensitivity analysis was carried out by varying the degradation rate and CBM threshold. The instance that was chosen for sensitivity analysis is 'ta11'. The parameters that were used for sensitivity analysis are the same as in the final run. The total number of runs was set to 25.

6.3.1 Degradation Rate

This experimentation is carried out by varying the degradation rate to investigate the impact on the optimal value, i.e., the average makespan and the 90th percentile of makespan. The degradation rate was varied by 10% of the base case, i.e., a 10% increase and a 10% decrease in the base case. The degradation rate is deterministic while executing the oracles, whereas it is random while performing the simulation. The base case is considered here at 0.25, so the 10% increase and decrease values of the base case are 0.275 and 0.225, respectively. Figure 6.22 shows the average results obtained by varying the degradation rate. While varying the degradation rate by 10% from the base case, the average makespan varies by 0.2% and 1.59% for the increasing and decreasing cases, respectively. Similarly, the 90th percentile of the makespan varies by 3.55% and 0.75%, respectively, for increasing and decreasing cases. It can be said that the average makespan and the 90th percentile of makespan are not very sensitive to a 10% change in the degradation rate.

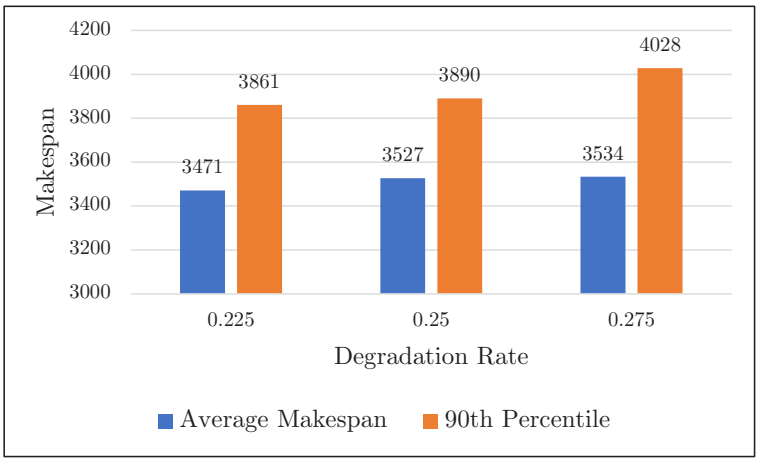


Figure 6.22: Impact of degradation rate.

6.3.2 CBM Threshold

Another sensitivity analysis is carried out by varying the CBM threshold. Similar to what was carried out in the degradation rate experimentation, the CBM threshold was varied by 10% of the base case. The average results obtained when varying the CBM threshold are shown in Figure 6.23. The base case was set to 40, so the 10% of the increasing and decreasing values were 44 and 36, respectively. When the CBM threshold is varied, the average makespan changes by 0.17% and 1.98%, respectively, for increasing and decreasing scenarios, whereas the 90th percentile of makespan varies by 0.44% and 1.18%. We can conclude that a 10% change in the CBM threshold has very little effect on the average makespan and 90th percentile of makespan.

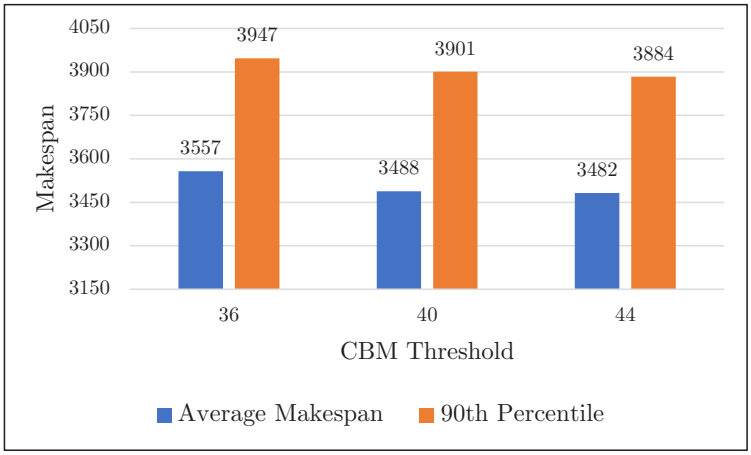


Figure 6.23: Impact of CBM threshold.

Chapter 7

Conclusion and Future Research

Considering real-life scenarios, this thesis proposes a simulation-optimization algorithm for the JSSP with CBM and random breakdowns. The proposed approach integrates surrogate functions, simulation and two metaheuristics to generate high-quality schedules. Generally, metaheuristics generate a near-optimal solution. We assume that the solution obtained from the proposed simulation-optimization approach is also near-optimal. The evaluation criteria were the combinations of average makespan and its 90th percentile. The algorithm was coded using Python in Anaconda Jupyter notebook. Taillard instances were used for the test instances.

The proposed algorithm showed good performance with a maximum average improvement of 8.68% for 15-job and 15-machine and a minimum average improvement of 5.46% for 50-job and 15-machine over the initial solution, with an average run time of 1.88 hours and 6.01 hours, respectively. It is hard to improve the solution with the proposed simulation-optimization algorithm when the instance is large. Numerical experimentation demonstrated that the proposed approach can effectively generate high-quality schedules.

We note that higher improvements could be reached by allowing the algorithm to run for longer time such as running the algorithm with higher number of iterations or running with higher number of runs. The decision maker can adjust the maximum number of iterations based on the time available to generate a schedule. Furthermore, the algorithm can be terminated at any time with a feasible schedule.

7.1 Future Research

Objective Function: The objective function considers the average makespan and its 90th percentile. Other performance metrics that could be investigated further include incorporating cost (i.e., production loss due to machine downtime in terms of cost, cost of CBM and CM) into the objective function with the makespan. In that

case, we would need to calculate the cost for an equivalent amount of makespan time.

Metaheuristics: It would be interesting to implement other metaheuristics in the inner and outer loop of the algorithm. A good suggestion is to implement the HSA in the inner loop and TS in the outer loop. Then, investigate the impact on the objective function as well as computational performance.

Maintenance Policy: The proposed approach considers CBM based on the processing time of the operations. Further analysis could be done by developing CBM model using Markov chain, Cox's Proportional Hazards Model (PHM). Imperfect maintenance consideration would be another interesting direction to be further analyzed.

Bibliography

[1] Mishra R. C. and Pathak K. *Maintenance Engineering and management*. PHI Learning Private Limited, New Delhi, 2012.

[2] Bevilacqua M. and Braglia M. The analytic hierarchy process applied to maintenance strategy selection. *Reliability Engineering and System Safety*, 70(1):71–83, 2000.

[3] Holmberg K., Komonen K., Oedewald P., Peltonen M., Reiman T., Rouhiainen V., Tervo J., and Heino P. Safety and reliability. Technical report, 2004.

[4] Tu P. Y. L., Yam R., Tse P., and Sun A. O. W. An integrated maintenance management system for an advanced manufacturing company. *The International Journal of Advanced Manufacturing Technology*, 17:692–703, 2001.

[5] Gertsbakh B. I. *Models of Preventive Maintenance*. North-Holland Publishing, Amsterdam, 1977.

[6] Johnson S. M. Optimal two and three stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[7] Smith W. E. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[8] McNaughton R. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

[9] Jackson J. R. An extension of johnson's results on job idt scheduling. *Naval Research Logistics Quarterly*, 3(3):201–203, 1956.

[10] Roy B. and Sussmann B. Les problèmes d'ordonnancement avec contraintes disjonctives. Note D. S. 9, SEMA, Paris, 1964.

[11] Németi L. Das reihenfolgeproblem in der fertigungs-programmierung und linearplanung mit logischen bedingungen. *Mathematika*, 6:87–99, 1964.

[12] Brooks G. H. and White C. R. An algorithm for finding optimal or near-optimal solutions to the production scheduling problems. *Journal of Industrial Engineering*, 16(1):34–40, 1965.

[13] Fisher H. and Thompson G. L. Probabilistic learning combinations of local job-shop scheduling rules. In Muth J. F. and Thompson G. L., editors, *Industrial Scheduling*, pages 225–251. Prentice-Hall, Englewood Cliffs, NJ, 1963.

[14] Lenstra J. K., Kan A. H. G. R., and Brucker P. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[15] Gonzalez T. and Sahni S. Flowshop and jobshop schedules: complexity and approximation. *Operations Research*, 26(1):36–52, 1978.

[16] Sotskov Y. N. and Shaklevich N.V. Np-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59(3):237–266, 1995.

[17] Zhang R. A simulated annealing-based heuristic algorithm for job shop scheduling to minimize lateness. *International Journal of Advanced Robotic Systems*, 10(4):1–9, 2013.

[18] Krim H., Zufferey N., Potvin J. Y., Benmansour R., and Duvivier D. Tabu search for a parallel-machine scheduling problem with periodic maintenance, job rejection and weighted sum of completion times. *Journal of Scheduling*, 25(1):89–105, 2022.

[19] Valente J. M. S. and Gonçalves J. F. A genetic algorithm approach for the single machine scheduling problem with linear earliness and quadratic tardiness penalties. *Computers and Operations Research*, 36(10):2707–2715, 2009.

[20] Neto R. F. T., Filho M. G., and da Silva F. M. An ant colony optimization approach for the parallel machine scheduling problem with outsourcing allowed. *Journal of Intelligent Manufacturing*, 26(3):527–538, 2015.

[21] Kohler W. H. and Steiglitz K. Exact, approximate, and guaranteed accuracy algorithms for the flow-shop problem n/2/f/f. *Journal of the Association for Computing Machinery*, 22(1):106–114, 1995.

[22] Zhang J., Ding G., Zou Y., Qin S., and Fu J. Review of job shop scheduling research and its new perspectives under industry 4.0. *Journal of Intelligent Manufacturing*, 30:1809–1830, 2019.

[23] Wagner H. M. An integer linear programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140, 1959.

[24] Lomnicki Z. A. A "branch-and-bound" algorithm for the exact solution of the three-machine scheduling problem. *Journal of the Operational Research Society*, 16(1):89–100, 1965.

[25] Manne A. S. On the job-shop scheduling problem. *Operations Research*, 8(2):219–223, 1960.

[26] Garey M. R., Johnson D. S., and Sethi R. The complexity of flow shop and job shops scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

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

[28] Lei D. Simplified multi-objective genetic algorithms for stochastic job shop scheduling. *Applied Soft Computing*, 11(8):4991–4996, 2011.

[29] Karimi H., Rahmati S. H. A., and Zandieh M. An efficient knowledge-based algorithm for the flexible job shop scheduling problem. *Knowledge-Based Systems*, 36:236–244, 2012.

[30] Rahmati S. H. A., Zandieh M., and Yazdani M. Developing two multi-objective evolutionary algorithms for the multi-objective flexible job shop scheduling problem. *International Journal of Advanced Manufacturing Technology*, 64:915–932, 2013.

[31] Akers S. B. Letter to the Editor—A graphical approach to production scheduling problems. *Operations Research*, 4(2):244–245, 1956.

[32] Gonçalves J. F. and Resende M. G. C. An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. *International Transactions in Operational Research*, 21(2):215–246, 2014.

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

[34] Liu S. Q. and Kozan E. Parallel-identical-machine job-shop scheduling with different stage-dependent buffering requirements. *Computers and Operations Research*, 74:31–41, 2016.

[35] Liu S. Q., Kozan E., Masoud M., Zhang Y., and Chan F. T. S. Job shop scheduling with a combination of four buffering constraints. *International Journal of Production Research*, 56(9):3274–3293, 2018.

[36] Yazdani M., Aleti A., Khalili S. M., and Jolai F. Optimizing the sum of maximum earliness and tardiness of the job shop scheduling problem. *Computers and Industrial Engineering*, 107:12–24, 2017.

[37] Nouri H. E., Driss O. B., and Ghédira K. Solving the flexible job shop problem by hybrid metaheuristics-based multiagent model. *Journal of Industrial Engineering International*, 14:1–14, 2018.

[38] Pongchairerks P. A two-level metaheuristic algorithm for the job-shop scheduling problem. *Complexity*, Article ID 8683472, 2019.

[39] Pongchairerks P. An enhanced two-level metaheuristic algorithm with adaptive hybrid neighborhood structures for the job-shop scheduling problem. *Complexity*, Article ID 3489209, 2020.

[40] Zarrouk R., Bennour I. E., and Jemai A. A two-level particle swarm optimization algorithm for the flexible job shop scheduling problem. *Swarm Intelligence*, 13:145–168, 2019.

[41] Mohammadi G. and Moaddabi E. Using two metaheuristic algorithms for scheduling parallel machines with sequence dependent set-up times in job shop industries. *International Journal of Systems Science*, 52(14):2904–2917, 2021.

[42] Gohareh M. M. and Mansouri E. A simulation-optimization framework for generating dynamic dispatching rules for stochastic job shop with earliness and tardiness penalties. *Computers and Operations Research*, 140:1–14, 105650, 2022.

[43] Wang S. and Yu J. An effective heuristic for flexible job-shop scheduling problem with maintenance activities. *Computers and Industrial Engineering*, 59(3):436–447, 2010.

[44] Moradi E., Ghomi S. M. T. F., and M. Zandieh. Bi-objective optimization research on integrated fixed time interval preventive maintenance and production for scheduling flexible job-shop problem. *Expert Systems with Applications*, 38(6):7169–7178, 2011.

[45] Hasan S. M. K., Sarker R., and Essam D. Genetic algorithm for job-shop scheduling with machine unavailability and breakdowns. *International Journal of Production Research*, 49(16):4999–5015, 2011.

[46] Harrath Y., Kaabi J., Sassi M., and Ali M. B. Multiobjective genetic algorithm-based method for job shop scheduling problem. In *2012 4Th Conference on Data Mining and Optimization (DMO)*, pages 13–17, Langkawi, Malaysia, 2012.

[47] Dalfard V. M. and Mohammadi G. Two meta-heuristic algorithms for solving multi-objective flexible job-shop scheduling with parallel machine and maintenance constraints. *Computers and Mathematics with Applications*, 64(6):2111–2117, 2012.

[48] Xiong J., Xing L. N., and Chen Y. W. Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns. *International Journal of Production Economics*, 141(1):112–126, 2013.

[49] Ye J. and Ma H. Multiobjective joint optimization of production scheduling and maintenance planning in the flexible job-shop problem. *Mathematical Problems in Engineering*, Article ID 725460, 2015.

[50] Ahmadi E., Zandieh M., Farrokh M., and Emami S. M. A multi objective optimization approach for flexible job shop scheduling problem under random machine breakdown by evolutionary algorithms. *Computers and Operations Research*, 73:56–66, 2016.

[51] Fitouri C., Fnaiech N., Varnier C., Fnaiech F., and Zerhouni N. A decison-making approach for job shop scheduling with job depending degradation and predictive maintenance. *IFAC-PapersOnLine*, 49(12):1490–1495, 2016.

[52] Nouiri M., Bekrar A., Jemai A., Trentesaux D., Ammari A. C., and Niar S. Two stage particle swarm optimization to solve the flexible job shop predictive scheduling problem considering possible machine breakdowns. *Computers and Industrial Engineering*, 112:595–606, 2017.

[53] Sajadi S. M., Alizadeh A., Zandieh M., and Tavan F. Robust and stable flexible job shop scheduling with random machine breakdowns: multi-objectives genetic algorithm approach. *International Journal of Mathematics in Operational Research*, 14(2):268–289, 2019.

[54] Zhai S., Riess A., and Reinhart G. Formulation and solution for the predictive maintenance integrated job shop scheduling problem. In *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pages 1–8, San Francisco, USA, 2019.

[55] Fnaiech N., Fitouri C., Varnier C., Fnaiech F., and Zerhouni N. A new heuristic method for solving joint job shop scheduling of production and maintenance. In *Symposium on Information Control Problems in Manufacturing*, Ottawa, Canada, 2015.

[56] Wang H., Sheng B., Lu Q., Yin X., Zhao F., Lu X., Luo R., and Fu G. A novel multi-objective optimization algorithm for the integrated scheduling of flexible job shops considering preventive maintenance activities and transportation processes. *Soft Computing*, 25(4):2863–2889, 2021.

[57] Gupta S. and Jain A. Analysis of integrated preventive maintenance and machine failure in stochastic flexible job shop scheduling with sequence-dependent setup time. *Smart Science*, pages 1–23, 2021.

[58] Baykasoğlu A. and Madenoğlu F. S. Greedy randomized adaptive search procedure for simultaneous scheduling of production and preventive maintenance activities in dynamic flexible job shops. *Soft Computing*, 25:14893–14932, 2021.

[59] Souza R. L. C., Ghasemi A., Saif A., and Gharaei A. Robust job-shop scheduling under deterministic and stochastic unavailability constraints due to preventive and corrective maintenance. *Computers and Industrial Engineering*, 168:1–15, 2022.

[60] Zheng Y., Mesghouni K., and Dutilleul S. C. Condition based maintenance applied to reduce unavailability of machines in flexible job shop scheduling problem. *IFAC-PapersOnLine*, 46(9):1405–1410, 2013.

[61] Zheng Y., Lian L., and Mesghouni K. Comparative study of heuristics algorithms in solving flexible job shop scheduling problem with condition based maintenance. *Journal of Industrial Engineering and Management*, 7(2):518–531, 2014.

[62] Zandieh M., Khatami A. R., and Rahmati S. H. A. Flexible job shop scheduling under condition-based maintenance: improved version of imperialist competitive algorithm. *Applied Soft Computing*, 58:449–464, 2017.

[63] Rahmati S. H. A., Ahmadi A., and Govindan K. A novel integrated condition-based maintenance and stochastic flexible job shop scheduling problem: simulation-based optimization approach. *Annals of Operations Research*, 269:583–621, 2018.

[64] Ghaleb M., Taghipour S., and Zolfagharinia H. Real-time integrated production-scheduling and maintenance-planning in a flexible job shop with machine deterioration and condition-based maintenance. *Journal of Manufacturing Systems*, 61:423–449, 2021.

[65] Hoos H. H. and Stützle T. Scheduling Problems. *Stochastic Local Search Foundations and Applications*, pages 417–465, 2005.

[66] Baker K. R. and Trietsch D. *Principles of Sequencing and Scheduling, 2nd Edition*. John Wiley & Sons, New Jersey, 2018.

[67] Graham R. L., Lawler E. L., Lenstra J. K., and Kan A. H. G. R. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[68] Schmidt G. Scheduling with limited machine availability. *European Journal of Operational Research*, 121(1):1–15, 2000.

[69] Ma Y., Chu C., and Zuo C. A survey of scheduling with deterministic machine availability constraints. *Computers and Industrial Engineering*, 58(2):199–211, 2010.

[70] Pinedo M. L. *Scheduling Theory, Algorithms, and Systems, 5th Edition*. Springer, New York, 2016.

[71] Griffin K. R. Job shop scheduling. In *Production and Inventory Management*, pages 65–76. 1971.

[72] Ben-daya M., Kumar U., and Murthy D. N. P. *Introduction to Maintenance Engineering: Modelling, Optimization and Management*. John Wiley & Sons, New Jersey, 2016.

[73] Kobbacy K. A. H. and Murthy D. N. P. *Complex System Maintenance Handbook*. Springer, London, 2008.

[74] Ebeling C. E. *An Introduction to Reliability and Maintainability Engineering, 3rd Edition*. Waveland Press, 2019.

[75] Barlow R. E. and Proschan F. *Mathematical Theory of Reliability*. John Wiley & Sons Inc., New York, 1996.

[76] Peres F. and Castelli M. Combinatorial optimization problems and metaheuristics: review, challenges, design, and development. *Applied Sciences*, 11:1–39, 2021.

[77] Glover F. Paths for integer programming. *Computers and Operations Research*, 13(5):533–549, 1986.

[78] Boussaïd I., Lepagnot J., and Siarry P. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

[79] Nesmachnow S. An overview of metaheuristics: accurate and efficient methods for optimisation. *International Journal of Metaheuristics*, 3(4):320–347, 2014.

[80] Xu J. and Zhang J. Exploration-exploitation tradeoffs in metaheuristics: survey and analysis. In *Proceedings of the 33rd Chinese Control Conference (CCC)*, pages 8633–8638, Nanjing, China, 2014.

[81] Parejo J. A., Ruiz-Cortés A., Lozano S., and Fernandez P. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing*, 16:527–561, 2012.

[82] Blum C. and Roli A. Hybrid metaheuristics: An introduction. In Blum C., Aguilera M. J. B., Roli A., and Sampels M., editors, *Studies in Computational Intelligence*, pages 1–30. Springer, Germany, 2008.

[83] Crepinsek M., Liu S. H., and Mernik M. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys*, 45(3):1–33, 2013.

[84] Sörensen K. and Glover F. W. Metaheuristics. In Gass S. I. and Fu M. C., editors, *Encyclopedia of Operations Research and Management Science*. Springer, USA, 2013.

[85] Holland J. H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Michigan, USA, 1975.

[86] Fraser A. S. Simulation of genetic systems by automatic digital computers II. Effects of linkage on rates of advance under selection. *Australian Journal of Biological Sciences*, 10(4):492–500, 1957.

[87] Davis L. Job shop scheduling with genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications*, pages 136–140, New Jersey, USA, 1985.

[88] Bierwirth C., Mattfeld D. C., and Kopfer H. On permutation representations for scheduling problems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1141:310–318, 1996.

[89] Metropolis N., Rosenbluth A. W., Rosenbluth M. N., Teller A. H., and Teller E. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[90] Kirkpatrick S., Gelatt C. D., and Vecchi M. P. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[91] Dowsland K. A. Simulated annealing. In Reeves C.R., editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 20–69. John Wiley & Sons Inc., New York, USA, 1993.

[92] Cui W., Lu Z., Li C., and Han X. A proactive approach to solve integrated production scheduling and maintenance planning problem in flow shops. *Computers and Industrial Engineering*, 115:342–353, 2018.

[93] Jones D. R. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001.

[94] Regis R. G. and Shoemaker C. A. Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. *Engineering Optimization*, 45(5):529–555, 2013.

[95] Tan M. H. Y. Sequential bayesian polynomial chaos model selection for estimation of sensitivity indices. *SIMA/ASA Journal on Uncertainty Quantification*, 3:146–168, 2015.

[96] www.merriam-webster.com/dictionary/oracle. Merriam-Webster Dictionary.

[97] Bierwirth C. A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spektrum*, 17(2-3):87–92, 1995.

[98] Sastry K., Goldberg D. E., and Kendall G. Genetic algorithms. In Burke E. K. and Kendall G., editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, 2nd Edition*, pages 1–30. Springer, New York, 2014.

[99] Baker J. E. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pages 14–21, New Jersey, USA, 1987.

[100] Jun J., Kang J., Jeong D., and Lee H. An efficient approach for optimizing full field development plan using monte-carlo simulation coupled with genetic algorithm and new variable setting method for well placement applied to gas condensate field in Vietnam. *Energy Exploration and Exploitation*, 35(1):75–102, 2017.

[101] Kuo C. C., Glover F., and Dhir K. S. Analyzing and modeling the maximum diversity problem by zero-one programming. *Decision Sciences*, 24(6):1171–1185, 1993.

[102] Taillard E. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.