ROBUST OPTIMIZATION ALGORITHMS FOR THE FLOW SHOP AND JOB SHOP SCHEDULING PROBLEMS WITH RANDOM FAILURES AND PREVENTIVE MAINTENANCE

by

Rafael Lucas Costa Souza

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

at

Dalhousie University
Halifax, Nova Scotia
December 2020

Table of Contents

List of Tables

List of Figures

Abstract

Inspired by a real-life problem in the kitchen cabinet manufacturing industry, this thesis proposes a suite of algorithms for solving the flow shop and the job shop scheduling problems (FSP & JSP) with both scheduled (preventive) maintenance and random breakdowns. These algorithms aim at obtaining schedules that strike a good balance between performance quality (i.e., shortest expected makespan) and solution robustness (i.e., least affected by breakdowns).

The proposed scheduling framework approximates the fitness function of the original problem using three surrogate functions. The first considers only the actual jobs, the second adds scheduled maintenances and the third adds both scheduled maintenances and deterministic breakdowns based on the mean time to failure of machines. For the FSP, a local optimum solution of each surrogate problem is found either through a local search heuristic or a simulated annealing algorithm. For the JSP, given the extremely large search space, a genetic algorithm is used to find local optimal solutions. These solutions are then simulated with random breakdowns and the best among them is compared to the incumbent solution of the original problem. In different variants of the algorithm, it either terminates once the new solution is found to be worse than the incumbent, or non-improving solutions are accepted, yet with a decreasing probability, in a simulated annealing style.

The first algorithm of the FSP showed no improvement over the initial solution for the 25-machine, 75-job problem. The second algorithm did not perform well due to premature termination. The third algorithm showed marginal improvement with an average of 1.25% over the initial solution with a much higher average run time. The first algorithm for the JSP showed an average marginal improvement of 5.33% and a quick run time of 10.01 minutes for the 50-job, 15-machine problem. The second algorithm showed good performance with an average improvement of 6.71% in an average time of 5.11 hours. These results show that the proposed framework can generate high-quality schedules while taking scheduled maintenance and random breakdowns into consideration.

Acknowledgements

I would like to start by thanking the person who motivated me in all the steps of this work. She was always there for me and provided me with unconditional love and emotional support. This work is dedicated to my mom, Gislene.

I owe my deepest gratitude to Dr. Ghasemi and Dr. Saif for providing invaluable guidance, mentoring, encouragement, constructive criticism, and inspiration throughout this whole project. Your patience, understanding, and resilience during this hard time always made me feel confident to keep pushing forward.

I cannot forget my family away from home. Thanks Ken, Priscilla, Ian, Boston, Luana, Lauryne, Thiago, Camila, Braeden, and João. They all kept me going, and this thesis would never have been possible without them.

Finally, a great thanks to my family, friends, and colleagues whose names are not mentioned here but had me in their minds.

Chapter 1

Introduction

Scheduling happens at all times in our lives. We are always planning and allotting portions of time to activities either at home planning our chores or at work planning our daily activities. Pinedo [40] states that scheduling is a method of decision-making that is a vital part of the service and manufacturing industries.

The work presented here was inspired by a project in Triangle Kitchen Ltd., a company located in Atlantic Canada which is known to be a leader in the manufacturing and wholesale distribution of cabinetry products and accessories. The production planning team was facing numerous challenges when trying to properly schedule their clients' orders on the shop floor. The management was interested in the creation of an automatic tool to generate good schedules considering the machinery maintenance and the randomness of machine failures as part of their new scheduling system. This problem was the spark that drove the development of this thesis.

For a company to thrive in its sector, it has to be competitive. Achieving and maintaining a relationship of trust with clients is fundamental for any company to accomplish competitiveness. Failing to completely deliver an order or failing to meet a deadline leads to a penalty that can be incurred in monetary form or lost of trust and goodwill. Only through proper scheduling and job sequencing, a company can avoid logistics conflicts that impact the final delivery date of a product, project, or service. Especially, in manufacturing, the production usually can be broken into several smaller units of work (tasks) and these jobs can be then allocated to a limited number of machines or people (resources). The allocation of tasks to limited resources is denoted as "scheduling problem" Baker [6].

The majority of researches in machine scheduling, a more specific kind of scheduling problem, consider that machines are available for processing at all times during the scheduling horizon. In real industrial scenarios, machines are subject to stochastic and deterministic unavailable periods of time. The stochastic stoppage during a machine's scheduling horizon may be due to machine breakdowns or any other unpredictable reason that makes the machine unavailable for a period of time. Deterministic stoppage may be due to, e.g., periodical machine maintenance, tool modification, adjustments. with fixed and foreknown starting time and duration. Deterministic stoppages, stochastic stoppages, and jobs compete for time in a set of machines at the same time. In this thesis, the deterministic stoppage is going to be determined by the unavailable period of time a machine stops for Preventive Maintenance (PM). Breakdowns (BD) are randomly generated as the machine age increases over time.

Intuitively, any consideration regarding the schedule planning has to be done in an integrated manner. Integrating the PM decisions with the jobs' sequencing regarding the average behaviour of random breakdowns will provide a smart robust schedule capable of being used as a management tool for decision making inside a company. To solve this problem, this thesis presents a suite of algorithms that aims to integrally generate robust schedules applied to the flow shop scheduling problem and two algorithms focused on the job shop scheduling problem.

The remainder of this Chapter presents the main objective of this thesis in section 1.1 and its organization in section 1.2.

1.1 Thesis Objective

The main objective of this thesis is to propose algorithms that optimize the sequence of jobs, while taking into consideration scheduled preventive maintenance and random machine breakdowns in a flow shop and job shop setting.

1.2 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 presents a literature review relevant to the flow shop and job shop scheduling problems. In Chapter 3, a brief background and problem definitions are provided. In Chapter 3.1.2, FSP algorithms are presented and a sensitivity analysis is conducted. In Chapter 3.1.3, a genetic algorithm is presented in detail, followed by the JSP algorithms. Finally, in Chapter 5.3 the results are shown, the thesis is concluded, and directions for future research are suggested.

Chapter 2

Literature Review

The scheduling theory field grew considerably after the publication of two papers by Johnson [26] and Smith [43] during the mid 1950's. Johnson [26] in 1954 presented a famous algorithm to solve a two-machine job shop problem minimizing makespan. Later in 1959, Wagner [45] presented the first known integer linear programming mathematical model for the job shop scheduling problem. In 1960, Manne [35] proposed a mixed integer linear program that has a smaller number of variables compared to the one presented by Wagner [45]. Still on the exact formulation, other researchers released their own formulations based on enumeration methods. Work published by Brooks et. al. [10] and Lomnicki [32] are examples of this early research. Graham [21] in 1979 outlined and classified the machine scheduling field in a 3-field problem classification scheme $\alpha|\beta|\gamma$.

Garey et. al. [19] in 1976 proved the complexity of the flow shop (FSP) and job shop (JSP) scheduling problems. FSP is NP-complete when the number of machines is greater than 3 and the JSP is NP-complete when the number of machines is greater than 2. Exact methods that solve these classes of problems cannot provide optimal solutions of large instances in reasonable times. Only small size instances could be solved to proven optimality in the literature. Alternatively, approximate algorithms were developed to effectively handle realistic instance sizes.

Constructive algorithms usually start with a set of parameters with no previous feasible solution and create a proper feasible one by expanding itself step-by-step. Nawaz et. al. [37] created an efficient algorithm that deals with the flow shop scheduling problem when minimizing makespan, which, until this day is reported in the literature (Ruiz et. al. [41]) to be one of the best construction algorithms regarding solution quality and computational time for FSPs. For JSPs, the most used constructive algorithm cited in the literature is known as the shifting bottleneck procedure and was proposed by Adams et. al. [1]. Based on the disjunctive formulation presented by Balas [7], Adams created a constructive algorithm that solves the JSP minimizing makespan.

Deterministic scheduling problems with availability intervals have drew a considerable attention from researchers at the beginning of the 21st century. The work published by Lee [30], Schmidt [42] and Ma et. al. [34] summarized this field. Aggoune [2] demonstrated that, specifically for the FSP, there are two main branches when describing machine unavailability: (a) the unavailability intervals have a fixed length and the start times of each unavailability period are known in advance; (b) the unavailability intervals have a predetermined time window so that the start time have to be determined; Ma et. al. [34] pointed out that even though there are a lot of research being conducted regarding single machine scheduling problem, parallel machines scheduling problem, flow shop scheduling problem, and job shop scheduling problem, there is a lack of research around the availability constraints.

On the deterministic unavailable periods of time, Kubzin et. al. [29] proposed an algorithm that can solve in polynomial time for a two-machine flow shop scheduling problem $(F2|no-wait, m(0,1)|C_{max})$ when one of the machines was subjected to obligatory maintenance with the objective to minimize the makespan. Ruiz et. al. [41] presented the performance evaluation of six different metaheuristic methods to solve the flow shop scheduling problem with multiple types of maintenance policies. One of the policies analyzed by Ruiz et. al. [41] is used in this thesis and was introduced by Cassady et. al. [12]. Hadda et. al. [22] showed an improved heuristic for the two-machine flow shop problem with unavailable periods on the first machine assuming resumable activity, and proved it could be solved in a polynomial time. Li et. al. [31] presented a discrete artificial bee colony algorithm to solve a multi objective flexible job shop scheduling problem with maintenance activities. Khoukhi et. al. [18] described a "dual-ant" colony algorithm that solves the flexible job shop scheduling problem with one preventive period maintenance for each machine.

On non-deterministic unavailable periods of time, when the duration and start time of these disruptions are stochastic (non-deterministic), Gholami and Zandieh [20] integrated simulation with a genetic algorithm to solve the job shop scheduling problem with stochastic breakdowns. Ming Wang [46] created a genetic algorithm to handle the flexible job shop scheduling problem with known disruption periods of time. Al-hinai et. al. [4] showed a two-stage hybrid genetic algorithm to solve a bi-objective flexible job shop problem with random breakdowns. Nouiri [38] proposed a two-stage particle swarm optimization algorithm to solve a flexible job shop problem minimizing makespan, where the created schedule was proven more robust and stable than the various benchmark data from the literature varying from Partial FJSP to Total FJSP. Ahmadi et. al. [3] reported the performance evaluation of different evolutionary metaheuristics when solving a multi-objective job shop scheduling problem with random machine breakdowns. GA was declared to be computationally faster and provide better results when compared to particle swarm optimization and randomized procedures.

Robustness is an important performance metric when dealing with scheduling problems because it is a clear way to show the schedule's adherence to the management goal. Herroelen and Roel [23] specified that robustness can be divided into two groups: (a) quality robustness and (b) solution robustness. Quality robustness is the measure of the insensitivity of a given performance parameter like makespan or total tardiness to the presence of uncertainty. Solution robustness, usually described as "stability", represents the insensitivity of each operation start time to changes in the original baseline.

In a real-life scenario, the deterministic and non-deterministic unavailable periods of time are not mutually exclusive. Even when scheduling maintenance ahead of time periodically, machine failure can still impact the shop floor unpredictably. To the best of our knowledge, there is no review paper available around this topic and the research is still in its infant stage. Cui et. al. [14] addressed the single machine scheduling problem with robustness integrating a maintenance policy. Le et. al. [33] proposed a genetic algorithm approach for the single machine scheduling problem with robustness regarding preventive maintenance and failure uncertainty. Cui et al. [15] developed a proactive approach to solve integrated production scheduling and maintenance planning in flow shops by creating a two-loop algorithm that tries to optimize a robust surrogate function that inserts buffer times in a schedule to absorb any unexpected breakdown. Their proposed strategy was simplistic and does not represent the computational capabilities of modern systems. This thesis expands on the work done by them and proposes three new algorithmic simulation optimization strategies for the flow shop scheduling problem and proposes two more algorithms to the job shop scheduling problem.

When compared to the work done by Cui et al. [15], the first proposed algorithm for the JSP uses an additional surrogate model, called oracles though out this thesis, to analyze the performance of a given neighbourhood with added preventive maintenances and breakdowns. The breakdowns happen at the computed mean time to failure (MTTF) of the machines based on the provided machine parameters. The other two algorithms try to get a faster conversion by using metaheuristics to explore the neighbourhood of solutions instead of performing a complete neighbourhood creation. Inspired by the FSP algorithms, two new algorithms were created using the same oracles and simulation structure but now ported to the JSP with added genetic algorithm metaheuristic to smartly search for solutions.

Chapter 3

Background and Problem Definition

In this Chapter, the standard classification scheme for the machine scheduling research field is described, then the basic concepts of the flow shop scheduling problem and the job shop scheduling problem are presented. Next, the overall preventive maintenance and random breakdowns assumptions are shown, followed by the description of the metaheuristic strategies used, finally the objective function and all its impacting metrics are outlined.

3.1 Machine Scheduling Preliminaries

Baker [6] describes that "machine scheduling problem" is a specialized scheduling problem where the sequencing of jobs completely describes a schedule. In the next subsections, we explore the machine scheduling field by initially describing the standard classification scheme. Next, we describe the flow shop scheduling problem (FSP) and finally describe the job shop scheduling problem (JSP) .

3.1.1 A classification scheme

Early work by Graham [21] neatly organized and classified the scheduling study field using a 3-field problem classification scheme $\alpha|\beta|\gamma$. Schmidt [42] and Ma et al.[34] expanded this classification scheme to the machine scheduling with unavailability period problem. Let us consider m machines M_i and n jobs J_j and assume that each machine can process one job at a time and that each job can be processed on at most one machine at a time.

The field $\alpha = \{\alpha_1, \alpha_2, \alpha_3\}$ specifies the machine environment. Parameter α_1 can assume the following values:

- Single machine $\{\alpha_1 = \varnothing\}$, a problem where only one machine exists;

- Identical parallel machines $\{\alpha_1 = P\}$, a system where machines are the same (i.e. with the same speed factor) and work in parallel;

- Uniform and unrelated parallel machines $\{\alpha_1 = Q\}$, a system where machines are different (i.e. with different speed factor) and work in parallel;

- Flow shop environment $\{\alpha_1 = F\}$, a set of m machines in series that have to process a set of jobs. Each job has to be processed at every machine once and all jobs obey the same sequencing path through the machines. When a machine completes a job, this job joins a First In First Out (FIFO) queue on the next machine of the designed path;

- Job shop environment $\{\alpha_1 = J\}$, a set of m machines that have to process a set of jobs. Each job may be processed at every machine once and has its own predetermined sequencing path through the machines. Re-circulation, when a job has to come back to a previous machine that it already has been through is not permitted;

- Open shop environment $\{\alpha_1 = O\}$, a set of m machines that have to process a set of jobs. Each job may be processed at every machine once and do not have a predetermined sequencing path through the machines. It is open to the scheduler to find the optimal sequencing of operations;

If Parameter α_1 was set to P which represents a system with parallel machines, Parameter α_2 has to assume a value of $\{\alpha_2 = k\}$ which represents the number of parallel machines or the number of dedicated machine stages.

Parameter α_3 is related to machine availability. Unavailable periods of time in a machine are described in the literature [34] as "holes". Parameter α_3 can assume the following values:

- $\{\alpha_3 = \varnothing\}$ when no holes exists and machines are available continuously;

- $\{\alpha_3 = h_{jk}\}$ which specifies an arbitrary number of holes on each machine. If j assumes the value of a positive integer, only that a specific machine has holes on it. If no value is assigned to j, holes will be allowed on all machines. If k assumes the value of a positive integer, it represents how many holes there are on that corresponding machine. If no value is given to k, the number of holes is considered arbitrary. The j subscript may not be added in the case of single machine and only k holes will occur.

The second field $\beta = \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6\}$ indicates a number of job and resource characteristics. To better understand β_1, we present the concept of preemption. Preemption as defined by Pinedo [40] is the ability to stop the processing of a job on a machine at any given time and substitute it by any other job. Any time that the machine has spent executing that preempted job is not lost and it can be given to another machine to continue working on it from that point forward. Parameter β_1 can assume the following values:

- $\{\beta_1 = \varnothing\}$ when no preemption is allowed;
- $\{\beta_1 = t - pmtn\}$ when there are operations preemption;
- $\{\beta_1 = pmtn\}$ when there are arbitrary number of preemption;

Parameter β_2 is related to resource availability and can assume the following values:

- $\{\beta_2 = \varnothing\}$ when there is no resource constraints;
- $\{\beta_2 = res\}$ when there are a limited number of resources R_h where every job requires at least r_{hj} units of R_h during its execution;
- $\{\beta_2 = res1\}$ when there is only a single resource available;

Parameter β_3 is related to the precedence of jobs. The precedence of jobs was described by Pinedo [40] as the requirement that a job or a set of jobs may be finished before the other job is permitted to begin in a specified machine. β_3 can assume the following values:

- $\{\beta_3 = \varnothing\}$ when there is no precedence relation;
- $\{\beta_3 = chains\}$ when a job has at most one predecessor and at most one successor;

- $\{\beta_3 = intree\}$ when a job has at most one successor;

- $\{\beta_3 = outtree\}$ when a job has at most one predecessor;

- $\{\beta_3 = prmu\}$ may appear only if in a flow shop setting and establishes that the precedence of the jobs in the queue in front of every machine follow a First In First Out (FIFO) structure.

Parameter β_4 is related to the release date r_j which is described as the earliest date that a job is available to be processed in a machine. β_4 can assume the following values:

- $\{\beta_4 = \varnothing\}$ when every job is available at time 0 i.e. $r_j = 0$;

- $\{\beta_4 = r_j\}$ when the release dates are specified;

Parameter β_5 is related to a constant upper bound on the number of a job's operation in a machine (m_j) and can only occur if $\alpha_1 = J$. β_5 can assume the following values:

- $\{\beta_5 = \varnothing\}$ when there is no such bound;

- $\{\beta_5 = m_j <= \bar{m}\}$ when the number of job's operations have to be smaller than $\bar{m}$;

Parameter β_6 is related to the problem's processing time. β_6 can assume the following values:

- $\{\beta_6 = \varnothing\}$ when there is no restriction on the processing times;

- $\{\beta_6 = p_{ij} = 1\}$ when unit processing times are considered;

- $\{\beta_6 = p_{lower} < p_{ij} < p_{upper}\}$ when a lower and an upper bound are specified;

The third field γ refers to the chosen optimality criteria. The most common minimization functions are:

- Makespan (C_{max}) which represents the total length of the schedule i.e. the time which the last job leaves the system;

- Maximum Lateness (L_{max}) which represents a measurement of the worst violation of the due dates;
- Total weighted completion time ($\sum w_j C_j$) which represents the total holding of inventory costs induced by the schedule also referred as "weighted flowtime" in the literature [6];
- Total weighted tardiness ($\sum w_j T_j$) which represents the penalties given to all the jobs which failed to be completed on time;
- Weighted number of tardy jobs ($\sum w_j U_j$);

α			β						γ
α_1	α_2	α_3	β_1	β_2	β_3	β_4	β_5	β_6	
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	C_{max}
P	k	h_{jk}	$t-pmtn$	res	$chains$	r_j	$m_j <= \bar{m}$	$p_{ij}=1$	L_{max}
Q			$pmtn$	$res1$	$intree$			$p_{lower} < p_{ij} < p_{upper}$	$\sum w_j C_j$
F					$outtree$				$\sum w_j T_j$
J					$prmu$				$\sum w_j U_j$
O									

Table 3.1: Summary of classification scheme

3.1.2 Flow shop scheduling problem (FSP)

Baker [6] and Pinedo [40] describe a flow shop scheduling problem in the following manner: A flow shop considers that the machines are organized in series. Jobs that arrive to be processed on those machines are broken down into small operations. Each operation has to be performed by at least one machine and the machines need to be different from one another. The sequence, i.e., the precedence structure in which these operations will be processed on the machines, has to be described by the job. Each job is a compilation of operations with the arrangement in which the operations will be executed on the machine. A job is considered completed when all its operations have been executed on that specific machine arrangement. All operations in all jobs have to have the same machine sequencing, thus they will always follow the same route within the machines.

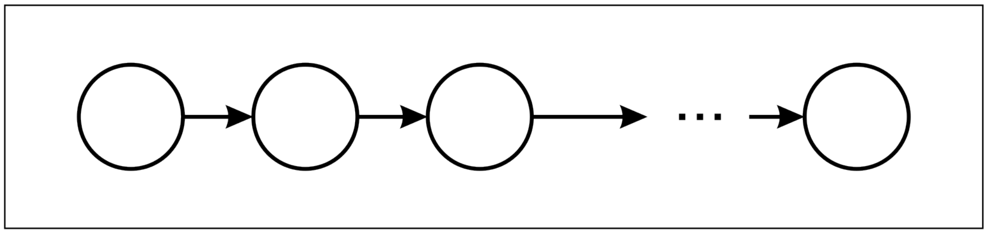


Figure 3.1: The operations arrangement of a job in a flow shop. Source: Baker [6]

The flow of work in a FSP is considered unidirectional since the machines are paired in series. The flow of work of a pure flow shop system is illustrated in Figure 3.2.

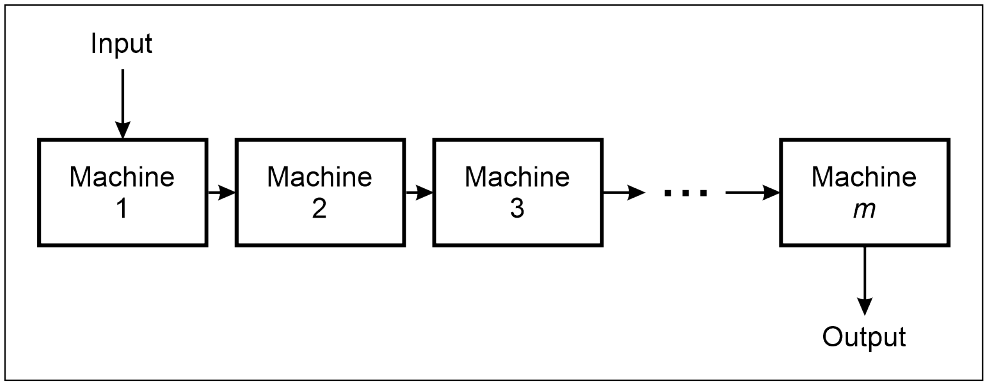


Figure 3.2: The workflow of a pure flow shop. Source: Baker [6]

There is a more general case of the FSP called “flexible” FSP. The only difference between the pure FSP and the flexible FSP is that jobs may not use all machines. In this case, a job is allowed to skip a machine if needed.

The basic pure flow shop problem has a set of important considerations:

- All jobs are available to be processed at time 0.
- The setup time, the time that a machine requires to be ready for an operation, is considered 0 or is included in the processing time of that operation.
- All machines are available at all times.
- The machine sequencing and processing times of all operations are known in advance.

- No preemption is permitted.

Baker [6] showed the following example of a flow shop problem containing $j = 2$ jobs and $k = 4$ machines. The processing times (p_{kj}) can be seen on Table 3.2.

Job j	1	2
p_{1j}	1	4
p_{2j}	4	1
p_{3j}	4	1
p_{4j}	1	4

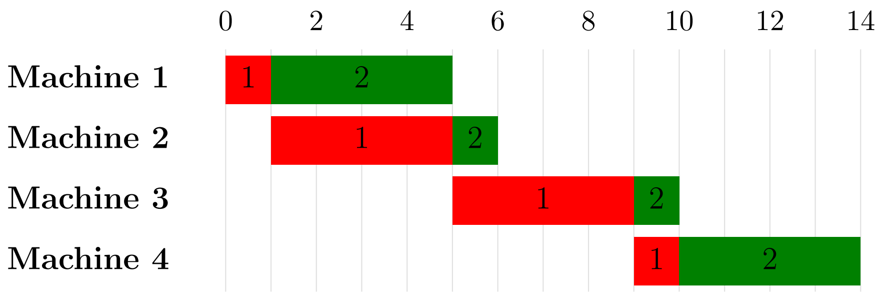
Table 3.2: Example of a flow shop problem processing times. Source: Baker [6]

The Gantt charts in Figure 3.3 show three possible schedules. The job's operation sequencing in Figure 3.3 (a) and Figure 3.3 (b) are the same, $X_{1,2,3,4} = \{1, 2\}$ and $X_{1,2,3,4} = \{2, 1\}$ respectively, but in Figure 3.3 (c) it is not. Note that the job sequencing changes from $X_{1,2} = \{1, 2\}$ to $X_{3,4} = \{2, 1\}$ on Machine 3 and 4. Job 2 was given the opportunity to break the queuing FIFO rule that exists between every machine and was processed ahead of job 1, thus Figure 3.3 (c) would not be considered a feasible solution if the system being analyzed is considered a *permutation* flow shop system. Pinedo [40] explains that finding optimal solutions when such changes are allowed is significantly harder even though changing the job sequencing throughout the schedule can lead to a shorter makespan (as illustrated on Figure 3.3). In this thesis, only the permutation flow shop is being considered and can be denoted as $(F|prmu|\gamma)$.

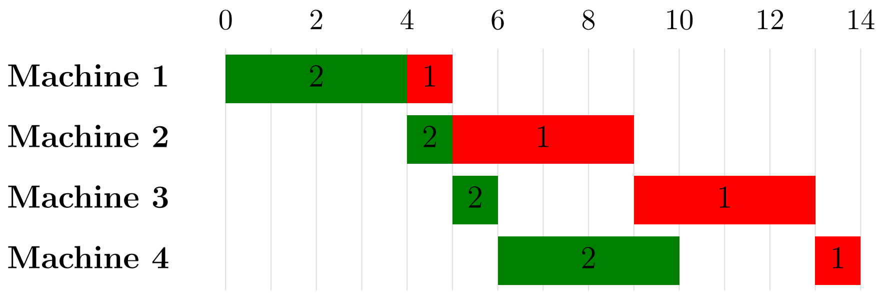
The NEH algorithm

A constructive algorithm differs from an optimization algorithm in the sense that it is able to create a solution only from the input data without knowing any previous feasible solution. To generate an initial feasible permutation flow shop schedule solution, the NEH algorithm was used in this thesis. It was initially introduced by Nawaz et. al. [37] and it is known until this day to be one of the best algorithms to construct schedules for the $F|prmu|C_{max}$ problem.

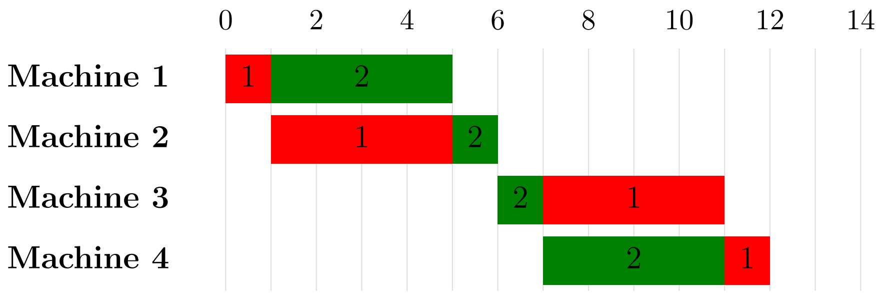
As Mokotoff [36] explains, the NEH algorithm works on the premise that a higher priority has to be given to jobs with the longest total processing time. When enumerating and positioning your jobs in a schedule, the jobs with the longest total



(a) $C_{max} = 14$. Source: Baker [6]



(b) $C_{max} = 14$. Source: Baker [15]



(c) $C_{max} = 12$. Source: Baker [15]

Figure 3.3: Example of some flow shop schedules. Source: Baker [15]

processing time should be placed first. Based on this simple idea, the algorithm is presented on Figure 1.

```
Input:
    The number of jobs $n$
    The number of machines $m$
    The processing times of job $j$ on machine $k$ as
    $P_{jk}(j = 1, \dots, n; k = 1, \dots, m)$
Output: $X, C_{max}$
    Step 1. For each job $j$ calculate $P_j = \sum_{k=1}^{m} P_{jk}$.
    Step 2. Arrange the jobs in descending order of $P_j$.
    Step 3. The first two jobs from the list of Step 2 are selected and the best
      sequence that minimizes the makespan ($C_{max}$) is then calculated. The
      sequence found relative to them is not changed for the other steps of the
      algorithm. Set $i = 3$.
    Step 4. Select the $i^{th}$ job from the list of Step 2. Calculate the best
      sequence by inserting the job in all $i$ available positions from the
      previous partial sequence without changing any already assigned order.
    Step 5. If $n = i$, STOP, Otherwise set $i = i + 1$ and go to Step 4.
```

Algorithm 1: NEH Algorithm. Source: Nawaz et. al. [37]

3.1.3 Job shop scheduling problem

Baker [6] and Pinedo [40] describe the job shop scheduling problem in the following manner: The main difference between the FSP and the JSP is that there is no unidirectional flow of work. All elements that characterize the FSP are still present on the JSP. There are a number of machines and a number of jobs to be processed. Each job still carries its precedence arrangement. Every job has its own machine sequencing. Re-circulation, when a job can visit a machine more than once, is also allowed but not used in this thesis. Baker [6] provided the following example with 3 machines and 4 jobs of a job shop problem. Table 3.3 shows an example where (a) represents the processing time and (b) represents each individual job routing.

To proper represent the problem in a Gantt chart, a triplet ijk was used to

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

Table 3.3: An example of a job shop problem. Source: Baker [6]

represent that job i, operation j requires machine k. Figure 3.4 shows a feasible schedule based on Table 3.3.

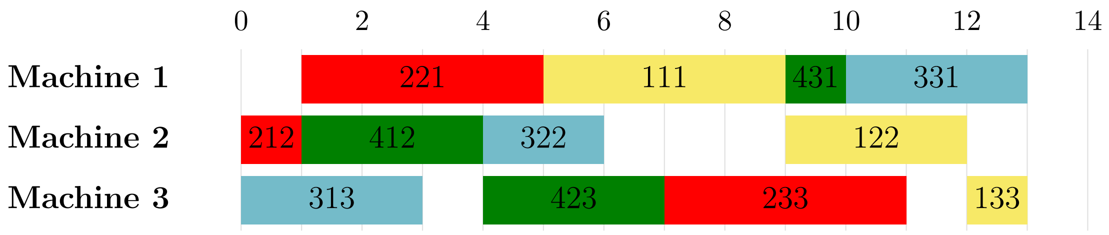


Figure 3.4: Example of a job shop schedule. Source: Baker [15]

If for the FSP the maximum number of combinations that need to be analyzed for an optimal solution is $n!$, we can infer that for the JSP this number increases to at least $(n!)^m$. Making it a generally harder problem when compared to FSP.

3.2 Maintenance Preliminaries

Machines will ultimately fail. Industrial machinery are not excluded from this rule. The importance of taking into account the different failure behaviours of our schedule is because they directly impact the availability of our resources. This thesis considers preventive maintenance and random breakdowns to try to approximate as much as possible to a real life scenario where machines undoubtedly fail. To better understand the maintenance approaches used in this thesis, we present the concept of reliability and maintainability. According to Ebeling [48] reliability is the probability that a system or component will be operational for a certain period of time given a specified set of parameters and settings. Maintainability, also defined by Ebeling [48], is the probability that a system or component will be back to a specific functioning state after a restoration or repair process was carried out given some recommended methods. In a system where machines are allowed to fail and then repaired, the interaction of reliability and maintainability will provide the total run time, i.e., the availability of our machines given a specified scenario. Deterministic downtime are introduced as preventive maintenance and stochastic failures are introduced as random breakdowns in this thesis. The next two subsections explain important concepts regarding Preventive Maintenance (PM) in subsection 3.2.1 and the random breakdowns in subsection 3.2.2.

3.2.1 Preventive Maintenance

The health state and age of a machine will dictate when it will be prone to stoppage due to failure. Theoretically, a machine will be stopped for preventive maintenance after the reliability threshold has been exceeded, after which failure may occur. Preventive maintenance is scheduled to keep the reliability at or above a threshold. This way we take control of when a machine will stop and restore its reliability to its original state. When dispatching the operations of a job to the shop floor with many machines, the preventive maintenance tasks will also compete for time on a machine. Therefore, all considerations regarding the scheduling of preventive maintenance and the sequencing of the operations of the jobs have to be done integrally to avoid logistic conflicts on the machine floor.

In this thesis, the failure rates of the machines are dictated following a Weibull distribution rate because it is one of the most studied and widely used in the literature. Considering t_j as the machine age, Ebeling [48] shows that the reliability of a machine can be calculated following equation 3.1.

$$R_j = e^{-(t_j/\theta_j)^{\beta_j}} \tag{3.1}$$

Consider that t_{pj} is the time that it takes to perform a preventive maintenance on machine M_j, t_{rj} is the time that it takes to perform a corrective maintenance (CM) on machine M_j, β_j is the shape parameter of the Weibull distribution failure function and θ_j is the scale parameter of the Weibull distribution failure function.

The challenge is to guarantee an optimal interval between each preventive maintenance task. T_j^* is the optimal time interval that a preventive maintenance task has to be scheduled so that the availability of the machine is considered optimal. Cui et. al. [15] derived equation (3.2) based on the Weibull distribution.

$$T_j^* = \theta_j\left(\frac{t_{pj}}{t_{rj} * (\beta_j - 1)}\right)^{(\frac{1}{\beta_j})} \tag{3.2}$$

By scheduling the preventive maintenance tasks as late as possible in the machine's scheduling horizon while keeping its age smaller than T_j^*, availability is maximized. Every time a preventive maintenance is carried on a machine, its age is restored to "as good as new" state. During the interval that exists between each preventive maintenance task, the machine is available to process the job's operations normally. These intervals are identified as batches. The operations are scheduled in such a way that they will always finish processing before a preventive maintenance task is carried. If an operation would be "cut" by a preventive maintenance task, it is postponed to the next batch.

Figure 3.5 illustrates the scheduling horizon of a machine after proper preventive maintenance task scheduling. This is a single machine example with $t_{pj} = 2$, $T_j^* = 10$ and the processing time of job i is $p_i = \{3, 2, 3, 4, 4, 2, 3, 2, 3, 4, 4, 2, 3, 5\}$.

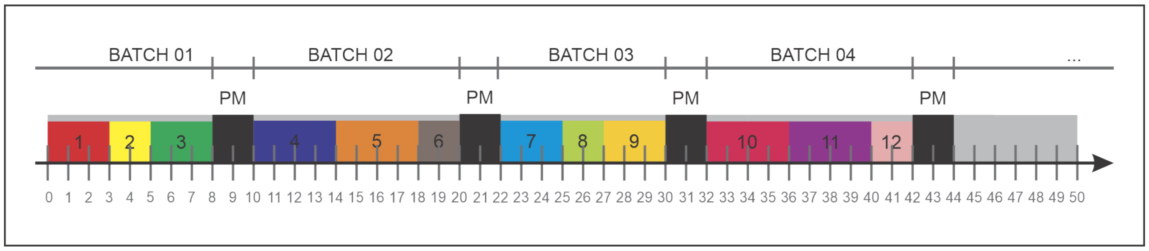


Figure 3.5: An example of the schedule horizon with proper preventive maintenance scheduling.

Note that jobs 4 and 10 had their start time postponed until the next available batch, so they would not be interrupted midway by a preventive maintenance task. The total run time T_j^* was never exceeded in a batch.

3.2.2 Breakdown

Even with all effort to predict the specific time of failure deterministically, random breakdowns are still likely to happen when a machine is running in real life. Cui et. al. [15] explained that in each preventive maintenance batch, a breakdown can be modelled as a stochastic point process, a non-homogeneous Poisson process to be more specific. When a breakdown happens, corrective maintenance (minimal repair) is carried out immediately to fix the machine. After a corrective maintenance, the age of the machine is not restored to "as good as new", but instead is assumed to be restored to its age at the time of failure.

Based on the reverse function of the reliability (Equation (3.3)) it is possible to create a sampling method to get the sampled breakdown times in a given machine M_j during a batch.

$$F^{-1}(p) = \theta[-\ln(1-p)]^{\frac{1}{\beta}} \tag{3.3}$$

Cui et. al. [15] described Equation (3.4) that is the τ^{th} breakdown time in a batch, where ς is an uniformly distributed random variable $\varsigma_\chi \sim U(0,1)$.

$$bt_{\tau j} = R_j^{-1}[\prod_{\chi=0}^{\tau}(1-\varsigma_\chi)] \tag{3.4}$$

Let us reconsider the preventive maintenance example from subsection 3.2.1. Figure 3.6 represents the same scheduling horizon now with random breakdowns being generated and inserted along it. $BD_\tau = \{\{6,13\},\{16\},\{26,31\},\{\}\}$

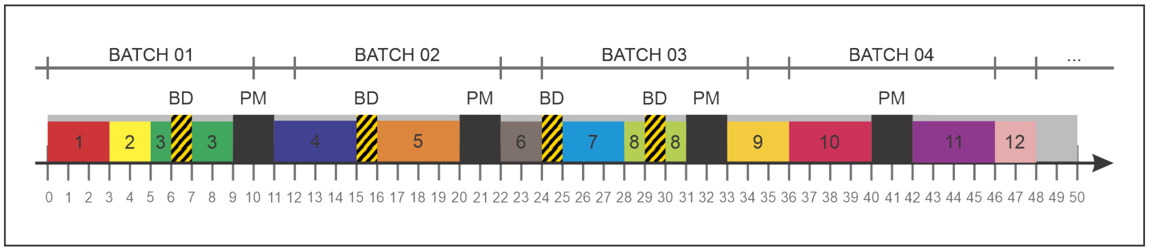


Figure 3.6: An example of the schedule horizon with proper preventive maintenance scheduling and random breakdowns.

Random breakdowns are generated within each batch separately. It is possible for more than one breakdown to happen in a given batch (note batches 1 and 3). If the sampled start time of a breakdowns happens during or after a preventive maintenance task (second breakdown during the first batch $BD_1 = \{6,13\}$) these are discarded and not considered. Only breakdowns that happen inside the batch impact the result of that specific batch on the schedule. After the corrective maintenance, the job operation that was being executed can resume without any further penalty.

A simplified way to estimate a specific time of a failure is to consider the mean time to failure (MTTF) of a machine. Ebeling [48] provided equations 3.5 and 3.6 to compute the MTTF in a Weibull distribution governed process.

$$MTTF_j = \theta_j \Gamma(1 + \frac{1}{\beta_j}) \tag{3.5}$$

$$\Gamma(x) = \int_0^\infty y^{x-1} e^{-y} dy \tag{3.6}$$

Computing the MTTF will prove sufficient to compute a scenario where both preventive maintenance and breakdown are added to the scheduling horizon. A machine will stop working as if it has failed at the calculated MTTF. The MTTF is considered as a point of failure along the machine accumulated run-time. Only a corrective maintenance is carried out at the MTTF, the machine's age is untouched, and the machine run-time is set to zero so that if its accumulated value reaches the MTTF again, another failure is added. The machines will restore theirs age following the previous preventive maintenance schedule.

3.3 Metaheuristics Preliminaries

Many useful algorithms rely on mathematical models such as linear, integer, and nonlinear programming to achieve optimal solutions. For complex and large problems, these techniques might not be able to provide an optimal solution, or even a feasible solution within a reasonable time. When dealing with this category of problem a heuristic method is commonly developed. A heuristic method as described by Hiller et al. [24] tries to discover a very good feasible solution (not exactly optimal) for a given problem. It is expected from a good heuristic that it can produce good or near optimal solutions for large or complicated problems. The majority of the algorithms try to search for a better solution in the solution space of the problem. Since many of them are commonly written as an iterative algorithm, a better result is expected to be found from a previously found best solution. Different algorithms have different stopping criteria, the most commons are the total run time or iteration limit. The first one stops the execution of the heuristic after a determined amount of time passed and the later one stops the execution when a pre-determined number of runs are reached.

Even though heuristics have been proven useful, a well-constructed unique heuristic has to be made for every single problem at hand. A new paradigm on how to design such optimization heuristics has been researched recently and are called metaheuristics. Metaheuristics are not specifically designed for a problem, instead it lays out the general structure and the rules that have to be followed to design a good heuristic that will fit a specific problem. Two different kinds of metaheuristics were used to design parts of the algorithms presented in this thesis. The next two Subsections explain the Simulated Annealing (SA) in Subsection 3.3.1 and Genetic Algorithm in Subsection 3.3.2.

3.3.1 Simulated Annealing

The Simulated Annealing metaheuristic, as the name suggests, tries to mimic the heat treatment process that are used to change the physical and electrical properties of materials such steel, aluminum, cooper and brass. It is known to be one of the most widely used metaheuristics that started with work from Kirkpatrick [28] and Cerny [13]. The main goal is to avoid local optimum and broadly search the solution space for better solutions. Local optima in this context mean a solution that is optimal only for the solutions that are immediately close to it. Due to the limited knowledge of the solution space, one might fall into a local optimum if the solution space is not well searched. The simulated annealing process tries to avoid that by occasionally accepting a worse solution than the actual good solution based on a neighbourhood search algorithm. This random search is intimately tied to the temperature parameter. When the procedure starts, the temperature is set to a high value which increases the probability of acceptance of new “trial solutions” even if they are not better than the actual solution. After each iteration (or n iterations), the temperature parameter is updated and the temperature drops down causing fewer and less worse solutions to be accepted. After some time, the procedure stops at a low temperature and the last solution found is reported as the approximation of the optimum objective function value. The way the temperature parameter is updated is called “temperature schedule” and is one of the most important parameters in this procedure. It directly impacts not just the execution time of the algorithm but the quality of the best solution.

```
Input:
    Initial solution $s_0$;
    Initial temperature $t_0 > 0$;
    Temperature reduction function $\alpha$;
    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 2: Simple Simulated Annealing Procedure. Source: Dowsland [17]

Let us denote by s_0 the initial solution, t_0 the initial temperature, α the temperature reduction function and $N(s_0)$ the neighbourhood of solutions from s_0. A simple Simulated Annealing is described in Algorithm 2. This simplistic approach to Simulated Annealing is memory-less, meaning that the solutions found throughout the execution of the algorithm are discarded and only the last trial solution is declared as the approximation to the optimal solution for the objective function. In this thesis, a list was used to store every trial solution and the best solution found throughout the execution of the algorithm is then chosen as the approximation of the optimal value of the objective function.

3.3.2 Genetic Algorithm

Introduced by Holland [25] and later applied to the job shop problem by Davis [16] the Genetic Algorithm, inspired by the theory of evolution from Charles Darwin, uses breeding, selection, and mutation over many different generations to produce a population with individuals that represent good solutions. The algorithm tries to mimic the behaviour observed naturally called survival of the fittest, where individuals that carry good genes have a better chance of survival.

A basic genetic algorithm starts with a finite set of individuals that are represented by a list of chromosomes. These chromosomes represent the traits of each individual and a fitness function (cost function) can be used to evaluate the fitness of each one of these individuals independently. The algorithm then randomly matches two individuals forming pairs, then parts of the chromosomes are exchanged between them, creating new individuals in a procedure called crossover. These new individuals are expected to carry traits from both parents that would potentially lead to a better solution. Next, each one of the newly created individuals is given a chance to suffer a mutation, this will guarantee that some random variability is added to the population. The individuals are evaluated and the worst performing ones are “killed”, meaning that they are not added to the initial population of the next generation. This procedure cycles creating new generations until some stopping criteria are reached, the diversity of the population is minimal, or the result is good enough. The population is expected to converge since its size is finite. Figure 3.7 shows how a basic genetic algorithm works.

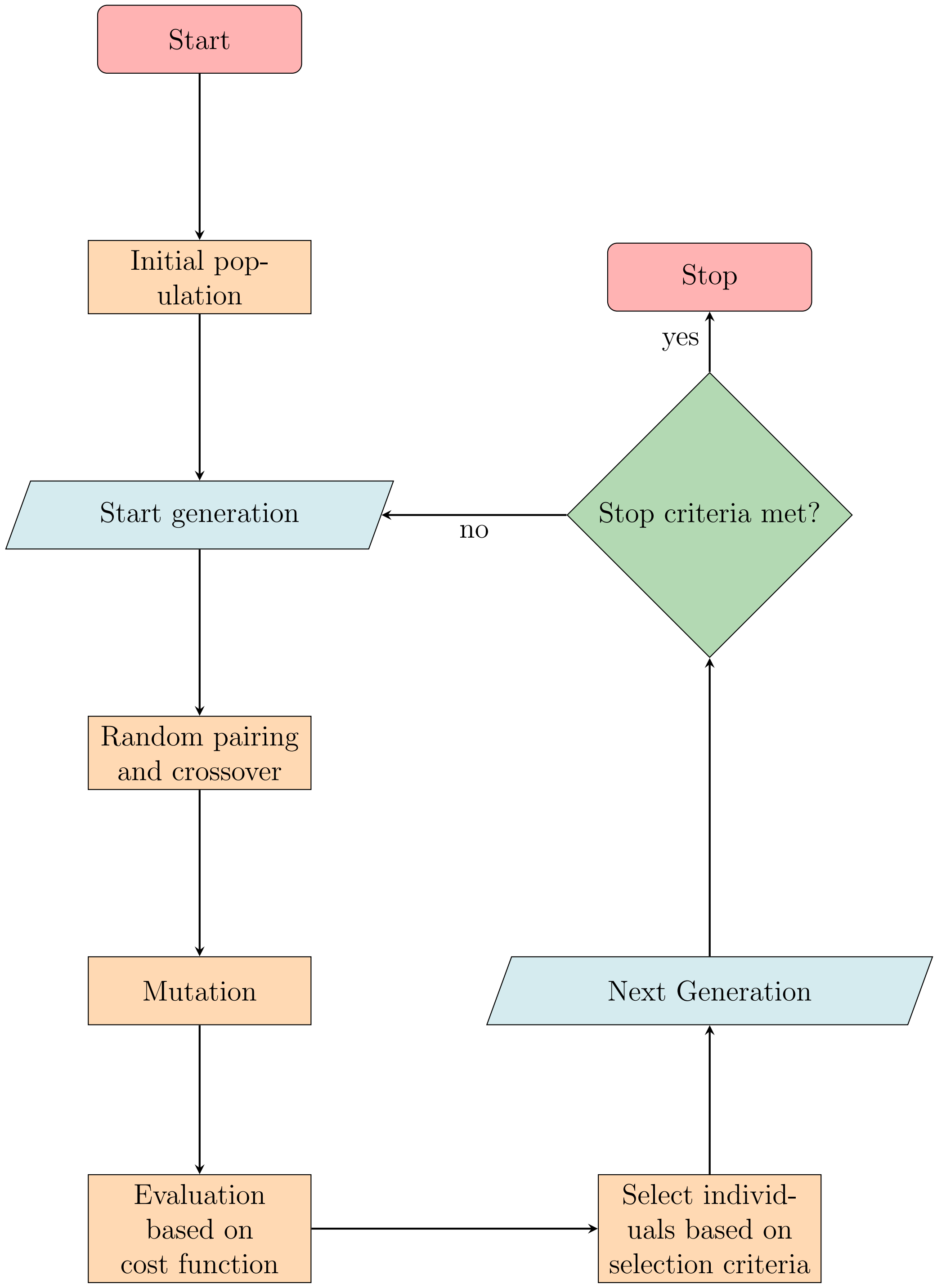

Start
Initial population
Start generation
Random pairing and crossover
Mutation
Evaluation based on cost function
Select individuals based on selection criteria
Next Generation
Stop criteria met?
no
yes
Stop

Figure 3.7: A block diagram of a simple genetic algorithm procedure.

This sub-section gives only the basic idea of a genetic algorithm, a more in-depth explanation is widely available in the literature and is not covered here. Evolutionary algorithms like the genetic algorithm are known to produce good results for the job shop scheduling problem as Bierwirth et. al. [9] observed.

3.4 Objective Function and Robustness

The optimization algorithms proposed here need a way to quantify the solution of the analyzed problems. This quantification has to be done in a way that the described algorithms are able to minimize its resulting value. We define an Objective Function (OF) that quantifies the metrics that need to be minimized in order to improve the resulting schedules. This section briefly summarizes all the metrics that impact the objective function, describes the concept of robustness, and finally present the derived objective function. The following notations are used.

Indices:

i index of jobs;

j index of operation;

k index of machines;

Sets:

J set of jobs;

M set of machines;

O set of operations of J_k;

Parameters:

n number of jobs;

m number of machines;

For the flow shop problem, p_{ik} processing time of job i on machine k;

For the job shop problem, p_{ijk} processing time of job i, operation j on machine k;

t_{pk} average preventive maintenance time of the machine M_k;

t_{rk} average corrective maintenance time of the machine M_k;

β_k the shape parameter of the failure function of the machine M_k;

θ_k scale parameter of the failure function of the machine M_k;

A set of jobs J is supposed to be processed on a set of machines M for minimizing a bi-objective function. Every job J_i consists of a sequence of o operations $O_{i0}, O_{i1}, \ldots, O_{ik}$ for the flow shop problem or $O_{ij0}, O_{ij1}, \ldots, O_{ijk}$ for the job shop problem. Each operation has a fixed processing time p on machine M_k. Only one job can be processed on a machine and a job can only be processed by one machine at a time. The jobs are available at the beginning of the scheduling horizon. There are an infinity capacity for the machines, therefore there is no buffers or waiting between any machines. In the flow shop case, the jobs' sequence is the same at every machine (permutation flow shop), when dealing with the job shop case the jobs' sequence is unique for every machine and a machine sequence matrix is also provided to show the sequence which each job J_i has to follow.

The machines are prone to failure and run time until a failure for said machines are governed by a Weibull probability distribution. Machine M_k has the following set of parameters: t_{pk}, t_{rk}, β_k and θ_k. Since the machines deteriorate over time, we consider $\beta_k > 1$. Random breakdown failures occur and impact product quality and system stability. To overcome these breakdowns, preventive maintenance is carried out during the machine's up time, which improves the machinery condition and reduces the chances of occurrence of breakdowns. The machine's age is reset to 0 after each preventive maintenance is carried out; in other words, the machine is restored to an "as good as new" state. Even with all preventive measures, random (not predicted) breakdowns may occur and, in this scenario, corrective maintenance should occur. In the case of corrective maintenance, a minimal repair is carried out which will restore the machine to a functioning state, but its age will stay the same as when the failure happened. No additional time will be inserted in the job after a corrective maintenance and the machine will resume the job it was doing prior to the failure.

The jobs' sequencing must be decided as traditional flow shop and job shop scheduling problems without any maintenance, yet maintenance policy decisions should be taken into consideration. The number of preventive maintenance and their start times for each machine in the scheduling horizon must be decided. The algorithms in this thesis integrate the flow shop job sequencing decision or the job shop sequencing decision with the preventive maintenance policy while also considering the random breakdowns that make corrective maintenance necessary. A job schedule σ^0 is created and then executed on the shop floor. If a corrective maintenance happens, the schedule is shifted and revised following the rescheduling policy. An actual schedule σ^w is then obtained at the end of the scheduling horizon.

Robustness is an important metric that guarantees the schedule's adherence to the management's goal. In this thesis, two different robustness metrics were used and analyzed. The first robustness metric is defined as "quality robustness", which represents the schedule's performance towards a defined metric. The chosen metric for this study is the total length of the schedule, i.e., the makespan. The second robustness metric is defined as "solution robustness", which represents the stability of the schedule. A schedule is considered stable when there are less divergence between the start times of jobs of the planned schedule σ^0 and the start times of jobs of the realized or simulated schedule σ^w.

For the flow shop problem, the robustness of the solution is evaluated as following: for the quality robustness we declare $Z_1 = E_w[C_{ik}^w]$ that minimizes the expected makespan of the schedule and for the solution robustness we declare $Z_2 = E_w[\sum_{i=0}^{n}\sum_{k=0}^{m}(S_{ik}^w - S_{ik}^0)]$ that minimizes the total divergence between the planned start time of every job in every machine to its realized start time based on σ^0 and σ^w respectively. For the job shop problem, the quality robustness is measured using $Z_1 = E_w[C_{ijk}^w]$ and the solution robustness using $Z_2 = E_w[\sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=0}^{m}(S_{ijk}^w - S_{ijk}^0)]$.

Since we are considering two performance measures, the objective function to be minimized is defined in Equation (3.7), where $\rho_2 = 1 - \rho_1$.

$$z = \rho_1 Z_1 + \rho_2 Z_2 \tag{3.7}$$

The parameter ρ_1 is a real number between 0 and 1 and serves as the weight given to the quality robustness objective.

Chapter 4

Methodology

In this Chapter, all the procedures and methods that are used to achieve the objective of this thesis are outlined. First the surrogate models defined as "Oracles" are described, then the simulation procedure is presented, followed by the formal description for the FSP optimization algorithms and its graphical representations, and finally the formal description for the JSP optimization algorithms and its graphical representations.

4.1 Oracles

The definition of the word "oracle" from the Collins English dictionary states: "4. a person who delivers authoritative, wise, or highly regarded and influential pronouncements." [47]. In this thesis, oracles are self-contained entities, similar to the formal definition of the word, that evaluate a feasible job sequence and return an expected performance metric. These results are later used by the algorithms to find the best performing job sequence. A complete solution is defined by $(x, \mathbf{Y}, \mathbf{ST})$, where x is a given job sequence, $\mathbf{Y}$ is the preventive maintenance position matrix and $\mathbf{ST}$ are the jobs start times matrix. We declared oracles to be used as criteria to analyze the job sequencing x and its impact on the schedule's makespan or objective function value. Oracles 1, 2 and 3 are used to analyze an entire given neighbourhood $\mathbf{X}$, which is a matrix of different job sequences, and return the best found regarding the schedule's makespan. Oracle 4 evaluate a single job sequence using simulation to estimate the average behaviour of the system.

Oracle 1, denoted as $Z_1(\mathbf{X})$, represents the best found schedule's makespan value without any consideration regarding maintenance and machine breakdown for a given neighbourhood $\mathbf{X}$. The schedule's makespan is calculated directly assuming that all machines are available at time 0 and throughout the horizon.

Oracle 2, denoted as $Z_2(\mathbf{X})$, represents the best found schedule's makespan including the preventive maintenance in every machine. According to maintenance theory, aiming at the maximization of the machine's availability, we build the optimal interval of preventive maintenance by deriving the following function $T_k^* = \theta_k\{t_{pk}/[t_{rk}\,(\beta_k - 1)]\}^{(\frac{1}{\beta_k})}$. Machines can work if their ages are smaller than T_k^*. A series of preventive maintenance are inserted as late as possible at every machine. The matrix $\mathbf{Y}$, which stores the preventive maintenance positions throughout the scheduling horizon, is created. Using a job sequence and the created matrix $\mathbf{Y}$ we can compute the schedule's makespan directly assuming that all machines are available at time 0.

Oracle 3, denoted as $Z_3(\mathbf{X})$, represents the best found schedule's makespan including the preventive maintenance and breakdowns that occur at the machine's Mean Time To Failure (MTTF). The preventive maintenance portion of this oracle is computed exactly like in $Z_2(\mathbf{X})$ and the matrix $\mathbf{Y}$ is created as a result. Governed by a Weibul probability distribution function, the average amount of time that a machine runs until it fails can be derived by computing $MTTF_k = \theta_k\Gamma(1 + (\frac{1}{\beta_k}))$. A machine is restored to age 0 after every preventive maintenance and the total time between the end of a preventive maintenance to the start of another preventive maintenance is called a batch. If a breakdown happens before the next preventive maintenance, i.e., inside of a batch, the job affected by it is immediately suspended, a corrective maintenance takes place and the job is immediately resumed. More than one breakdown may occur per batch. A matrix $\mathbf{ST}$ that represents how many breakdowns happened within each batch is created. Using a job sequence, the preventive maintenance matrix $\mathbf{Y}$, and the breakdown matrix $\mathbf{ST}$ we can compute the schedule's makespan directly assuming that all machines are available at time 0.

Oracle 4, denoted as $Z_4(x)$, represents the objective function value when the machines' failure uncertainty and preventive maintenance are considered. The preventive maintenance portion of this oracle is computed exactly like in $Z_2(\mathbf{X})$ and the matrix $\mathbf{Y}$ is created as a result. Breakdowns are created in a simulation process following Algorithm 3 presented by Cui et al. [15]. As described, the average objective function value over all scenarios ($\mathbf{W}$) is declared as the expected objective function value for the oracle.

4.2 Monte Carlo Simulation

A Monte Carlos simulation approximation of the expected realized schedule was developed by Cui et al. [15] for the flow shop problem and it is presented as follows. Consider the objective function, where C_{ik}^{w} is the realized finish time of operation O_{kj} in scenario w, S_{ik}^{w} realized start time of operation O_{kj} in scenario w, and ξ_{ik}^{w} is the realized number of breakdowns when processing O_{kj}, let us expand and analyze it below.

$$\begin{aligned}
z &= \rho_1 E_w[C_{ik}^{w}] + \rho_2 E_w[\sum_{k=0}^{m}\sum_{i=0}^{n}(S_{ik}^{w} - S_{ik}^{0})] \\
z &= \rho_1 E_w[S_{ik}^{w} + p_{ik} + \xi_{ik}^{w} t_{rk}] + \rho_2 E_w[\sum_{k=0}^{m}\sum_{i=0}^{n}(S_{ik}^{w} - S_{ik}^{0})] \\
z &= \rho_1 (E_w[S_{ik}^{w}] + p_{ik} + t_{rk} E_w[\xi_{ik}^{w}]) + \rho_2 \sum_{k=0}^{m}\sum_{i=0}^{n} E_w[S_{ik}^{w}] \\
&\qquad - \rho_2 \sum_{k=0}^{m}\sum_{i=0}^{n} E_w[S_{ik}^{0}] \qquad (4.1)
\end{aligned}$$

The exact value of $E_w(S_{ik}^{w})$ for any given solution cannot be obtained in polynomial time, therefore, evaluations that are based on the schedule realization cannot be obtained in reasonable time. S_{ik}^{0}, $E_w(\xi_{ik}^{w})$ and p_{ik} can be calculated in polynomial time. ξ_{ik}^{w} is governed by a Poisson probability distribution, according to maintenance theory, with $\lambda_{ik} = (\frac{a_{ik}}{\theta_k})^{\beta_k} - (\frac{b_{ik}}{\theta_k})^{\beta_k}$ and $Pr(\xi_{ik}^{w} = \eta) = (\lambda_{ik})^{\eta} e^{-\frac{\lambda_{ik}}{\eta!}} \forall \eta \in [0, +\infty)$. There is no close form to derive the expectation of the realized start time, however, the expected value of ξ_{ik}^{w} can be derived.

For $E[\xi_{ik}^{w}] = \lambda_{ik} = (\frac{a_{ik}}{\theta_k})^{\beta_k} - (\frac{b_{ik}}{\theta_k})^{\beta_k}$ the values for a_{ik} and b_{ik} can be easily calculated. a_{ik} represents the age of the machine after job i in machine k and b_{ik} represents the age of the machine before job i in machine k. After a preventive maintenance, the machine's age turns 0 and the machine starts to degenerate from that point forward. Every preventive maintenance should be treated separately as a batch. A batch is a collection of all operations in the same preventive maintenance period. Let B_{lk} be the number of batches in M_k, l be the l^{th} batch in M_k, L_k be the number of batches in M_k, n_{lk} be the number of operations in B_{lk}, and q_{lk} be the total processing time of operations in B_{lk}.

The machine random breakdowns are modelled as a stochastic point process model for each preventive maintenance period. Governed by a Weibull probability function, machine failure is known to be a non-homogeneous Poisson process during a preventive maintenance period. $R_k = e^{-\frac{t_k}{\theta_k}^{\beta_k}}$ is the machine reliability, where t_k describes the machine age. The modelled sampling method to acquire the τ^{th} breakdown time in the machine M_k is $bt_{\tau k} = R_k^{-1}[\prod_{\chi=0}^{\tau}(1-\varsigma_\chi)]$, where $\varsigma_\chi \sim U(0,1)$ is a uniformly distributed random variable, and $R_k^{-1}(g)$ is the reverse function of $R_k(g)$. The number of scenarios (sample size) is W, and $z(\sigma^0)$ is the solution's objective value. A detailed description of the simulation procedure is described in Algorithm 3.

```
Input:
    Number of scenarios $W$
    Set $z_w = \{0, 0, \ldots, 0\}(w = 0, 1, \ldots, W)$
    Jobs list $\mathbf{X}$
    Preventive maintenance matrix $\mathbf{Y}$
    Planned start times matrix $\mathbf{ST}$
Output: $z(\sigma^0)$
    Step 1. Calculate $a_{ik}$, $b_{ik}$ $\forall i, k$, and compute $q_{lk}$, $n_{lk}$ $\forall l, k$.
    Step 2. Set $w = 1$.
    Step 3. For $j = 1$ to $m$, do:
      3.1. Set $\{\xi_{ik} \forall k\} = \{0, 0, \ldots, 0\}$; set $l = 0$; set $n_0 = 0$.
      3.2. Set $\tau = 0$.
      3.3. Generate random sample $\varsigma_\chi \sim U(0,1)$, and set
      $bt_{\tau k} = R_k^{-1}[\prod_{\chi=0}^{\tau}(1-\varsigma_\chi)]$
      3.4. If $bt_{\tau k} < a_{[1+n_{l-1}]k}$, set $\xi^w_{[1+n_{l-1}]k} = \xi^w_{[1+n_{l-1}]k} + 1$; otherwise if $\exists\ \delta$ that
      satisfies $a_{[\delta+n_{l-1}]k} < bt_{\tau j} < a_{[\delta+1+n_{l-1}]k} (\forall \delta \leq n_{l-1})$, the set
      $\xi^w_{[1+n_{l-1}]k} = \xi^w_{[1+n_{l-1}]k} + 1$. Set $\tau = \tau + 1$. Go to 3.2.
      3.5. $l = l + 1$. If $l \leq L_k$, go to 3.2.
    Step 4. Compute $S^w_{ik} \forall k, \forall i$, and the objective function value
     $z^w = \rho_1(C^w_{ik}) + \rho_2(\sum_{k=0}^{m}\sum_{i=0}^{n}(S^w_{ik} - S^0_{ik}))$.
    Step 5. Set $w \leftarrow w + 1$. If $w < W$, go to Step 3.
    Step 6. Calculate $z(\sigma^0) = \frac{\sum_{w=1}^{W} z^w}{W}$.
```
Algorithm 3: Monte Carlo simulation procedure. Source: Cui et al [15].

The same approach was adapted and used to simulate the behaviour of the job shop system.

4.3 Flow Shop Optimization Algorithms

The Flow Shop Scheduling Problem (FSP) assume that a number of machines to process a number of jobs, the sequence in which the jobs are executed in the shop floor is the same for every machine, i.e., all jobs follow the same route in the shop floor. In this study, preventive maintenance and random breakdowns are added to the scheduling horizon to better represent the behaviour of this system to an actual industry setting.

Using the Oracles described in Subsection 4.1 three strategies were outlined to try to find the best job sequencing that minimizes an Objective Function. Since Monte Carlo simulation is very time consuming and computational hungry, there is a need to minimize the amount of simulation runs to be executed. The first three Oracles analyze a given neighbourhood of solutions (job sequences) and return the best job sequencing regarding their own metrics. By doing all this computational effort early, only three simulation procedures have to be carried to analyze these three good job sequences. The expected best job sequencing for the overall problem should be also the best job sequencing for the described Oracles.

The remainder of this Section outline in detail three different strategies created to try to find the best job sequencing taking into consideration all the described requirements for the flow shop problem with added preventive maintenance and random breakdowns.

4.3.1 The optimization algorithms

The construction algorithm described by Nawaz et al. [37] named NEH method, is known to be the best constructive heuristic regarding $F_m//C_{max}$ (flow shop scheduling problem minimizing total makespan). The first job's sequence x_0 is always obtained using the NEH method assuming that all machines are always available at time 0.

Algorithms 4, 6 and 7 try to optimize the jobs' sequence by performing a two-loop procedure. A inner loop that tries to improve the jobs' sequencing locally based on the evaluation of each oracle and an outer loop that focuses on the overall objective function in the broader scope. The algorithms are described and followed by their formal definitions below.

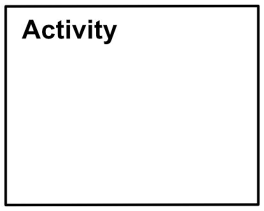
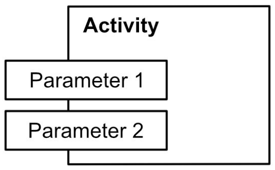
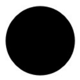
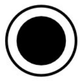
Algorithm 4 executes a full neighbourhood local search heuristic in the inner loop and then uses the simulated annealing metaheuristic on the outer loop to escape local optima and avoid the termination of the algorithm prematurely. The inner loop can be described as follows: the neighbourhood ($N(x_0)$) for a given jobs' sequence (x_0) is created by switching the position of every two jobs (pairwise exchange). The created neighbourhood has a size of $\frac{n(n-1)}{2}$. For every neighbour x in $N(x_0)$ the values of $Z_1(x)$, $Z_2(x)$ and $Z_3(x)$ are calculated using their respective oracles. The jobs' sequences with the minimum objective value for $Z_1(x)$, $Z_2(x)$ and $Z_3(x)$ are selected and declared as x_1^*, x_2^* and x_3^* respectively. Then $Z_4(x_0)$, $Z_4(x_1^*)$, $Z_4(x_2^*)$ and $Z_4(x_3^*)$ are calculated and the optimal solution x^* is declared being the minimum objective value found between $Z_4(x_1^*)$, $Z_4(x_2^*)$ and $Z_4(x_3^*)$. A simulated annealing process is initiated to loop over the inner portion of the algorithm to improve the quality of the result; therefore, the outer loop can be described as follows: the initial temperature is given by T, the update factor is described by α and the probability of acceptance is calculated as $Pr_{acceptance} = e^{\frac{Z_4(x)-Z_4(x^*)}{T}}$. If the optimal value found ($Z_4(x^*)$) is smaller than the initial solution ($Z_4(x_0)$) or if the probability of acceptance is greater than or equal to an uniformly random generated number between $(0,1]$, the solution (x^*) is accepted as a new trial solution. The initial solution x_0 is replaced by (x^*), the temperature is updated to $\alpha * T$, and the inner loop is repeated. If none of those conditions are met, the algorithm stops and best job sequence found throughout the algorithm's execution is declared as the approximation of the optimal solution.

Algorithm 6 executes a simulated annealing metaheuristic process with a k-pair neighbourhood optimization heuristic in the inner loop and the outer loop repeats the inner loop if a better solution is found. The inner loop can be described as follows: three simulated annealing procedures will start in parallel, one for each oracle ($Z_1(x)$, $Z_2(x)$ and $Z_3(x)$). Let us consider $Z_1(x)$, the initial temperature is T, the update factor is described by α and the probability of acceptance is calculated as $Pr_{acceptance} = e^{\frac{Z_1(x_k)-Z_1(x^{new})}{T}}$. Denoted as $N(x_0)$, the neighbourhood for a given jobs' sequence x_0, which is a list with nbk members created by switching the position of two randomly chosen jobs (pairwise exchange). For every neighbour (x) in $N(x_0)$ the values of $Z_1(x)$ are calculated and the minimum one is chosen and declared as x^*. If $Z_1(x^*)$ is smaller than the initial solution $Z_1(x_0)$ or if the probability of acceptance is greater than or equal to an uniformly random generated number between $(0,1]$, the solution x^* is accepted. The initial solution x_0 is replaced by x^*, the temperature is updated to $\alpha * T$, and the inner loop is repeated. If none of those conditions are met, the algorithm stops, and x^* is declared being the solution with minimal $Z_1(x)$ found during the execution. This procedure happens similarly for $Z_2(x)$ and $Z_3(x)$. The jobs' sequence with the minimum objective value for each oracle is finally declared as x_1^*, x_2^* and x_3^* respectively. Then $Z_4(x_0)$, $Z_4(x_1^*)$, $Z_4(x_2^*)$ and $Z_4(x_3^*)$ are calculated and the optimal solution x^* is declared being the minimum objective value found between $Z_4(x_1^*)$, $Z_4(x_2^*)$ and $Z_4(x_3^*)$. The outer loop can be described as follows: if the optimal value found $Z_4(x^*)$ is smaller than the actual initial solution $Z_4(x_0)$ the algorithm set the initial solution as x^* and it repeat the inner portion of the algorithm trying to improve the solution, if no better solution is found the algorithm declares the best job sequence found throughout the algorithm's execution as the approximation of the optimal solution.

Algorithm 7 executes a simulated annealing metaheuristic process with a k-pair neighbourhood optimization heuristic on the inner loop and then uses the simulated annealing metaheuristic on the outer loop to escape local optima and avoid the termination of the algorithm prematurely. The inner portion of this algorithm is the same as the one described in Algorithm 6. The outer loop can be described as follows: the initial temperature is given by T, the update factor is described by α and the probability of acceptance is calculated as $Pr_{acceptance} = e^{\frac{Z_4(x_k) - Z_4(x^*)}{T}}$. If the optimal value found $(Z_4(x^*))$ is smaller than the initial solution $(Z_4(x_0))$ or if the probability of acceptance is greater than or equal to an uniformly random generated number between $(0,1]$, the solution (x^*) is accepted. The initial solution is replaced by it, the temperature is updated to $\alpha * T$ and the inner loop is repeated. If none of those conditions are met the algorithm stops and the best job sequence found throughout the algorithm's execution as the approximation of the optimal solution.

Graphical representation

To better represent the logic behind the proposed algorithms, a graphical representation using the UML (Unified Modelling Language) activity diagrams are presented. UML activity diagrams are similar to flow charts which aim to represent the behaviour of a system, a complex operation, a complex business rule, single or multiple use cases, business, or software processes [5]. The following notation was used for the activity diagrams presented here. There are many other structures and standardized symbols that are not covered in this text.

Symbol	Name	Description
	Activity Symbol	Encapsulate the symbols that represents a model.
	Activity Parameters	Rectangles that sits on the border of the activity symbol that represents the parameters from outside of the activity that are used by the modelled activity.
	Start Symbol	A black circle that represents the start point of the modelled activity.
	End Symbol	A black circle inside a hollow circle with a border that represents the end point of the modelled activity.

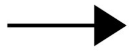
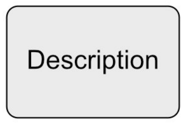
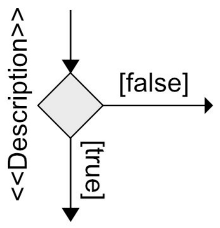
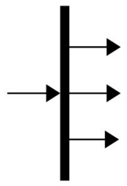
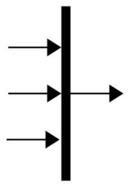
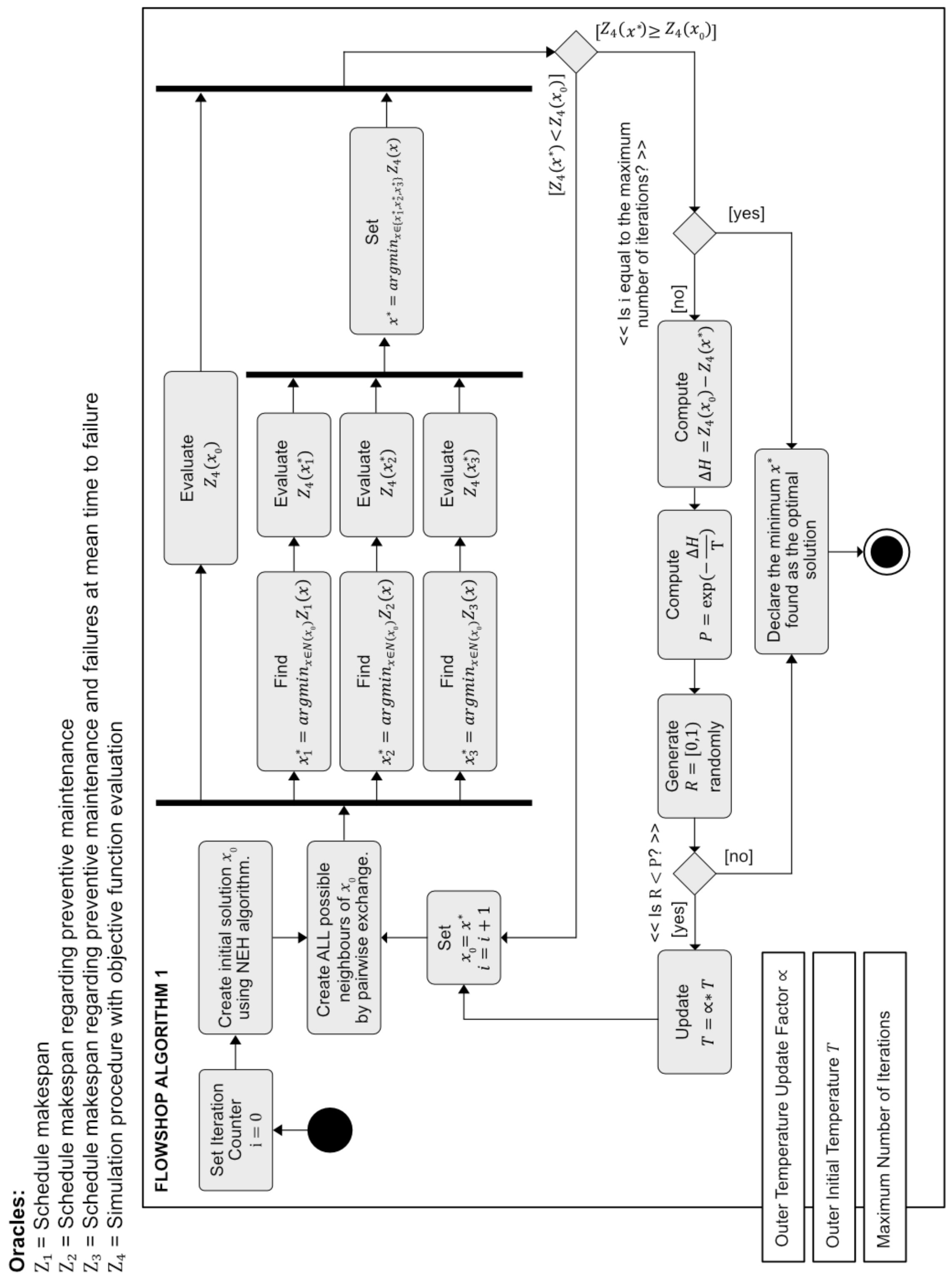
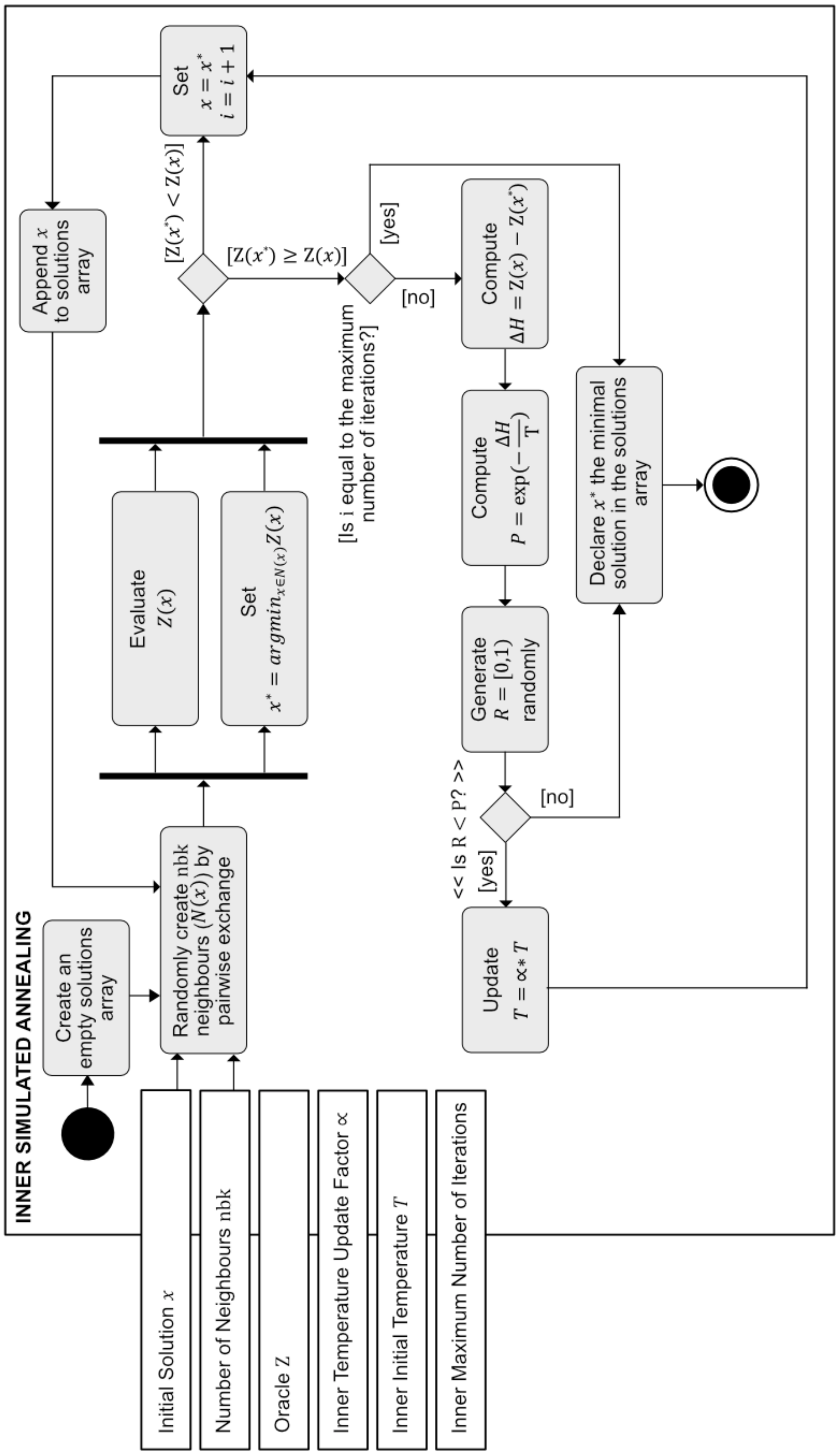
Symbol	Name	Description
	Flow Symbol	An arrow that represents the flow of information, action or data of the modelled activity.
	Action Symbol	A rounded rectangle with a description inside that represents an action of the modelled activity. This description can contain plain text or even code in a specific programming language if needed.
	Condition Symbol	A diamond-shaped symbol that represents a point of decision an branching. "<<" and ">>" are used to encapsulate a description, usually a question, that will lead to branching to one of the arrows leaving the symbol.
	Fork Symbol	A black bar with arrows pointing outwards from it. Splits a single activity flow into multiple concurrent and parallel activities.
	Joint Symbol	A black bar with arrows pointing to it. Joins two or more concurrent activities into a single flow.

Table 4.1: UML notation (adapted). Object Management Group [39]

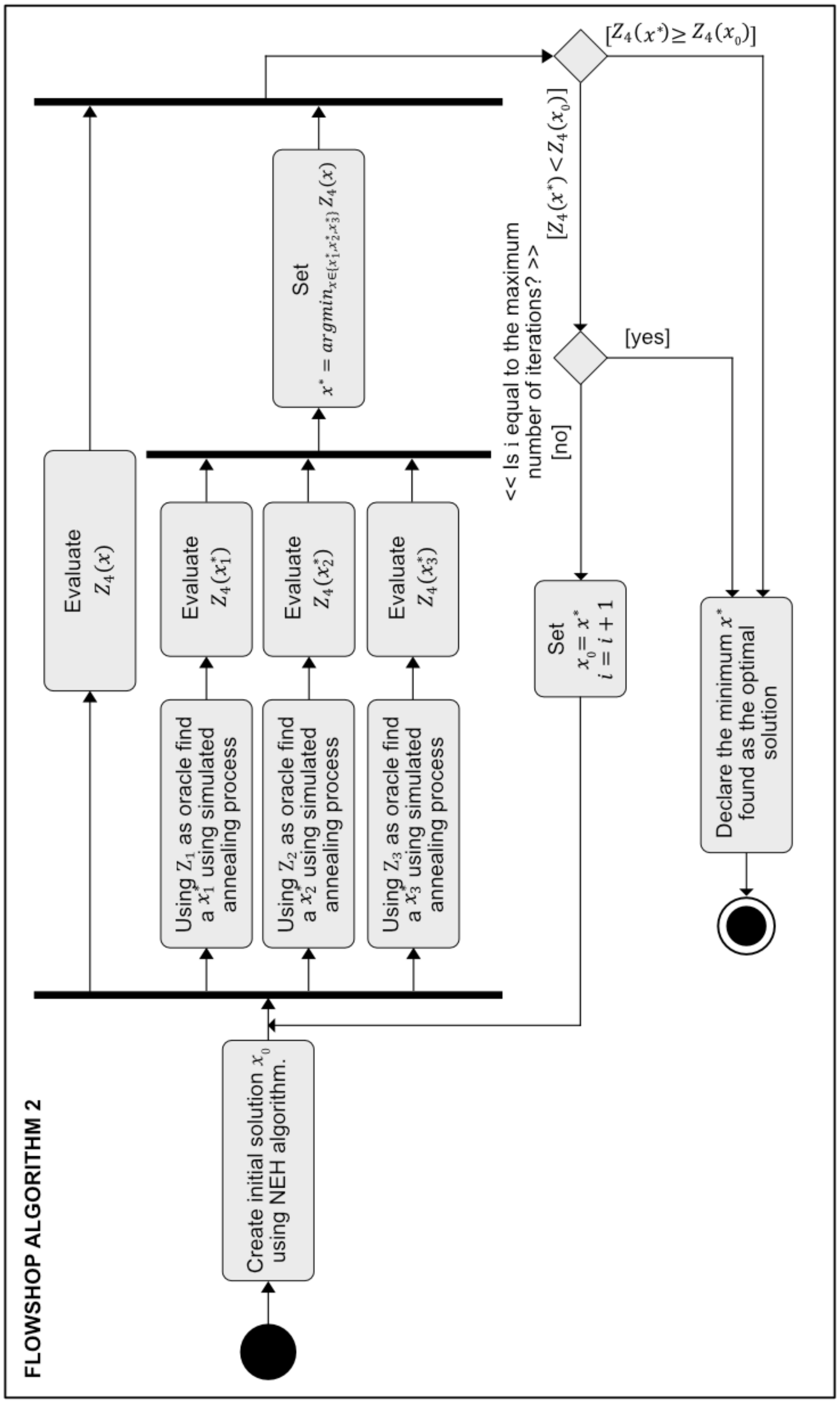

Oracles:
Z_1 = Schedule makespan
Z_2 = Schedule makespan regarding preventive maintenance
Z_3 = Schedule makespan regarding preventive maintenance and failures at mean time to failure
Z_4 = Simulation procedure with objective function evaluation
FLOWSHOP ALGORITHM 1
Set Iteration Counter i = 0
Create initial solution x_0 using NEH algorithm.
Create ALL possible neighbours of x_0 by pairwise exchange.
Evaluate $Z_4(x_0)$
Find $x_1^* = argmin_{x \in N(x_0)} Z_1(x)$
Find $x_2^* = argmin_{x \in N(x_0)} Z_2(x)$
Find $x_3^* = argmin_{x \in N(x_0)} Z_3(x)$
Evaluate $Z_4(x_1^*)$
Evaluate $Z_4(x_2^*)$
Evaluate $Z_4(x_3^*)$
Set $x^* = argmin_{x \in \{x_1^*, x_2^*, x_3^*\}} Z_4(x)$
$[Z_4(x^*) \geq Z_4(x_0)]$
$[Z_4(x^*) < Z_4(x_0)]$
Set $x_0 = x^*$ $i = i + 1$
<< Is i equal to the maximum number of iterations? >>
[yes]
[no]
Compute $\Delta H = Z_4(x_0) - Z_4(x^*)$
Compute $P = \exp(-\frac{\Delta H}{T})$
Generate $R = [0,1)$ randomly
<< Is R < P? >>
[yes]
[no]
Update $T = \propto * T$
Declare the minimum x^* found as the optimal solution
Outer Temperature Update Factor $\propto$
Outer Initial Temperature T
Maximum Number of Iterations

Algorithm 4: FSP optimization algorithm with complete neighbourhood exploration and outer simulated annealing

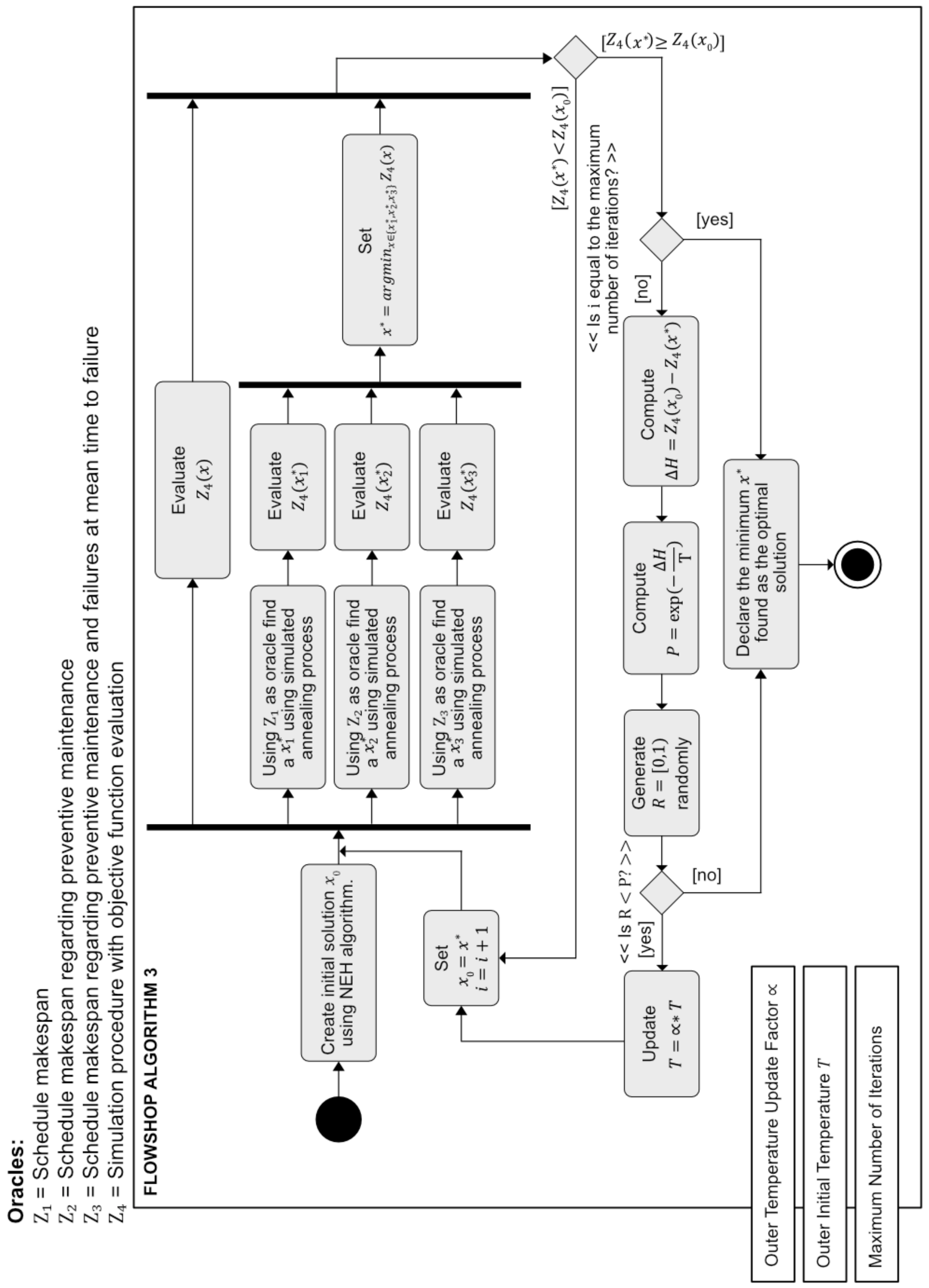

INNER SIMULATED ANNEALING
Initial Solution x
Number of Neighbours nbk
Oracle Z
Inner Temperature Update Factor $\propto$
Inner Initial Temperature T
Inner Maximum Number of Iterations
Create an empty solutions array
Randomly create nbk neighbours ($N(x)$) by pairwise exchange
Evaluate $Z(x)$
Set $x^* = argmin_{x \in N(x)} Z(x)$
$[Z(x^*) < Z(x)]$
$[Z(x^*) \geq Z(x)]$
Set $x = x^*$ $i = i + 1$
Append x to solutions array
[Is i equal to the maximum number of iterations?]
[yes]
[no]
Compute $\Delta H = Z(x) - Z(x^*)$
Compute $P = \exp(-\frac{\Delta H}{T})$
Generate $R = [0,1)$ randomly
<< Is $R < P$? >>
[no]
[yes]
Update $T = \propto * T$
Declare x^* the minimal solution in the solutions array

Algorithm 5: FSP local k-neighbour optimization simulated annealing

Oracles:
Z_1 = Schedule makespan
Z_2 = Schedule makespan regarding preventive maintenance
Z_3 = Schedule makespan regarding preventive maintenance and failures at mean time to failure
Z_4 = Simulation procedure with objective function evaluation


FLOWSHOP ALGORITHM 2
Create initial solution x_0 using NEH algorithm.
Evaluate $Z_4(x)$
Using Z_1 as oracle find a x_1^* using simulated annealing process
Using Z_2 as oracle find a x_2^* using simulated annealing process
Using Z_3 as oracle find a x_3^* using simulated annealing process
Evaluate $Z_4(x_1^*)$
Evaluate $Z_4(x_2^*)$
Evaluate $Z_4(x_3^*)$
Set $x^* = argmin_{x \in \{x_1^*, x_2^*, x_3^*\}} Z_4(x)$
$[Z_4(x^*) \geq Z_4(x_0)]$
$[Z_4(x^*) < Z_4(x_0)]$
<< Is i equal to the maximum number of iterations? >>
[yes]
[no]
Set $x_0 = x^*$ $i = i + 1$
Declare the minimum x^* found as the optimal solution

Algorithm 6: FSP optimization algorithm with inner k-neighbour simulated annealing



Oracles:
Z1 = Schedule makespan
Z2 = Schedule makespan regarding preventive maintenance
Z3 = Schedule makespan regarding preventive maintenance and failures at mean time to failure
Z4 = Simulation procedure with objective function evaluation
FLOWSHOP ALGORITHM 3
Create initial solution x_0 using NEH algorithm.
Evaluate $Z_4(x)$
Using Z_1 as oracle find a x_1^* using simulated annealing process
Using Z_2 as oracle find a x_2^* using simulated annealing process
Using Z_3 as oracle find a x_3^* using simulated annealing process
Evaluate $Z_4(x_1^*)$
Evaluate $Z_4(x_2^*)$
Evaluate $Z_4(x_3^*)$
Set $x^* = argmin_{x \in \{x_1^*, x_2^*, x_3^*\}} Z_4(x)$
$[Z_4(x^*) \geq Z_4(x_0)]$
$[Z_4(x^*) < Z_4(x_0)]$
Set $x_0 = x^*$ $i = i + 1$
<< Is i equal to the maximum number of iterations? >>
[yes]
[no]
Compute $\Delta H = Z_4(x_0) - Z_4(x^*)$
Compute $P = \exp(-\frac{\Delta H}{T})$
Generate $R = [0,1)$ randomly
<< Is R < P? >>
[yes]
[no]
Update $T = \propto * T$
Declare the minimum x^* found as the optimal solution
Outer Temperature Update Factor ∝
Outer Initial Temperature T
Maximum Number of Iterations

Algorithm 7: FSP optimization algorithm with inner k-neighbour simulated annealing and outer simulated annealing

4.4 Job Shop Optimization Algorithms

In a Job Shop Scheduling Problem setting, all the aspects that characterizes the FSP are still present. The main difference is that each job has its own machine requirement. These machines are also prone to preventive maintenance and random machine breakdowns. The use of evolutionary metaheuristic, as the genetic algorithm described in Section 3.3.2, is described in the literature as a good way to explore and find good job sequencing for the JSP. By combining the genetic algorithm metaheuristic with the same ideas behind the use of Oracles from the flow shop optimization algorithms, two new algorithms to be applied to the JSP were created.

This Section starts by describing the genetic algorithm metaheuristic applied to the JSP in detail, then outlines two new algorithms that tries to find the best job sequencing for the job shop scheduling problem with added preventive maintenance and random breakdowns.

4.4.1 Genetic Algorithm Applied to the Job Shop Scheduling Problem

In the next subsections, the main pieces of the genetic algorithm applied to the job shop scheduling problem are explored by initially describing the chromosome representation and the initial population creation, the crossover process is discussed, the gene selection procedure is explained, the mutation procedure is addressed, and finally the entire genetic algorithm algorithm is reviewed.

The Chromosome Representation and Initial Population

As discussed in section 3.3.2, a chromosome is an array that represents a solution. The standard coding technique used to represent a solution in the early studies of the genetic algorithm used binary or permutation representations. Using the same representation techniques in modern optimization problems we will add infeasible solutions (i.e. individuals) to the population set. To avoid this behaviour and not waste computation resources to check, fix, or penalize a bad individual a job permutation with repetition representation that covers all feasible solutions for the job shop scheduling problem was proposed by Bierwirth [8] and is used in this thesis.

Let us consider J_i as the set of jobs to be processed, each of these jobs can be further broken down to operations that require a predetermined machine M_k. The number of operations for each job is the same i.e. the total number of operations per job are the same as the number of available machines m. The sequence in which each job has to visit the machines in the shop floor are given as input for the problem. For the 3 jobs and 4 machines case, an example of a chromosome (individual) created by permutation with repetition of jobs J_i can be written as $(J_1, J_2, J_3, J_3, J_2, J_1, J_1, J_1, J_2, J_3, J_3, J_2)$. Note that J_i appears in the array at most m times and their positions are randomly chosen. Given the following machine sequence matrix $SEQ_{ik} = (1, 2, 3, 4; 3, 1, 4, 2; 4, 2, 1, 3)$ and by reading the randomly created chromosome from left to right, an array that represents a sequence of operations (O_{ik}) can then be written as $(O_{11}, O_{23}, O_{34}, O_{32}, O_{21}, O_{12}, O_{13}, O_{14}, O_{24}, O_{31}, O_{33}, O_{22})$. This sequence of operations can be given to a schedule builder to evaluate any performance metric of interest. Oracles described in section 4.1 are used as schedule builders to evaluate the different performance metrics of the population set.

The pseudo-code of how to create an initial population of size *popSize* is described bellow:

```
Input:
    Number of jobs $i$;
    Number of machines $k$;
    Population size $popSize$;
Output: An array $pop$ containing $popSize$ individuals.
Create an empty array $pop$;
while The size of pop $<$ popSize do
    Create an $individual$ incremental integer array starting from 0 up to $i * k$ ;
    Randomly permute the elements of the $individual$ array;
    foreach $ind \in individual$ do
        $ind \leftarrow ind \mod i$
    end
    Append $individual$ to array $pop$;
end
```

Algorithm 8: Initial population creation pseudo code.

To illustrate Figure 4.1 shows an example of implementation of the population creation procedure for $i = 5$ and $k = 2$.

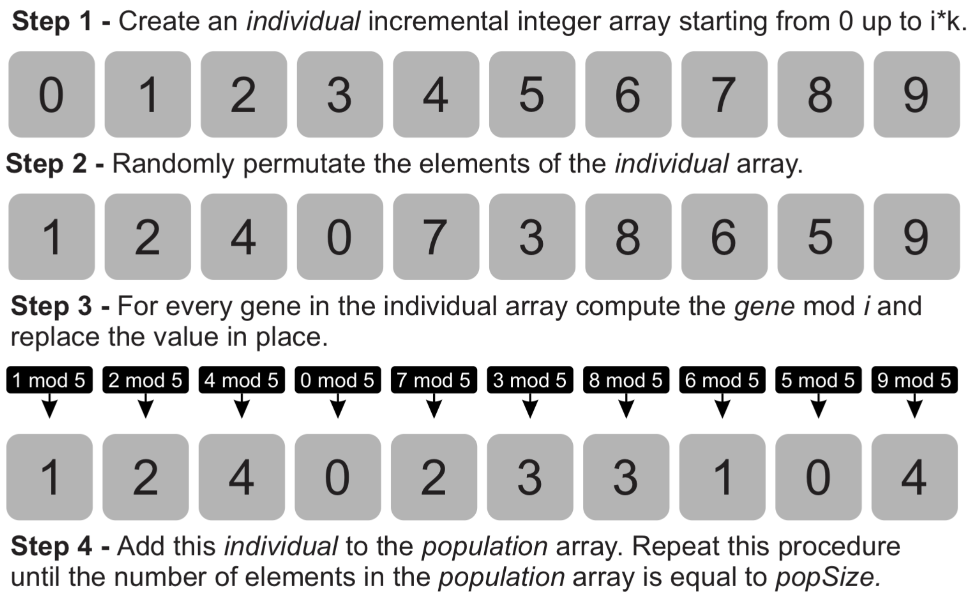

Step 1 - Create an individual incremental integer array starting from 0 up to i*k.
0 1 2 3 4 5 6 7 8 9
Step 2 - Randomly permutate the elements of the individual array.
1 2 4 0 7 3 8 6 5 9
Step 3 - For every gene in the individual array compute the gene mod i and replace the value in place.
1 mod 5
2 mod 5
4 mod 5
0 mod 5
7 mod 5
3 mod 5
8 mod 5
6 mod 5
5 mod 5
9 mod 5
1 2 4 0 2 3 3 1 0 4
Step 4 - Add this individual to the population array. Repeat this procedure until the number of elements in the population array is equal to popSize.

Figure 4.1: Population creation example.

The Crossover Process

The crossover procedure will create new chromosomes (children) with parts of two other chromosomes (parents) as described in section 3.3.2. A chromosome is a collection of genes, each position inside a chromosome is called a “gene” in this study. The Precedence Preservative Crossover (PPX) presented by Bierwirth et al. [9] was used and is described as follows: Consider n as the length of one of the parent chromosomes. To get the sequence in which the genes will be extracted from the parents, generate an array with size n randomly filled with elements of the set $\{1, 2\}$. Following the created gene sequence array, extract a gene from the selected parent, append it to a new child array, and then delete the same gene on the other parent array. Repeat until both parents arrays are empty and the child array has n elements. Figure 4.2 shows the step-by-step example of this procedure.

Chromosomes Selection

In this thesis, to determine which individuals will participate in future generation crossover process, the biased roulette wheel selection approach was used. In proportion to its fitness value, a chromosome is given a chance to be selected. “Fitness” in this context is the evaluation of a chromosome in a given metric of interest. Oracles

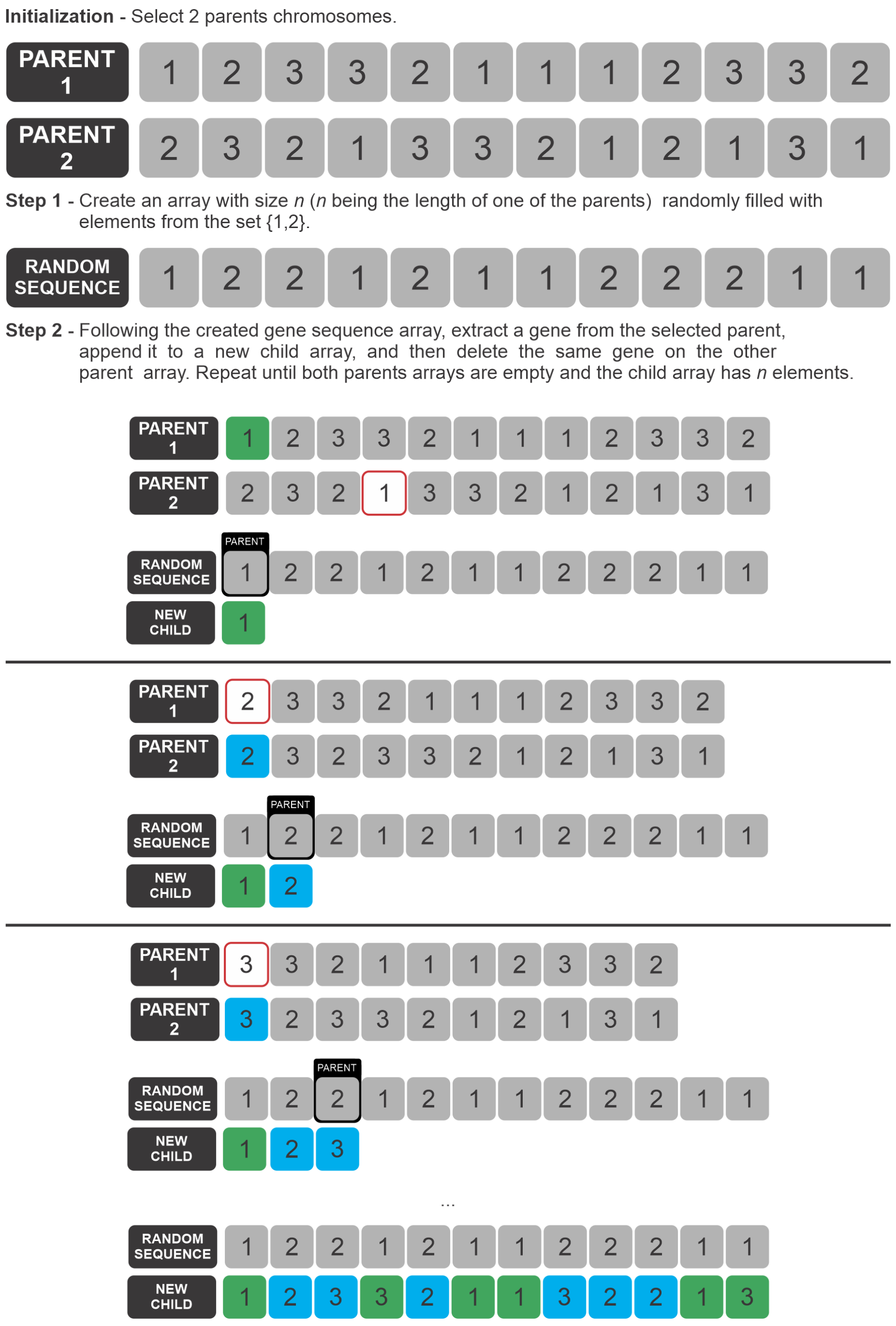


Figure 4.2: Example of a PPX crossover.

are used as schedule builders that provide a performance evaluation of a chromosome in the presented algorithm. The name “biased roulette wheel” comes from the analogy of a spinning roulette wheel where good individuals (the ones with better fitness value) have a bigger slot size than bad individuals. By randomly spinning the wheel, one by one, individuals are chosen to enter the new population array. Figure 4.3 illustrates the method.

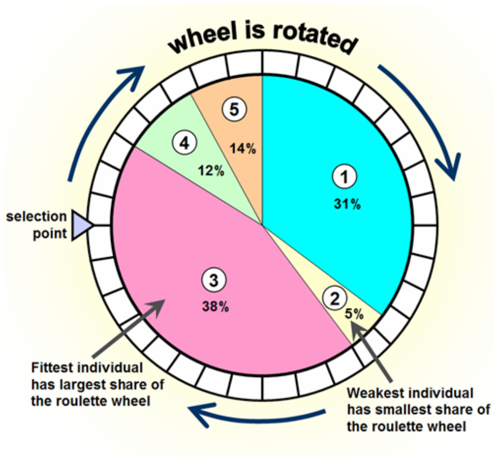

wheel is rotated
selection point
1
31%
2
5%
3
38%
4
12%
5
14%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

Figure 4.3: Roulette wheel selection . Source: Jun et al. [27]

It is clear that individuals that have better fitness values will also have a better chance of survival. The roulette wheel can be implemented by the following Algorithm 9 by Burke et al. [11].

Mutation

As it happens in real life, the chromosomes in a generated individual can suffer random mutations. Two parameters are used to control the mutation of the entire population: the population mutation ratio and the gene selection ratio. The first parameter stipulates the percentage of individuals (chromosomes) of the population that will be mutated and the second factor stipulates the percentage of genes in a single individual (chromosome) that will suffer mutation.

The mutation procedure starts by randomly selecting the individuals (chromosomes) from the population set that are going to be mutated based on the population mutation ratio. For every selected individual (chromosome), tag the genes that are going to suffer mutation following the gene selection ratio. The tagged genes shift to the right and the mutated individuals (chromosomes) substitute those selected initially. Figure 4.4 illustrates the right shift procedure.

Note that this procedure do not make the mutated individuals infeasible, so no computation efforts are wasted to check, validate, and fix a mutated individual.

```
Input:
    Population $pop$;
    Population size $n$ ;
    Maximum population size $popSize$;
Output: A new population $pop$
Step 1. Evaluate the fitness of each individual $f_i$;
Step 2. Compute the probability (slot size), $p_i$, of selecting each one of the
  individuals 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 a random number, $r \in (0, 1]$ ;
Step 5.
if $r < q_1$ then
    select the first chromosome
else
    select chromosome $x_i$ such that $q_{i-1} < r < q_i$
end
Step 6. Repeat steps 4 and 5 until $popSize$ chromosomes were selected;
```

Algorithm 9: Roulette wheel procedure. Source: Burke et al. (Adapted) [11]

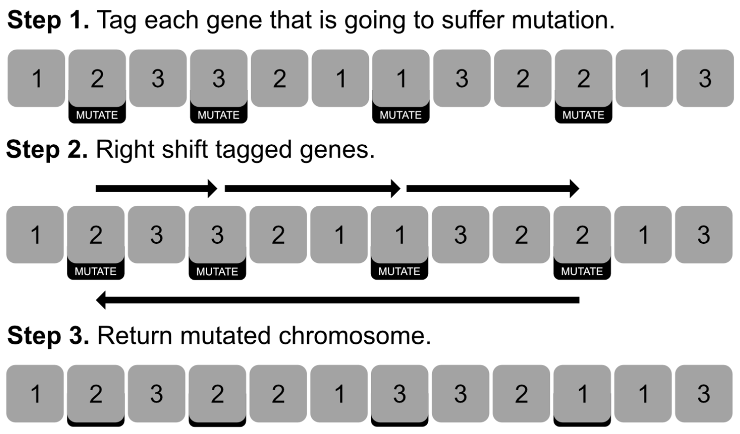

Step 1. Tag each gene that is going to suffer mutation.
1 2 3 3 2 1 1 3 2 2 1 3
MUTATE
Step 2. Right shift tagged genes.
1 2 3 3 2 1 1 3 2 2 1 3
Step 3. Return mutated chromosome.
1 2 3 2 2 1 3 3 2 1 1 3

Figure 4.4: Mutation Right Shift

The Genetic Algorithm

The genetic algorithm is used to explore the solution space looking for a good solution based on the performance evaluation given by one of the oracles described in section

4.1. The chosen stopping criteria is the maximum number of generations, as soon as the algorithm performs a determined amount of generations, it stops and the best individual is reported as the approximation of the optimal solution for that given oracle. The approach described here uses an “elite” array to carry forward to new populations members that were declared to produce good solutions for the job shop scheduling problem with added preventive maintenance and random breakdowns. These chromosomes are added at the beginning of the population creation step, and it is updated at every loop of the main algorithm. Algorithm 10 represents the genetic algorithm applied to job shop problem.

4.4.2 The optimization algorithms

Algorithms 11 and 12, similar to algorithms 4, 6 and 7, try to identify a job permutation with repetition individual that can be used to construct a robust schedule. The algorithm performs a two-loop procedure, the inner portion explores many different solutions using randomly generated individuals encapsulated in a genetic algorithm metaheuristic, and the outer loop focus on the overall objective function. An array of “elite” individuals that produce good solutions are carried forward, from one iteration of the algorithm to the next, as part of the inner population creation for the genetic algorithms procedures. The elite array has a fixed specified size and it follows a FIFO rule, meaning that as soon as the maximum number of individuals is reached, the first individual that enters the array will leave to make room for a newly found elite individual. This elite array will induce the creation of good individuals since its members’ genes will produce new individuals that inherit its good characteristics.

Algorithm 11 executes three parallel genetic algorithm metaheuristics with random population creation in the inner scope, and the outer scope will call the inner procedure again if a better solution is found aiming to improve the quality of the result. The inner loop can be described as follows: Three parallel genetic algorithms with independent random population creation procedures are started. If there are any individuals in the elite array, these individuals are added to the initial population of every genetic algorithm instance. Each one of the genetic algorithms tries to optimize the results using one of the Oracles $Z_1(x), Z_2(x)$ and $Z_3(x)$ as its performance metric. When the genetic algorithm procedures finishes, the individuals that

```
Input:
    Oracle $or$;
    Elite array $elite$;
    Maximum generation number $maxGen$;
    Maximum population size $popSize$;
    Population mutation ratio $popMutationRatio$;
    Gene selection ratio $geneSelectionRatio$;
Output: A job sequence with repetition ($x$) that represents the
        approximation of the Oracle's optimal solution.
Initialization. Randomly generate a population set ($oldPop$) with $popSize$
  individuals;
Randomly select and substitute individuals from $oldPop$ with individuals in
  $elite$;
Create an empty $bestScoreProgress$ array;
while $gen < maxGen$ do
    Create a $newPop$ empty array;
    while $newPop$ number of elements is smaller than $popSize$ do
        Append to $newPop$ individuals generated by the PPX crossover
          procedure by randomly selecting parents in $oldPop$.;
    end
    Mutate the entire $newPop$ array following the $popMutationRatio$ and
      $geneSelectionRatio$;
    Stack the $oldPop$ and the newly created $newPop$;
    Using the biased roulette wheel procedure, select up to $popSize$
      individuals and set it as $oldPop$;
    Append to the $bestScoreProgress$ array the minimum objective value
      found in $oldPop$ and it's corresponding chromosome;
end
Return $x$ as the chromosome with the least objective function value from the
  $bestScoreProgress$ array;
```
Algorithm 10: The Genetic Algorithm Applied to the Job Shop Problem.

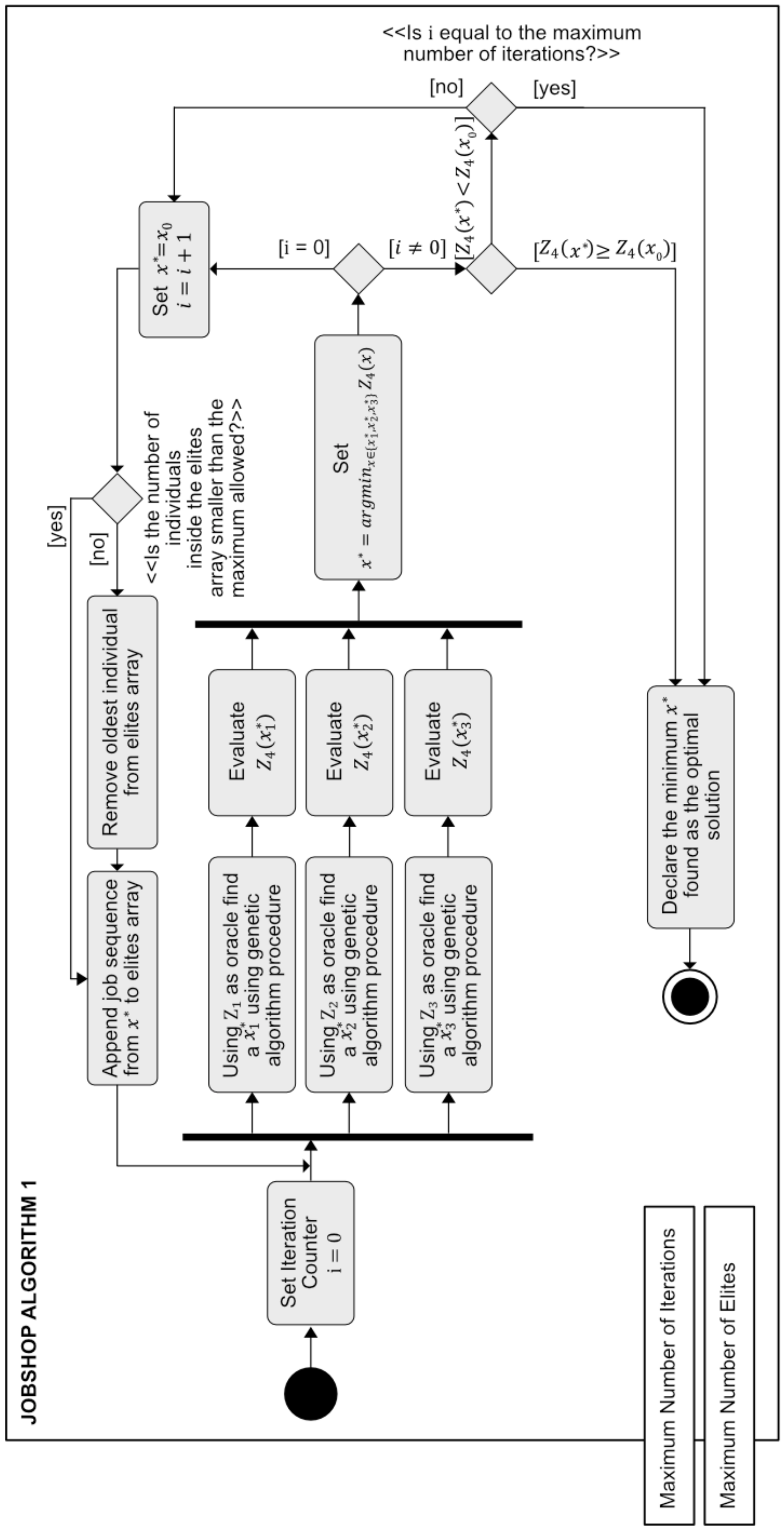
represent the optimal solution for every Oracle are then declared as x_1^*, x_2^* and x_3^* for $Z_1(x), Z_2(x)$ and $Z_3(x)$ respectively. Then $Z_4(x_1^*), Z_4(x_2^*)$ and $Z_4(x_3^*)$ are calculated and the optimal solution x^* is declared being the minimum objective function value found between them. The outer loop can be described as follows: The first x^* from the inner algorithm is declared as initial solution x_0, the individual is added to an elite array and the inner portion of the algorithm is called. From the second loop forward, if the value of $Z_4(x^*)$ is smaller than $Z_4(x_0)$ the algorithm tries to improve the solution again by appending the individual to the elite array, x_0 is set to x^* and the inner portion of the algorithm is called. This procedure repeats until $Z_4(x_0)$ is better than $Z_4(x^*)$, i.e., no better solution is found. The best job sequence found throughout the algorithm's execution is declared as the approximation of the optimal solution.

Algorithm 12 executes three parallel genetic algorithms metaheuristics with random population creation in the inner scope and the outer scope is a simulated annealing procedure where bad solutions are accepted to escape local optima and search for a better result. The inner loop of this algorithm is done exactly as described in algorithm 11. The outer loop can be described as follows: The first x^* from the inner algorithm is declared as initial solution x_0, the individual is added to an elite array, the temperature T is set following the initial temperature and the inner portion of the algorithm is called. From the second loop forward, if the value of $Z_4(x^*)$ is smaller than $Z_4(x_0^*)$ the algorithm accepts this solution and tries to improve the solution again by appending the individual to the elite array, x_0 is set to x^* and the inner portion of the algorithm is called. If $Z_4(x_0)$ is smaller or equal to $Z_4(x^*)$ a uniform random number between $R = (0, 1]$ is created and a probability of acceptance is calculated as $Pr_{acceptance} = e^{\frac{-(x_0^* - x^*)}{T}}$. If $R < Pr_{acceptance}$ the temperature T is updated following the temperature update factor α, x_0 is set to x^* and the inner portion of the algorithm is called. If $R \geq Pr_{acceptance}$ the algorithm stops, and the best job sequence found throughout the algorithm's execution is declared as the approximation of the optimal solution.

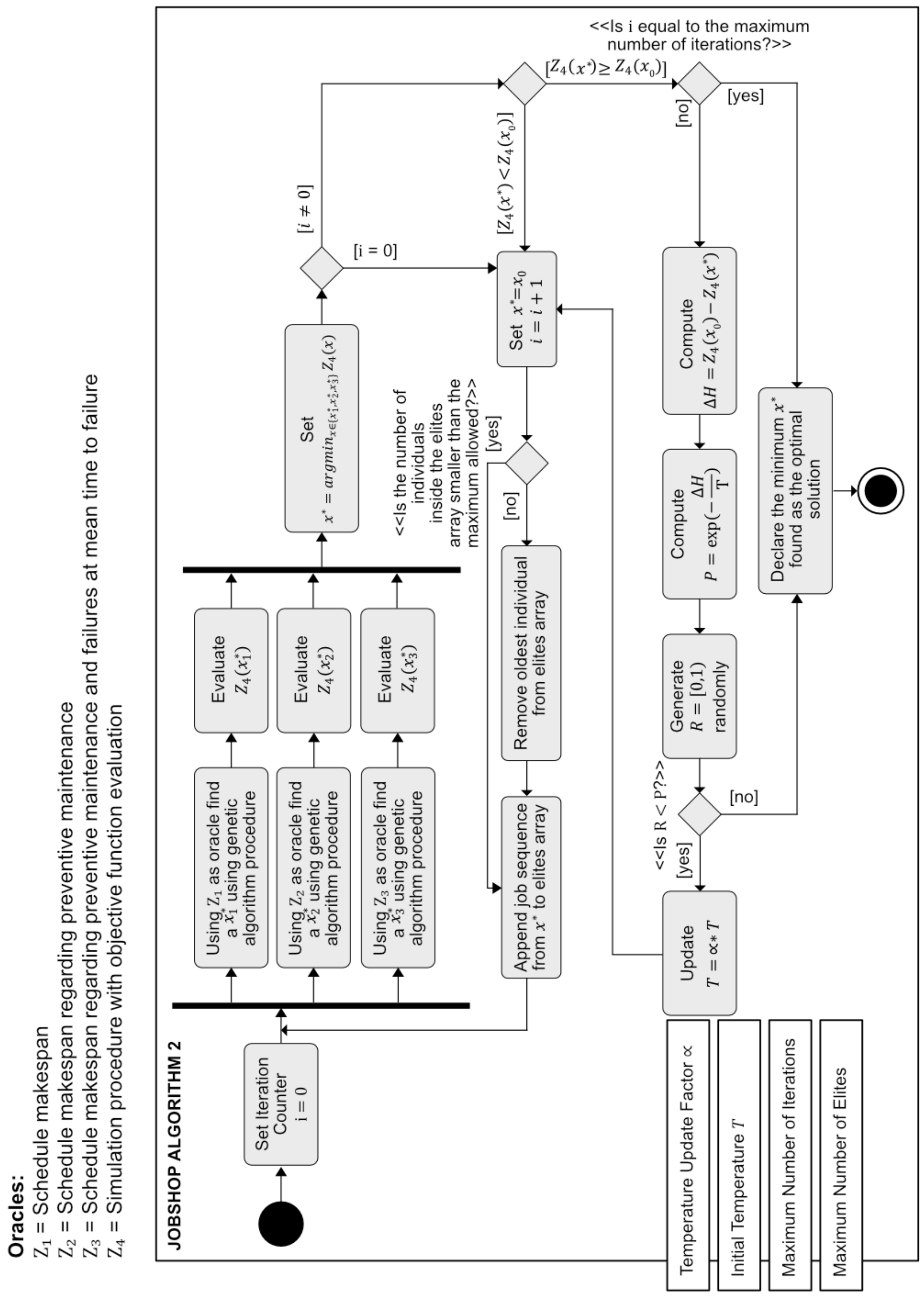
Graphical representation

Similar to what is shown on subsection 4.3.1, the JSP optimization algorithms are graphically presented using the UML activity diagram of Algorithm 11 and Algorithm 12.

Oracles:
Z_1 = Schedule makespan
Z_2 = Schedule makespan regarding preventive maintenance
Z_3 = Schedule makespan regarding preventive maintenance and failures at mean time to failure
Z_4 = Simulation procedure with objective function evaluation


JOBSHOP ALGORITHM 1
Set Iteration Counter i = 0
Append job sequence from x^* to elites array
Remove oldest individual from elites array
Using Z_1 as oracle find a x_1^* using genetic algorithm procedure
Using Z_2 as oracle find a x_2^* using genetic algorithm procedure
Using Z_3 as oracle find a x_3^* using genetic algorithm procedure
Evaluate $Z_4(x_1^*)$
Evaluate $Z_4(x_2^*)$
Evaluate $Z_4(x_3^*)$
Set $x^* = argmin_{x \in \{x_1^*, x_2^*, x_3^*\}} Z_4(x)$
<<Is the number of individuals inside the elites array smaller than the maximum allowed?>>
[yes]
[no]
Set $x^* = x_0$ $i = i + 1$
[i = 0]
[$i \neq 0$]
[$Z_4(x^*) < Z_4(x_0)$]
[$Z_4(x^*) \geq Z_4(x_0)$]
<<Is i equal to the maximum number of iterations?>>
[no]
[yes]
Declare the minimum x^* found as the optimal solution
Maximum Number of Iterations
Maximum Number of Elites

Algorithm 11: JSP optimization algorithm with inner genetic algorithm


Oracles:
Z_1 = Schedule makespan
Z_2 = Schedule makespan regarding preventive maintenance
Z_3 = Schedule makespan regarding preventive maintenance and failures at mean time to failure
Z_4 = Simulation procedure with objective function evaluation
JOBSHOP ALGORITHM 2
Set Iteration Counter i = 0
Using Z_1 as oracle find a x_1^* using genetic algorithm procedure
Using Z_2 as oracle find a x_2^* using genetic algorithm procedure
Using Z_3 as oracle find a x_3^* using genetic algorithm procedure
Evaluate $Z_4(x_1^*)$
Evaluate $Z_4(x_2^*)$
Evaluate $Z_4(x_3^*)$
Set $x^* = argmin_{x\in\{x_1^*,x_2^*,x_3^*\}} Z_4(x)$
[i = 0]
[$i \neq 0$]
[$Z_4(x^*) < Z_4(x_0)$]
[$Z_4(x^*) \geq Z_4(x_0)$]
<<Is i equal to the maximum number of iterations?>>
[no]
[yes]
Set $x^* = x_0$ $i = i + 1$
<<Is the number of individuals inside the elites array smaller than the maximum allowed?>>
[yes]
[no]
Remove oldest individual from elites array
Append job sequence from x^* to elites array
Compute $\Delta H = Z_4(x_0) - Z_4(x^*)$
Compute $P = \exp(-\frac{\Delta H}{T})$
Generate $R = [0,1)$ randomly
<<Is $R < P$?>>
[yes]
[no]
Update $T = \propto * T$
Declare the minimum x^* found as the optimal solution
Temperature Update Factor $\propto$
Initial Temperature T
Maximum Number of Iterations
Maximum Number of Elites

Algorithm 12: JSP optimization algorithm with inner genetic algorithm and outer simulated annealing

Chapter 5

Results and Discussion

This Chapter summarizes all the experiments and main discussions regarding the application of the proposed algorithms. There is a Section for every class of problem with their initial parameter estimation, followed by the final experiments and achieved result, finally a conclusion and future research are outlined.

5.1 FSP Experiments

To understand the behaviour of the proposed algorithms when varying individual parameters, the first part of this Section presents the initial parameter estimation for the FSP optimization algorithms. Then the final experimental procedures are outlined, and the results are discussed.

5.1.1 Initial Parameter Setting

The proposed test procedures were coded using Python 3.8 with Numpy 1.18, Numba 0.50.1, and Cython 0.29.21 frameworks and tested on a set of randomly generated instances for evaluating the performance of the proposed algorithms. A computer with a Ryzen 7 3800X with 3.9GHz clock speed processor and 32GB of 3600MHz RAM memory was used to run the instances of these tests.

Even with all the effort to compute good parameters for the algorithms presented here, they must be used as good starting points. The parameters found here are not optimal for all problem sizes or instances. There are many iterations and relationships that are not explored in this text and are out of the scope of this work.

Test 1 - Temperature Update Factor

When running a simulated annealing procedure, two factors are of extreme importance to the expected average rate of improvement and total computational time: the initial temperature T_0 and temperature update factor α. The following tests only focus on the temperature update factor. In Algorithms 4 and 6 there is only one simulated annealing step in the outer loop and the inner loop of the algorithm, respectively. Algorithm 4 generates $\frac{n(n-1)}{2}$ neighbours every time the inner loop is called. That being the case, the exploration of different neighbours are higher and the expected average improvement that is achieved at each iteration is also higher. On the other hand, Algorithm 6 tries to search for good neighbours in a smarter way, trying to achieve similar results in a faster manner. Algorithm 7 has simulated annealing in its inner and outer loops and is affected by both update factors at the same time. It was decided to extrapolate the decisions regarding the update factor from Algorithms 4 and 6 to Algorithm 7.

The following tests were designed to evaluate the impact of the temperature update α on the time to solve and the average rate of improvement of every scenario. The number of machines and the number of jobs were set to $m = \{15\}$ and $n = \{20, 50, 70\}$ respectively. A total of 5 instances were created for each machine/job combination and the average values for all scenarios are reported. The global parameters for each instance were set as follows: the weight of solution robustness $\rho = 1$; the processing times of each job are integers uniformly generated between $[9, 29]$; t_{pj} is uniformly generated between $[12, 15]$; t_{rj} was uniformly generated between $[8, 12]$; θ_k was uniformly generated between $[60, 100]$; β_j was uniformly generated between $[2, 3]$. The simulated annealing procedure had its parameters set as follows: for Algorithm 4 the initial temperature was set to $T_0 = 0.8Z_4(x^0)$ and for Algorithm 6 the initial temperature was set to $T_0 = 1Z_4(x^0)$ and the number of neighbours to be created nbk was set to 10% of $\frac{n(n-1)}{2}$. α_1 represents the update factor for algorithm 4 and α_2 represents the update factor for algorithm 6.

Figure 5.1 shows a summary of the results for Algorithm 4, looking at the average execution time from the chart on the left, specially for the 70 jobs problem, the time to solve tends to increase exponentially as α_1 goes higher. The average improvement is marginal and the bigger the instance, the more difficult it is to significantly improve the solution.

Figure 5.2 shows the summary of the results for Algorithm 6. The average time to run tends to go higher as α_1 increases. The average improvement also tends to go

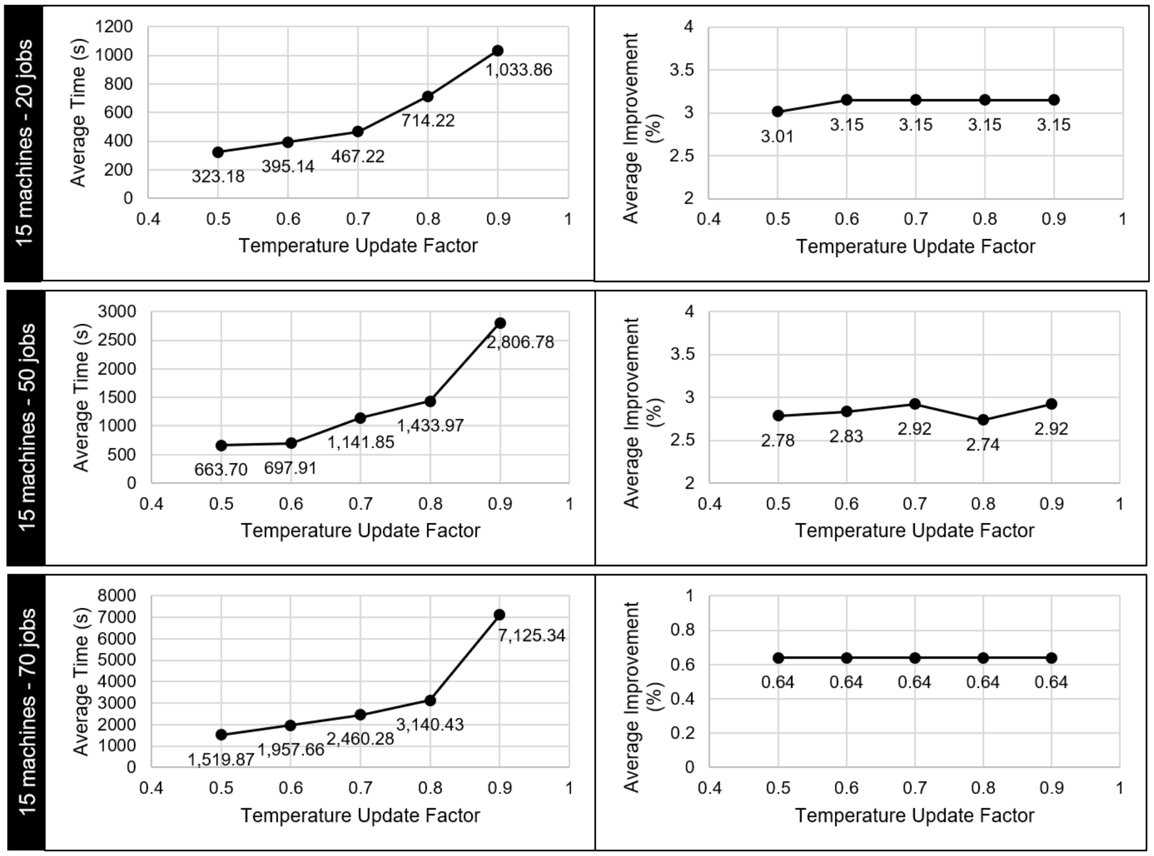


Figure 5.1: Sensitivity Analysis Temperature Update Factor Algorithm 4.

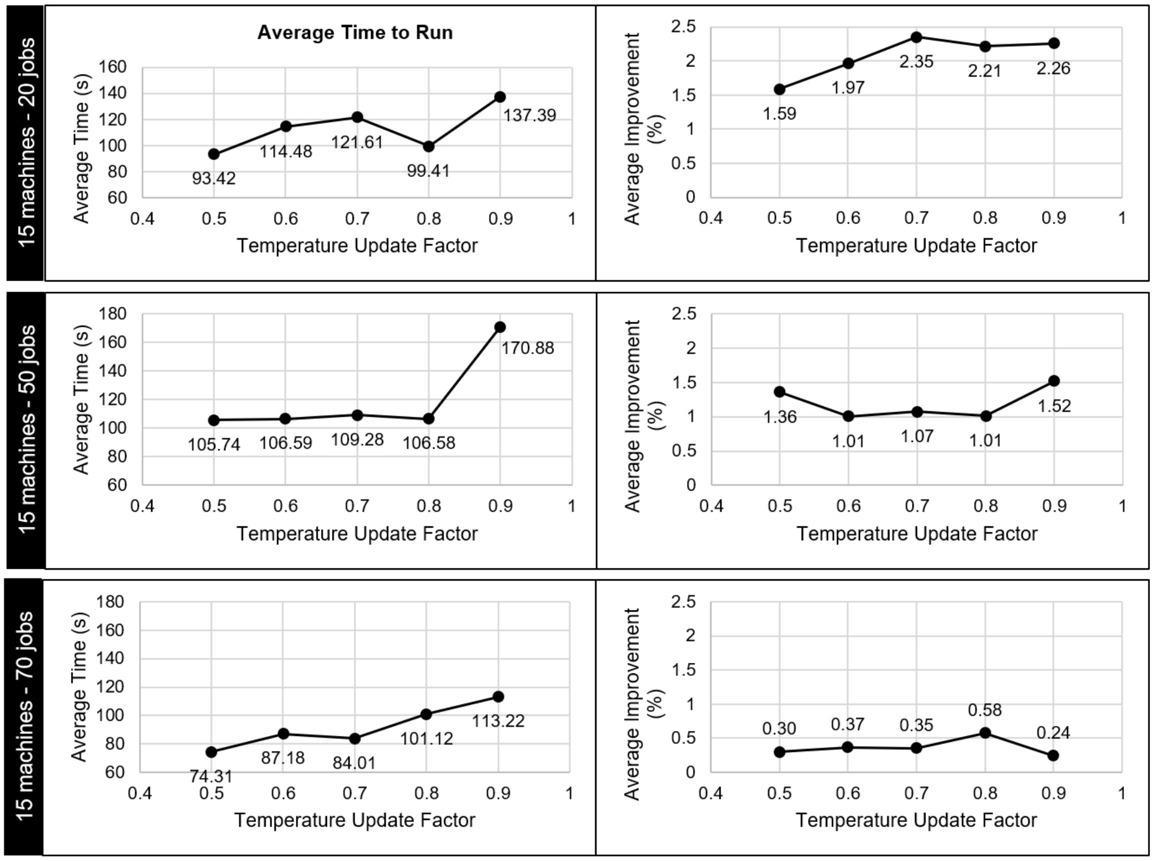


Figure 5.2: Parameter Estimation, Temperature Update Factor, Algorithm 6.

marginally higher as α_1 increases. As the size of the instance increases, the harder it is to find a significant improvement, similar to what happens in Algorithm 4.

5.1.2 Optimality Gap Analysis

To identify how close the algorithms approximate to the optimal solution, an enumeration toy experiment was performed. A small problem with $n = 5$ jobs and $m = 3$ machines was randomly created.

	Machine 1	Machine 2	Machine 3
Job 1	13	26	15
Job 2	23	20	25
Job 3	17	15	23
Job 4	10	10	9
Job 5	14	20	21

Table 5.1: Processing times (P_{ij}) example.

	Machine 1	Machine 2	Machine 3
θ_j	87	95	79
β_j	2	3	2
t_{pj}	12	12	15
t_{rj}	9	12	12

Table 5.2: θ_j, β_j, t_{pj}, and t_{rj} parameters example.

The total number of possible permutations of the job sequencing array is $n!$. In this example, the total number of possible permutations is $5! = 120$. By applying Oracle Z_4 to every possible job sequence, the optimal solution for the problem can then be found. The job sequence $[0, 2, 4, 1, 3]$ has the minimal objective function with a value of 114.3995. The same problem is then given to each one of the algorithms for the flow shop problem and the objective function (OF) values found for each algorithm are listed in Table 5.3.

The simulated annealing procedure had its parameters set as follows: for Algorithm 4 the initial temperature was set to $T_0 = 0.8Z_4(x^0)$; for Algorithm 6 the initial temperature was set to $T_0 = 1Z_4(x^0)$; the number of neighbours to be created *nbk* was set to 12 in every loop. The temperature update factor α_1 was set to 0.60 and α_2 was set to 0.70.

Algorithm	Job Sequence	OF Value	Optimality Gap	Execution Time
Algorithm 4	[4, 2, 3, 1, 0]	115.553	1.01%	422s
Algorithm 6	[2, 3, 4, 1, 0]	119.001	4.02%	43s
Algorithm 7	[2, 3, 4, 1, 0]	119.001	4.02%	657s

Table 5.3: Optimality gap example.

We can observe that Algorithm 4 has a closer gap to the optimal solution when compared to Algorithms 6 and 7. The job sequence [2, 3, 4, 1, 0] was found initially by the NEH algorithm and neither Algorithms 6 or 7, by executing the simulated annealing with a k-pairwise exchange neighbourhood construction strategy, was able to improve it. Algorithm 6 only did a single loop taking only 43 seconds to execute until completion, on the other hand Algorithm 7 reached the declared maximum number of trials with no improvement stopping criteria with 20 loops executed taking 657s.

5.1.3 Final Experiments

The instances for this section were randomly created; the number of machines and the number of jobs selected were $m = \{15, 30, 45\}$ and $n = \{25, 50, 75\}$ respectively. Since this thesis was inspired by a medium size company, this combination of machines/jobs is accurate to what one can experience in a real life scenario. A total of 5 runs were executed for each machine/job combination created and the average values for all scenarios are reported. The global parameters for each instance were set as follows: the weight of solution robustness $\rho = \{0.5\}$; the processing times of each job are integers uniformly generated between $[9, 29]$; the preventive maintenance time for every machine t_{pj} was uniformly generated between $[12, 15]$; the corrective maintenance time for every machine t_{rj} was uniformly generated between $[8, 12]$; the scale parameter θ_k was uniformly generated between $[60, 100]$; the shape parameter β_j was uniformly generated between $[2, 3]$. The simulated annealing procedure had its parameters set as follows: for Algorithm 4 the initial temperature was set to $T_0 = 0.8Z_4(x^0)$; for Algorithm 6 the initial temperature was set to $T_0 = 1Z_4(x^0)$; the number of neighbours to be created nbk was set to 10% of $\frac{n(n-1)}{2}$ for every different n. The total number of scenarios created during the simulation procedure was set to 1000. The temperature update factor had to be set differently for each size of the problem, for the 25 jobs problem, α_1 was set to 0.75, for the 50 jobs problem α_1 was set to 0.80, and finally for 75 jobs problems α_1 was set to 0.85. α_2 was set to 0.75 for every problem size. The α_1 and α_2 were extrapolated to Algorithm 7.

Figure 5.3 and Figure 5.4 show the summary of the results obtained after the execution of Algorithm 4. The improvement obtained from the initial solution is not impressive. This algorithm performs a complete neighbourhood search at the start of execution and finding a better solution through pairwise exchange is deemed to be challenging. The impact on the average time per loop is significant when moving from a smaller problem size to a bigger one, for instance, there is an increase of 946.58% on the execution time going from 25 jobs 15 machines to 75 jobs 45 machines.

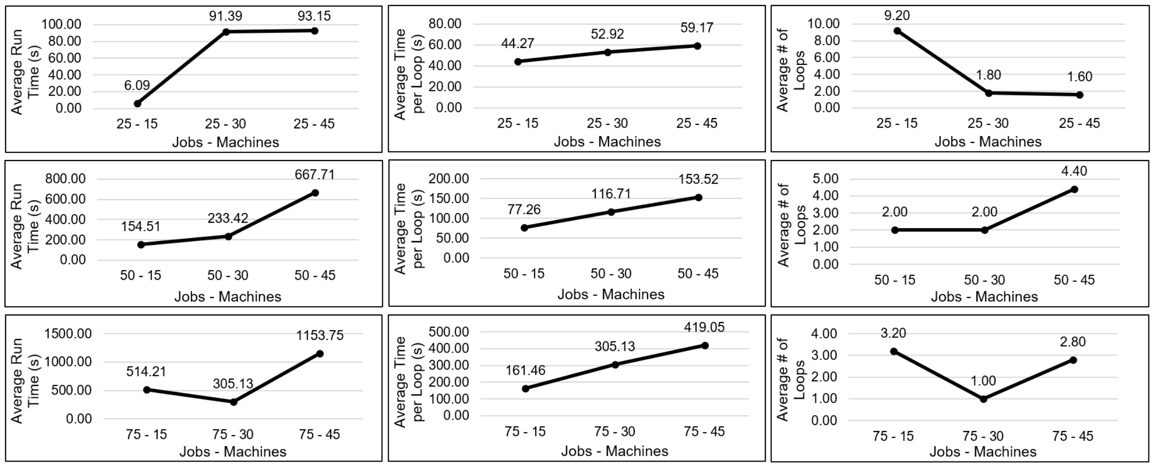


Figure 5.3: Execution time - FSP Algorithm 4

Figure 5.5 and Figure 5.6 show the summary for Algorithm 6. By default, Algorithm 6 stops if no improvements were found on the previous loop. Looking at the average number of loops the Algorithm stops prematurely only executing one iteration of the loop with none to minimal improvement over the initial solution.

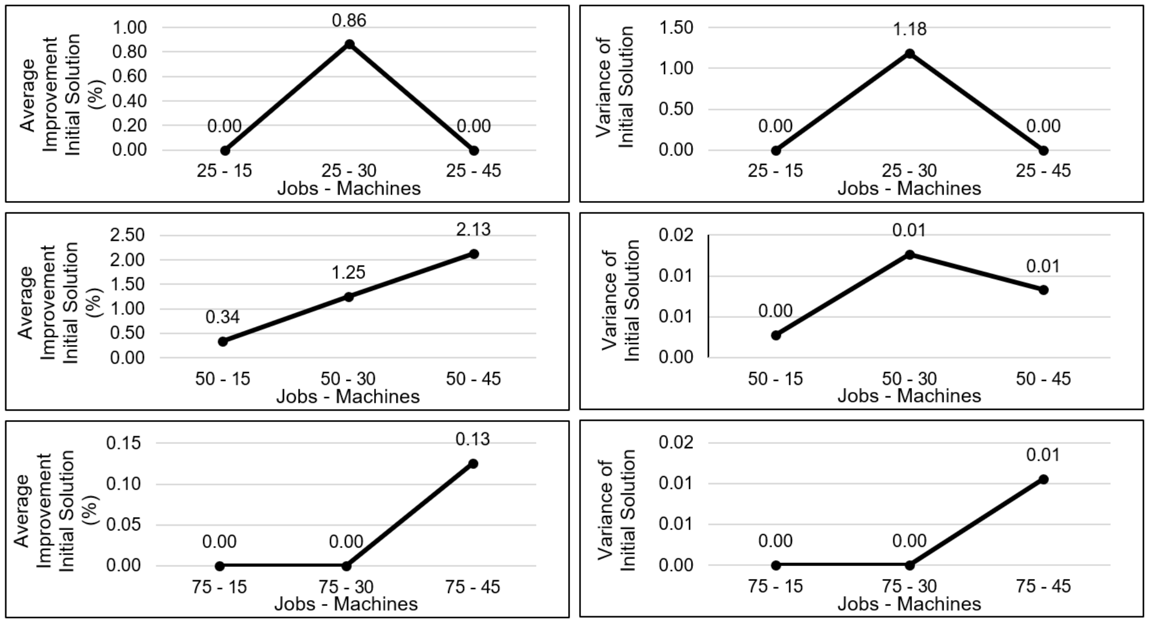


Figure 5.4: Improvement - FSP Algorithm 4

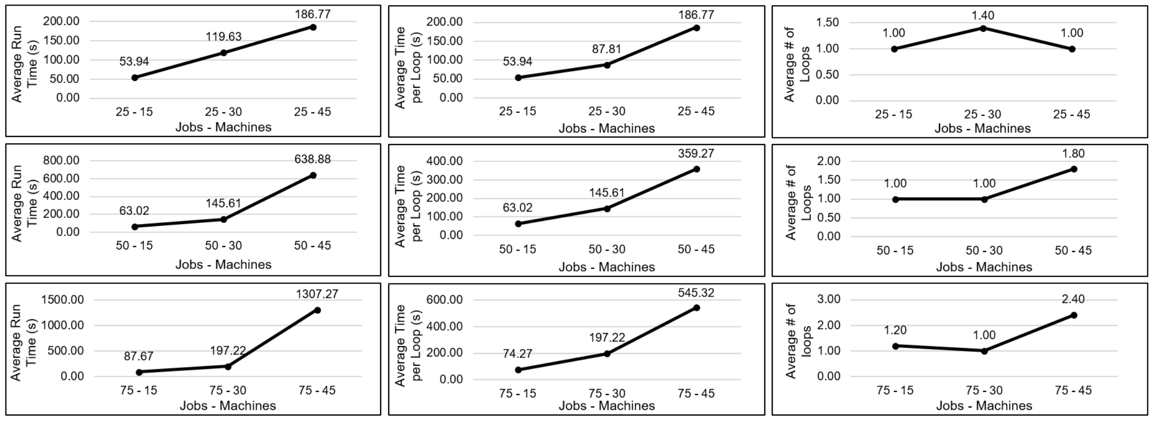


Figure 5.5: Execution time - FSP Algorithm 6

Figure 5.7 and Figure 5.8 represent the summary for Algorithm 7. The improvement over the initial solution observed on average is higher when compared to Algorithm 6 since the Algorithm do not stop early and search more the solution space. The average run time is higher in all cases when compared to Algorithm 6, for instance, the 30 machines 75 jobs problem had an execution time increase of 1859% taking on average 1.02 hours to complete.

Another important factor to point out is that the difficulty to find a better solution increases as the size of the problem increases. The NEH algorithm used to generate

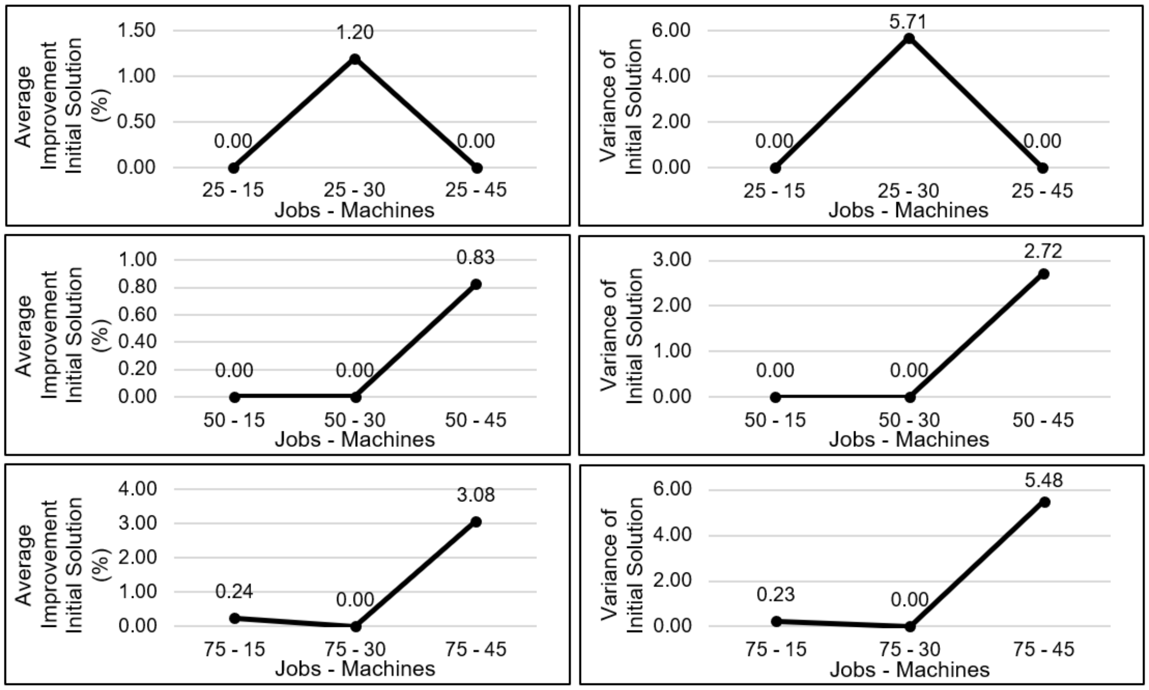


Figure 5.6: Improvement - FSP Algorithm 6

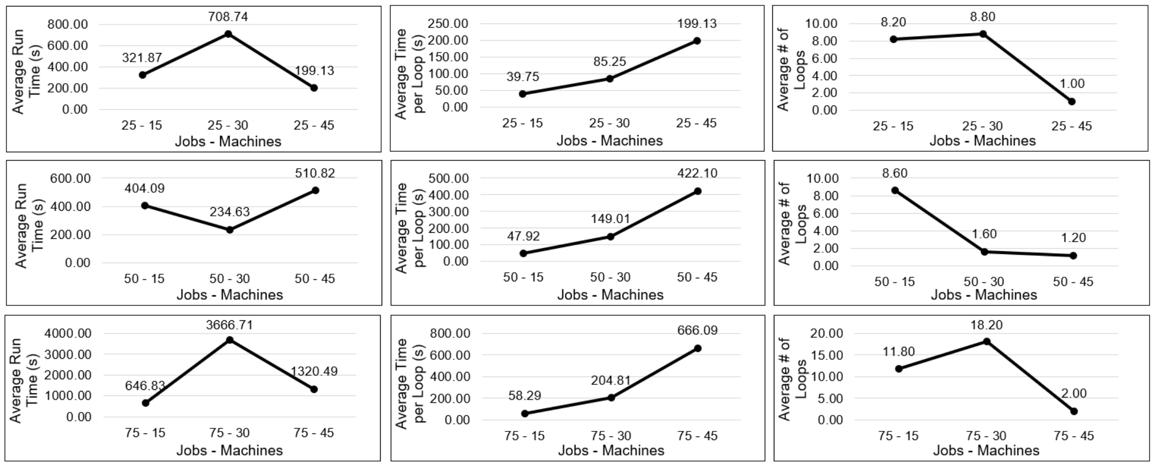


Figure 5.7: Execution time - FSP Algorithm 7

the initial feasible solution is known to be one of the best algorithms in finding a great initial solution for the FSP. In the tests presented here, only marginally better solutions were found when compared to the initial solution.

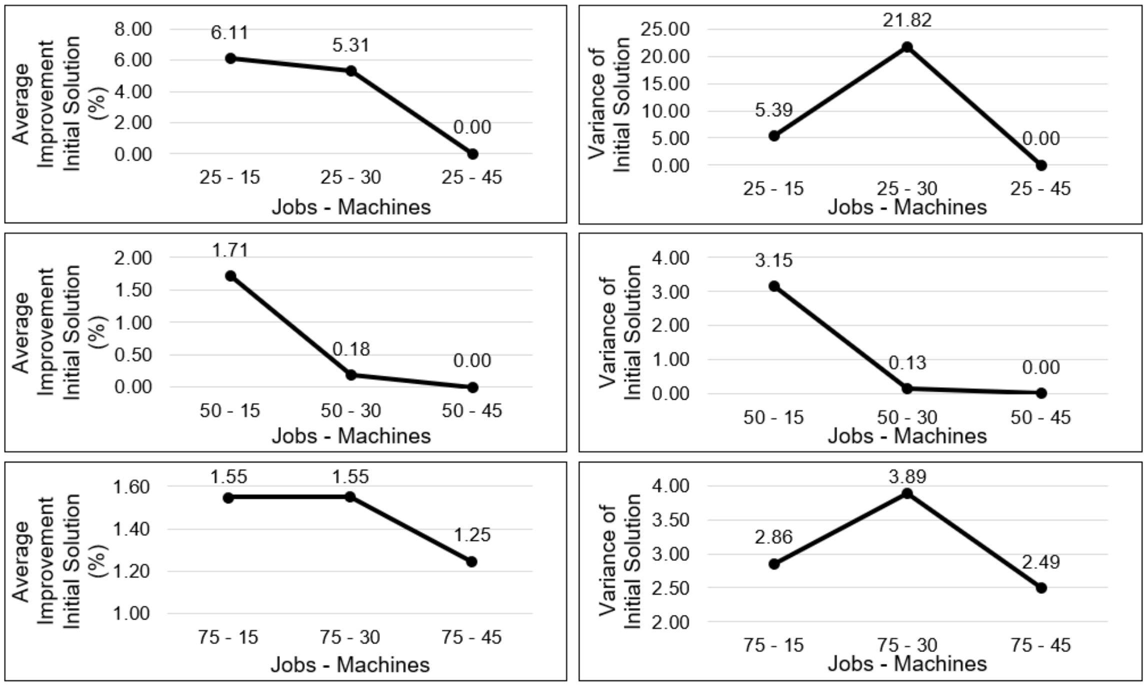


Figure 5.8: Improvement - FSP Algorithm 7

5.2 JSP Experiments

To understand the behaviour of the proposed algorithms when varying individual parameters, the first part of this Section presents the initial parameter estimation for the JSP optimization algorithms. Then the final experimental procedures are outlined, and the results are discussed.

5.2.1 Initial Parameter Setting

This section explores the behaviour of the genetic algorithm by varying its parameters one by one. The parameters estimated here are not optimal for all problem sizes or instances. There are many different relationships and iterations that are not explored here and are out of scope of this work. Problem instances were created, expanding the work done by Taillard [44] by adding the parameters needed to create preventive maintenance and random breakdowns. The instances created by Taillard provide the processing times of each operation in a machine and the machine sequence requirement for each job. The chosen instance for this test is 'ta01' with 15 jobs and

15 machines. The time to perform a preventive maintenance and corrective maintenance in a machine was calculated by taking the average of the processing times in a machine and multiplying it by a factor ranging from 0 to 1. The multiplication factor chosen for this section was 0.8 and 0.4 for the preventive maintenance and corrective maintenance times, respectively. Similarly, the scale parameter of the Weibull distribution which dictates the frequency that a preventive maintenance will happen, was derived by multiplying the average of all processing times in a machine by an integer. The chosen integer used in this section is 6 on average there will be the execution of 6 operations in a given machine between each preventive maintenance. The shape parameter was set to 2.5, the number of elite chromosomes was set to 4, ρ was set to 0.5 and a total of 25 runs was carried in each one of the tests.

The proposed test procedures were coded using Python 3.8 with Numpy 1.18, Numba 0.50.1, and Cython 0.29.21 frameworks. A computer with a Ryzen 7 3800X with 3.9GHz clock speed processor and 32GB of 3600MHz RAM memory was used to run the instances of these tests.

Test 1 - Gene Mutation Selection Rate

The gene mutation selection rate changes a specific chromosome as discussed in sub section 4.4.1 by randomly choosing some of the genes to be right shifted. This test was designed to investigate the impact of varying the gene mutation selection rate from 0.05 to 0.45 by increments of 0.05 on the overall performance of the job shop genetic algorithm procedure. The maximum number of generations was fixed in 200, the population size was fixed to 300 and the population mutation rate was fixed to 0.10. After 25 runs, the average of the results from Oracles Z_1, Z_2 and Z_3 are shown in Figures 5.9, 5.10, and 5.11 respectively.

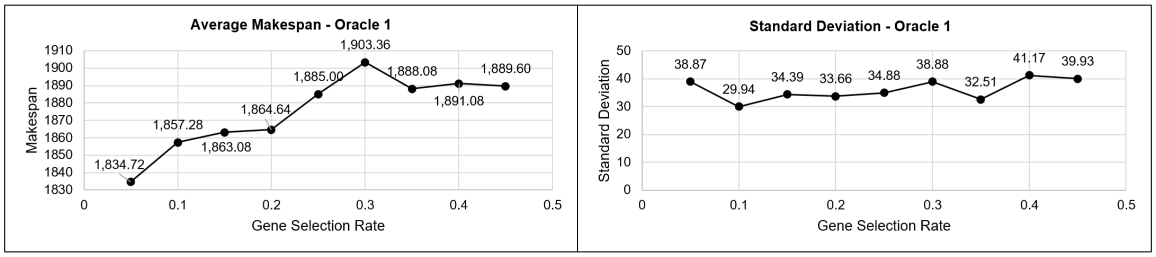


Figure 5.9: Test 1 - Gene Mutation Selection Rate - Oracle Z_1

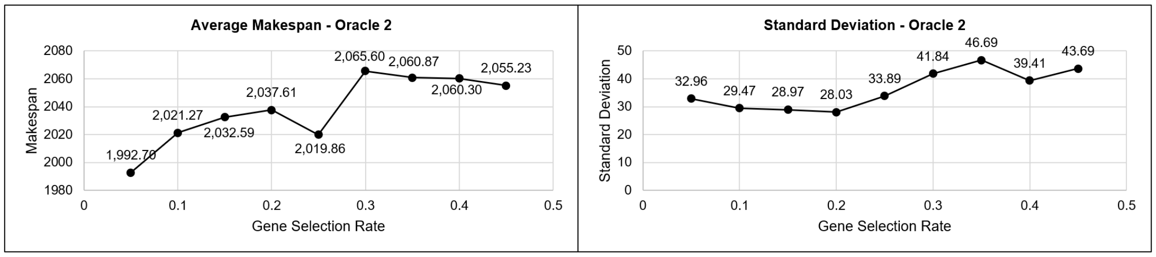


Figure 5.10: Test 1 - Gene Mutation Selection Rate - Oracle Z_2

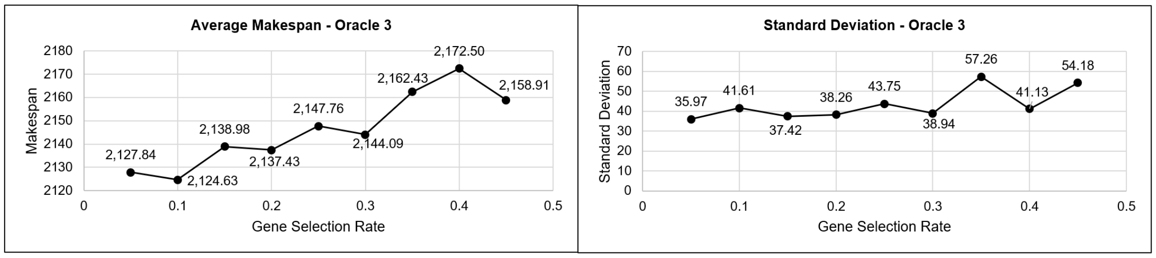


Figure 5.11: Test 1 - Gene Mutation Selection Rate - Oracle Z_3

The average of the schedule's makespan goes higher as the gene mutation selection rate increases. To avoid the degradation of the solution, a value around 0.05 is recommended. The impact on the run time was minimal, thus not evidenced here.

Test 2 - Population Mutation Rate

The population mutation rate determines how many individuals in a given population are randomly mutated. Test 2 was designed to evaluate the behaviour of the genetic algorithm procedure when the population mutation rate changes from 0.05 up to 0.25 by increments of 0.05. The maximum number of generations was fixed in 200, the population size was fixed to 300 and the gene mutation selection rate was fixed to 0.05. After 25 runs, the average of the results from Oracles Z_1, Z_2 and Z_3 are shown in Figures 5.12, 5.13, and 5.14 respectively.

The average makespan performance showed no meaningful improvement by setting the population mutation rate between 0.1 and 0.2. Oracle Z_2 behaved differently than the others when the population mutation rate was 0.25. By trying to reach a more stable population where good enough performance is observed, a population mutation rate of 0.15 is recommended. The impact on the run time was not meaningful, then it is not reported here.

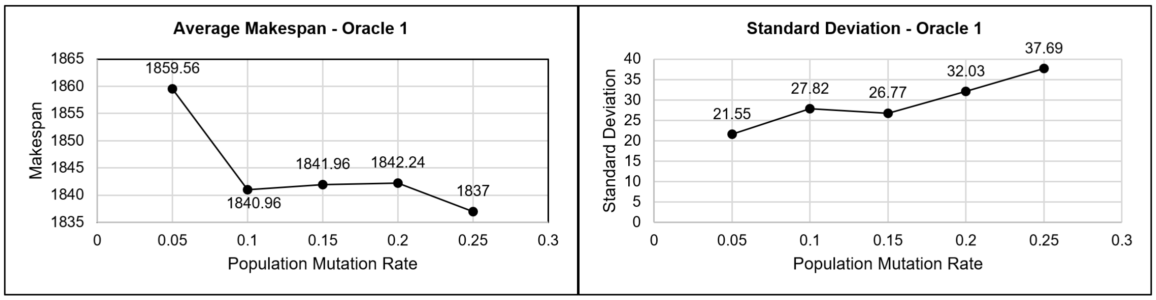


Figure 5.12: Test 2 - Population Mutation Rate - Oracle Z_1

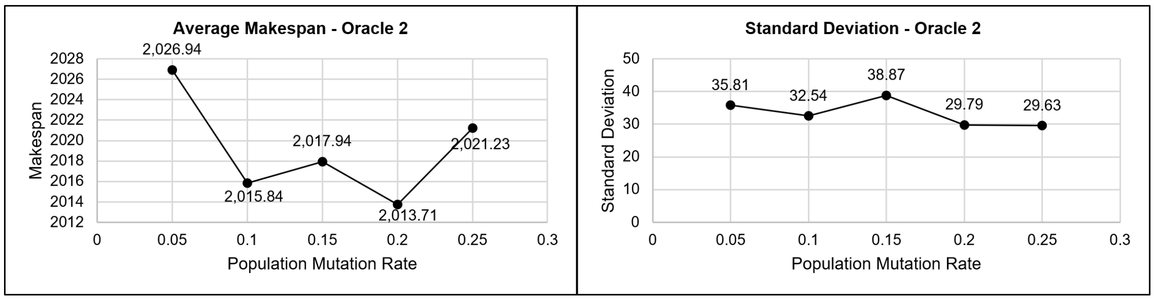


Figure 5.13: Test 2 - Population Mutation Rate - Oracle Z_2

Test 3 - Population Size

The population size dictates how many individuals are created and how many individuals survive at the end of each generation. The more individuals are added, the more calculations are made, requiring more computational resources and considerably increasing the average time to run. This parameter has to be balanced to achieve the desired level of performance with a feasible run time. Test 3 varies the population size from 300 up to 900 by increments of 150 individuals. The maximum number of generations was fixed in 200, the gene mutation selection rate was fixed to 0.05, and the population mutation rate was fixed to 0.15. After 25 runs, the average of the results from Oracles Z_1, Z_2 and Z_3 are shown in Figures 5.15, 5.16, and 5.17 respectively.

The variation of the population size did not increase the overall performance of the algorithm when compared to the impact on the average time to run specially for Oracle Z_3. For instance, if considering the 300 population size as a baseline for Oracle Z_3 an improvement of only 1.59% on average on the makespan was observed with an increase of 220.74% on the average run time. Since the observed improvement is marginal and the impact on the average time to run is considerable, a population

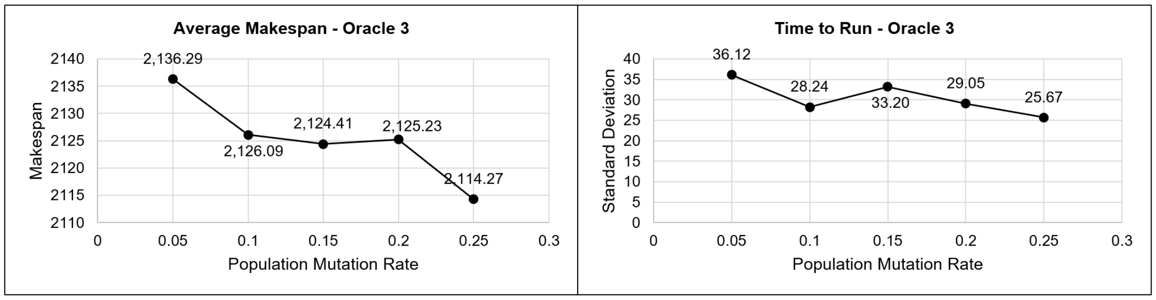


Figure 5.14: Test 2 - Population Mutation Rate - Oracle Z_3

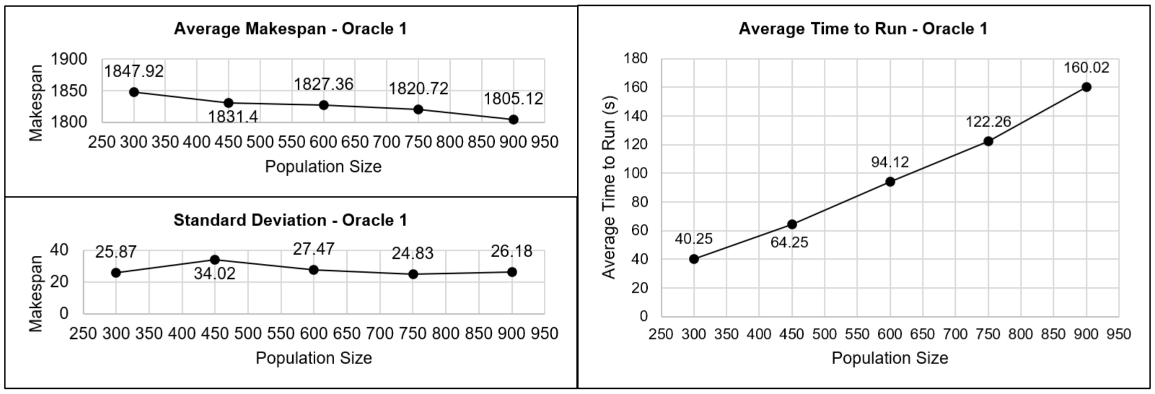


Figure 5.15: Test 3 - Population Size - Oracle 1

size of 300 is recommended if trying to save resources and time.

Test 4 - Maximum Number of Generations

The maximum number of generations specifies how many iterations of the crossover/mutation a population will go through until it stops. Test 4 was designed to analyze the impact of the number of generations by varying from 100 up to 900 by increments of 200. The population size was set to 600, the population mutation rate was set to 0.15, and the gene selection rate was set to 0.05.

Similarly to what was observed in Test 3, this parameter impacts deeply the average run time of the genetic algorithm. Considering 100 generations as a baseline for Oracle Z_1 the average improvement on the makespan is 3.91% compared to an increase of 767.38% in the average run time of the genetic algorithm. Since the decrease of the average makespan is slim when compared to the steep increase of the average time to run, this parameter should be set to 100.

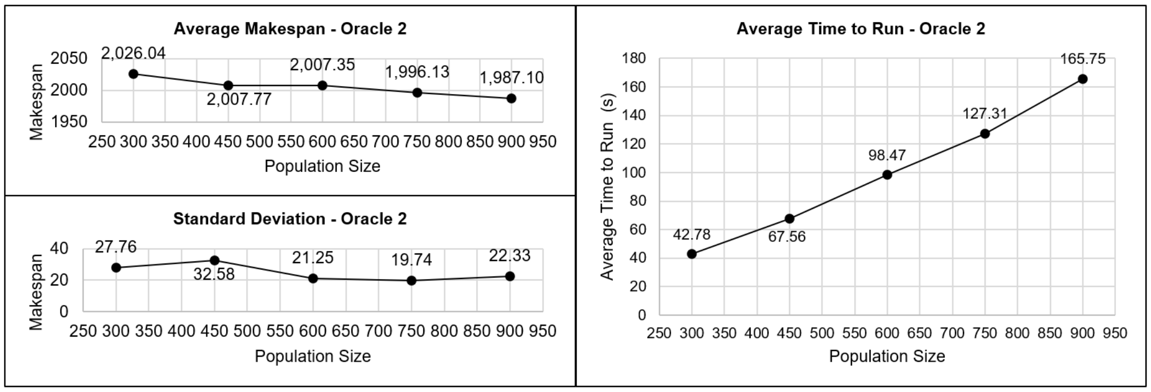


Figure 5.16: Test 3 - Population Size - Oracle 2

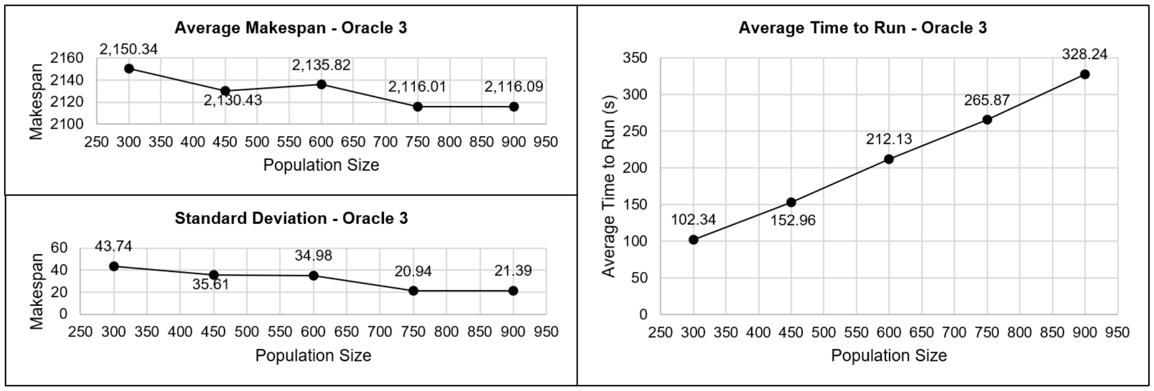


Figure 5.17: Test 3 - Population Size - Oracle 3

5.2.2 Final Experiments

For this section, Taillard's instances 'ta01', 'ta11', 'ta21', 'ta31', 'ta41' and 'ta51' were selected to test Algorithm's 11 and Algorithm's 12 performance. The instances sizes are listed in Table 5.4.

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

Table 5.4: Taillard's Instance Size. Source: Taillard [44]

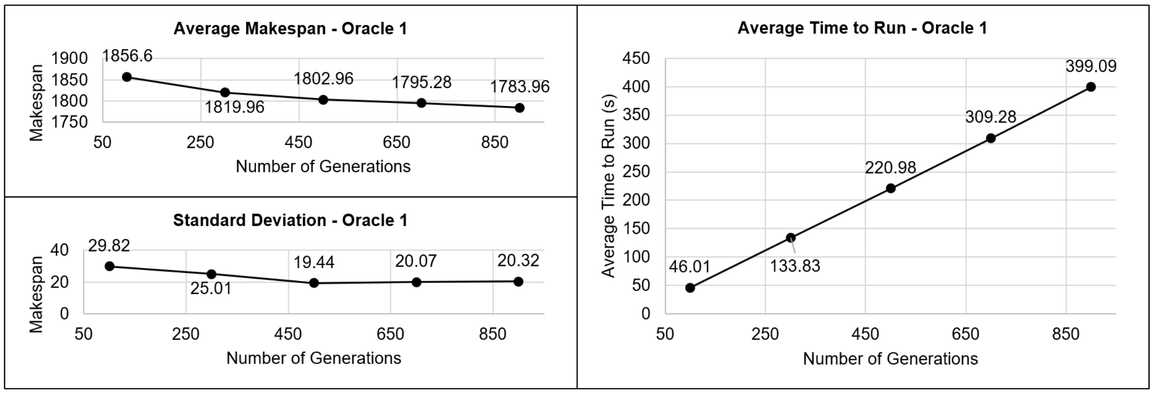


Figure 5.18: Test 4 - Maximum Number of Generations - Oracle 1

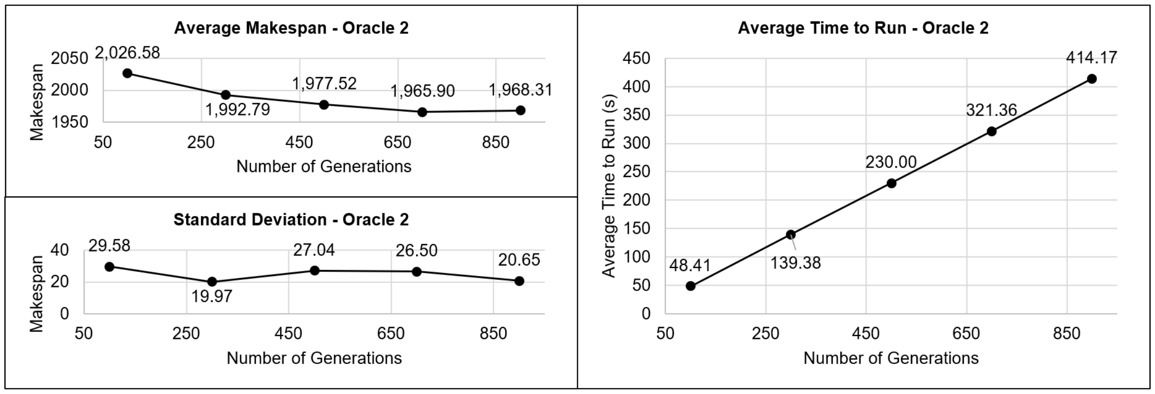


Figure 5.19: Test 4 - Maximum Number of Generations - Oracle 2

The global parameters of the algorithm were set similarly on how it was done in section 5.2.1. The preventive time factor was set to 0.8; the the corrective maintenance time factor was set to 0.4; the scale parameter factor was set to 6; the shape parameter was set to 2.5; the number of elite chromosomes was set to 4; ρ was set to 0.5; The genetic algorithm parameters were set as follows: the population size was set to 150; the gene selection rate was set to 0.05; the population selection rate was set to 0.15; the maximum number of generations was set to 200. The number of scenarios / iterations required by the simulation procedure was set based on the evaluation of a 95% confidence interval on the Normal distribution with an error bound of 1% of the true mean value and was set to 4970 scenarios / iteration. The simulated annealing parameters of Algorithm 12 were set as follows: the maximum number of iterations was set to 100; the temperature update factor was set to 0.8 and the temperature update run was set to 3. A total of 5 runs were executed for every instance and their

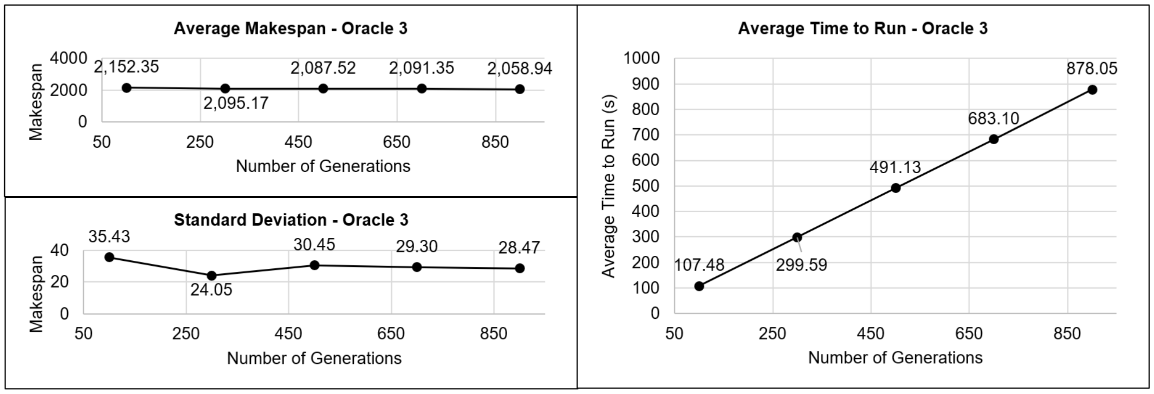


Figure 5.20: Test 4 - Maximum Number of Generations - Oracle 3

average behaviours are summarized in Figure 5.21 and Figure 5.22 for Algorithm 11 and Algorithm 12 respectively.

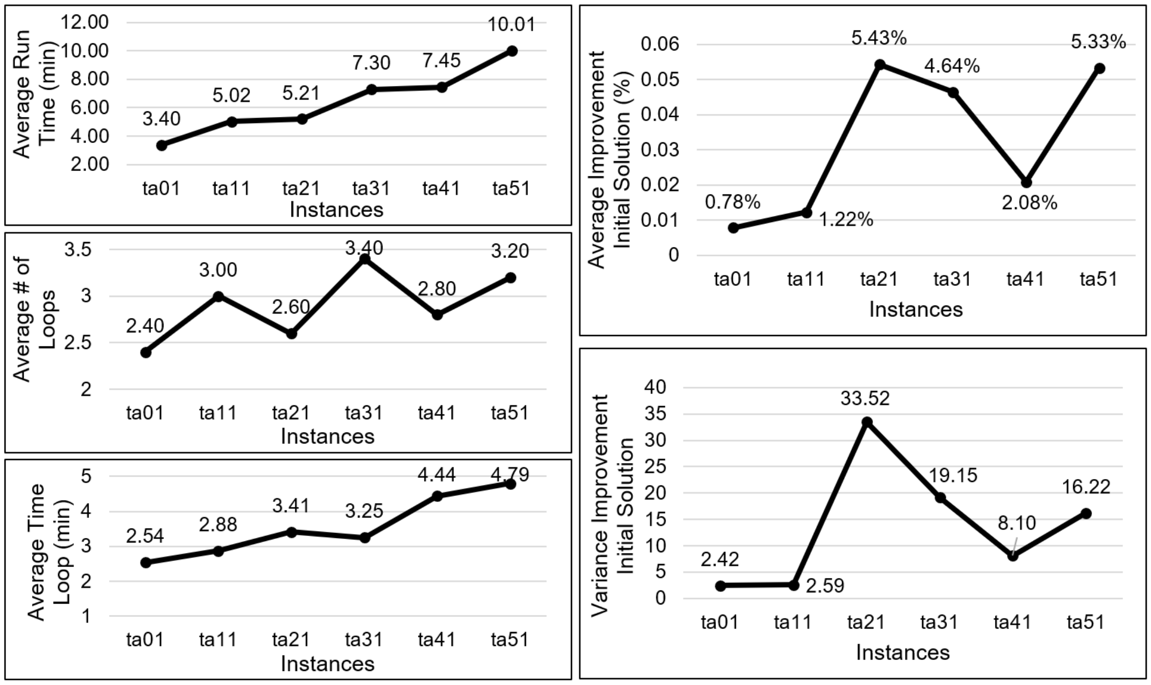


Figure 5.21: JSP Algorithm 11 result summary

For Algorithm 11 the average improvement over the initial solution is small, reaching at most 5.43% of improvement on the 'ta21' instance. This can be explained by the fact that only an average of 3.40 loops were done when solving this instance. Indicating that the algorithm is stopping prematurely, thus not reaching a good solution conversion. On the other hand, the execution time of only 5.21 minutes shows that

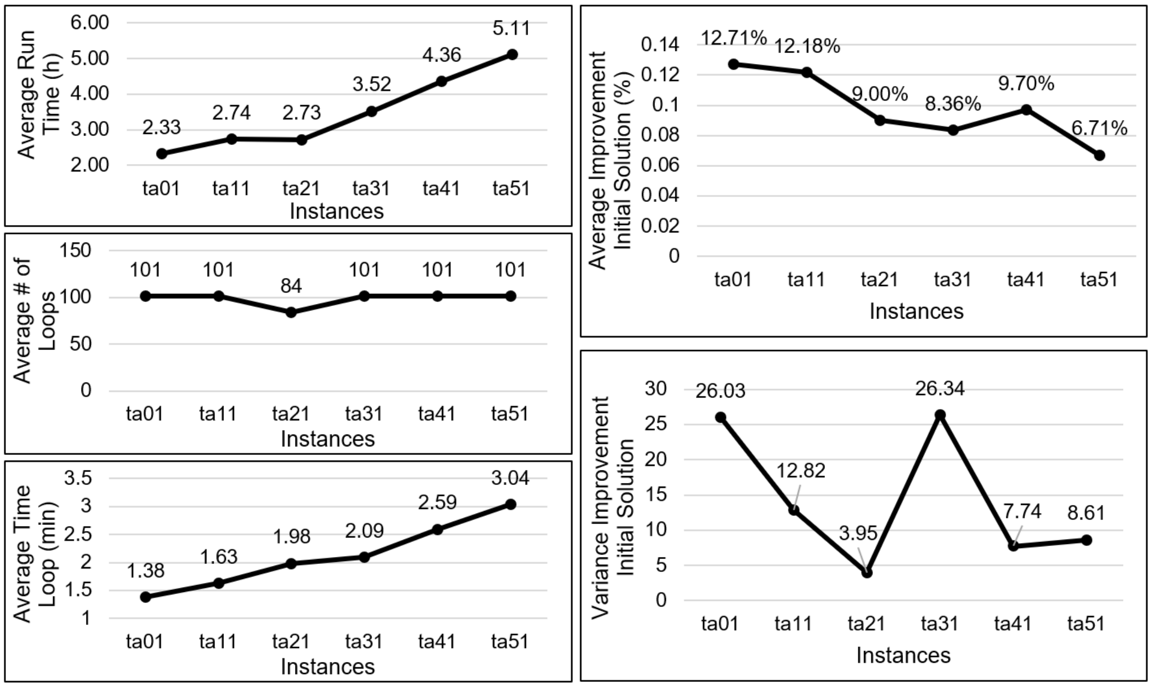


Figure 5.22: JSP Algorithm 12 result summary

this algorithm can be generally good to generate quick solutions that can be used by the management as a starting point for further improvements.

For Algorithm 12 the ‘ta01’ instance observed an improvement of 12.71% on average over the initial solution with only 2.33 hours of average execution time. The Algorithm was able to better improve problems with smaller size, only achieving an improvement of 6.71% on the ‘ta51’ instance.

Algorithm 12 showed significant solution improvement over Algorithm 11 but with relative higher execution time. The higher number of average loops allows Algorithm 12 to reach better conversion. The run time of 4.44h for the ‘ta41’ with 20 machines and 40 jobs is not unreasonable since that the production management team can execute this evaluation during the night in preparation for the next day’s shift.

5.3 Conclusion and Future Research

Inspired by a situation in a real-world company, this thesis proposes three algorithms for the FSP and two algorithms for the JSP with integrated considerations regarding preventive maintenance and random machine breakdown. Two performance metrics were described as quality robustness and solution robustness and combinations of both were used for schedule evaluation.

The algorithms were coded using Python 3.8 with Numpy 1.18, Numba 0.50.1, and Cython 0.29.21 frameworks. The flow shop test instances were randomly created using a set of predetermined parameters. The job shop test instances, on the other hand, were created based on Taillard's standard JSP library.

The first algorithm of the flow shop problem showed no improvement over the initial solution. The second algorithm did not perform well because of its premature termination, thus leading to a poor solution conversion. The third algorithm showed marginal improvements over the initial solution with a much higher average run time. The NEH algorithm showed to be a great start point and only pairwise neighbourhood exploration, even with all effort, was not enough to provide considerable improvement in a reasonable time.

The first algorithm of the job shop problem showed marginal improvements with quick run time. The solution for a 50-job, 20-machine problem takes an average of 10 minutes. By itself, the first algorithm can be used to generate good quick starting points for the management team analysis. The second algorithm showed good performance by running in reasonable times with considerable improvements over the initial solution. The bigger the instance, the harder it is to improve using the proposed algorithm.

5.3.1 Future Research

Quality Robustness: The objective function only takes into consideration the schedule's makespan as one of the objective function factors. Other different metrics are also important and should be further analyzed, such as the number of tardy jobs, weighted completion times, and maximum lateness.

Maintenance Policy: The presented preventive maintenance is a predictive scheduled time-driven model. Other policies as condition-based maintenance could be used instead and its interaction with random breakdowns could be further explored.

Metaheuristics: Specially on the outer scope of Algorithm 12, using a different metaheuristic could lead to better conversion of results. A good suggestion should be using Tabu Search instead of simulated annealing.

Implementation: new GPU computing technologies are becoming more widely available and coding these algorithms in a way that leverages the speed increase of these technologies can improve the run time of the algorithm significantly.

Bibliography

[1] Joseph Adams, Egon Balas, and Daniel Zawack. Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, 34(3):391–401, 1988.

[2] Riad Aggoune. Minimizing the makespan for the flow shop scheduling problem with availability constraints. In *European Journal of Operational Research*, volume 153, pages 534–543, 2003.

[3] Ehsan Ahmadi, Mostafa Zandieh, Mojtaba Farrokh, and Seyed Mohammad Emami. 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.

[4] Nasr Al-Hinai and Tarek Y. Elmekkawy. Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm. *International Journal of Production Economics*, 132(2):279–291, 2011.

[5] Scott W. Ambler. *The elements of UML^{TM} 2.0 style*, volume 9780521616. Cambridge University Press, 2005.

[6] Kenneth R. Baker and Dan Trietsch. *Principles of sequencing and scheduling: Second edition.* John Wiley & Sons, Inc., New Jersey, 2018.

[7] Egon Balas. Machine Sequencing Via Disjunctive Graphs: An Implicit Enumeration Algorithm. *Operations Research*, 17(6):941–957, 1969.

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

[9] Christian Bierwirth, Dirk C. Mattfeld, and Herbert Kopfer. 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.

[10] George H. Brooks and Charles R. White. An algorithm for finding optimal or near optimal solutions to the production scheduling problem. *Journal of Industrial Engineering*, 16(1):34, 1965.

[11] Edmund K. Burke and Graham Kendall. Search methodologies: Introductory tutorials in optimization and decision support techniques. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 1–620, 2005.

[12] C. Richard Cassady and Erhan Kutanoglu. Minimizing job tardiness using integrated preventive maintenance planning and production scheduling. *IIE Transactions (Institute of Industrial Engineers)*, 35(6):503–513, 2003.

[13] Vladimir Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[14] Wei Wei Cui, Zhiqiang Lu, and Ershun Pan. Integrated production scheduling and maintenance policy for robustness in a single machine. *Computers and Operations Research*, 47:81–91, 2014.

[15] Weiwei Cui, Zhiqiang Lu, Chen Li, and Xiaole Han. A proactive approach to solve integrated production scheduling and maintenance planning problem in flow shops. *Computers and Industrial Engineering*, 115:342–353, 2018.

[16] Lawrance Davis. Job Shop Scheduling with Genetic Algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 136–140, New Jersey,, 1985. L. Erlbaum Associates Inc.

[17] Kathryn A. Dowsland. Simulated annealing. In *Modern heuristic techniques for combinatorial problems*, chapter Simulated, page 315. John Wiley & Sons, Inc., New York, NY, 1993.

[18] Fatima El Khoukhi, Jaouad Boukachour, and Ahmed El Hilali Alaoui. The "Dual-Ants Colony": A novel hybrid approach for the flexible job shop scheduling problem with preventive maintenance. *Computers and Industrial Engineering*, 106:236–255, 2017.

[19] David S. Johnson Garey, Michael R. and Ravi Sethi. Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[20] Mansour Gholami and Zandieh Mostafa. Integrating simulation and genetic algorithm to schedule a dynamic flexible job shop. *Journal of Intelligent Manufacturing*, 20(4):481–498, 2009.

[21] Ronald L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5(C):287–326, 1979.

[22] Hatem Hadda, Najoua Dridi, and Sonia Hajri-Gabouj. An improved heuristic for two-machine flow shop scheduling with an availability constraint and nonresumable jobs. *4or*, 8(1):87–99, 2010.

[23] Willy Herroelen and Roel Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289–306, 2005.

[24] Frederick S. Hillier and Gerald J. Lieberman. *Introduction to operations research*. Number BOOK. McGraw-Hill Education, 2015.

[25] John H. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, Ann Arbor, 1992.

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

[27] Jongyoung Jun, Joomyung Kang, Daein Jeong, and Haeseon Lee. 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.

[28] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[29] Mikhail A. Kubzin and Vitaly A. Strusevich. Two-machine flow shop no-wait scheduling with machine maintenance. *4or*, 3(4):303–313, 2005.

[30] Chung Yee Lee, Lei Lei, and Michael Pinedo. Current trends in deterministic scheduling. *Annals of Operations Research*, 70(0):1–41, 1997.

[31] Jun Qing Li, Quan Ke Pan, and M. Fatih Tasgetiren. A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities. *Applied Mathematical Modelling*, 38(3):1111–1132, 2014.

[32] Z. A. Lomnicki. 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.

[33] Zhiqiang Lu, Weiwei Cui, and Xiaole Han. Integrated production and preventive maintenance scheduling for a single machine with failure uncertainty. *Computers and Industrial Engineering*, 80:236–244, 2015.

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

[35] Alan S. Manne. On the Job-Shop Scheduling Problem. *Operations Research*, 8(2):219–223, 1960.

[36] E. Mokotoff. Multi-objective simulated annealing for permutation flow shop problems. *Studies in Computational Intelligence*, 230:101–150, 2009.

[37] Muhammad Nawaz, E. Emory Enscore, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[38] Maroua Nouiri, Abdelghani Bekrar, Abderrazak Jemai, Damien Trentesaux, Ahmed C. Ammari, and Smail Niar. 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.

[39] Object Managemt Group. About the Unified Modeling Language Specification Version 2.5.1, 2019.

[40] Michael L. Pinedo. *Scheduling: Theory, algorithms, and systems, fifth edition.* Springer, Cham, Gewerbestr, fifth edit edition, 2016.

[41] Rubén Ruiz, J. Carlos García-Díaz, and Concepción Maroto. Considering scheduling and preventive maintenance in the flowshop sequencing problem. *Computers and Operations Research*, 34(11):3314–3330, 2007.

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

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

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

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

[46] Yong Ming Wang, Hong Li Yin, and Kai Da Qin. A novel genetic algorithm for flexible job shop scheduling problems with machine disruptions. *International Journal of Advanced Manufacturing Technology*, 68(5-8):1317–1326, 2013.

[47] www.collinsdictionary.com/. Collins English Dictionary - Complete & Unabridged, 2014.

[48] William Zimmer. *An Introduction to Reliability and Maintainability Engineering*, volume 31. Waveland Press, 1999.

Appendix A

Supplementary Material

Description:

The accompanying zip folder contains all the algorithms implemented following what is described during this thesis. All instances and data relevant to the execution of said algorithms are also included.

Filename:

code.zip