FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Meta-Heuristics for Job-Shop Rescheduling

**Gonçalo Santos Oliveira**


Mestrado em Engenharia Informática e Computação

Supervisor: Luís Paulo Reis

July 30, 2022

# Meta-Heuristics for Job-Shop Rescheduling

**Gonçalo Santos Oliveira**

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

President: Daniel Silva
Referee: José Manuel Torres
Supervisor: Luís Paulo Reis

July 30, 2022

# Abstract

Nowadays, manufacturers must allocate their resources optimally to maximise their output while minimising costs and keeping up with an increasingly competitive market. This goal can be achieved using manufacturing management processes such as Advanced Planning and Scheduling, which aims to allocate raw materials and production capacity to meet demand optimally. Specifically, manufacturers must understand the specifications and requirements of each order, represent them on the shop floor [1, 2, 3, 4] and create an optimal schedule around that information. In Computer Science, this problem is known as the Job-shop Scheduling problem, and there have been many attempts to solve it. However, although strategies that find optimal schedules are known, they suffer from having a search space that grows exponentially. Because of this, they do not scale well, making them unsuitable for real-world use cases where efficiency is vital for the success of the manufacturing operation. Besides, the dynamic nature of shop floor production represents a significant obstacle for traditional strategies, whose rigidity prevents the adaptation of the schedule to unpredictable real-time events, such as machine failures, changes in requirements, and new orders [1, 4]. In this project, we study a strategy to solve the Job-shop Scheduling problem that addresses the issues of traditional solutions using innovative algorithms and metaheuristics from Artificial Intelligence. We develop testing scenarios from data gathered from real-world manufacturers to evaluate the performance and quality of results achieved by the algorithms we implemented. Our results confirm that it is possible to improve the results achieved by current optimisation techniques for large instances.

**Keywords**: Advanced Planning and Scheduling, Complex Job-shop Scheduling Problem, Manufacturing Management

**ACM Classification**:
CCS > Computing Methodologies > Artificial Intelligence > Planning and Scheduling > Planning for deterministic actions
CCS > Applied Computing > Operations Research > Multi-criterion optimization and decision-making
CCS > Theory of Computation > Design and analysis of algorithms > Mathematical optimization > Discrete optimization > Optimization with randomized search heuristics

# Resumo

Nos dias de hoje, as indústrias têm de alocar recursos de forma a maximizar a sua produção, enquanto minimizam os custos associados à sua atividade e sobrevivendo num mercado cada vez mais competitivo. Este objetivo pode ser obtido recorrendo a processos de gestão da produção, tais como o Advanced Planning and Scheduling, que tenta alocar matérias-primas e capacidade de produção de forma a cobrir a procura. Concretamente, este processo traduz-se na contextualização das especificidades e requisitos de cada encomenda no chão de fábrica e, com essa informação, criar um escalonamento das atividades. Em Ciência de Computadores este problema é conhecido como Job-shop Scheduling e existem várias técnicas de resolução. Apesar disso, embora estas técnicas consigam encontrar soluções ótimas, o conjunto de soluções possíveis cresce exponencialmente com o número de tarefas e de máquinas, o que torna inviável o seu uso em situações do mundo real onde a eficiência é de vital importância para o sucesso da operação fabril. Por outro lado, a natureza dinâmica da produção no chão de fábrica representa um grande obstáculo para soluções tradicionais que, em virtude da sua rigidez, são incapazes de se adaptar a eventos imprevisíveis no chão de fábrica, tais como avarias em máquinas, mudanças nos requisitos e a receção de novas encomendas. Nesta dissertação, estudaremos uma estratégia para resolver o problema de Job-shop Scheduling, endereçando as lacunas das soluções tradicionais usando algoritmos tradicionais e metaheurísticas de Inteligência Artificial. Desenvolveremos cenários de teste a partir de dados obtidos em fábricas reais, de forma a avaliar o desempenho e a qualidade dos resultados obtidos pelos algoritmos implementados. Os resultados confirmarão que é possível melhorar os resultados obtidos por técnicas de otimização tradicionais para instâncias de elevada dimensão.

**Palavras-chave**: Advanced Planning and Scheduling, Complex Job-shop Scheduling Problem, Manufacturing Management

**Classificação ACM**:
CCS > Computing Methodologies > Artificial Intelligence > Planning and Scheduling > Planning for deterministic actions
CCS > Applied Computing > Operations Research > Multi-criterion optimization and decision-making
CCS > Theory of Computation > Design and analysis of algorithms > Mathematical optimization > Discrete optimization > Optimization with randomized search heuristics

# Acknowledgements

I would like to thank Professor Luis Paulo Reis for the knowledge on scheduling algorithms shared which helped give insight into the scheduling problem.

I would also like to thank Francisco Andrade, my supervisor at Critical Manufacturing, for guiding the solution development process and for helping in the review process of this dissertation report.

Finally, I would like to thank my friends and family for their constant support.

Gonçalo Oliveira

*"Better three hours too soon than a minute too late"*

William Shakespeare

# Contents

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| B&B | Branch and Bound |
| BS | Beam Search |
| ACO | Ant-Colony Optimization |
| CPS | Cyber-Physical System |
| FBS | Filtered Beam Search |
| GA | Genetic Algorithm |
| HTTP | Hypertext Transfer Protocol |
| IoT | Internet of Things |
| IIoT | Industrial Internet of Things |
| JSP | Job-shop Scheduling Problem |
| JSON | JavaScript Object Notation |
| MES | Management Execution System |
| ERP | Enterprise Resource Planner |
| NN | Neural Network |
| XML | Extensible Markup Language |

# Chapter 1

# Introduction

## 1.1 Context and Motivation

As customer expectations of product delivery quality keep growing, manufacturing companies must adopt increasingly sophisticated manufacturing processes to protect their position in the highly competitive industrial production market. Besides, the demand for customized products is a challenge to traditional mass production processes, given the necessity of representing each order's specifications and requirements on the shop floor.

Nonetheless, with the increasing digitization of industrial processes enabled by advances in technology and materialized in the Fourth Industrial Revolution, factories are capable of leveraging manufacturing execution systems (MES) to plan resource allocation [1, 2, 3, 4] according to a manufacturing management process known as Advanced Planning and Scheduling. Specifically, scheduling problems on the shop floor are often instances of the Job-Shop Scheduling Problem, a subclass of the optimization problem known as Optimal Job Scheduling.

Scheduling refers to the allocation of finite resources to activities which must be completed before a pre-determined deadline or, as is usual in the manufacturing context, the allocation of machines to tasks while considering both monetary and temporal constraints. Over time, researchers have shown that many scheduling algorithms can compute high-quality solutions in the context of shop-floor production. However, such algorithms tend to be rigid and often struggle to adapt to unpredictable events such as machine failures, changes in requirements and new orders, leaving the rescheduling problem unsolved [1, 4].

## 1.2 Aim and Goals

This dissertation aims to compare the effectiveness of three approaches based on meta-heuristics to solve the Job-Shop Scheduling Problem versus a greedy strategy. Each of the implemented approaches is contained in a module which communicates with an MES and is capable of creating a new schedule in the event of unexpected machine failures, changes in requirements and new orders, thus delivering better tolerance to failure. Besides, the solution attempts to significantly

reduce the size of the search space, thus diminishing the overall execution time of the algorithm, unsurprisingly, at the cost of not guaranteeing that a globally optimal solution can be found. In order to run each of the implemented meta-heuristics, it is necessary to recompile the program using the appropriate flag.

This contribution aids in the creation of robust solutions for real-world corporations in the manufacturing business, which consequently results in reduced costs for manufacturers as it allows factories to adapt much more quickly to unexpected events while fulfilling monetary and temporal requirements - budgets and deadlines. Finally, this dissertation presents a system capable of building a schedule which is adjusted to the characteristics of the shop floor and, therefore, not only does it result in better resource management but also confers the factory an improved ability to deliver customized products on time.

This dissertation was done in partnership with Critical Manufacturing SA, a Maia, Porto-based software company specialized in the development of Manufacturing Execution Systems (MES) for the Semiconductor and Electronics industry. The proposed solution integrates with their product.

## 1.3 Document Structure

Chapter 2 focuses on the context of the dissertation, presenting an overview of the Fourth Industrial Revolution and the technologies that enable the use of digital scheduling systems before introducing the state of the art in optimization algorithms applied to the Job-Shop Scheduling Problem and strategies to handle unexpected events in the shop floor. Afterwards, Chapter 3 describes the problem at hand in detail, introducing its mathematical formulation and how it applies to Critical Manufacturing's goal of solving the Job-Shop Scheduling Problem. This dissertation also addresses the rescheduling problem. Next, Chapter 4 showcases the architecture of the proposed solver and how it integrates with the MES, as well as a description of the meta-heuristics that were implemented. Meanwhile, Chapter 5 describes multiple test scenarios and compares the results achieved by the proposed solution with those reached by a greedy solver. Ultimately, Chapter 6 presents final comments and suggestions for future work.

# Chapter 2

# Literature review

The purpose of this chapter is to describe the context of the dissertation by introducing relevant concepts and exploring the state of the art in the scientific fields it is related to. Firstly, section 2.1 presents the Industry 4.0 paradigm, a new industrial archetype permitted by the latest advances in technology. Secondly, section 2.2 focuses on the Job-Shop Scheduling Problem and presents a historical overview of the various techniques that have been used to solve it. Next, section 2.3 aims to present a solution to the Job-Shop Scheduling Problem which formulates the Job-Shop Scheduling Problem as a decentralized and multi-agent based environment. Finally, section 2.4 takes a closer look at a concrete application of Genetic Algorithms to solve the JSP.

## 2.1 Industry 4.0

This section focuses on the Fourth Industrial Revolution, specifically the digitization of industrial processes, by first presenting an overview of its history and, afterwards, the concepts and technologies that explain the necessity of implementing an automated scheduling system in the new industrial context known as Industry 4.0. In the end, the section concludes with a mention of the consequences of the revolution on society.

### 2.1.1 History

Industry 4.0 refers to the embodiment of rapid technological development and consequent change in industrial processes and societal patterns in the $21^{st}$ century enabled by increasing inter-connectivity and smart automation. The term was popularized by Klaus Schwab, Founder and Executive Chairman of the World Economic Forum, who claims that "the fourth industrial revolution creates a world in which virtual and physical systems of manufacturing globally cooperate with each other in a flexible way" [10].

Incidentally, the first industrial revolution, which spanned from the later half of the $18^{th}$ century to the early half of the $19^{th}$ century, enabled by the large-scale construction of railroads and the invention of the steam engine, saw a significant increase in mechanical production. Afterwards, in the late $19^{th}$ century, the second industrial revolution empowered by the mass adoption

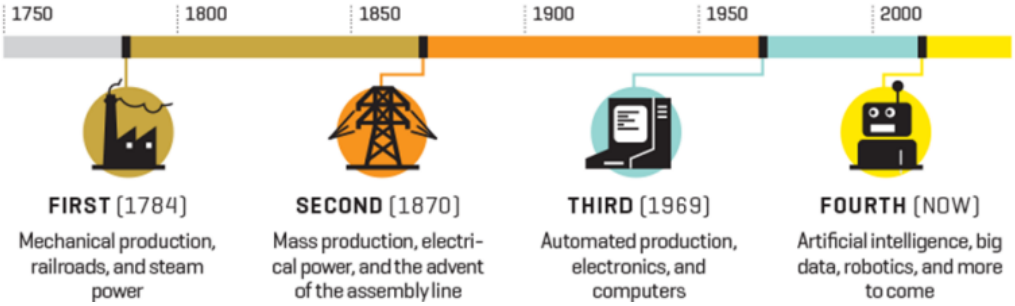


Figure 2.1: The four industrial revolutions [5]

of electricity and the creation of the assembly line, resulted in the creation of industrial processes geared towards mass production. Next, in the 1960s, with the development of semiconductors, mainframe and personal computing and with the invention of the internet came the third industrial revolution.

In fact, the Fourth Industrial Revolution represents a major change in economic systems and societal structures just like the revolutions of the past. According to Klaus Schwab [10] the latest shift in the industrial production paradigm, which started in the beginning of the century, is characterized by an increasing digitalization of industrial processes not only powered by ubiquitous access to the internet and low costs on the acquisition of sensors but also permitted by the latest research in the fields of Artificial Intelligence, Machine Learning and Cyber-Physical Systems, which includes areas of study such as the Internet of Things, Big Data, Cloud Computing [10, 11], Digital Twin and Agent-based Systems [12]. The term Industry 4.0, "the new buzzword in the manufacturing industry" [13], which refers to processes regarding the entire value chain, manufacturers and service providers [13], was coined at the 2011 Hannover Fair. Figure 2.1 shows an overview of the four industrial revolutions described in this section.

### 2.1.2 Internet of Things and Cyber-Physical Systems

The Internet of Things or IoT, commonly referred to as Industrial Internet of Things (IIoT) when applied in an industrial context, is a core component of the Industry 4.0 paradigm. IIoT focuses on interconnected devices, usually sensors and instruments on the shop floor, which collect, exchange and analyse information sent over the Internet in real-time. In fact, the transmitted data may represent information about the state of a machine or tool - its temperature, rotation speed, power consumption and vibration - which can then be used to predict when a machine might fail and reschedule the operations in the shop floor to take such failure into account.

Furthermore, some researchers have also focused on Cyber-Physical Systems, which attempt to bridge the gap between the physical and digital worlds [11]. Such systems effectively create a digital twin of a set of interconnected devices by simulating the interactions between physical devices in a digital context.

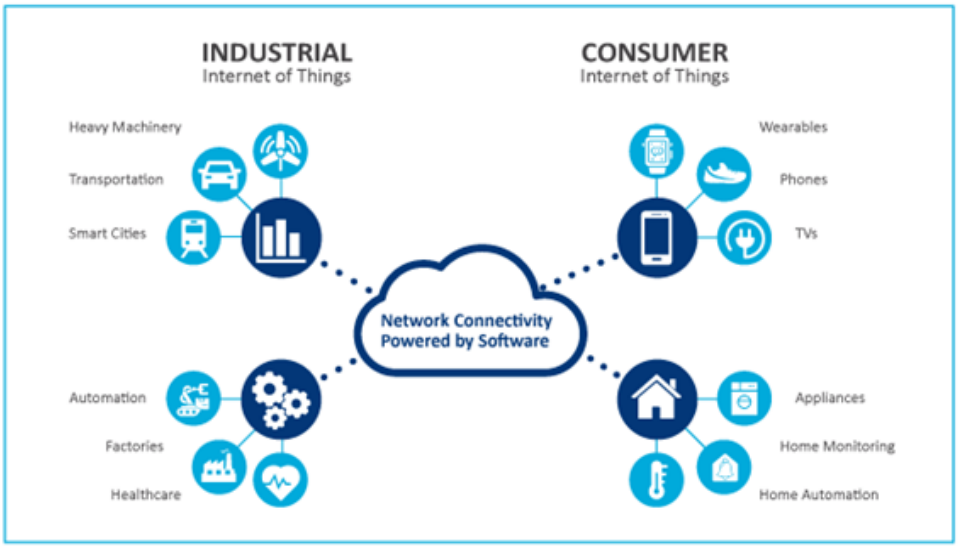


Figure 2.2: The IoT and IIoT application domains [6]

In addition, although the Internet of Things and Cyber-Physical Systems are related, they serve different purposes. While IoT is mainly focused on the interconnectedness of devices on the shop floor and how they communicate with each other, the study of CPSs is mostly aimed at the interactions between the physical and the digital worlds [14].

### 2.1.3 Smart Manufacturing

Smart Manufacturing is a category of manufacturing which not only enables the mass-production of highly personalized products [15] at a competitive cost, but also enables dynamic changes in production throughput based on demand [16], supply chain optimization [16] as well as efficient production and recyclability [17]. The Smart Manufacturing Leadership Coalition defines this concept as a set of manufacturing practices which rely on information and communication technology to shape future manufacturing operations [16]. Meanwhile, Wallace and Riddick [18] define smart manufacturing as a "data-intensive application of information technology at the shop floor level and above to enable intelligent, efficient and responsive operations". Finally, according to Lu et al, Smart Manufacturing can be characterized by "digitization and service-orientation, smart and connected automation devices and collaborative manufacturing networks" [15].

The definitions presented above imply that Smart Manufacturing relies heavily on automation and on the digitization of industrial processes. In fact, the latest advances in industrial robotics and artificial intelligence allow workers and robots to cooperate with each other by learning from human demonstrations and using the gathered knowledge to reason about different problems beyond what they were originally programmed to solve [19].

### 2.1.4 Scheduling in Smart Manufacturing

Scheduling is a management technique which enables the optimization of production processes and the efficient allocation of resources, specifically, it is a "process of arranging, controlling and optimizing work on the shop floor" [20]. In traditional manufacturing, scheduling relies on manual computations or on the use of simple, inefficient and limited programs, which lead to wrong or incomplete solutions due to a lack of knowledge on the current conditions of the shop floor. In fact, implementing a scheduling system capable of using real-time data is highly complex and represents a non-deterministic polynomial-time hard combinatorial optimization problem [21]. Under the mass production paradigm, products are easy to copy and are made in series according to the same method, meaning that the time needed to finish a product is approximately the same for every copy of the same product. In fact, scheduling systems built for the second and third industrial revolution were designed to achieve high performance and high delivery rates.

However, Industry 4.0 requires factories to output large quantities of highly customized products, in other words to transition from mass production to mass individualization, which calls for more efficient and robust scheduling solutions, not only on the shop floor but also along the value chain [12]. In fact, disturbances in the production line such as the arrival of new orders, canceled orders or machine breakdowns become more significant under such a paradigm given that, in such circumstances, a sudden rescheduling becomes necessary and may stop the shop floor [5]. For these reasons, new scheduling systems must strive for increased autonomy, adaptability and flexibility due to the highly dynamic environment of the modern shop floor [12]. In a systematic review of the state of the art of scheduling in the context of smart manufacturing by Alemão et al [12] the authors found that 65% of articles focus on the minimization of the makespan, 37% focus on the minimization of total tardiness and 15% focused on minimizing energy consumption.

In conclusion, scheduling in Smart Manufacturing is key to a successful industrial venture operating according to the Industry 4.0 paradigm but is still in an early stage of development and most research on the topic does not reflect real-world shop floor environments but rather simplified and heavily controlled ones [12].

### 2.1.5 Manufacturing Execution System

According to Critical Manufacturing, a Manufacturing Execution System or MES is a computerized system "designed to execute production flow, which means a keen focus on the process functions and related data collection and reporting" [7]. Indeed, an MES stores information gathered from sensors and machines on the shop-floor which can be used to monitor the state of the production line and report when a failure happens. In particular, an MES collects real-time data about the parts and materials that are flowing through the production line at a given time and about maintenance steps that may occur [22]. All this data can be used to drive decision support systems and aid workers in understanding their role in the shop-floor, which ultimately results in "better decision making, operational performance and process improvements" [7]. In fact, an MES also allows the factory to arrange, control and optimize work and workloads, a process known as

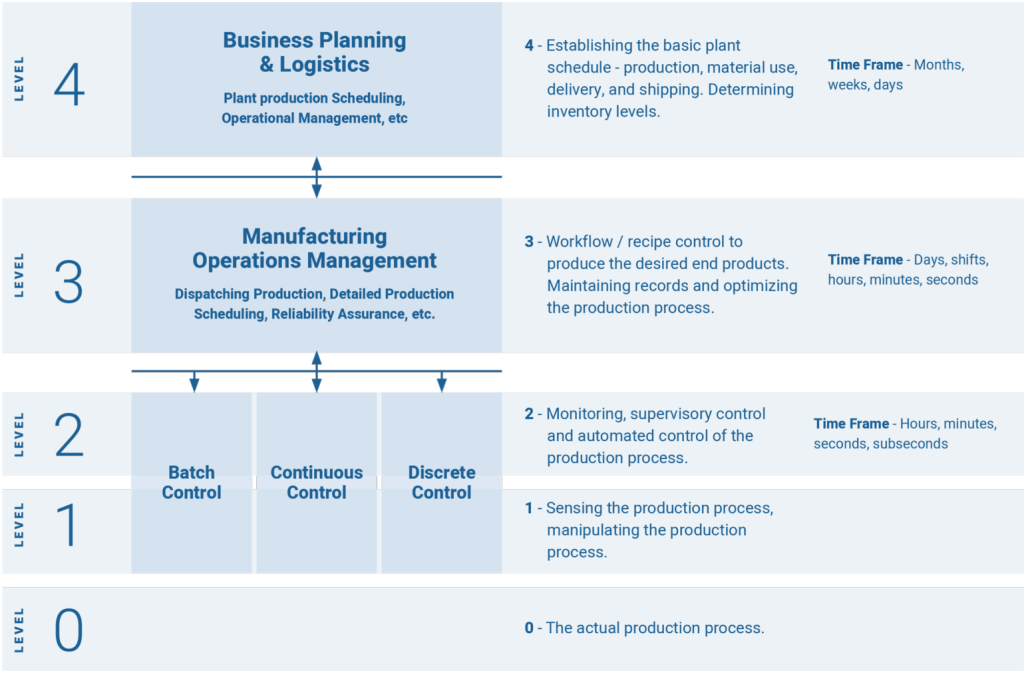


Figure 2.3: ISA-95 functional hierarchy of levels of manufacturing decision making [7]

scheduling, which is used to plan machinery resources, human resources, production processes and purchase of materials.

Besides, according to the ISA-95 functional hierarchy of levels of manufacturing decision making present in Figure 2.3, an MES belongs to the third level of the hierarchy as it is highly connected with the process of Manufacturing Operations Management. In fact, an MES differs from a traditional Enterprise Resource Planner or ERP given that where an MES is "the ideal choice for a complex production process with multiple variations and a massive number of transactions" [23], an ERP "is generally designed to support a homogeneous process with business operating information" [23] and corresponds to the fourth level in the hierarchy of Figure 2.3.

Finally, according to Critical Manufacturing, an MES plays a critical role in the context of Industry 4.0. Incidentally, implementing such a system in a factory is critical to reducing manufacturing cycle time, order lead time, direct labor costs, data entry time and significantly reduces the amount of paperwork needed, the amount of work in progress inventory and, finally, promotes machine utilization. Consequently, these benefits result in increased customer satisfaction and better regulatory compliance [23].

### 2.1.6 Social Impacts

The Fourth Industrial Revolution, like those of the past, will fuel major changes in how customers interact with products but also in the way factories make their products. Indeed, manufacturing times and costs can be significantly reduced with the increased level of automation, improved monitoring and better planning, which results in cheaper products for buyers. Besides, Industry

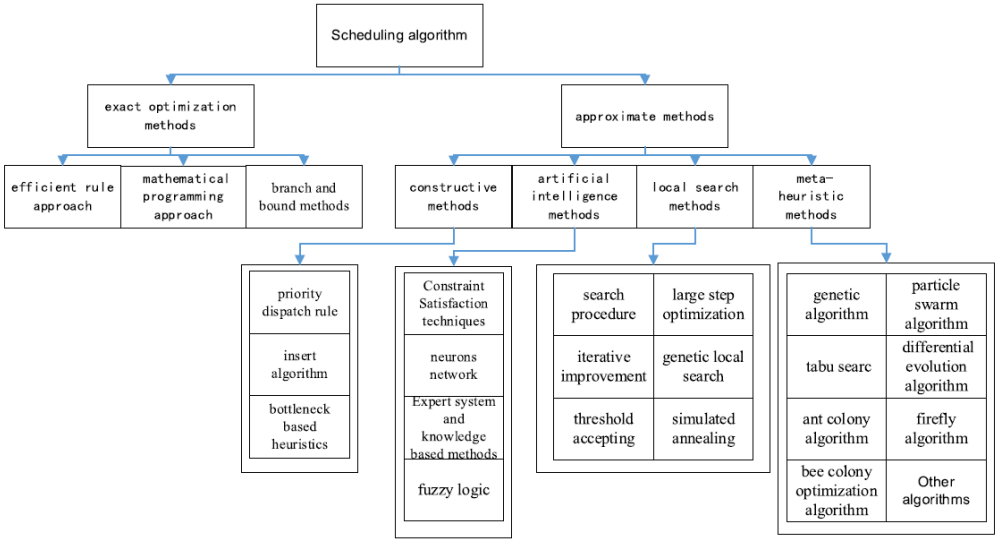


Figure 2.4: Taxonomy of scheduling algorithms [5]

4.0 is capable of increasing the customization of products due to a more streamlined production process, thus providing customers with unique experiences, contrary to traditional mass production which does not promote such variety [24].

On the other hand, the increasing digitization of industrial processes may become a vector for malicious attacks which can shutdown a factory. Additionally, organizations must guarantee reliability and stability in machine-to-machine communication [25]. Besides, there are other consequences of the widespread adoption of the new industrial paradigm put forth by the Fourth Industrial Revolution in the economic, political and social dimensions of society.

## 2.2 Job-Shop Scheduling

This section focuses on the Job-Shop Scheduling Problem, a subclass of the Optimal Job Scheduling problem, and presents an overview of the various approaches for solving this problem, including their advantages, disadvantages and limitations, according to the taxonomy found in Figure 2.4.

### 2.2.1 Problem Description

The Job-Shop Scheduling Problem is an optimization problem in computer science and operations research. Job-Shop Scheduling refers to the scheduling of tasks in the shop floor, which is the place in a factory where workers complete jobs with the aid of machines and tools. Besides, because a factory has finite resources at its disposal - number of workers, machines and raw materials - its efficient use is crucial to avoid unnecessary expenses [1].

Furthermore, scheduling is the act of allocating resources to a set of tasks or, in the manufacturing context, workers and raw materials to machines. Apart from that, an important problem for

| Job | Machine A | Machine B |
|---|---|---|
| Job 1 | 6 | 4 |
| Job 2 | 1 | 3 |
| Job 3 | 3 | 6 |

Table 2.1: Example Input

factories to solve is that of what to do in the face of unexpected events. In fact, modern factories can leverage data gathered from sensors on the shop floor to periodically generate a new schedule that fulfills their goals.

### 2.2.2 Exact Methods

The earliest attempts at solving the Job-Shop Scheduling Problem relied on deterministic methods to calculate optimal results. Despite their success, these solutions do not scale well, making them unsuitable for real-world scenarios.

#### 2.2.2.1 Efficient Rule

Efficient rule methods can obtain an optimal solution by establishing rules that depend on their input data, thus defining the processing order that guarantees an optimal solution. Johnson [26], in his paper, identifies the minimization of the makespan as a goal for any approach for this problem and proposes the following set of rules to achieve it. Firstly, the task processing time must be constant. Secondly, all jobs must be processed in the same order on each machine, and, finally, there must be no priority hierarchy between jobs. Nonetheless, these rules are unsuitable for real world scenarios. Table 2.1 shows an example input to a Job-Shop Scheduling solved using Johnson's efficient rules, where for each job and machine the table shows the cost of allocating a job to a machine.

Other researchers have published their own efficient rules [27, 28, 29]. However, many have found that the computational complexity of the JSP becomes NP-Hard [30] once the number of machines and the number of operations per job is greater than two, meaning that there is no efficient rule for such scenario. Exact methods cannot solve most instances of the JSP efficiently, taking too long to complete. Further exact methods rely on constraints to ignore results that do not improve the solution.

#### 2.2.2.2 Mathematical Programming

Wagner [31], in his 1959 paper, used mathematical programming to solve the JSP. Although Wagner found that mathematical programming can find optimal solutions, further research shows that using such an approach requires excessive computing time or results in poor quality solutions.

| Num Jobs | Num Machines | Combinations |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1 | 2 |
| 2 | 2 | 4 |
| 3 | 2 | 36 |
| 3 | 4 | 1296 |
| 4 | 4 | 331776 |

Table 2.2: Growth in the size of the search space in a Job-Shop Scheduling Problem

In 1960, a paper published by Manne [32] proposed a formulation of the JSP which involved fewer variables and could find a solution in less time than Wagner's proposal by combining integer and linear programming. Integer programming was used for the decision variables and linear programming was used for the constraints.

#### 2.2.2.3 Branch and Bound

Branch and bound was initially used to address the issues of Johnson's efficient rules by building a search tree and discarding solutions which are known to not be optimal, or known to not improve the current best solution, without testing them. In fact, the total number of combinations in a Job-Shop Scheduling Problem with $m$ machines and $n$ jobs grows according to the expression $(n!)^m$. This exponential growth in the size of the search space means that the use of exact methods is infeasible even in small scenarios. Table 2.2 shows how a small increment in the number of jobs or machines results in an explosion in the total number of combinations of allocations of jobs to machines, representing a form of combinatorial explosion.

Brooks and White [33] and Lomnicki [34] used B&B methods to find the optimal solution for the JSP. Further work by Hefetz and Adiri [29] introduced an efficient optimal approach for the two machine JSP.

### 2.2.3 Constructive Methods

Constructive methods are similar to exact ones. They establish a set of rules to allocate tasks to machines. However, these methods relax some of the rules and can find a solution in less time at the expense of the quality of the solution.

#### 2.2.3.1 Dispatch Rules

Dispatch rule methods assign a priority to each of the jobs and create a schedule, taking that priority in consideration. Over time, researchers have identified numerous relevant rules such as the job delivery date, the shortest processing time and the least total work remaining. Nonetheless, there is no universal set of rules for every scenario, so it is necessary to understand the requirements of each problem to get the best results [35].

#### 2.2.3.2 Insert Algorithms

Over time, researchers have found that insert algorithms can be a better alternative to dispatch rules. Instead of establishing a priority between jobs, these algorithms insert jobs into an empty or an existing schedule wherever best fits the scheduling goals.

Research by Werner and Winkler [36] showed that the insert algorithm can be combined with a beam search step to build an initial scheduling and continuously improve it by re-inserting jobs, generating possible solutions for the beam search step. In 1999, Sotskov, Tautenhahn and Werner [37] applied insert algorithms to the batch production problem, where several jobs can be executed at the same time. Afterwards, in 2014, Zheng, Lian and Mesghouni [38] assessed the performance of various algorithms in handling the scheduling of maintenance operations. Finally, in 2019, Bekkar, Belalem and Beldjilali [39] applied a greedy insert algorithm in a scenario with transportation time constraints.

#### 2.2.3.3 Bottleneck Heuristics

Bottleneck-based heuristics present a more balanced trade-off between result quality and execution time, when compared to previous methods. A Bottleneck-based strategy called Shifting Bottleneck Process was first applied to the JSP by Adams et al [40] by simplifying and decomposing the original problem into single machine scheduling problems. In practice, the proposed method repeatedly sets the order of each job until the set of jobs for the entire order are scheduled.

Further research by Dauzere-Peres and Lasserre [41] presents an improvement on Adams's method which considers delay precedence constraints (DPC), that is, delays between the end of a job and the beginning of the next job. In 1995, Balas, Lenstra and Vazacopoulos [42] combined Shifting Bottleneck with Branch and Bound to solve a scenario with delay precedence constraints.

Recently, researchers have focused on hybrid approaches, combining Shifting Bottleneck with other heuristics. In 2014, a paper by Zhang, H. Manier and M. Manier [43] proposed a Bottleneck-based heuristic to handle scenarios with transportation constraints. In 2016, papers by Zhou and Peng [44] and Cubillos and Cabrera Paniagua [45] used a Bottleneck-based approach for large instances and combined Shifting Bottleneck with Taboo search, respectively.

### 2.2.4 Artificial Intelligence

Artificial Intelligence (AI) was initially proposed in 1956 in Dartmouth College as an area of knowledge dedicated to the study of algorithms and techniques to confer human-like intelligence to computational units. This section is dedicated to an overview of AI-based techniques to solve the Job-Shop Scheduling Problem.

#### 2.2.4.1 Constraint Satisfaction

Constraint Satisfaction methods use constraints to reduce the search space and consequently the number of possible solutions, thus significantly reducing the execution time [5].

Mark Fox, in his 1983 PhD thesis [46], describes how constraint satisfaction can be used to solve the Job-Shop Scheduling Problem, discussing the possible ways to represent knowledge, select constraints and manage conflicts between constraints. Another contribution of Fox's thesis is the Intelligent Scheduling and Information System (ISIS), a knowledge system to support his solution, on top of which the Opportunistic Intelligent Scheduler (OPIS) was built [47].

#### 2.2.4.2 Neural Networks

Neural Networks (NN) are akin to biological neural systems and are a collection of interconnected units or nodes which connect to each other, transmitting signals to other nodes.

The application of Neural Networks to the JSP was first proposed by Foo and Takefuji [48] in 1988 who presented a modified Tank and Hopfield neural network running an integer linear programming model which minimizes the sum of the start time of each task. An example of such a network can be found in Figure 2.5. Further research in 1991 by Zhou, Cherkassky, Baldwin and Olson, [49] used Hopfield networks and a modified cost function to improve the original solution by Takefuji. This modification involved the use of a linear cost function instead of a quadratic one, which significantly reduces the complexity of the network while increasing the quality of the solutions.

Lately, Neural Networks have been used in hybrid solutions. In 2003, Foo and Takefuji [50] used a NN approach to select dispatch rules. In 2008, Weckman, Ganduri and Koonce [51] used NN alongside genetic algorithms. In 2015, Xanthoupoulos and Koulouriotis [52] used a NN approach to cluster analysis, allowing the selection of dispatch rules. In 2017, Yang and Zhang [53] used NN alongside genetic algorithms to predict job end dates.

### 2.2.5 Local Search Methods

Local Search methods rely on the iterative improvement of a solution in order to find the optimal solution. Iterative improvement works by introducing small changes to the solution and reevaluating it, discarding bad solutions, which significantly decreases the size of the search space. However, these methods are prone to getting stuck in a local optimum.

#### 2.2.5.1 Beam Search

Beam Search (BS) is a technique for searching decision trees, similar to Branch and Bound but it only expands the most promising k nodes, making it faster but less accurate than Branch and Bound. Filtered Beam Search (FBS), a common variant of Beam Search, uses a beam of size b and a filter of size f. It selects the f best nodes and then selects the b best nodes for expansion. Figure 2.6 shows a graph representing a beam search using a filter of size 6 and a beam of size 3.

Research by Ow and Morton [54] introduces a FBS method for the single machine problem with weighted and non-weighted tardiness, comparing their results with other heuristics and search methods. In 2008, Shi-jin, Bing-hai and Li-feng [55] combined Beam Search with Dispatch Rules.

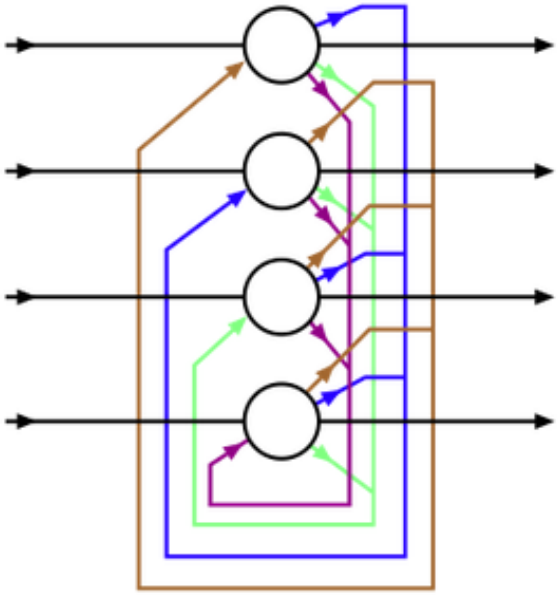

Figure 2.5: Hopfield network with 4 units

In 2015, Birgin, Ferreira and Ronconi [56] used Beam Search to solve a scenario of the JSP involving sequence flexibility.

#### 2.2.5.2 Simulated Annealing

Simulated Annealing was first presented by Kirkpatrick, Gelatt and Vecchi [57] in 1983 and was inspired by the process of a solid annealing from its maximum energy state to its minimum energy state gradually. The energy state also works as a measure of how likely it is for the algorithm to move to a worse solution, which allows it to escape local optimums [58].

In 1989, Osman and Potts [59] applied Simulated Annealing to solve the permutation flow-shop scheduling. In this scenario, jobs are processed in the same order in all steps of production. In 1992, Laarhoven, Aarts and Lenstra [60], using an approach based on Simulated Annealing, achieved high quality solutions at the expense of longer running times.

More recently, in 2005, Xia and Wu [61] devised a method based on Simulated Annealing and particle swarm optimization to improve search efficiency in a multi-objective flexible job shop scheduling scenario. In 2014, Zorin and Kostenko [62] used Simulated Annealing to solve an instance of the JSP with unknown task duration. In 2016, Harmanani and Ghosn [63] introduced a solution for the non-preemptive open JSP problem using Simulated Annealing.

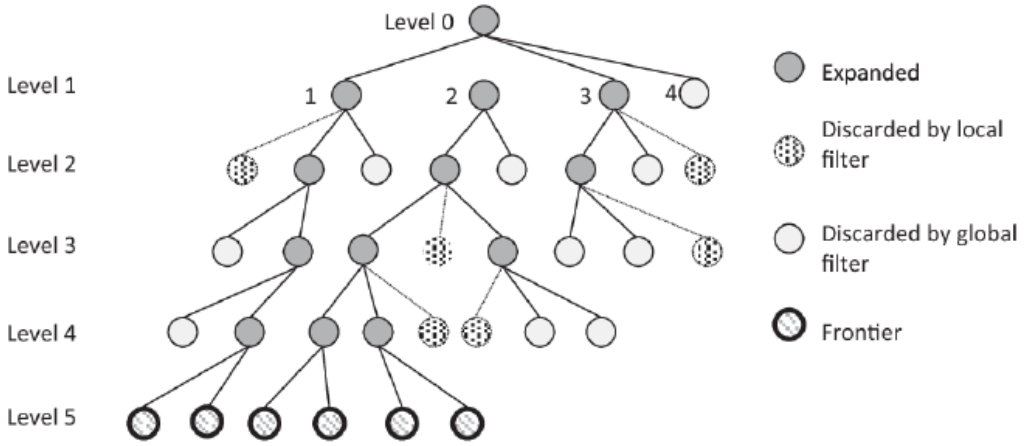


Figure 2.6: Beam Search using a filter of size 6 and a beam of size 3

### 2.2.6 Meta-heuristics

#### 2.2.6.1 Tabu Search

Tabu Search is similar to the previous local search methods, except for the fact that it stores the search space that was already explored. TS-based methods block moves to states that have been recently visited but allows moves to those same states once a pre-determined number of iterations has passed. That number is known in the literature as tabu tenure.

Initial research into Tabu Search and its use to solve the JSP was conducted by Glover [64] in 1986. Further research by Meeran and Morshed [65] combined Genetic Algorithms and Tabu Search to solve the JSP by leveraging Tabu Search's parallel search and the Genetic Algorithm's ability to avoid local optima. In 2015, Peng et al. [66] used Tabu Search and path relinking to build a path connecting the initial solutions to the optimized solution and aid in the process of choosing candidate solutions, which improved the quality of the output.

#### 2.2.6.2 Genetic Algorithms

Genetic Algorithms, are an instance of Evolutionary Algorithms, and are based on the biological process of evolution. These algorithms were originally proposed by John Holland [67] in the 1970s.

Such type of algorithm starts by creating a population of candidate solutions or individuals, the size of which is highly dependent on the type of problem being modelled. Afterwards, an iterative process is performed, simulating the biological process of evolution. In fact, in each iteration or generation, the fittest individuals are selected from the current population, according to a fitness function, such as the value of the objective function, and each individual's genome is modified, either through recombination - or crossover - with a different individual or through random mutation. The new generation of candidate solutions will then be used in the next iteration of the algorithm. Finally, the algorithm terminates when either a pre-defined number of iterations or an

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

Figure 2.7: A genetic algorithm [8]

acceptable level of fitness has been reached for the population. Figure 2.7 shows the pseudocode of a basic Genetic Algorithm.

Genetic Algorithms were initially used to solve the JSP by Davis [68] and, since then, many researchers have successfully combined it with other meta-heuristics to achieve high-quality solutions in a reasonable amount of time. The efficiency of these algorithms depends on the structure of the information and on the quality of the initial population. In 1991, Falkenauer and Bouffouix [69] improved Davis's solution by encoding each machine's operations as a string of symbols. In 2015, Kuczapski et al. [70] studied the effects of using dispatch rules to generate the initial population for the Genetic Algorithm. In the same year, Jalilvand-Nejad and Fattahi [71] combined a Genetic Algorithm with an integer linear programming model to improve a benchmark solution based on Simulated Annealing. In 2017, Zhang et al. [72] used a Genetic Algorithm to solve the multi-objective Flexible Job-Shop Scheduling Problem.

In 1995, Bierwirth [73] introduced a crossover operator called Generalized Order Crossover (GOX), where the offspring inherits a heuristically chosen portion of the chromosome of each parent, so that the same amount of information is incorporated into the new chromosome. This technique yielded results that amounted to, on average, between 94.8% and 99.3% of the best known solution to problems in the Muth and Thompson benchmark.

#### 2.2.6.3 Ant Colony Optimization

Ant Colony Optimization (ACO) is based on the process of ant colony foraging, where ants find the best paths from their colonies to necessary resources by leaving a trail of pheromones. Pheromones wear off after some time, so the path with the highest concentration of pheromones will be the

best path. ACO can be parallelized, which allows the usage of multiple computational units and a significant speedup in processing time [74].

ACO was originally proposed by Colorni, Dorigo and Maniezzo [74], who used it to solve the Traveling Salesman Problem, a NP-Hard problem. Further researchers have found that ACO can be used to solve the JSP and that combining it with other algorithms, such as Tabu Search [75] and Beam Search [76] significantly increases the efficiency of the algorithm and the quality of its solutions. ACO has also been used in conjunction with a knowledge system by Xing et al. [77], immunity algorithms by Xue et al. [78] and two-generation pareto by Zhao et al. [79]. Finally, in 2017, Huang and Yu proposed a solution for dynamic JSP.

## 2.3 Job-Shop Scheduling in the Industry

A 2019 Master's dissertation by Eduardo Leite introduces a "decentralized, multi-agent based, service-oriented solution to the scheduling problem" [4], which is described as a "multi-objective, multi-dimension environment". This solution is capable of reducing the response time to customer orders and requirements due to a more efficient use of resources in the shop floor. In fact, this decreased response time, and overall lower execution time, is the result of breaking down the problem into smaller parts which are solved by different computational agents running a modified Genetic Algorithm in parallel. Finally, the main goal of this dissertation is to minimize total setup time, average tardiness and the makespan or a combination of the three optimization criteria [4].

Despite all these contributions, the approach presented in the dissertation does not take into account the integration of resources such as employees and tools. Besides, although it provides some insight into the rescheduling problem in a decentralized context, it does not differentiate non-working times from maintenance, which are both times when some machines may not be available but represent different events in the shop floor and require different strategies to handle efficiently.

## 2.4 A Genetic Algorithm Approach to Job-Shop Scheduling

Genetic Algorithms, in general, are more complex than what was presented in section 2.2.6.2. Indeed, a robust implementation of a GA must not only feature good selection and crossover operators but also a representation that effectively encodes the target problem, which is challenging as there is no universally favourable representation or operators for any problem.

In 1993, Fang et al [9] proposed a novel approach to solve the JSP based on a Genetic Algorithm, which is capable of producing good results in a short amount of time when run against a benchmark designed by Muth and Thompson [80].

In fact, the proposed solution relies on a modified representation of the Traveling Salesman Problem originally introduced by [81], which encodes a JSP scenario as a sequence of $j \times m$ chunks, where $j$ corresponds to the number of jobs and $m$, to the number of machines. According to this encoding, the sequence *abc* has the following meaning: "put the first untackled task of the

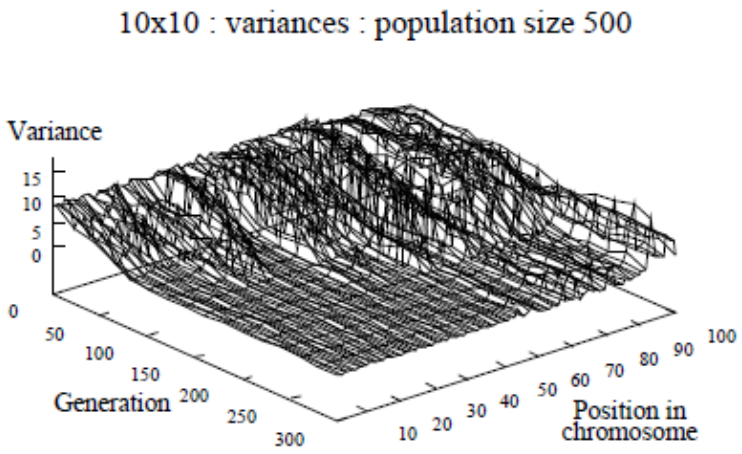


Figure 2.8: Plot of variance of chunks of the genome with time, and with genome position, on the $10 \times 10$ JSSP [9]

*a*-th uncompleted job into the earliest place where it will fit in the developing scheduling, then put the first untackled task of the *b*-th uncompleted job into the earliest place where it will fit in the developing schedule, and so on" [9], which guarantees that all possible encodings represent legal schedules.

Using this representation, the authors found that as the number of iterations of the GA increases, the variance of the early positions of the genome in the gene pool - or the diversity of genes at early positions of the genome - decreases very quickly, suggesting premature convergence, although the variance of later positions stays high for longer, as shown in figure 2.8. In fact, the authors suggest that promoting mutation in fast-converging areas and crossover in slow-converging areas improves the quality of the results. Such strategy, a form of Gene Variance-based Operator Targeting, gives results almost as good as those given by techniques based on branch and bound, when run against a benchmark proposed by Muth and Thompson [80].

Although this strategy only gives results marginally better than previous GA-based methods, it is much more straightforward to apply and presents a much smaller computational complexity than other strategies. The authors also compared the performance of the GVOT-based crossover operator with traditional ones such as fixed one-point crossover with fixed mutation rate per chromosome, fixed one-point crossover with increasing mutation probability and uniform crossover operators [9]. In fact, the Gene Variance-based Operator Targeting strategy resulted in schedules with the least makespan as can be seen in figure 2.9.

Besides, in case of rescheduling due to a change in the processing time of a task or a change in the start time of a task, this representation enables the scheduler to only take into consideration the affected tasks and keep all other genes in the genome unchanged.

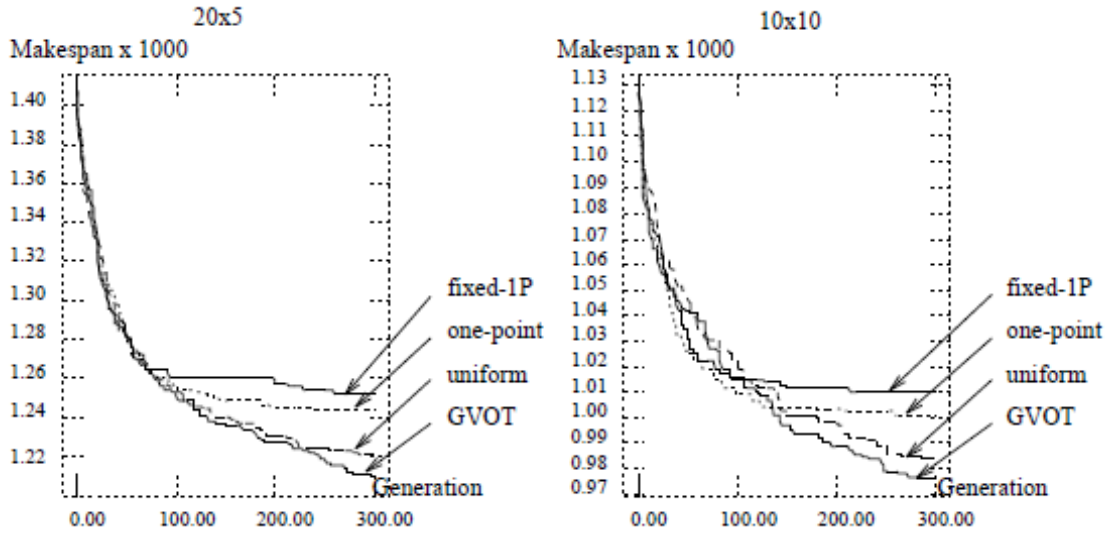


Figure 2.9: Relative performance of different crossover operators on the $10 \times 10$ and $20 \times 5$ [9]

## 2.5 Critical Overview

The present chapter shows an overview of the context of this dissertation when it comes to the concepts related to the Fourth Industrial Revolution which justify and make it possible to use an automated scheduling system in a shop floor.

Besides, the chapter presents the multitude of algorithms which have been used to solve the Job-Shop Scheduling Problem. As could be seen, most exact methods either take too long to find a solution in relatively small scheduling scenarios or cannot find solutions of acceptable quality in a reasonable time. For this reason, researchers have turned to AI, local search and meta-heuristics based methods which make no guarantee of finding the optimum solution but can closely approximate it, while taking little time to execute.

This dissertation focuses on the algorithms which have been found to be the most promising meta-heuristics when applied to the JSP: Tabu Search, Simulated Annealing and Genetic Algorithm.

# Chapter 3

# Problem Definition

This chapter includes two sections. On the one hand, the first section presents a formal definition for the Job-Shop Scheduling Problem, including an analysis of the multitude of constraints imposed on the generic formulation of the problem, so as to better model real-world scheduling scenarios. On the other hand, the second section focuses on the definition and description of the type of scheduling problems that the solution proposed in this dissertation addresses.

## 3.1 Job-Shop Scheduling Problem

A JSP is generically defined as an optimization problem where a set of $n$ jobs $J = \{J_1, J_2, ..., J_n\}$ of varying processing time must be allocated to a set of $m$ machines $M = \{M_1, M_2, ..., M_m\}$ with varying processing power. The resulting schedule minimizes a cost function $C$, usually representative of the total makespan of the schedule. Apart from that, other possible objective functions include the tardiness and earliness of a job as well as the total throughput or the number of jobs that complete before their deadline. Furthermore, each job represents a sequence of operations. The set $O = \{O_{11}, O_{12}, ..., O_{nm}\}$ represents the union set of the operations of all jobs, where $n$ represents the index of the job in set $J$ and $m$ represents the index of the machine in set $M$ that must process it. The completion time of operation $O_{nm}$ can be given by $F_{nm}$. In addition, let $F_{ji}$ represent the completion time of the last operation to finish in the entire schedule and $d_{ji}$ denote the processing time of operation $O_{ji}$, that is, the last scheduled operation. Given these definitions, the end time of an operation subtracted by its processing time must be greater or equal than the end time of the previous operation.

Formally, and taking into account the definitions above [82], the problem can be defined as follows:

$$min\, F_{ji}$$

Subject to:

$$F_{kl} \leq F_{ji} - d_{ji}$$

$$F_{ji} \geq 0$$

The following sections present the multitude of constraints that researchers and real-world manufactories have imposed on the generic definition of the JSP, in order to model shop-floor conditions as faithfully as possible.

### 3.1.1 Constraints on the number of machines and resources

The simplest formulation of the JSP features a single machine and a set of jobs with a well-defined execution time and due date. In this scenario, the function to be optimized usually represents either the total tardiness or the total earliness of the resulting schedule, given that the total makespan is constant regardless of the ordering of jobs and that setup times are irrelevant. Mathematically, tardiness and earliness of a job are respectively defined by the following expressions: $max(0, C_i - d_i)$ and $max(0, d_i - C_i)$ where $C_i$ represents the completion time of job $i$ and $d_i$ represents the deadline for job $i$. However, not only is there no efficient rule capable of optimizing this problem but the number of possible states also grows at a rate of $n!$ with $n$ corresponding to the number of jobs. Table 3.1 represents an example input for a scenario with only one machine and a list of jobs with a well-defined processing time (in minutes) and due date representing the number of minutes elapsed since the beginning of the first job.

Furthermore, adding machines to a JSP scenario does not increase the complexity of the problem but significantly widens the search space. Indeed, the total number of assignments of machines to jobs, assuming that a job can be assigned to any machine, is given by the expression $n! \times m^n$, where $n$ represents the number of jobs and $m$, the number of machines. Such a problem where a job can be assigned to any machine is known in the literature as the Flexible Job-Shop Scheduling Problem. In addition, there are scenarios where each job is divided into as many operations as there are available machines and where the $i$-th operation of a job must be processed on the $i$-th machine. Such a scenario is known as the Flow-Shop Scheduling Problem.

| **Job** | **Processing Time** | **Due Date** |
|---|---|---|
| Job 1 | 40 | 60 |
| Job 2 | 20 | 80 |
| Job 3 | 30 | 130 |

Table 3.1: Job information with processing times and due dates

Nonetheless, in real-world scenarios each machine can only perform a specific kind of operation to an object. In fact, it is usual to attach a list of provided services to each machine or resource, so as to create a better model of shop floor conditions.

### 3.1.2 Constraints on machine availability and capacity

In most theoretical formulations of the Job-Shop Scheduling Problem, machines are assumed to be always available for job processing. However, this is not the case in real-world scenarios as

machines can stop for long periods of time for a variety of reasons - non-working times, scheduled or unscheduled maintenance and unexpected machine failures. Because of this, the delay between operations of the same job must be taken into account when finding a makespan minimizing schedule by biasing the scheduler towards choosing more reliable machines. Furthermore, it is often too expensive to keep machines idling for long periods of time, so a decent scheduler must not only minimize the idle time of a machine but also the delay between the execution of the operations of a job.

Incidentally, non-working times differ from maintenance downtime due to their periodic nature, which makes them a lot more predictable and easier to incorporate into a developing schedule. In fact, non-working times are well-defined and known before a schedule is made, while maintenance time can be unexpected in case it is caused by an abrupt failure in a machine. Besides, although no machine may start processing an operation while in a non-working time, some machines allow operations to end during this period.

In addition, even though the concept of job corresponds to an abstraction of a task which must be processed on some machine, in reality a job is associated with a physical material with shape and volume. In fact, machines in a shop floor have a limit to the amount of unfinished goods that they can process. In such environments, the total execution time of an operation on a machine is the result of the sum of a fixed time interval and a variable part which is proportional to the quantity of unfinished goods processed by the machine.

### 3.1.3 Constraints on job precedence and priority

Some formulations of the Job-Shop Scheduling Problem, in which a job is represented as a set of operations, allow operations within each job to be processed at an arbitrary order. Such a scenario is known in the literature as the Open-Shop Scheduling Problem. Nevertheless, in most real-world scenarios, a job and the operations in it represent the construction process of a product, meaning that such operations must be completed in a pre-defined order or sequence, which models the transition from raw materials to a finished product.

Furthermore, in more complex scenarios, operations in the same job may be done in parallel and may even follow alternative paths in the production flow. Indeed, in order to model such scenarios, an adequate formulation of the problem must take into account the multitude of machines and resources that a material can be processed on.

Finally, machines may need some time to transition between jobs, which is known as the setup time. In fact, in real-world scenarios, the setup time of a machine may depend on the type and characteristics of the previous machine in the production line, which requires the creation of a setup-matrix representing these times. Besides, jobs may be assigned a priority depending on the goals of the manufacturing operation. All of these requirements must be taken into account when developing a scheduler in order to make it capable of modelling real-world shop floor conditions.

## 3.2 CMF Job-Shop Scheduling

The sections above have presented a generic formulation for the Job-Shop Scheduling Problem and the multitude of modifications - constraints and restrictions - which have been imposed on it by researchers and manufacturing businesses to better model complex shop floor environments.

The problem addressed by this dissertation is yet another variant of the general Job-Shop Scheduling Problem. In fact, in an instance of this variant, there are multiple jobs composed of an arbitrary number of operations which must be processed on a machine that provides a specific service. There is also a number of machines accompanied by a list of the services each of them provide. Besides, each operation in a job must be done sequentially and determines the amount of unfinished goods that should be processed by a machine. In addition, each machine has a fixed cycle time (FCT), a variable cycle time (VCT) and a setup time (ST). The execution time of an operation is given by the following formula:

$$ExecutionTime = FCT + (VCT * OperationQuantity)$$

Meanwhile, the total time for an operation is given by the formula:

$$TotalTime = SetupTime + ExecutionTime$$

However, this formulation assumes that the setup time of a machine is constant regardless of the machine that previously processed an operation and that a machine has infinite capacity and is always available to process a job except when it is already processing one. In addition, the criteria to optimize correspond to a weighted combination of the makespan and the total setup time of the resources allocated to steps in the schedule.

Furthermore, this variant uses different labels for the various concepts related to the Job-Shop Scheduling Problem. For example, a job is referred to as a flow which is divided into a variable number of operations called steps which represent the production process of a material.

# Chapter 4

# Proposed Solution

The present chapter describes the implementation of the proposed solution, which addresses the Job-Shop Scheduling Problem described in section 3.2, and presents how it interacts with the Critical Manufacturing MES. In addition, the chapter introduces, in detail, the meta-heuristics that were implemented.

## 4.1 System Architecture

### 4.1.1 Overview

The proposed solution is a scheduling engine available as a module for the Critical Manufacturing MES and accessible through a web server that receives scheduling requests from an MES in an appropriate format, representing a scheduling scenario. The scheduling engine then executes one of the three implemented meta-heuristics - Tabu Search, Simulated Annealing, Genetic Search - minimizing a weighted average of the makespan and the resource setup time. The scheduling engine, in turn, after executing the implemented meta-heuristic algorithm, returns the solved schedule scenario which contains information regarding the start times, end times and selected resource for all operations.

Furthermore, as mentioned in Chapter 3, the Job-Shop Scheduling Problem requires the definition of multiple entities in order to model a shop floor scenario. In fact, each entity - Flow, Schedule, Resource and Context - are all represented as JSON objects which are then concatenated to form a scheduling scenario. Table 4.1 shows the properties of the Flow object, which represents a job to which a material is assigned.

As mentioned above, a flow has a name and a material assigned to it as well as a collection of steps. Table 4.2 shows the properties of a Step object, which represents an atomic operation in a job or flow. In turn, a step requires a service which must be performed by a resource that can execute it. Table 4.3 shows the properties of a Resource object, which defines the characteristics of a machine on the shop floor. Finally, Table 4.4 shows the properties of a Context object, which is responsible for encoding other types of crucial information for the scheduler.

| Parameter | Type | Description |
|---|---|---|
| name | string | The flow's unique identifier. |
| material | string | The unique identifier of the material which is going through the present production flow. |
| steps | array | Represents an array of Step objects. The order in which they appear in the array is assumed to be the order in which they must be processed. |

Table 4.1: Flow schema

| Parameter | Type | Description |
|---|---|---|
| name | string | The step's unique identifier. |
| required_service | string | The unique identifier of the step's required service. |
| op_state | enum | Represents the current status of a step. Valid values include NOT_DONE, DONE and IN_PROGRESS. |
| op_start | timestamp | The timestamp corresponding to the start time of the step. |
| op_end | timestamp | The timestamp corresponding to the start time of the step. |
| selected_resource | string | The unique identifier of the resource was picked to perform a step. |
| quantity | int | The amount of unfinished goods that must be processed in the step. |
| scheduling_direction | char | For internal use of the Critical Manufacturing MES. Always set to "F". Unused by the proposed solution. |
| operationID | long | For internal use of the Critical Manufacturing MES. Unused by the proposed solution. |

Table 4.2: Step schema

| Parameter | Type | Description |
|---|---|---|
| name | string | The resource's unique identifier. |
| fixed_cycle_time | int | The amount of time, in minutes, that corresponds to the fixed part of the execution time formula found in section 3.2. |
| variable_cycle_time | int | The amount of time, in minutes, that corresponds to the variable part of the execution time formula found in section 3.2. |
| setup_time | int | The amount of time, in minutes, that a resource takes to setup. Corresponds to the setup time part of the execution time formula found in section 3.2. |
| services | array | An array of the unique identifiers of the services provided by the current resource. |

Table 4.3: Resource schema

| Parameter | Type | Description |
|---|---|---|
| current_date | string | A timestamp corresponding to the start of the desired schedule. |
| planning_horizon | int | The amount of time, in minutes, that must be scheduled. No job may start after the planning horizon but may end after this period. |
| timeroud | int | Rounds all times to the closest multiple of this value. |
| down_resources | array | An array of the unique identifiers of the resources in the shop floor which are not working. The scheduler attempts to schedule all steps while rejecting allocations of any of the resources. |
| optimization_criteria | array | An array of objects representing an optimization criteria. Each object contains a name and a weight. By default the only possible names are "Execution Time" and "Setup Time". Besides, the total sum of the weight parameter on all optimization criteria must be equal to 100%. |

Table 4.4: Context schema

Once the program starts, an HTTP server is initialized and listens for scheduling requests on a predefined port. In every request or response, the information in the schemas presented is concatenated and encoded in JavaScript Object Notation (JSON) format and inserted in the body

| Status Code | Message | Description |
|---|---|---|
| 200 | OK | The schedule scenario sent in the body is valid and was successfully used to solve a scheduling problem. The response contains the solved scheduling scenario. |
| 400 | Bad Request | The request was not understood by the server possibly due to an error when parsing the schedule scenario file or when the requested endpoint is not available. The response contains information about the exception thrown during the processing of the request. |

Table 4.5: Possible HTTP responses to a scheduling request

of such event. Table 4.5 shows the possible HTTP status codes which can be returned for a given scheduling request.

### 4.1.2 Integration with the Critical Manufacturing MES

The scheduling engine presented in this chapter, named APSRescheduling, was designed to be used as a module for the Critical Manufacturing MES. Figure 4.1 shows the modules responsible for processing a scheduling request both on the MES and on the proposed scheduler and illustrates the communication between such modules.

The module named MESHTML corresponds to the graphical user interface of the Critical Manufacturing MES and is where a user can press a button which starts the scheduling request process. The MESHTML module, in turn communicates with the HostService, which processes requests for MES operations, and sends it the identifier of the scheduling scenario to solve. Because the requested operation corresponds to a scheduling request, this identifier is passed to the CallSchedulingEngineAPI module, which in turn calls the CallSchedulingEngine module. The latter module is responsible for generating the JSON representing the scheduling scenario and sending it to the web server which allows access to the APSRescheduling scheduling engine.

Once a schedule is found, the web server in the APSRescheduling module responds with the schedule scenario in the format described in Section 4.1.1. The information follows the path described in the previous paragraph in reverse until it reaches the MESHTML module which decodes the solved schedule scenario and presents the user a Gantt chart of the schedule found by the scheduling engine.

Furthermore, the purpose of the proposed solution is to replace a scheduling engine developed by INESCTEC for Critical Manufacturing which is available as a module for the Critical Manufacturing MES and implements a greedy strategy to solve the Job-Shop Scheduling Problem. Similarly to APSRescheduling, the INESCTEC scheduler is available through a web server, however it communicates with the MES using files in the Extensible Markup Language (XML) format, which is generated by proprietary software, AltovaXML, and is not easily understood by humans. Indeed, the reason for choosing JSON over XML is due to the fact that not only is JSON

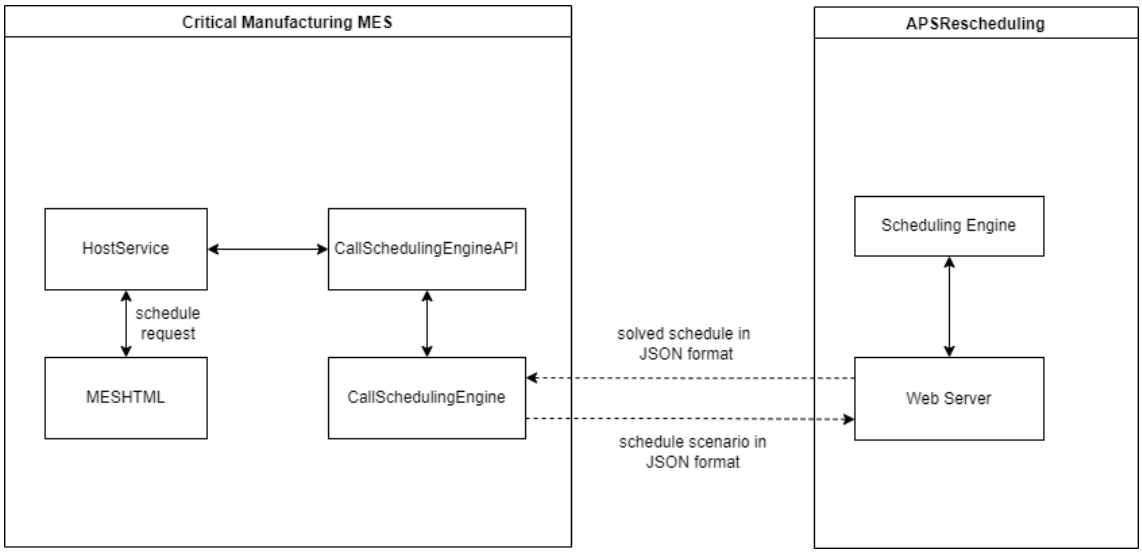


Figure 4.1: Proposed System Architecture and Communication with the Critical Manufacturing MES

human-readable but it is also free to use, easy to parse and enables the representation of simple data structures such as objects and arrays.

## 4.2 System Implementation

This section describes in detail the implementation of the scheduler. In fact, the scheduler is capable of solving an instance of the Job-Shop Scheduling Problem using one of the following meta-heuristics at a time: Tabu Search, Simulated Annealing and Genetic Search. Tabu Search and Simulated Annealing are both instances of local search meta-heuristics which follow a similar structure, involving the creation of an initial state and of neighbour states. Section 4.2.1 goes into detail about the creation of an initial state from a schedule scenario while Section 4.2.2 describes the strategy used by the scheduler to generate neighbour states.

Besides, because the goal of the scheduler is to minimize a weighted combination of the makespan and the setup time, the solution also defines a fitness function, which is presented in detail in Section 4.2.4.

Finally, Section 4.2.3 presents the implementation of the genetic operators for the Genetic Search meta-heuristic in detail, given that this algorithm belongs to the class of evolutionary algorithms and follows a different approach to optimize the problem.

### 4.2.1 Initial State Generation

As was seen in Chapter 2, the initial state is key to the effectiveness of a meta-heuristic. For this reason, the scheduler implements a strategy for generating the initial state of a scheduling scenario. This strategy is an example of a dispatch rule which can be formulated in the following way: for

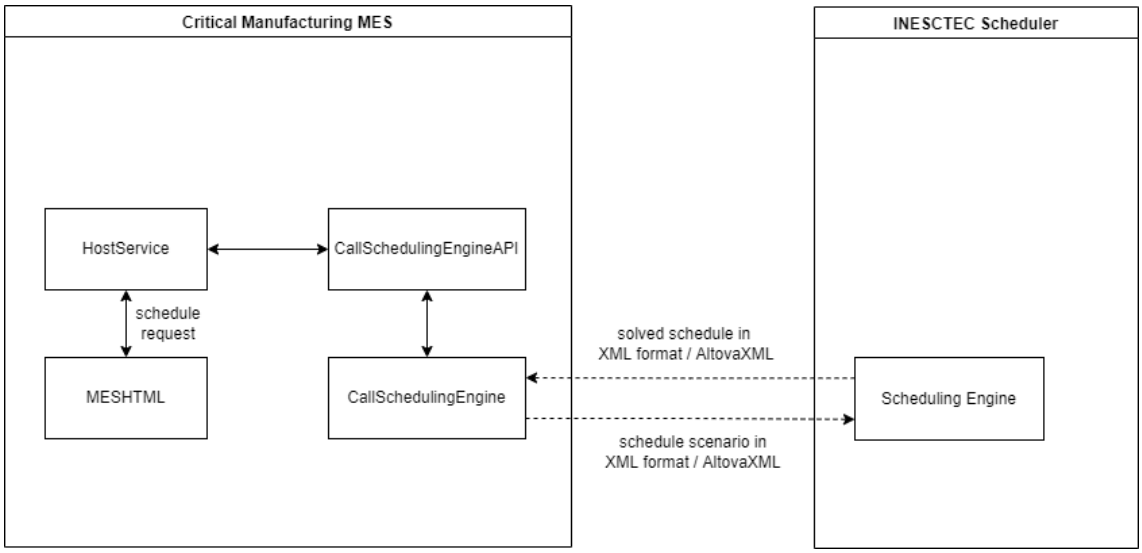


Figure 4.2: Architecture of the Original Critical Manufacturing MES Scheduling System

each step that must be scheduled or rescheduled, randomly choose a resource that provides the service required by the step and allocate it to the step, inserting the latter in the first possible time when the resource is available.

The scheduler takes into account that steps must be completed sequentially so, for any step and resource, the first possible allocation time must come after the end of the previous step.

### 4.2.2 Neighbour State Generation

Another key part of a scheduler which relies on meta-heuristics to solve an optimization problem is the strategy to generate neighbour states. Indeed, on every iteration of the Tabu Search and the Simulated Annealing algorithms, the scheduler generates a pre-determined number of neighbour states from a state previously identified as the best so far. Such neighbours are then scored by a fitness function and the best one is stored and used in the following iterations. For more information about the fitness function see Section 4.2.4.

Furthermore, a neighbour state is generated by randomly picking a step in the schedule and then change the resource allocated to it to a different randomly chosen resource that provides the service required by the step. This requires the scheduler to recalculate the start and end time of all steps after the step that was modified, given that changing the resource that was allocated to a step may change its start and end time and even duration. In fact, this may happen if the new resource has a different fixed or variable cycle time or setup time. Figure 4.2 shows the current architecture of the existing solution implemented in the Critical Manufacturing MES.

### 4.2.3 Genetic Search Implementation

Genetic Search is a type of Evolutionary Algorithm and simulates the biological process of evolution, thus defining genetic operators such as crossover, mutation and selection. The implementation of this meta-heuristic is based on the approach described in Section 2.4. In fact, the schedule is represented as a sequence of strings where the $i$-th element in it represents the unique identifier of the resource that has been allocated to the $i$-th step in the schedule. Such a sequence, in the context of evolutionary algorithms, is known as a chromosome and each of its elements is known as a gene. With such a representation, all chromosomes represent valid schedules given that the scheduler is guaranteed to not insert a gene which refers to a resource that does not provide the service required by the step in question. Furthermore, building a schedule from this representation is straightforward and is based on the dispatch rule presented in Section 4.2.1: for every gene $i$ in the chromosome, allocate the $i$-th step to the specified resource and insert a task in the growing schedule in the first possible time when the resource is available.

Unlike local search meta-heuristics, the Genetic Search algorithm calls for an initial population of chromosomes rather than a single initial state. However, such population, of a pre-determined size, is generated by repeatedly calling the initial state generation procedure described in Section 4.2.1. This scheduler was designed for scheduling scenarios where a service is provided by multiple resources and, for this reason, if the number of resources is large enough the population will always have chromosomes which are distinct from each other.

The algorithm selects a pair of chromosomes for selection using a roulette-wheel selection strategy, also known as fitness proportionate selection, which assigns each chromosome the likelihood of being chosen, which is proportional to the fitness of the chromosome. This guarantees that the selection function does not keep selecting the same chromosomes for crossover.

Another aspect of the Genetic Search approach present in this dissertation is that the scheduling engine computes the uniqueness of each gene, taking into account all the chromosomes in the population. The purpose of this calculation is to simulate the behaviour of the Gene Variance-based Operator Targeting crossover and mutation operators described in Section 2.4. The uniqueness of a gene is ill-defined in the literature. However, given that each gene represents a resource, it is possible to use the setup time of the resource and its execution time as a measurement of the uniqueness of a gene. Because of this, the scheduler uses the following sum which represents the inverse of this measure:

$$\sigma^2_{executionTime} + \sigma^2_{setupTime} + BlauIndex * \overline{executionTime} + 1$$

the purpose of such a sum is to prevent two resources with the same execution time and setup time to have the same uniqueness value as they may represent different resource types on the shop floor. The BlauIndex parameter is a measure of the diversity of the information encoded in each gene and is mathematically defined as the probability that two genes randomly selected from a group belong to different categories and is given by the following expression:

$$\sum \frac{1}{f^2}$$

where $f$ represents the number of times a gene occurs in the population.

Furthermore, genes with a high value of the measure defined above are selected for mutation while genes with a low value are selected for crossover. The selected gene is computed using a roulette-wheel selection strategy. Finally, the algorithm relies on the use of an elite of size 3 where the 3 best chromosomes in the population are kept in the gene pool every generation.

In conclusion, this implementation of Genetic Search is similar to the one described in Section 2.4, which, according to the original authors, found good makespan minimizing schedules.

### 4.2.4 Fitness Function

In addition to the information presented in Table 4.2, it must be noted that the solution presented in this chapter is not just designed to deal with scheduling scenarios where no operation has been scheduled yet. Indeed, the proposed scheduler is capable of processing scenarios where some resources have already been allocated to a step, in which case the parameters concerned with the start and end time of a step will already be filled in. In such a situation, which requires the scheduler to reschedule some steps, the scheduler will attempt to keep the allocations for steps occurring very close to the start time of the first step to be scheduled, which is sent by the MES, and only change allocations which occur later.

Besides, given that meta-heuristics only approximate the optimum of a function and make no guarantees about actually finding it, the scheduler computes the fitness of a solution according to a scoring function. Assuming a scheduling scenario has a set of flows $F = \{f_1, f_2 ... f_n\}$, the number of steps in each flow $f$ is given by $n\_steps(f)$ and the set of ordered steps in each flow $f$ is $S_f = \{s_{f_1}, s_{f_2} ... s_{f_{n\_steps(f)}}\}$. For this reason, the non-idle time $NIT$ of the schedule, that is the total amount of time in which some resource is not idle, is given by the following formula:

$$NIT = \sum_{f \in F} \sum_{s \in S_f} TotalTime(s) = \sum_{f \in F} \sum_{s \in S_f} (startTime(s) - endTime(s))$$

And the total delay $TD$ between steps in all flows is given by the following formula.

$$TD = \sum_{f \in F} \sum_{i=1}^{n\_steps(f)-1} (startTime(s_{f_{i+1}}) - endTime(s_{f_i}))$$

This represents a period of time in the execution of the flow where no progress is being made in the construction process of the material.

In order to compute the total running time $TRT$ of the schedule, the non-idle time value is added to the total delay, giving the expression:

$$TRT = NIT + TD$$

Furthermore, in order to address the rescheduling problem, the scheduling engine relies on a penalty function to penalize solution states in which the resource that was previously allocated to a step is modified. The penalty is given by the following formula:

$$PenaltyFunction(dateTime) = frozenPeriod \times \frac{60}{(dateTime - currentTime) + 1}$$

where the $frozenPeriod$ represents the number of minutes during which the scheduler must prevent the reallocation of a different resource to a step, $dateTime$ depicts the start time of the step to be rescheduled and $currentTime$ represents the time the schedule was generated. The Critical Manufacturing MES also features a penalty function that is used in a similar situation and the concept of "frozen period" is similar to the one presented in this dissertation. However, its scheduler will not use the frozen period after the period has passed and the APSRescheduling engine will take into account steps much later in time because its penalty function is much smoother rather than step-wise.

As described in Table 4.4, the scheduler allows the definition of a weight to each optimization criteria - $w_e$ for execution time and $w_s$ for setup time - the sum of which must be equal to 1. STD indicates the setup time delay defined to be one third of the previously computed task delay TD:

$$STD = \frac{TD}{3}$$

Using the definitions provided above, the fitness function is given by the following formula:

$$Fitness = PenaltyFunction + TRT + w_s \times (STD + SetupTime) + w_e \times (TD + ExecutionTime)$$

The incorporation of such concepts in the fitness function procedure aids the scheduler in preferring solution states which not only minimize the total running time of the schedule but also the delay between tasks of the same flow. The combination of these factors works as a proxy for the makespan.

# Chapter 5

# Experiments and Analysis of Results

The present chapter describes the multiple test scenarios which were created to verify the proposed scheduler's performance in multiple contexts with a variable number of jobs and resources.

## 5.1 Test Scenario Description

The test scenarios were created with the aid of Python 3's implementation of the Mersenne Twister pseudo-random number generator [83] found in package *random*. The random number generator was used to generate a multitude of numerical values:

- The number of services that a resource can provide.
- The fixed cycle time and setup time of a resource.
- The service required by a step.

The procedure to generate test scenarios attempts to guarantee that there are at least two resources capable of providing a given service. However, in rare cases, this may not happen due to the random nature of the generator. The proposed scheduler can handle these scenarios by refusing to schedule steps whose required service is not provided by any resource, as well as steps coming after that one in the same flow.

Several tests were created and each of them was run on each of the implemented meta-heuristics. The basis for comparison is a greedy scheduler developed by INESCTEC and incorporated as a module on the Critical Manufacturing MES. This scheduler builds a schedule by inserting a step in the first available interval of time of the first resource in the resource list that provides the service required by the step. The created test scenarios have the following characteristics regarding the number of jobs and the number of resources:

- 5 flows and 10 resources.
- 10 flows and 15 resources.
- 15 flows and 20 resources.

The number of provided services per resource is variable but averages 3 per resource.

In these scenarios, the number of steps in a job is equal to the number of jobs. Table 5.1 shows the parameters used in each meta-heuristic.

| Meta-Heuristic Name | Number of Iterations | Other parameters |
|---|---|---|
| Simulated Annealing | 10.000 | Not applicable |
| Tabu Search | 10.000 | Uses a tabu tenure of 5. Meaning that every solution state inserted into the tabu list is removed after 5 iterations. |
| Genetic Search | 50 | Uses an initial population size of 10, a mutation probability of 0.01 and an elite size of 3. |

Table 5.1: Parameters for each of the implemented meta-heuristics

For the purpose of these test scenarios, the variable cycle time was set to 0 and thus the quantity parameter in each step is irrelevant to compute the total execution time of a step. This was done to improve the quality of the comparison of results versus an existing scheduler which does not minimize total execution time but rather the total fixed cycle time.

## 5.2 Analysis of Results

### 5.2.1 Total Scheduling

This subsection describes and presents an analysis of results for total scheduling scenarios, in which no step has previously been allocated to a resource. In Scenarios 1 to 3, the goal is the minimization of the makespan.

#### 5.2.1.1 Scenario 1

Scenario 1 corresponds to a small scenario where there are 5 flows, each with 5 steps with a total of 25 schedulable steps, and 10 resources. Table 5.2 shows the makespan of the best schedule found by the INESCTEC greedy scheduler and the three meta-heuristics implemented in the proposed solution where it can be seen that any of the metaheuristics produces better results than a greedy approach.

| Meta-Heuristic Name | Makespan |
|---|---|
| INESCTEC | 300 minutes |
| Simulated Annealing | 188 minutes |
| Tabu Search | 201 minutes |
| Genetic Search | 219 minutes |

Table 5.2: Makespan for the three implemented meta-heuristics

Furthermore, Figure 5.1 shows the Gantt chart for the schedule found by the INESCTEC greedy scheduler while Figure 5.2 shows a similar chart for the schedule found by the solver presented in this dissertation when running a Simulated Annealing meta-heuristic. The figures show the tasks that have been assigned to each resource. Because the APSRescheduling solver is capable of seeing the total makespan of the schedule as the schedule grows, it also features an improved capability of picking the resource which is best at minimizing the makespan as this measure is taken into account in all iterations of the algorithm. Indeed, by preferring a different resource, the scheduler can obtain a significant improvement in makespan when compared to the greedy scheduler. In addition, given the architecture of the system it is difficult to obtain an accurate measurement of the execution of each of the schedulers so this measurement was disregarded, as they are similar for this scenario. Nonetheless, the makespan of the schedule found by the meta-heuristics is on average 32.5% shorter than that of the solution found by the greedy scheduler.

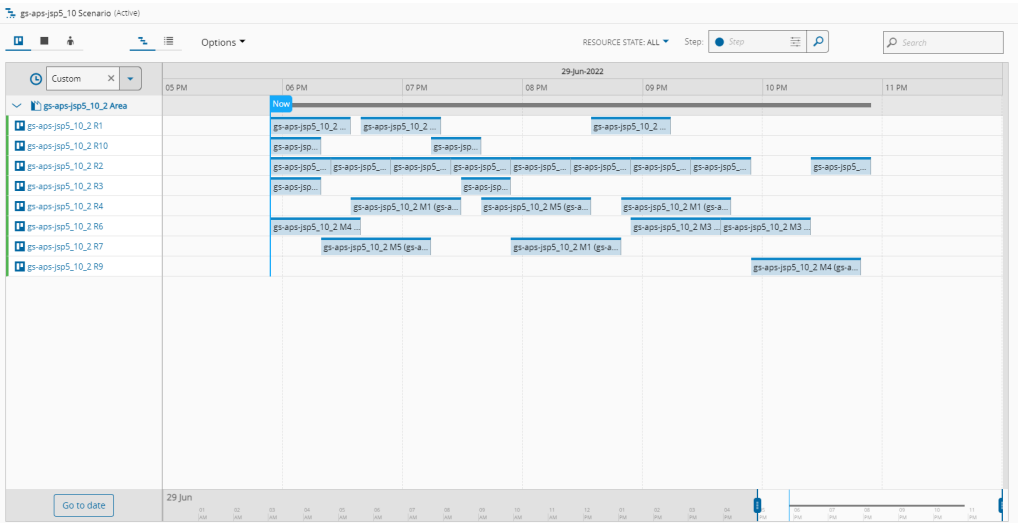


Figure 5.1: Gantt Chart for the schedule found by the INESCTEC solver on a $5 \times 10$ scenario

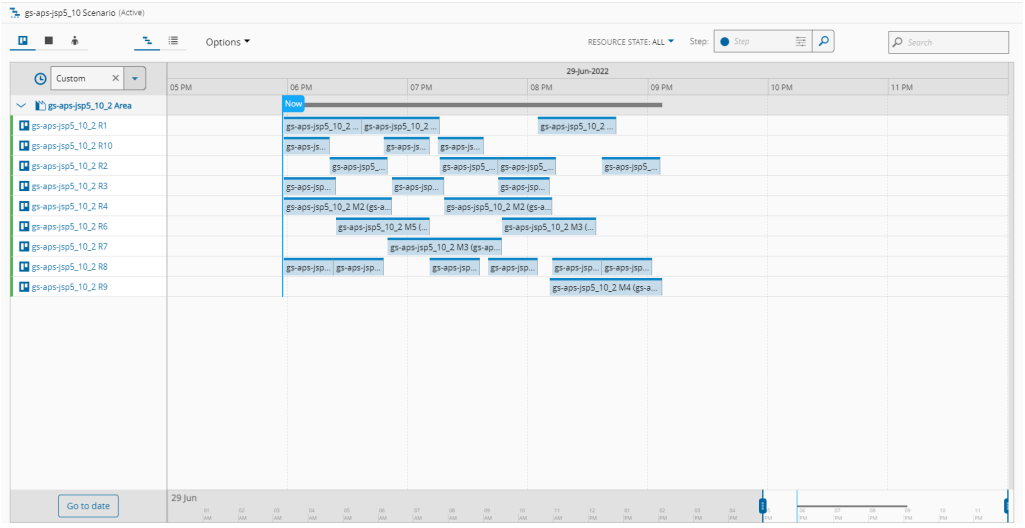


Figure 5.2: Gantt Chart for the schedule found by the APSRescheduling solver for a $5 \times 10$ scheduling problem and running the Simulated Annealing algorithm

Finally, Table 5.3 shows the execution time of each meta-heuristic. The Simulated Annealing meta-heuristic was the fastest of the run while Tabu Search was much slower for a comparable number of iterations possibly due to the computation of the tabu list. Besides, even though the Genetic Search algorithm took as much time to finish as Tabu search, the solution is significantly worse, having a longer makespan than the other algorithms.

| Algorithm/Solver Name | Execution time |
|---|---|
| Simulated Annealing | 0.48 seconds |
| Tabu Search | 15.17 seconds |
| Genetic Search | 16.1 seconds |

Table 5.3: Execution times for the three implemented meta-heuristics

#### 5.2.1.2 Scenario 2

Scenario 2 corresponds to a medium-sized scenario where there are 10 flows, each with 10 steps with a total of 100 schedulable steps, and 15 resources. Table 5.4 shows the makespan of the best schedule found by the INESCTEC greedy scheduler and the three implemented meta-heuristics.

| Algorithm/Solver Name | Makespan |
|---|---|
| INESCTEC | 690 minutes |
| Simulated Annealing | 465 minutes |
| Tabu Search | 468 minutes |
| Genetic Search | 814 minutes |

Table 5.4: Makespan for the three implemented meta-heuristics

In addition, Figure 5.3 shows the Gantt chart for the schedule found by the INESCTEC greedy scheduler while Figure 5.4 shows a similar chart for the schedule found by the solver presented in this dissertation when running a Tabu Search meta-heuristic. The figures show results relatively similar to those found for Scenario 1 given that the Tabu Search and Simulated Annealing meta-heuristics have successfully found a better makespan minimizing schedule than a greedy scheduler. Indeed, the reason for such a difference is similar: the scheduler, by having a wider view of the growing schedule, can pick the best resources at minimizing the makespan. Furthermore, the makespan of the schedule found by the meta-heuristics is on average 33% shorter than that of the solution found by the greedy scheduler.

However, the Genetic Search algorithm is significantly slower than Tabu Search and the Simulated Annealing ones due to the fact that the implementation of this algorithm must compute the uniqueness of the resources in the gene pool, which is computationally intensive. Besides, this algorithm was unable to find a solution with a lower makespan than a greedy scheduler. Even if the number of iterations was increased, the execution time of this algorithm is already very high, which implies that the approach is not adequate for this problem.

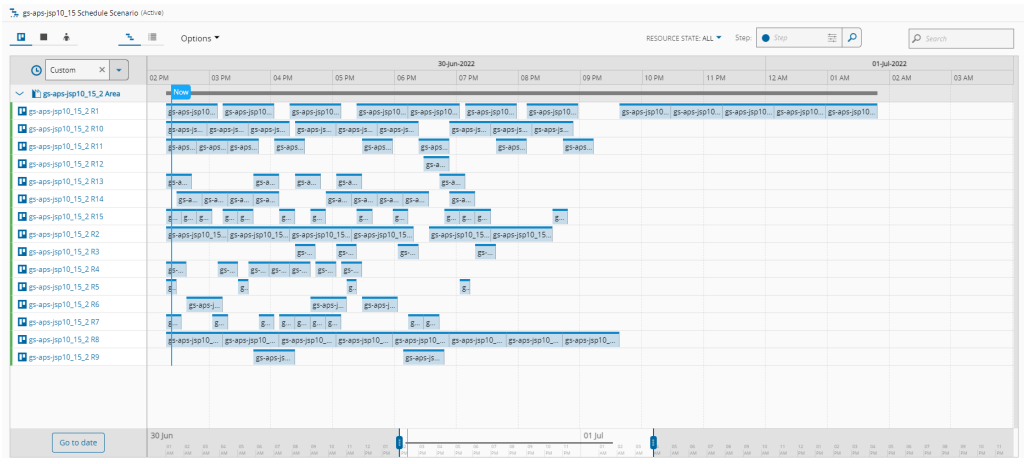

Figure 5.3: Gantt Chart for the schedule found by the INESCTEC solver on a $10 \times 15$ scenario

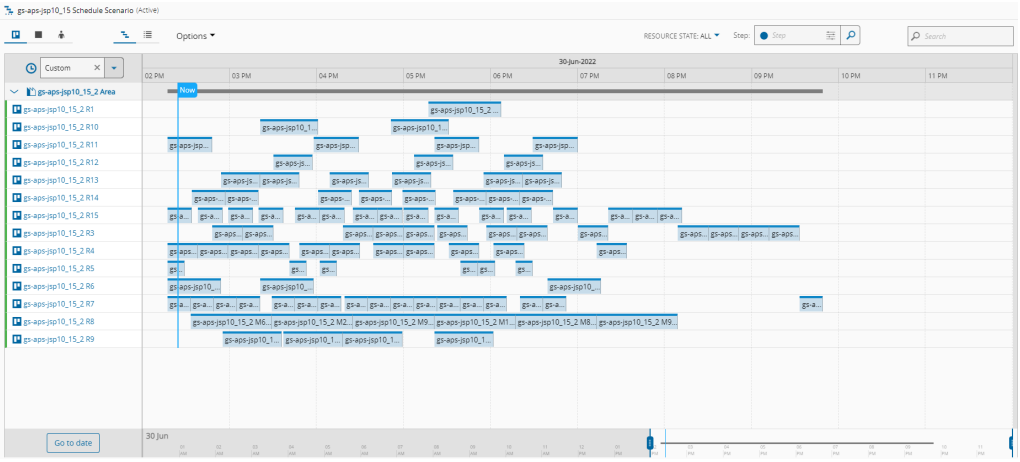

Figure 5.4: Gantt Chart for the schedule found by the APSRescheduling solver for a $10 \times 15$ scheduling problem and running the Tabu Search algorithm

Finally, Table 5.5 shows the execution time of each meta-heuristic. Like in Schenario 1, the Simulated Annealing meta-heuristic was the fastest of the run while Tabu Search was much slower for a comparable number of iterations. Besides, even though the Genetic Search algorithm took less time to finish as Tabu search, the solution is significantly worse, having a longer makespan than the other algorithms, including a greedy strategy. In fact, judging by the number of generations of the Genetic Search algorithm and its execution time, it is highly likely that in order to reach the same makespan as the Simulated Annealing algorithm, this approach would have to run for several minutes.

| Algorithm/Solver Name | Execution time |
|---|---|
| Simulated Annealing | 1.55 seconds |
| Tabu Search | 53.06 seconds |
| Genetic Search | 32.43 seconds |

Table 5.5: Execution times for the three implemented meta-heuristics

#### 5.2.1.3 Scenario 3

Scenario 3 corresponds to a large scenario where there are 15 flows, each with 15 steps for a total of 225 schedulable steps, and 20 resources. Table 5.6 shows the makespan of the best schedule found by the INESCTEC greedy scheduler and the three meta-heuristics implemented in the proposed solution where it can be seen that any of the metaheuristics produces better results than a greedy approach.

| Algorithm/Solver Name | Makespan |
|---|---|
| INESCTEC | 790 minutes |
| Simulated Annealing | 498 minutes |
| Tabu Search | 540 minutes |
| Genetic Search | 728 minutes |

Table 5.6: Makespan for the three implemented meta-heuristics

Figure 5.5 shows the Gantt chart for the schedule found by the INESCTEC greedy scheduler on the perspective of the material rather than resource. The results are in line with the ones achieved by the Simulated Annealing and Tabu Search meta-heuristic in previous scenarios. Furthermore, the makespan of the schedule found by the meta-heuristics is on average 35% shorter than that of the solution found by the greedy scheduler.

The Genetic Search algorithm, on the other hand, is significantly slower than Tabu Search and the Simulated Annealing just as in Scenario 2.

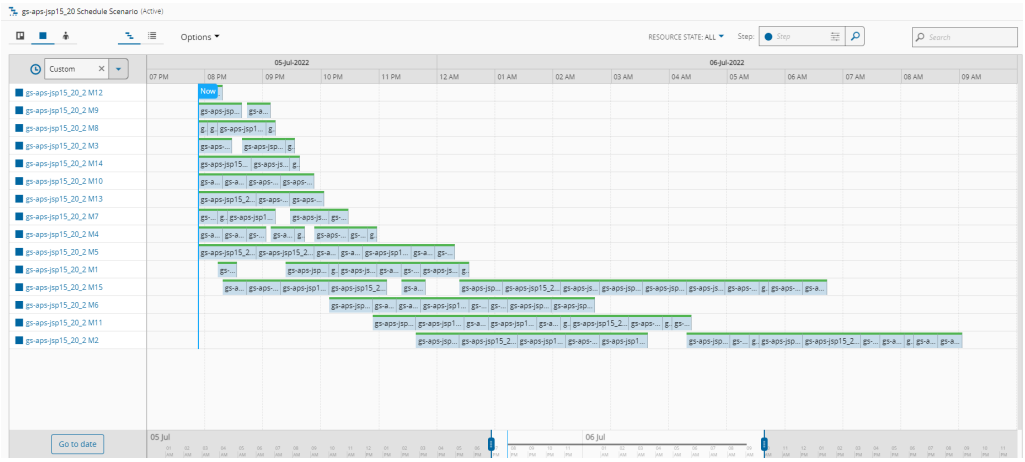


Figure 5.5: Gantt Chart (material view) for the schedule found by the INESCTEC solver on a $15 \times 20$ scenario

Finally, Table 5.7 shows the execution time of each meta-heuristic. Like the previous scenarios, the Simulated Annealing meta-heuristic was still the fastest of the run while Tabu Search was much slower for a comparable number of iterations possibly due to the computation of the tabu list. The conclusions are similar to the other scenarios given that even though the Genetic Search algorithm took as much time to finish as Tabu search, the solution is significantly worse, having a longer makespan than the other algorithms.

| Algorithm/Solver Name | Execution time |
|---|---|
| Simulated Annealing | 1.57 seconds |
| Tabu Search | 56.03 seconds |
| Genetic Search | 32.65 seconds |

Table 5.7: Execution times for the three implemented meta-heuristics

### 5.2.2 Conclusions

As was seen in the test scenarios and analysis done above, the Tabu Search and Simulated Annealing meta-heuristics have better results than the INESCTEC greedy scheduler. On average, the meta-heuristics mentioned gave results 33% better than those achieved by a greedy scheduler. However, the Genetic Search algorithm takes a very long time to converge and its performance is highly dependant on the fitness of the initial population as theorized in many articles presented in Chapter 2, which means that this approach is inadequate to solve this problem. Besides, in some situations, this meta-heuristic gives worse results than a greedy approach. Tabu Search does not have a low execution time either, averaging a 41 second execution time. Finally, Simulated Annealing was the fastest meta-heuristic of the three and presents the best execution time to makespan ratio.

# Chapter 6

# Conclusions and Future Work

## 6.1 Main Contributions

The Job-shop Scheduling problem is very complex and has many possible formulations to fit most scenarios in manufacturing. A significant amount of work has already been done in an attempt to solve it, using the multitude of techniques presented in this document, each with their own advantages and disadvantages, with no single technique being suitable for every formulation of the JSP.

This dissertation and the solution included in it attempt to approach an instance of the Job-Shop Scheduling Problem which is not usually seen in the literature but represents an important, although simplified, model of shop floor conditions common in the semiconductor and electronics industry. In addition, this dissertation shows that meta-heuristics based approaches such as Tabu Search and Simulated Annealing present an improvement in the ability to minimize the makespan as well as the total setup time of a schedule. Nonetheless, even though the implementation of Genetic Search is based on a novel approach to evolutionary algorithms with good results, it did not result in an improvement in makespan or setup time minimization in the context of the problem presented in this dissertation.

In conclusion, this dissertation contributes with the following:

- A definition of the problem as a multi-objective environment where there are multiple rather than a single machine which can be allocated to an operation.
- An application of concepts from the scientific literature to build a Genetic Search approach to the problem described in the dissertation.
- A presentation of an approach for dealing with the necessity to compute schedules after the dispatch of a previous schedule to machines on the shop floor.
- A comparison of the performance of multiple meta-heuristics regarding the minimization of the makespan and setup times compared to a greedy strategy, thus corroborating the effectiveness and usefulness of such techniques in the context of Job-Shop Scheduling.

## 6.2 Future Work

The solution presented in this dissertation aims to be as complete as possible. However, some improvements to the solution are possible, for instance:

- The representation of non-machine resources such as employees and tools in the model of the scheduler.
- The representation of other constraints such as certifications and scheduled resource down-time in the model of the scheduler.
- The addition of the possibility to schedule flows with alternate paths.
- An investigation on how different genetic encodings, or representations, of the problem and genetic operators such as crossover, mutation and selection can improve the performance of the Genetic Search meta-heuristic, not only in terms of execution time but also when it comes to the effectiveness of the minimization goal.
- An analysis of the effectiveness of meta-heuristics such as Beam Search and Ant Colony Optimization, and how they compare to the meta-heuristics implemented in this dissertation.

# References


[1] Yue Wang, J Yan, and Mitchell Miendger Tseng. An auction based negotiation protocol for resource allocation in customized housing construction. *Procedia CIRP*, 28:161–166, 2015.

[2] Thomas Edson Espindola Goncalo and Danielle Costa Morais. Agent-based negotiation protocol for selecting transportation providers in a retail company. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pages 263–267. IEEE, 2015.

[3] Nelson Rodrigues, Eugenio Oliveira, and Paulo Leitão. Decentralized and on-the-fly agent-based service reconfiguration in manufacturing systems. *Computers in Industry*, 101:81–90, 2018.

[4] Eduardo Miguel Bastos Leite. Industry 4.0-shop-floor negotiation, 2019.

[5] Jian Zhang, Guofu Ding, Yisheng Zou, Shengfeng Qin, and Jianlin Fu. Review of job shop scheduling research and its new perspectives under industry 4.0. *Journal of Intelligent Manufacturing*, 30(4):1809–1830, 2019.

[6] ELE Times Bureau. Consumer iot to be bigger market than industrial iot., 2017. Last accessed 27 June 2022.

[7] Critical Manufacturing. Erp vs. mes: Still a standoff?, 2020. Last accessed 27 June 2022.

[8] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[9] Hsiao-Lan Fang, Peter Ross, and David Corne. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. 1993.

[10] Klaus Schwab. *The fourth industrial revolution*. Currency, 2017.

[11] Guoping Li, Yun Hou, and Aizhi Wu. Fourth industrial revolution: technological drivers, impacts and coping methods. *Chinese Geographical Science*, 27(4):626–637, 2017.

[12] Duarte Alemão, André Dionisio Rocha, and José Barata. Smart manufacturing scheduling approaches—systematic review and future directions. *Applied Sciences*, 11(5):2186, 2021.

[13] Holger Kinzel. Industry 4.0–where does this leave the human factor? *Journal of Urban Culture Research*, 15:70–83, 2017.

[14] Chen Kan, Hui Yang, and Soundar Kumara. Parallel computing and network analytics for fast industrial internet-of-things (iiot) machine information processing and condition monitoring. *Journal of manufacturing systems*, 46:282–293, 2018.

[15] Yuqian Lu, Xun Xu, and Lihui Wang. Smart manufacturing process and system automation–a critical review of the standards and envisioned scenarios. *Journal of Manufacturing Systems*, 56:312–325, 2020.

[16] Smart Manufacturing Leadership Coalition. Implementing 21st century smart manufacturing. In *Workshop summary report*, pages 1–36, 2011.

[17] Stephanie S Shipp, Nayanee Gupta, Bhavya Lal, Justin A Scott, Christopher L Weber, Michael S Finnin, Meredith Blake, Sherrica Newsome, and Samuel Thomas. Emerging global trends in advanced manufacturing. Technical report, INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA, 2012.

[18] E Wallace and F Riddick. Panel on enabling smart manufacturing. *State College, USA*, 2013.

[19] Weitian Wang, Rui Li, Yi Chen, Z. Max Diekel, and Yunyi Jia. Facilitating human–robot collaborative tasks by teaching-learning-collaboration from human demonstrations. *IEEE Transactions on Automation Science and Engineering*, 16(2):640–653, 2019.

[20] Yongkui Liu, Lihui Wang, Xi Vincent Wang, Xun Xu, and Lin Zhang. Scheduling in cloud manufacturing: state-of-the-art and research challenges. *International Journal of Production Research*, 57(15-16):4854–4879, 2019.

[21] Banu Çaliş and Serol Bulkan. A research survey: review of ai solution strategies of job shop scheduling problem. *Journal of Intelligent Manufacturing*, 26(5):961–973, 2015.

[22] CFO. Plucking more profit from production with mes, 1999. Last accessed 27 June 2022.

[23] Critical Manufacturing. What is an mes, 2016. Last accessed 27 June 2022.

[24] Sunil Luthra and Sachin Kumar Mangla. Evaluating challenges to industry 4.0 initiatives for supply chain sustainability in emerging economies. *Process Safety and Environmental Protection*, 117:168–179, 2018.

[25] Hendrik Sebastian Birkel and Evi Hartmann. Impact of iot challenges and risks for scm. *Supply Chain Management: An International Journal*, 2019.

[26] Selmer Martin Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1):61–68, 1954.

[27] Sheldon B Akers Jr. A graphical approach to production scheduling problems. *Operations Research*, 4(2):244–245, 1956.

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

[29] N Hefetz and I Adiri. An efficient optimal algorithm for the two-machines unit-time jobshop schedule-length problem. *Mathematics of Operations Research*, 7(3):354–360, 1982.

[30] Teofilo Gonzalez and Sartaj Sahni. Flowshop and jobshop schedules: complexity and approximation. *Operations research*, 26(1):36–52, 1978.

[31] Harvey M Wagner. An integer linear-programming model for machine scheduling. *Naval research logistics quarterly*, 6(2):131–140, 1959.

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

[33] George H Brooks. An algorithm for finding optimal or near optimal solutions to the production scheduling problem. *The Journal of Industrial Engineering*, 16(1):34–40, 1969.

[34] ZA 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.

[35] Chandrasekharan Rajendran and Oliver Holthaus. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European journal of operational research*, 116(1):156–170, 1999.

[36] Frank Werner and Andreas Winkler. Insertion techniques for the heuristic solution of the job shop problem. *Discrete applied mathematics*, 58(2):191–211, 1995.

[37] Yuri N Sotskov, Thomas Tautenhahn, and Frank Werner. On the application of insertion techniques for job shop problems with setup times. *RAIRO-Operations Research-Recherche Opérationnelle*, 33(2):209–245, 1999.

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

[39] Azzedine Bekkar, Ghalem Belalem, and Bouziane Beldjilali. Iterated greedy insertion approaches for the flexible job shop scheduling problem with transportation times constraint. *International Journal of Manufacturing Research*, 14(1):43–66, 2019.

[40] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3):391–401, 1988.

[41] S Dauzere-Peres and J-B Lasserre. A modified shifting bottleneck procedure for job-shop scheduling. *The international journal of production research*, 31(4):923–932, 1993.

[42] Egon Balas, Jan Karel Lenstra, and Alkis Vazacopoulos. The one-machine problem with delayed precedence constraints and its use in job shop scheduling. *Management Science*, 41(1):94–109, 1995.

[43] Qiao Zhang, Hervé Manier, and Marie-Ange Manier. A modified shifting bottleneck heuristic and disjunctive graph for job shop scheduling problems with transportation constraints. *International Journal of Production Research*, 52(4):985–1002, 2014.

[44] B. H. Zhou and T. Peng. Modified shifting bottleneck heuristic for scheduling problems of large-scale job shops. *Journal of Donghua University (English Edition)*, 2016.

[45] Rafael Mellado Silva, Claudio Cubillos, and Daniel Cabrera Paniagua. A constructive heuristic for solving the job-shop scheduling problem. *IEEE Latin America Transactions*, 14(6):2758–2763, 2016.

[46] Mark S Fox. Constraint-directed search: A case study of job-shop scheduling. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1983.

[47] Stephen F Smith, Mark S Fox, and Peng Si Ow. Constructing and maintaining detailed production plans: Investigations into the development of kb factory scheduling. *AI magazine*, 7(4):45–45, 1986.

[48] Foo Yoon-Pin Simon et al. Integer linear programming neural networks for job-shop scheduling. In *IEEE 1988 International Conference on Neural Networks*, pages 341–348. IEEE, 1988.

[49] Deming N Zhou, Vladimir Cherkassky, TR Baldwin, and DE Olson. A neural network approach to job-shop scheduling. *IEEE Transactions on Neural Networks*, 2(1):175–179, 1991.

[50] Foo Yoon-Pin Simon et al. Stochastic neural networks for solving job-shop scheduling. i. problem representation. In *IEEE 1988 International Conference on Neural Networks*, pages 275–282. IEEE, 1988.

[51] Gary R. Weckman, Chandrasekhar V. Ganduri, and David A. Koonce. A neural network job-shop scheduler. *Journal of Intelligent Manufacturing*, 19(2):191–201, 2008.

[52] A. S. Xanthopoulos and D. E. Koulouriotis. Cluster analysis and neural network-based metamodeling of priority rules for dynamic sequencing. *Journal of Intelligent Manufacturing*, 29(1):69–91, 2018.

[53] Donghai Yang and Xiaodan Zhang. A hybrid approach for due date assignment in a dynamic job shop. In *2017 9th International Conference on Modelling, Identification and Control (ICMIC)*, pages 793–798, 2017.

[54] Peng Si Ow and Thomas E. Morton. Filtered beam search in scheduling†. *International Journal of Production Research*, 26(1):35–62, 1988.

[55] Wang Shi-jin, Xi Li-feng, and Zhou Bing-hai. Filtered-beam-search-based algorithm for dynamic rescheduling in fms. *Robotics and Computer-Integrated Manufacturing*, 23(4):457–468, 2007.

[56] E.G. Birgin, J.E. Ferreira, and D.P. Ronconi. List scheduling and beam search methods for the flexible job shop scheduling problem with sequencing flexibility. *European Journal of Operational Research*, 247(2):421–440, 2015.

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

[58] Peter J. M. van Laarhoven and Emile H. L. Aarts. *Simulated annealing*, pages 7–15. Springer Netherlands, Dordrecht, 1987.

[59] IH Osman and CN Potts. Simulated annealing for permutation flow-shop scheduling. *Omega*, 17(6):551–557, 1989.

[60] Peter J. M. van Laarhoven, Emile H. L. Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.

[61] Weijun Xia and Zhiming Wu. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers  Industrial Engineering*, 48(2):409–425, 2005.

[62] Daniil A. Zorin and Valery A. Kostenko. Simulated annealing algorithm for job shop scheduling on reliable real-time systems. In Eric Pinson, Fernando Valente, and Begoña Vitoriano, editors, *Operations Research and Enterprise Systems*, pages 31–46, Cham, 2015. Springer International Publishing.

[63] Haidar M. Harmanani and Steve Bou Ghosn. An efficient method for the open-shop scheduling problem using simulated annealing. In Shahram Latifi, editor, *Information Technology: New Generations*, pages 1183–1193, Cham, 2016. Springer International Publishing.

[64] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers  Operations Research*, 13(5):533–549, 1986. Applications of Integer Programming.

[65] S. Meeran and M. S. Morshed. A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study. *Journal of Intelligent Manufacturing*, 23(4):1063–1078, 2012.

[66] Bo Peng, Zhipeng Lü, and T.C.E. Cheng. A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers  Operations Research*, 53:154–164, 2015.

[67] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[68] Lawrence Davis. Job shop scheduling with genetic algorithms. volume 140, 1985.

[69] Emanuel Falkenauer, S Bouffouix, et al. A genetic algorithm for job shop. In *ICRA*, pages 824–829. Citeseer, 1991.

[70] Artur M Kuczapski, Mihai V Micea, Laurentiu A Maniu, and Vladimir I Cretu. Efficient generation of near optimal initial populations to enhance genetic algorithms for job-shop scheduling. *Information Technology and Control*, 39(1), 2010.

[71] Amir Jalilvand-Nejad and Parviz Fattahi. A mathematical model and genetic algorithm to cyclic flexible job shop scheduling problem. *Journal of Intelligent Manufacturing*, 26(6):1085–1098, 2015.

[72] W Zhang, JB Wen, YC Zhu, and Y Hu. Multi-objective scheduling simulation of flexible job-shop based on multi-population genetic algorithm. *International Journal of Simulation Modelling*, 16(2):313–321, 2017.

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

[74] Alberto Colorni, Marco Dorigo, Vittorio Maniezzo, et al. Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life*, volume 142, pages 134–142. Paris, France, 1991.

[75] Kuo-Ling Huang and Ching-Jong Liao. Ant colony optimization combined with taboo search for the job shop scheduling problem. *Computers & operations research*, 35(4):1030–1046, 2008.

[76] Christian Blum. Beam-aco hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591, 2005.

[77] Li-Ning Xing, Ying-Wu Chen, Peng Wang, Qing-Song Zhao, and Jian Xiong. A knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing*, 10(3):888–896, 2010.

[78] P. Zhang H. Xue and S.Wei. Appplying hybrid algorithm of immunity and ant colony in job-shop scheduling. In *Proceedings of the 2014 international conference on industrial engineering and manufacturing technology*, volume 4, page 91. Shanghai, China, 2015.

[79] Boxuan Zhao, Jianmin Gao, Kun Chen, and Ke Guo. Two-generation pareto ant colony algorithm for multi-objective job shop scheduling problem with alternative process plans and unrelated parallel machines. *Journal of Intelligent Manufacturing*, 29(1):93–108, 2018.

[80] John F Muth and Gerald Luther Thompson. *Industrial scheduling*. Prentice-Hall, 1963.

[81] John Grefenstette, Rajeev Gopal, Brian Rosmaita, and Dirk Van Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, volume 160, pages 160–168. Lawrence Erlbaum, 1985.

[82] Jorge Magalhaes-Mendes. A comparative study of crossover operators for genetic algorithms to solve the job shop scheduling problem. *WSEAS transactions on computers*, 12(4):164–173, 2013.

[83] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.