Vrije Universiteit Brussel

Faculty of Science and Bio-Engineering Sciences
Department of Computer Science
Computational Modeling Lab

A Generic Multi-Agent Reinforcement Learning Approach for Scheduling Problems

Yailen Martínez Jiménez

Dissertation Submitted for the degree of Doctor of Philosophy in Sciences

Supervisors: Prof. Dr. Ann Nowé
Prof. Dr. Rafael Bello

Print: Silhouet, Maldegem

2012 Uitgeverij VUBPRESS Brussels University Press
VUBPRESS is an imprint of ASP nv (Academic and Scientific Publishers nv)
Ravensteingalerij 28
B-1000 Brussels
Tel. +32 (0)2 289 26 50
Fax +32 (0)2 289 26 59
E-mail: info@vubpress.be
www.vubpress.be

ISBN 978 90 5718 220 4
NUR 984
Legal deposit D/2012/11.161/156

To my family and my friends
A mi familia y mis amigos

Committee members:

Supervisors:

Prof. Dr. Ann Nowé
Vrije Universiteit Brussel

Prof. Dr. Rafael Bello Pérez
Universidad Central "Marta Abreu" de Las Villas

Internal members:

Prof. Dr. Viviane Jonckers
Vrije Universiteit Brussel

Prof. Dr. Bernard Manderick
Vrije Universiteit Brussel

Prof. Dr. Bart Jansen
Vrije Universiteit Brussel

External members:

Prof. Dr. Ir. Greet Vanden Berghe
KAHO Sint-Lieven

Prof. Dr. El-Houssaine Aghezzaf
Ghent University

Abstract

Scheduling is a decision making process that is used on a regular basis in many real life situations. It takes care of the allocation of resources to tasks over time, and its goal is to optimize one or more objectives. Depending on the problem being solved, tasks and resources can take many forms, and the objectives can also vary. In this dissertation we are interested in manufacturing scheduling, which is an optimization process that allocates limited manufacturing resources over time among parallel and sequential manufacturing activities. Customer orders have to be executed, and each order is composed by a number of operations that have to be processed on the resources or machines available. Each order can have release and due dates associated, and typical objectives functions involve minimizing the tardiness or the makespan.

In real world scheduling problems the environment is so dynamic that all this information is usually not known beforehand. For example, manufacturing scheduling is subject to constant uncertainty, machines break down, orders take longer than expected, and these unexpected events make the original schedule fail. That is the reason why companies prefer to have robust schedules rather than optimal ones. A key issue is to find a proper balance between these two performance measures.

Scheduling problems can be interpreted as a distributed sequential decision-making task, which makes the use of multi-agent reinforcement learning algorithms possible. These algorithms are of high relevance to various real-world problems, as they allow the agents to learn good overall solutions through repeated interactions with the environment.

Our main contribution is a generic multi-agent reinforcement learning approach that can easily be adapted to different scheduling settings, such as the job shop scheduling with parallel machines, online problems or the hybrid flow shop scheduling. It is possible to increase the robustness of the solutions and to look at different objective functions, like the tardiness or the makespan. Furthermore, the proposed approach allows the user to define certain parameters involved in the solution construction process, in order to define the balance between robustness and optimality.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisors Ann Nowé and Rafael Bello, for accepting me as a PhD student and for all the guidance and the help during the past years.

I would like to thank the members of the examination committee: Bart Jansen, Viviane Jonckers, Bernard Manderick, El-Houssaine Aghezzaf and Greet Vanden Bergue, for finding the time to read this dissertation and also for the insightful comments and suggestions.

The colleagues and friends from the CoMo lab also deserve a big acknowledgement. Thank you, Abdel, Allan, Bert, Cosmin, David C., David S., Joke, Jonatan, Katja, Kevin, Kris, Kristof, Maarten D., Maarten P., Madalina, Marjon, Matteo, Mike, Nyree, Peter, Ruben, Pasquale, Saba, Stijn, Sven, Steven, Tim, Yann-Aël and Yann-Michaël, for making CoMo such a nice working environment. Either talking about research, playing with the wii, ping pong, or enjoying the BBQs, I hope you will manage to prepare the mojitos next year ;-) Also thanks to the colleagues from the ARTI side: Frederik, Lara, Katrien, Pieter and Luc.

During the PhD I had the opportunity to collaborate with great researchers from different institutions: Tony, Katja, Greet, Elsy, Dmitriy, Patrick, Stefan, Tommy and all the colleagues from the DiCoMAS project. Thank you for the opportunity of working together and also for spending time together either attending conferences or summer schools.

Sports were also part of my activities in Brussels, especially badminton and beach volley (when the weather allowed it). This, together with some other special occasions, gave me the opportunity to meet people from other labs and other countries. I would like to thank Elisa, Christophe, Jorge, Engineer, Nicolas, Stefan, Andoni, Lode, Beat, Edgar, Wolf, Theo, Simonne, Anna, Magda, Kasia, Lina, Sebastian, Frank, Anne-Sonia, Kimberly, Eve, Marilyn, Ioana, Virginia and Raül, for making Brussels a more sporty and friendly environment :)

It was thanks to the support of many people that I was able to get used to Belgium and somehow feel like at home. I want to thank Gabriella, Andrea and the whole family in Hungary. Borys, Françoise and also many cubans who live in Belgium and who helped me and supported me in many different ways during my stay in Brussels. Muchísimas gracias a Gilberto, Lellani, Abelito, Olga y familia, Francisco y Paola.

I also had the opportunity to meet wonderful people from the Cuban Embassy in Brussels: Maiza, Eduardo, Yurielkys, Michel, Martica, Michel Rodríguez, Arturo, Elvira, Patricia, Niurka, Puig, Yamile, Jorge, Elio y Gilma, Jorge Alfonso, Mirta y Oriol, Lourdes, Baby, Elizabeth y Alfredo.

A mis colegas y amigos de la Universidad Central "Marta Abreu" de Las Villas, tanto de la Facultad de Matemática, Física y Computación como de otras facultades y departamentos. Igualmente a los amigos y vecinos que son casi parte de la familia. Son muchos para mencionarlos a todos, pero dentro de poco tendré la oportunidad de agradecerles en persona!

I am also grateful to the Vrije Universiteit Brussel for providing the funding for the PhD scholarship and to the VLIR project for the support.

Last but not least, I want to thank my family for their endless support. A toda mi familia un agradecimiento immenso por siempre creer en mi y por apoyarme incondicionalmente. Un beso grande a todos.

Thank you all!!
Muchas Gracias!!

Brussels, October 2012
Yailen

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A schedule is commonly defined as 'a plan for performing work or achieving an objective, specifying the order and allotted time for each part'. With this definition in mind, if we think about scheduling, which is the action of making a schedule, it is easy to notice that it is a type of task that we have all performed at some point, maybe without noticing it. Let us start with a simple example: how many times have you decided beforehand the tasks you want to do in one day? and even the order in which you want to accomplish them? Yes, that is scheduling. And of course, you prepare your schedule with one objective in mind, maybe you have to finish a report by 4pm, which means that you have a deadline to meet, or maybe you just want to go home early, meaning that you want to finish with everything as soon as possible.

But even this simple scenario can get a bit more complicated. For example, when some types of tasks start to repeat, because there are specific things you have to perform every day, then you start to evaluate how good the schedule you made the previous day was (again, maybe without noticing it). You might change the order in which you perform the tasks, because you think there is a slight chance of finishing earlier if you make some changes in your daily actions, which means that you adjust your decisions depending on the outcome you were able to obtain, then we can say that you are learning from experience. Now imagine a system composed by several persons like you, performing tasks in specific orders, but now you have to use common resources to accomplish the tasks. As you can notice, a good coordination is needed in order to make sure that all the tasks are executed and the objective of the whole system is optimized.

Scheduling is a decision making process that is used on a regular basis in every situation where a specific set of tasks has to be performed on a specific set of resources. Some examples are: airport gate scheduling, crew scheduling, processor scheduling and manufacturing scheduling. In this dissertation we will focus on manufacturing scheduling, where the schedule construction process plays an important role, as it can have a major impact on the productivity of the company. In the current competitive environment, effective scheduling has become a necessity for survival in the market place, as companies have to meet the due dates and use the resources in an efficient manner [Pinedo (2008)].

1.1 Manufacturing Scheduling

Manufacturing scheduling is defined as an optimization process that allocates limited manufacturing resources over time among parallel and sequential manufacturing activities. This allocation must obey a set of constraints that reflect the temporal relationships between activities and the capacity limitations of a set of shared resources [Wang & Shen (2007)].

In other words, the problem can be defined as a set of jobs that has to be processed on a set of machines, with the objective of finding the best schedule, that is, an allocation of the jobs to time intervals on the machines that minimizes the chosen objective. Each job consists of a set of operations that have to be scheduled in a predetermined order, and each of these operations needs a certain amount of time (processing time) to be executed. This time depends on the machine where the operation will be processed. The jobs can have release and due dates associated, which means that they are expected to be finished in a specific amount of time.

If we are solving a deterministic scheduling problem, then all the information is assumed to be known beforehand: the number of jobs, the number of machines, the number of operations per job, the processing times of the operations, the precedence constraints and problem constraints (which can change depending on the type of scheduling problem being solved). If, on the other hand, the scheduling problem being solved is stochastic, then the information is not complete, because the processing times of the operations belonging to the different jobs are modeled as random variables. This means that the processing time of an entire job is not known until it is fully executed.

The problems can be classified according to different characteristics, for example, the number of machines (one machine, parallel machines), the job characteristics (preemption allowed or not, equal processing times) and so on. In this dissertation we will assume the classification scheme proposed in [Graham et al. (1979)], where the scheduling problems are specified using a classification in terms of three fields $\alpha|\beta|\gamma$, where α specifies the machine environment, β the operation characteristics, and γ the criteria to optimize.

The combination of all these features makes the scheduling problem a challenging task, that is why it has captured the interest of many researchers from different research communities, for example, Operations Research (OR) and Artificial Intelligence (AI).

1.2 Operations Research and Artificial Intelligence

Manufacturing scheduling problems have been addressed using combinatorial optimization techniques, both approximate methods and exact methods. According to [Billaut et al. (2008)], solving a particular problem may require the use of modeling tools for complex systems (simulation, Petri nets, etc.), thus leading to the definition of matchings between these methods.

Different OR techniques (Linear Programming, Mixed-Integer Programming, etc) have been applied to scheduling problems. These approaches usually involve the definition of a model, which contains an objective function, a set of variables and a set of constraints. OR based techniques have demonstrated the ability to obtain optimal solutions for well-defined problems, but OR solutions are restricted to static models. AI approaches, on the other hand, provide more flexible representations of real-world problems, allowing human expertise to be present in the loop [Gomes (2000)].

The vast majority of the research in scheduling has focused on the development of exact and suboptimal procedures for the generation of a baseline schedule assuming complete information and a deterministic environment [Herroelen & Leus (2005)]. However, the real world is not so stable, projects may be subject to unexpected events during execution, which may lead to numerous schedule disruptions, for example, resources can become unavailable (breakdowns or scheduled maintenances), new orders can arrive, operations could take longer than expected, etc.

One approach to deal with such disruptions is to generate robust schedules, where robustness refers to the performance of an algorithm under uncertainties [Davenport et al. (2001)]. The objective is to generate schedules that are able to absorb some level of uncertainty without having to reschedule. There are some techniques that can deal with such problems, in this dissertation we will consider slack-based techniques, where the main idea is to provide each activity with some extra execution time (slack time) in such a way that part of the uncertainty can be absorbed. The slack time that will be added to each activity or task is proportional to its duration and the resource where it will be processed.

A very important issue is to focus the slack time on areas where it will be really needed. For example, let us assume that a new machine arrives in the system, which can work for long periods before breaking, then it is not wise to make the schedule more robust at the beginning of the scheduling period. The key point is then to identify these areas that are more likely to need slack time in order to deal with a machine breakdown or other unexpected events.

In order to summarize all the features that have been described, Figure 1.1 shows a perspective of integration between AI and OR techniques.

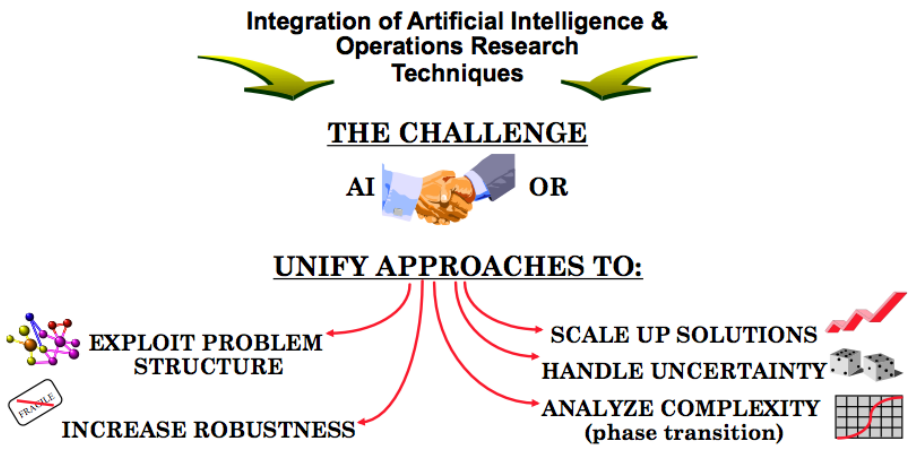

Figure 1.1: Integration of Artificial Intelligence and Operations Research techniques, taken from [Gomes (2000)].

This idea was presented in [Gomes (2000)], where the author explains how OR methods have focused on tractable representations, which have demonstrated the ability to identify optimal and locally optimal solutions, but they are restricted to

rigid models with limited expressive power. AI methods, on the other hand, provide richer and more flexible representations, which can lead to intractable problems. The challenge is to achieve some sort of unification between the approaches in order to come up with good representations of the problem and at the same time deal with uncertainty, scalability issues and increase the robustness.

1.3 Our Approach

Many scheduling problems suggest a natural formulation as distributed decision-making tasks. Hence, the employment of multi-agent systems represents an evident approach. Furthermore, given the well-known inherent intricacy of solving scheduling problems, decentralized approaches for solving them may yield a promising option [Gabel (2009)]. The research presented in this dissertation is based on this idea. We propose a generic multi-agent reinforcement learning approach for scheduling problems, which attempts to solve some of the issues mentioned in the previous section by focusing on finding robust solutions to different scheduling problems using a combination of different techniques, mainly reinforcement learning, optimization, and slack-based approaches.

Reinforcement Learning (RL) is learning what to do (how to map situations to actions) so as to maximize a numerical reward signal [Sutton & Barto (1998)]. It allows an agent to learn optimal behavior through trial-and-error interactions with its environment. By repeatedly trying actions in different situations the agent can discover the consequences of its actions and identify the best action for each situation. For example, when dealing with unexpected events, learning methods can play an important role, as they could 'learn' from previous results and change specific parameters for the next iterations, allowing not only to find good solutions, but more robust ones.

A common approach to solve scheduling problems is to use dispatching priority rules, which work by assigning priorities to the tasks that can be executed on a resource depending on a specific criteria. The learning algorithms could benefit from this idea, by associating the priorities with the feedback signal the agents receive when executing the actions, and this is something that will be studied in this dissertation.

Another problem that has been identified in the scheduling community is the fact that most of the research concentrates on optimization problems that are a simplified version of reality. As the author points out in [Urlings (2010)]: *"this allows for the use of sophisticated approaches and guarantees in many cases that optimal solutions are obtained. However, the exclusion of real-world restrictions harms the applicability of those methods. What the industry needs are systems for optimized production scheduling that adjust exactly to the conditions in the production plant and that generate good solutions in very little time"*.

It is well known that companies prefer to have robust schedules, even though robustness implies extra time in the schedule. The inclusion of extra time means that you lose in optimality, but the real world is so dynamic that you have to expect the unexpected.

In this thesis we help to close the gap between literature and practice. The generic multi-agent reinforcement learning approach being proposed allows to simulate, for example, what will happen if a resource is not available during a specific amount of time, or to estimate the probable end time of an extra order that arrives to the system after some hours of work, having a very high priority. This helps to identify the 'critical' parts of the schedule and based on that incorporate different levels of robustness to the solution in those places where it will be needed.

In the next section we give an overview of the structure of this dissertation.

1.4 Outline of the Dissertation

The research presented in this dissertation is divided in seven chapters.

In Chapter 2 we present an overview of the different scheduling problems that will be addressed in this dissertation. We start explaining the Job Shop Scheduling Problem, then we gradually move towards more challenging problems, like the Parallel Machines Job Shop Scheduling, where identical parallel machines can execute the same type of tasks, and the Flexible Job Shop Scheduling Problem, where there are also multiple machines that can execute the same type of tasks, but in this case the machines are not identical, for example, they can differ in speed, and this gives an extra level of complexity to the sequencing process.

Then we switch to more stochastic scenarios. We introduce a version of the Flexible Job Shop Scheduling Problem where release dates and due dates are added

to the jobs. Some perturbations are also introduced in the system, mainly represented by machine breakdowns, which gives the possibility of looking at different objective functions, as there is more information involved. We also present an online scheduling problem, which is based on a chemical production plant with two decision levels. Each level has four machines which differ in speed and jobs enter the system according to specific stochastic distributions.

Finally, we describe a scheduling scenario that is based on a real-world case, which according to its specifications belong to the category of the hybrid flow-shop scheduling problems (jobs have to be processed following the same production flow in a series of stages). It is an enzyme production process with four different stages: 1) seed fermentation, 2) main fermentation, 3) broth preparation and 4) recovery. Each of these stages has multiple machines, except the recovery line. The orders from the customers can involve the production of different types of enzymes, and all of them have to follow the production path in the same order (stages 1 to 4). To conclude the chapter, some related work is presented.

Chapter 3 provides the main ideas behind Reinforcement Learning and the theory on Multi-Agent Reinforcement Learning. The concepts of *agent* and *multi-agent systems* are introduced, as well as different representations of agents systems. We explain how the agents interact with the environment in order to learn how to solve a specific task, and how multiple agents can act in a cooperative or competitive way. Different solution methods are introduced, together with three possible action selection strategies. The last section of the chapter defines the main concepts that have to be taken into account when solving a scheduling problem using a Multi-Agent Reinforcement Learning Approach.

In Chapter 4 we summarize the main results of the application of Reinforcement Learning in the solution of the different scheduling problems introduced in Chapter 2. We start by introducing a basic approach, which is modeled for the JSSP, and then we explain how to adapt this model in order to deal with the extra constraints that gradually appear in the other scheduling scenarios. Each section of the chapter presents the approach corresponding to one of the scheduling problems and some of the results obtained by experimenting and comparing with other approaches reported in literature.

As discussed before, there are several unexpected events that can affect the scheduling process. In the real-world things are constantly changing and this is something inevitable, machines can break down, customers can show up with new orders, jobs can take longer than expected, and these events can occur at any time, making the original schedule fail. In Chapter 5 we give some details about the different sources of uncertainty that can affect the scheduling process. We introduce the concept of robustness and how to incorporate it in the solutions. We present some existing techniques in this area and discuss how each of them introduces the slack time (extra time assigned to the activities) that can be used for the algorithm to deal with unexpected events.

As was mentioned before, it is important to take unexpected events into account. This is the sole focus of the research presented in Chapter 6: how to schedule under uncertainty. This chapter is divided in two parts. First, we present the results obtained when solving a stochastic scheduling problem. This study was developed in collaboration with Elsy Kaddoum, from the Systèmes Multi-Agents Coopératifs (SMAC) lab - IRIT, Université Paul Sabatier, Toulouse, France, and Tony Wauters, from the Combinatorial Optimisation and Decision Support (CODeS) Research Group, KAHO Sint-Lieven, Gent. The results of this collaboration were presented in [Kaddoum et al. (2010)]. In the second part of the chapter, a new slack-based approach which aims at incorporating robustness in the solution construction process is introduced. Our idea is based on the methods introduced in the previous chapter, but it changes the way the slack time is added. The term 'criticality' is introduced and its influence in the way of providing the activities with extra time is explained. It is important to mention that this approach allows the user to define the level of robustness that will be included in the solution. Of course this is more useful when the user has domain knowledge, and it could be achieved by letting him tune the parameters that define when a specific part of the schedule can be considered as critical.

Finally, we conclude in Chapter 7 with a summary of the presented research, together with some ideas for future work, including a potential real-world application in the Cuban industry.

Figure 1.2 provides a graphical representation of this outline. The arrows represent the dependency between the different chapters.

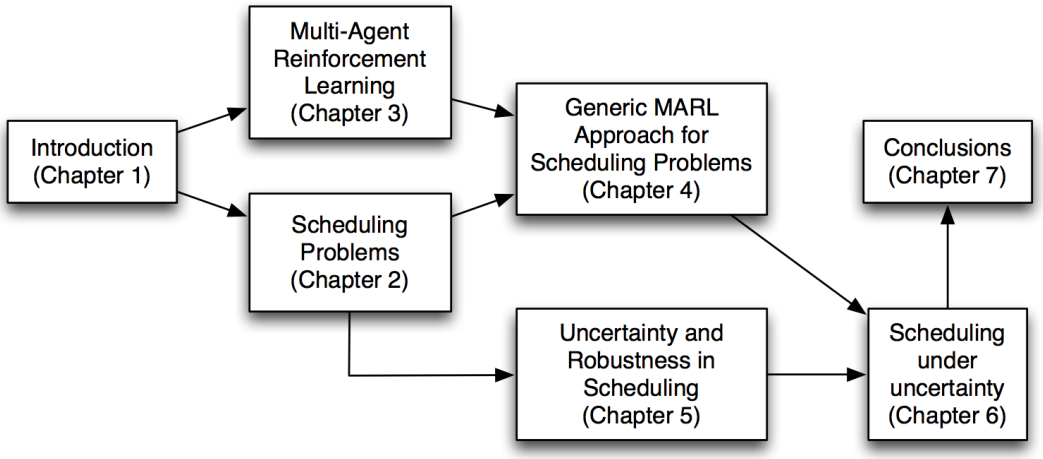

Figure 1.2: Graphical representation of the outline of this dissertation.

Chapter 2

Scheduling Problems

Scheduling concerns the allocation of limited resources to tasks over time.
It is a decision-making process that has as a goal the optimization of one or more objectives
- [Pinedo (2008)] -

Scheduling problems are present in every situation where a given set of tasks has to be performed and these tasks require the allocation of resources to time slots. This is a common procedure that we usually perform in our daily life, but when the constraints that have to be met increase and the number of tasks and resources grow, then we realize that constructing a schedule that satisfies all the requirements is not so straightforward. This chapter starts by giving an introduction to scheduling and some of the factors that can influence this kind of problem. The different types of schedules that can be obtained are defined, as well as the different scheduling problems that will be addressed by the research presented in this dissertation.

2.1 Introduction to Scheduling

Scheduling is a decision making process that is used on a regular basis in many real life situations. It deals with the allocation of resources to tasks over time, and its goal is to optimize one or more objectives [Pinedo (2008)]. For example,

scheduling problems occur routinely in publishing houses, universities, hospitals, airports, transportation companies, manufacturing and services industries, and so on. The resources and tasks can take many forms, depending on the problem being solved, and the objectives can also vary, for example, one of the most common goals is to minimize the makespan, which is the completion time of the last task.

The problem of scheduling has captured the interest of many researchers from a number of research communities: management science, industrial engineering, operations research (OR) and artificial intelligence (AI). The growing complexity of manufacturing processes and fierce competition in the market place drive enterprises to optimize their operations as much as possible. In particular, optimal scheduling of production orders on limited resources is an important issue and this type of optimization problem has fascinated researchers for years [Grossmann (2005)].

Manufacturing scheduling, in short, is an optimization process that allocates limited manufacturing resources over time among parallel and sequential manufacturing activities. This allocation must obey a set of constraints that reflect the temporal relationships between activities and the capacity limitations of a set of shared resources. The allocation also affects the optimality of a schedule with respect to different criteria.

Figure 2.1 shows a typical diagram of the information flow in a manufacturing/production system.

In general, in a manufacturing system, the scheduling function has to interact with other decision making functions. It is affected by the middle-range planning, which examines the stock levels, the demand forecasting and the requirements plan, in order to achieve the optimization of the combination 'Production-Allocation of Resources'. In this context, the construction of a feasible, optimized production schedule, without the support of an advanced decision support tool, is a very difficult and time consuming procedure that requires not only deep knowledge of all the data and parameters of the production system, but also specific knowledge in the particular field [Metaxiotis et al. (2005)].

These type of problems are typically NP-hard [Garey et al. (1976), Brucker (2007)] and the computational time increases exponentially with the problem size, being manufacturing scheduling one of the most difficult scheduling problems [Shen (2002)].

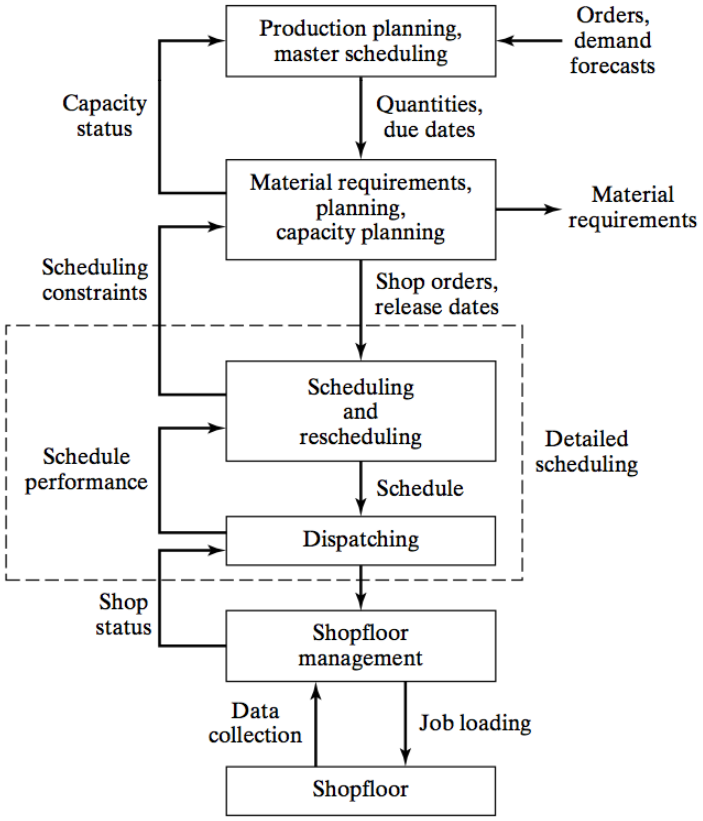

Figure 2.1: Information flow diagram in a manufacturing system, taken from [Pinedo (2008)].

Research in this field has made an impressive progress, from simple models that are only useful from an academic point of view to approaches that can be used in complex realistic settings like batch production, where groups of items (batches) move through the production process together, a stage at a time, and the next stage can not start until the previous one is completed[1]. Batch processes are widely used in the pharmaceutical, chemical, food, paint, and agrichemical industries, because they provide the flexibility to produce various products using the same processing facility [Charnprasitphon (2007)].

Figure 2.2 shows several factors that can influence a scheduling problem, of course these factors do not necessarily appear at the same time.

Every real-life production process has its own specific characteristics and constraints which are hard to generalize into a one-size-fits-all mathematical problem

[1] the order of the stages is fixed

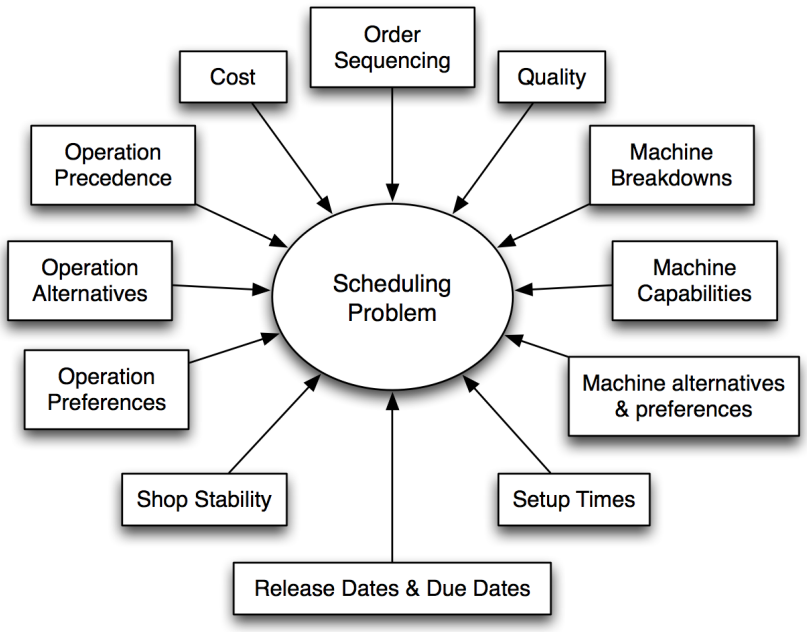

Figure 2.2: Several factors that can influence a scheduling problem.

formulation, so effective optimization approaches remain in high demand.

The classical approach to solve scheduling problems consists of setting up a mathematical model, often based on a Mixed Integer Linear Programming formulation, and then applying a search algorithm like Branch and Bound to calculate the (guaranteed) optimal solution [Méndez et al. (2006)]. However, as the problem grows in the number of available resources, operations to be scheduled, and other constraints to be taken into account, this approach will no longer yield an optimal solution in a reasonable time span. In this situation, researchers often relax constraints that actually exist in the real-life problem. This makes the problem solvable, but it is no longer of practical use. As a consequence, only a fraction of published scheduling research deals with actual real-world problems [Ruiz et al. (2008)].

Figure 2.3 summarizes the different types of scheduling problems that will be addressed in this dissertation. The left side of the picture shows the deterministic problems and the right side groups the stochastic ones and a real-world case. It is worth to mention that the online scheduling problem addressed in this dissertation is based on a real scenario, although it is a simplified version of the original problem.

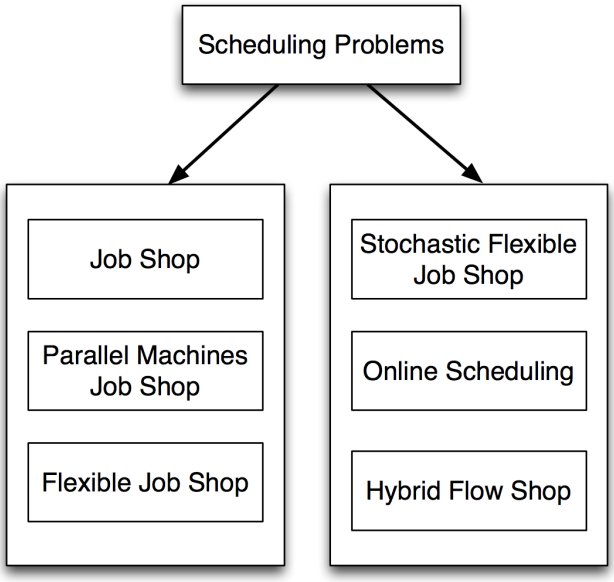

Figure 2.3: Different types of Scheduling Problems

All these scheduling problems will be explained in detail in the next sections of the chapter (from section 2.4 to 2.9). But first we introduce some basic notation and properties (sections 2.2 and 2.3) that will be used in the rest of the dissertation.

2.2 Notation

The definition of a scheduling problem includes several variables and properties, in this section we provide a list of the ones that will be used in this dissertation:

m	Number of machines
n	Number of jobs
M_i	Machine i, where $i = \{1, ..., m\}$
J_j	Job j, where $j = \{i, ..., n\}$
C_j	The completion time of job j
P_j	Processing time of job j
o_{ij}	The i_{th} operation of job j
s_{ij}	Start time of the operation i, job j
c_{ij}	Completion time of the operation i, job j
p_{ij}	Processing time of operation i, job j
r_j	Release date of job j
d_j	The due date of job j
C_{max}	The makespan, maximum C_j over all jobs, $max(C_1, ..., C_n)$
L_j	Lateness of job j
E_j	Earliness of job j
T_j	Tardiness of job j
w_j	The weight associated with job j

Figure 2.4 shows an example of a timing diagram for operation o_{ij}, where some of the properties mentioned before are summarized. The rest of the properties will be described in the next section.

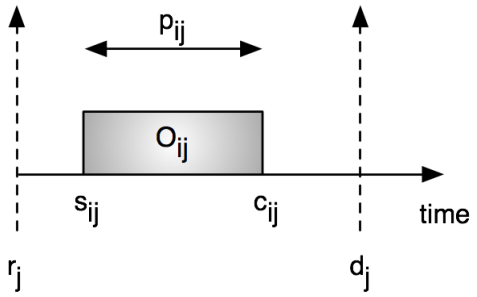

Figure 2.4: Timing diagram of one operation.

2.3 Classification of Scheduling Problems

Scheduling problems are commonly classified according to the machine environment, the job characteristics and the objective function [Herrmann et al. (1993)]. This classification is commonly referred to as the triplet $\alpha|\beta|\gamma$, proposed in [Graham et al. (1979)], where:

- α describes the machine environment (single machine, multiple machines), see subection 2.3.1.
- β denotes the constraints that have to be met (subsection 2.3.2).
- γ refers to the objective function, for example, the makespan, which is the more commonly used in literature, for more explanation about possible objectives functions see subsection 2.3.3.

2.3.1 Machine Environment

The first element of the problem description is the machine environment. The case of a single machine is the simplest of all possible machine environments, as jobs have only one operation to be performed and there is only one machine that can execute it. But when there are multiple machines that can process the operations, the environments get more complicated, as parallel machines could be identical, but they could also differ in speed. The possible environments are summarized as follows:

- Identical Parallel Machines: process jobs with the same speed.
- Non-Identical Machines: have machine-dependent speeds.
- Unrelated Parallel Machines: have machine and job dependent speeds.

When each job has a fixed number of operations requiring different machines we are dealing with a shop problem, and depending on the constraints it presents, can be included in one of the following categories:

- Open Shop: There are m machines and each job has to be processed on each of them. There are no ordering constraints between the operations of each job, which means that jobs can follow different routes.
- Job Shop: In a job shop with m machines, jobs have to visit each machine once, but in this case the operations of a job are totally ordered. This order is not the same for all the jobs.
- Flow Shop: In a Flow Shop with m machines, all jobs go through all the machines in the same order.
- Flexible Job Shop: A generalization of the job shop and the unrelated parallel machines environment. Each operation can be processed by any among a set of possible machines, and these machines differ in speed.

2.3.2 Job Characteristics

Each problem has a set of job characteristics, which may occur in any combination. A list of the more commonly used is given below:

- Preemption: refers to environments where the processing of a job may be interrupted and resumed later (possibly on another machine).
- Precedence Constraints: The jobs can have precedence constraints, that is, some jobs can not start until others are completed.
- Release Dates: A job can not start to be processed before its release date.
- Due Dates: Time by which a job is expected to be finished.
- Setup Times: There might be a setup time involved between the execution of two different jobs on the same machine, which means that the machine needs some time to get ready for the execution of the next job.

2.3.3 Objective Functions

The objective function to be minimized or maximized is the third element of a problem description. This may be a sum of variables or the maximum or minimum of some variable function. Typical objective functions include the following performance measures:

- Flowtime: sum of the completion times of the jobs, $\sum_j C_j$.
- Makespan: maximum completion time, which is equivalent to the time when the last job leaves the system (also known as C_{max}).
- Lateness: difference between the due date and the completion time. This measure can be positive (tardiness) or negative (earliness), $L_j = C_j - d_j$. Tardiness measures the difference if the job is late, otherwise it takes value zero. Similarly, earliness measures the difference if the job is finished early, otherwise it is zero.
  - Earliness: $\sum_{j=1}^{n} E_j$, where $E_j = max(d_j - C_j, 0)$
  - Tardiness: $\sum_{j=1}^{n} T_j$, where $T_j = max(C_j - d_j, 0) = max(L_j, 0)$
- Earliness-Tardiness: This measure has been the focus of some studies where it is desirable that jobs finish close to their due dates, $\sum_{j=1}^{n} E_j + \sum_{j=1}^{n} T_j$, penalties could also be associated when jobs are either too early or too late.
- Total weighted tardiness: Similar to the tardiness but this time the weight of the jobs is taken into account: $\sum_j w_j T_j$.

2.3.4 Other Criteria

Another distinction to be made is between deterministic and stochastic scheduling problems. In the deterministic class, all parameters are known without uncertainty, whereas this is not the case in the latter. Stochastic scheduling is concerned with scheduling problems in which the processing times of tasks are modeled as random variables.

Finally, scheduling methodologies can also be classified according to the way the schedule is obtained. The methods can obtain the solutions in a constructive way or by iteratively repairing a complete schedule. The constructive methods incrementally extend a partial schedule until every task has been scheduled. The

repair methods iteratively modify a complete schedule to remove conflicts or to further optimize the solution. Most optimization methods and constraint-directed search methods follow the constructive approach.

2.3.5 Types of Schedules

According to the schedule properties, any feasible schedule (i.e. a solution to the scheduling problem) can be categorized into three major kinds: non-delay, semi-active and active. The definitions of these categories are adapted from [Conway et al. (1967)] and [Pinedo (2008)]. We first present the three definitions and afterwards summarize the main differences between them with one general example.

Definition 1 (Non-Delay Schedule)**.** *A feasible schedule is called non-delay, if no resource is kept idle, while there is at least one operation waiting for further processing on that resource.*

The non-delay category is probably one of the most used in literature when solving a scheduling problem where the objective is to minimize the makespan. However, at the end of the section we will show that keeping the resources working does not always lead to optimal solutions.

Definition 2 (Semi-Active Schedule)**.** *A feasible schedule is called semi-active, if an earlier completion of any operation could only be achieved by changing the processing order on at least one resource.*

This means that the operations start to be executed as early as possible, which implies that the only way to execute them earlier is by changing the processing order on the resources. A semi-active schedule can be obtained from any arbitrary schedule by moving each operation to the left, until it gets blocked either by the preceding operation on that machine or the preceding operation of that job (procedure called 'limited-left-shift'). The ordering of the operations cannot be altered.

Definition 3 (Active Schedule)**.** *A feasible schedule is called active if it is not possible to construct another schedule, through changes in the order of processing on the resources, with at least one operation finishing earlier and no operation finishing later.*

In other words, a feasible schedule is said to be active if no operation can be moved into an empty time slot earlier in the schedule without violating the ordering

constraints. Active schedules can be obtained by performing what is called a 'left-shift' operation, which is any decrease in the time at which the operation starts that does not require an increase in the starting time of any other operation [Conway et al. (1967)]. In this case an operation is allowed to 'jump over' another operation into an interval of idle time, if that interval is large enough to accommodate the shifted operation.

The following example helps to understand the differences between the three definitions. It shows three possible solutions to the same scheduling instance, and we will analyze to which of the categories defined above they belong.

Example 1: *L*et us assume that we are solving a scheduling problem with two jobs and two resources. Each job has two operations and the information about which machine can execute them is given in the following table:

Table 2.1: Scheduling instance with 2 resources and 2 jobs.

Job	**Op$_1$**	**Op$_2$**
1	M_2,2	M_1,1
2	M_1,2	M_2,1

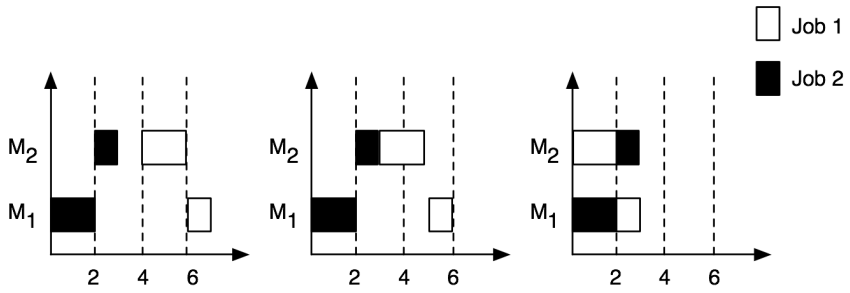

Figure 2.5: Three different feasible solutions to the instance presented in Table 2.1.

Analyzing the schedules presented in Figure 2.5, it is possible to see that schedule 1 (left hand side) is a feasible schedule, because the ordering constraints of the problem are satisfied, but it does not belong to any of the categories described before. If we look at machine M_2 we will see that it remains idle while there are

operations it can execute (for example from time 3 to 4), which means it is not a non-delay schedule.

It is not a semi-active schedule either, as a limited-left-shift can be performed on the first operation of J_1. This operation is being executed on M_2 at time 4, when it could start its processing at time 3. If we perform limited-left shifts on schedule 1, then we will obtain schedule 2 (middle), which means that schedule 2 is semi-active.

But neither schedule 1 or schedule 2 are active, because a left-shift can be performed on both of them on the first operation of J_1. This operation can be moved into the initial idle time on M_2 without modifying the start of the second operation of job 2 (jumping over it), as the idle time of the machine is exactly the amount of time it needs to be executed.

If we perform this, then the second operation of J_1 could be subject of a limited-left shift, resulting in schedule 3 (right hand side), which is an active schedule and coincidentally, the only non-delay schedule in the example.

If we look again at Figure 2.5, we will see that the non-delay schedule results in better makespan, which makes sense as there is a higher utilization of the resources. But if we also analyze Figure 2.6, which shows the hierarchy of the different types of schedules, we will see that the optimal schedules can be found in the set of active schedules, but not necessarily in the non-delay class, which will be demonstrated through the next example:

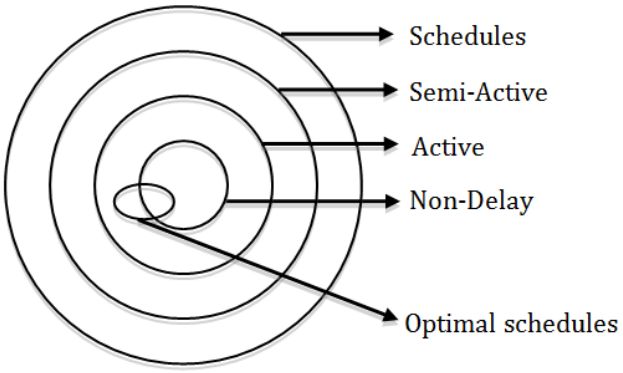

Figure 2.6: Different types of schedules. The optimal solutions can be found within the active schedules, but not necessarily in the non-delay class.

Example 2: *G*iven an instance with three machines and two jobs (data for the operations is given in table 2.2), where the goal is to minimize the makespan, figure 2.7 illustrates that looking at non-delay schedules is not sufficient for finding the optimum. The operations of the jobs need to be processed in numerical order. Machines and processing times are specified for each operation. From the table below we can infer that Job_1 should be processed in M_1 first for one time step, then go to M_2 for three time steps and finally go to M_3 for being successfully processed after five time steps (the same for Job_2).

Table 2.2: Instance with 3 resources and 2 jobs. Job 1 consists of 3 operations, operation 1 (Op_1) has to be executed on machine M_1 during 1 time step, Op_2 on machine M_2 for 3 times steps and so on.

Job	**Op$_1$**	**Op$_2$**	**Op$_3$**
1	M_1,1	M_2,3	M_3,5
2	M_2,3	M_1,2	M_3,1

Figure 2.7 shows the optimal schedule together with the best (and in fact the only) non-delay one. Job_1 is represented in white and Job_2 in black.

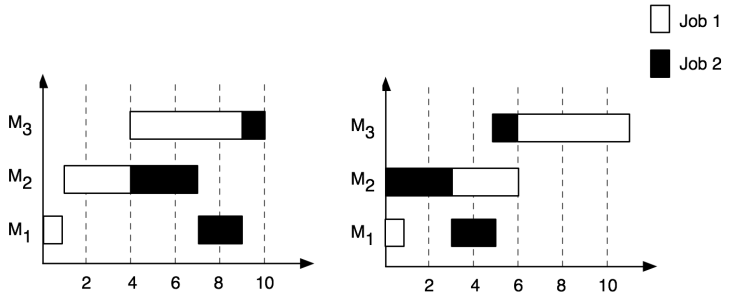

Figure 2.7: Left: Optimal schedule. Right: Non-delay schedule

In this case the optimal schedule, with $C_{max} = 10$, is not a non-delay one. If we look for a non-delay schedule, that is, a schedule with no idle times between operations, then our best result would be C_{max} =11.

If we decide to obtain the non-delay solution, it means that machines keep working if there are operations to execute. That is the reason why in the first time step M_1 immediately starts with the first operation of Job_1 and M_2 with the first operation of Job_2 (right hand side schedule).

But if we analyze the optimal solution (left hand side schedule) it is possible to notice that it is better if M_2 waits instead of immediately start executing the first operation of Job_2. This introduces idle times in the schedule, but in the long term leads to a better solution. This illustrates that looking at non-delay schedules is not sufficient for finding the optimum.

After introducing the terminology and the different properties that must be taken into account when solving a scheduling problem, the next sections will describe the different scheduling scenarios that will be addressed in this dissertation.

2.4 Job Shop Scheduling Problem

A well-known manufacturing scheduling problem is the classical job shop scheduling (JSSP), which involves a set of jobs and a set of machines with the purpose of finding the best schedule, that is, an allocation of the operations to time intervals on the machines that has the minimum duration required to complete all jobs (in this case the objective is to minimize the makespan). The total number of possible solutions for a problem with n jobs and m machines is $m(n!)$. In this case, exact optimization methods fail to provide timely solutions. Therefore, we must turn our attention to find methods that can efficiently produce satisfactory (but not necessarily optimal) solutions.

The general job shop problem is an interesting challenge. In many cases it is easy to visualize what is required, but even for small problem instances it could be extremely difficult to make progress towards a solution. Classical approaches to solve job shop scheduling problems are, for example, branch-and-bound algorithms. Besides, there are also several local search procedures that have been applied in this field, for instance, simulated annealing and tabu search.

2.4.1 Problem Definition

The terminology of job-shop scheduling arose in the manufacturing industries. In the fundamental theory of scheduling (for example [Baker (1974)] and [Garey & Johnson (1979)]) job-shop scheduling defines a set of scheduling problems in the following mathematical formulation. Given a set M of m machines that can pro-

cess only one job at a time, and given a set J of n jobs that must be processed on each of these machines in a prescribed, job-dependent order, find a feasible schedule that minimizes the total processing time. Each job J_j consists of m chained operations $\{o_{1j}, o_{2j}, ..., o_{mj}\}$ that have to be scheduled in a predetermined given order (precedence constraints) [Zhang (1996)].

There is a total of $N = n * m$ operations where o_{ij} is the i_{th} operation of job J_j and has to be executed during an uninterrupted processing time p_{ij}. The workflow of each job throughout the machines is independent of the other jobs'. At any given time, each machine is able to carry out a single job and each job can only be processed by a single machine.

The objective is to determine the starting time ($s_{ij} \geq 0$) for each operation so as to entail a minimization of the makespan in such a way that all of the constraints are met:

$$C^*_{max} = min(C_{max}) = min_{feasible-schedules}(C_j, \forall J_j \in J) \tag{2.1}$$

Some of the restrictions inherent in the definition of the JSSP are the following:

- Only one operation from each job can be processed simultaneously.
- No preemption (i.e. process interruption) of operations is allowed.
- No job is processed twice on the same machine.
- Each job must be processed to completion.
- Jobs may be started at any time, i.e., no release times exist.
- Jobs may be finished at any time, i.e., no due dates exist.
- Machines can not process more than one operation at a time.
- There is only one machine of each type.
- Machines may be idle within the schedule period.
- Jobs must wait for the next machine in the processing order to become available.
- The machine processing order of each job is known in advance and it is immutable.

The JSSP is widely accepted as one of the most difficult NP-hard problems [Garey et al. (1976)] and consequently, as noted in [Garrido et al. (2000)], presents a constant intellectual challenge: despite over 40 years of effort, resulting in hundreds of published journal articles and dozens of dissertations, even state-of-the-art algorithms often fail to consistently locate optimal solutions to relatively small problem instances.

2.4.2 JSSP Instances

There are multiple benchmark problems for the JSSP available in the OR-Library [Beasley (1990)], which is a library of operations research problems available on the internet. Some of these well-known instances will be used in this work to measure the performance of the algorithms we will introduce in this dissertation. Figure 2.8 shows an example of a JSSP instance from this library.

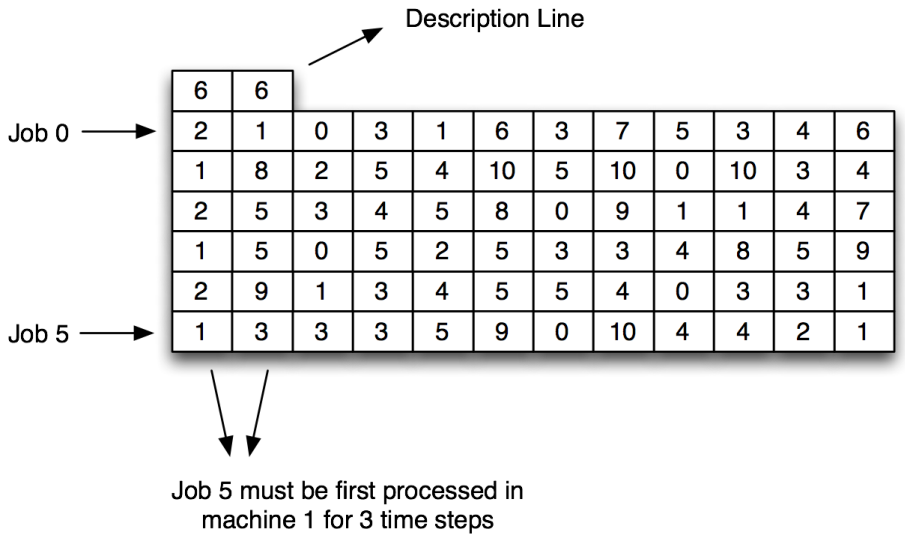

Figure 2.8: Example of a JSSP instance (ft06) from the OR-Library.

As it can be seen in Figure 2.8, the data is given in a specific format, which can be summarized in the following way:

Each instance consists of a line of description, which contains the number of jobs and the number of machines, and then one line for each job, listing the machine number and processing time for each operation of the job. The machines are numbered starting with 0.

In this case, the first line stands for 6 jobs and 6 machines, and then, for example, job 1 (third line) must be processed in resource 1 for 8 time units, then go to resource 2 for 5 time units, and so on. What is important here is to remember (from the restrictions of the problem mentioned above) that the processing order (the order in which each job has to be processed by the machines) cannot be violated. For example, from the data shown in Figure 2.8 we can infer that Job 5 has to be processed in the following order: $\{M_1,\ M_3,\ M_5,\ M_0,\ M_4,\ M_2\}$ for $\{3, 3, 9, 10, 4, 1\}$ time units respectively.

Each line of the instance can be summarized as depicted in Figure 2.9.

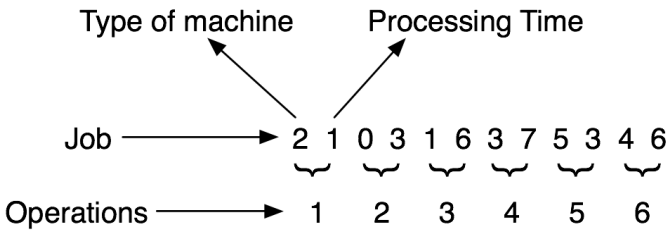

Figure 2.9: Job description in the JSSP environment.

The Gantt Chart representation in Figure 2.10 shows an optimal solution for the previously shown 6x6 instance ft06.

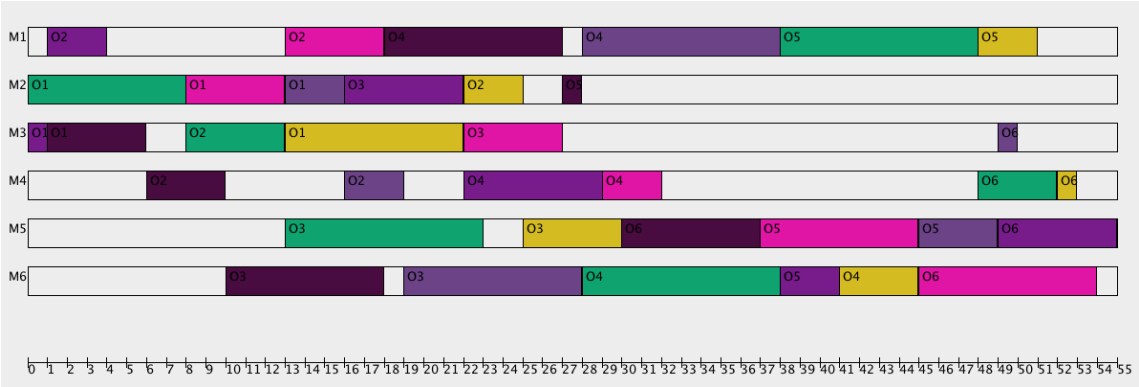

Figure 2.10: Optimal schedule for the instance ft06. Each color represents a different job.

In this solution the makespan is 55 time units (C_{max}=55). It is important to notice that the optimal solution is actually a delayed schedule: After 6 time steps,

resource 3 (M_3) remains idle and waits until it can process the second operation of job 2 (green) at $t = 8$, although it could have immediately continued to process the first operation of job 5 (yellow), which had been waiting at resource M_3 since the beginning of the scheduling process.

In the next section we will increase the complexity of the scheduling problem, by incorporating parallel machines that can execute the same type of operations.

2.5 Parallel Machines Job Shop Scheduling

The job-shop scheduling problem with parallel machines (JSSP-PM) represents an important problem encountered in current practice of manufacturing scheduling systems. It consists of assigning any operation for each job to a resource of a candidate set of identical parallel machines, in addition to the classic job shop scheduling problem (JSSP) where the operations must be arranged on each (assigned) resource in order to minimize a certain objective [Rossi & Boschi (2009)].

Different terms are used to name the candidate set of identical parallel machines, it can be termed a machine type, a workcenter or also a flexible manufacturing cell. The difference with the classical Job Shop is that instead of having a single resource for each machine type, in flexible manufacturing systems a number of parallel machines are available in order to both increase the throughput rate and avoid production stop when, for example, machines fail or maintenance occurs.

2.5.1 Problem Description

In the JSSP-PM, operations of n jobs have to be scheduled on m pools of machines $G_j (j = 1, ..., m)$, each including certain number of identical parallel machines. Each job has a total number of O operations. These operations should be processed in a given order and each of them has to be executed by a specific machine. The problem with equal-size pools, i.e. k machines for each pool G_j, is also referred to as the JSSP-kPM problem. Job-shop scheduling is a particular case of JSSP-kPM where the number of machines in each pool is one (k=1). The assignment of operations to a machine of a pool gives a sort of further flexibility, as there are multiple machines that can execute the same type of tasks, but this also increases the complexity of the problem [Rossi & Boschi (2009)]. The objective function in this case is the minimization of the makespan.

Basically the restrictions for the JSP-kPM are the following:

- No two operations of one job may be processed simultaneously.
- No job is processed twice on the same machine or workcenter.
- Each job must be processed to completion.
- Jobs must wait for the next machine in the processing order to become available.
- No machine may process more than one operation at a time.
- Machines may be idle within the schedule period.
- Preemption is not allowed.
- The machine processing orders of each job is known in advance and is immutable.

2.5.2 Problem Instances

Figure 2.11 shows an example of an instance in the JSSP-PM environment.

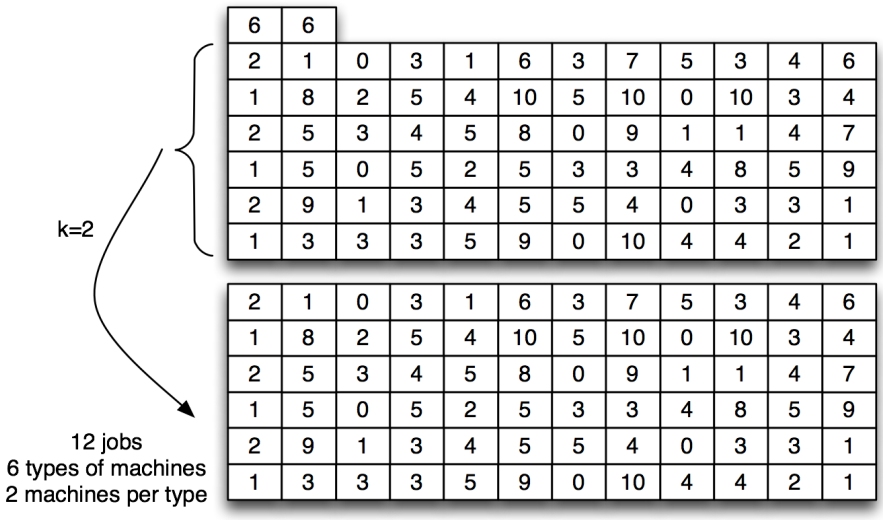

Figure 2.11: Example of a JSSP-PM instance with k=2.

In this case the instances are generated by taking a basic JSSP instance and replicating the jobs based on the number k. For example, let us assume that we have the 6x6 instance shown before, if k=2 it means that the new instance will have

12 jobs, 6 types of machines, and each of these 6 groups will have 2 identical parallel machines.

2.6 Flexible Job Shop Scheduling

The flexible job shop scheduling problem (FJSSP) consists of performing a set of n jobs $J = (J_1, J_2, ..., J_n)$ on a set of m machines $M = (M_1, M_2, ..., M_m)$. A job J_j has an ordered set of o_j operations $o_j = (o_{1j}, o_{2j}, ..., o_{o_j j})$. Each operation $o_{i,j}$ can be performed on any among a subset of available machines ($M_{ij} \subseteq M$). The main difference between this problem and the version with parallel machines described in the previous subsection is that in this case the machines are not identical, they might differ in speed and this gives an extra level of complexity to the sequencing. Executing operation $o_{i,j}$ on machine M_k takes $p_{i,j,k}$ processing time. Operations of the same job have to respect the finish-start precedence constraints given by the operation sequence. A machine can only execute one operation at a time. An operation can only be executed on one machine and can not leave it before the treatment is finished.

2.6.1 Flexible Job Shop Instances

There is also a set of instances available for tests for the case of the Flexible Job Shop, in this case in the FJSPLIB (http://www.idsia.ch/~monaldo/fjsp.html).

Figure 2.12 shows the structure of the FJSSP instances. In the first line there are (at least) 2 numbers: the first is the number of jobs and the second the number of machines. The third number is not necessary, which means that it is not always present, and it represents the average number of machines per operation.

Every row represents one job: the first number is the number of operations of that job, the second number (for example $k \geq 1$) is the number of machines that can process the first operation; then according to k, there are k pairs of numbers (machine, processing time) that specify which are the machines and the processing times; then the data for the second operation and so on.

For example, the second line in Figure 2.12 indicates that job 1 has 3 operations. The first operation can be executed in 2 machines, M_1 for 20 time units or M_3 for 25 time units, and so on. Figure 2.13 shows how each line (representing a job) can be read.

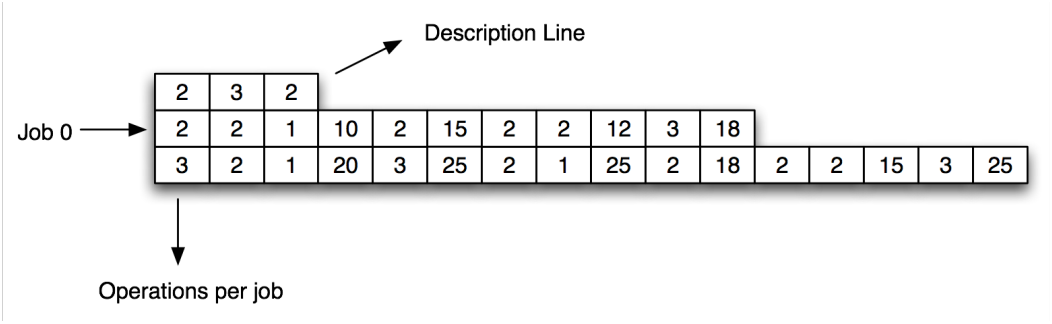

Figure 2.12: Flexible Job Shop Scheduling Instance

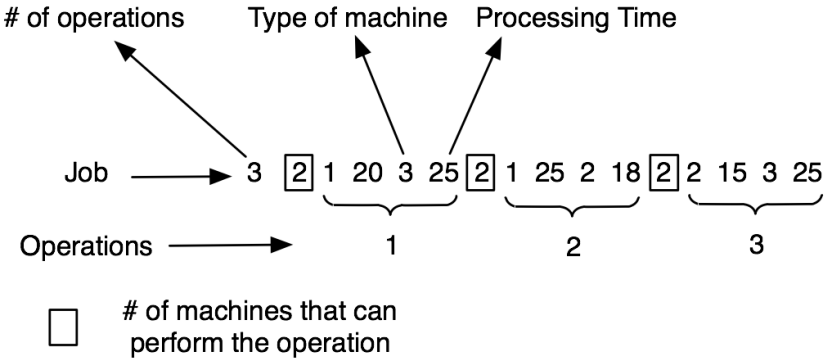

Figure 2.13: Job description in the Flexible Job Shop environment

Different heuristic procedures have been developed in the last years for the FJSSP, for example, tabu search, dispatching rules, simulated annealing and genetic algorithms. All these methods can be classified into two main categories: hierarchical approaches and integrated approaches.

The hierarchical approaches are based on the idea of decomposing the original problem in order to reduce its complexity. A typical decomposition is 'assign then sequence', meaning that the assignment of operations to machines and the sequencing of the operations on the resources are treated separately. Once the assignment is done (each operation has a machine assigned to execute it), the resulting sequencing problem is a classical JSSP.

Integrated approaches consider assignment and sequencing at the same time. The methods following this type of approach usually give better results but they are also more difficult to implement.

2.7 Stochastic Flexible Job Shop Scheduling

The stochastic flexible job shop scheduling problem addressed in this work is an extension of the previously described FJSSP. It consists of performing a set of n jobs $J = \{J_1, J_2, ..., J_n\}$ on a set of m machines $M = \{M_1, M_2, ..., M_m\}$. A job J_j has an ordered set of o_j operations $o_j = \{o_{1j}, o_{2j}, ..., o_{ojj}\}$. Each operation $o_{i,j}$ can be performed on any among a subset of available of machines ($M_{i,j} \subseteq M$). Executing operation $o_{i,j}$ on machine M_k takes $p_{i,j,k}$ processing time. Operations of the same job have to respect the finish-start precedence constraints given by the operation sequence. A machine can only execute one operation at a time. An operation can only be executed on one machine and can not leave it before the treatment is finished.

A job J_j is released at time r_j and is due at time d_j. A machine M_k can have perturbations (e.g. breakdowns) which cause already started operations to suspend their execution. The interrupted operation can continue when the perturbation is finished. Once an operation has started on a machine it can not move to another machine.

We denote the scheduled start and completion time of an operation $o_{i,j}$ as $s_{i,j}$ and $c_{i,j}$. The completion time of a job C_j is equal to the completion time of its last operation $c_{o_j j}$. The tardiness of a job J_j is $T_j = max(C_j - d_j, 0)$. If a job J_j has a tardiness larger than zero ($T_j > 0$), then we say that it is tardy and $U_j = 1$ else $U_j = 0$. The following objectives are used:

- $C_{max} = max\{C_j | 1 \leq j \leq n\}$: makespan or completion time of the last job that leaves the system,
- $T_{max} = max\{T_j | 1 \leq j \leq n\}$: maximum tardiness,
- $\overline{T} = (1/n) \sum_{j=1}^{n} T_j$: mean tardiness, and
- $T_n = \sum_{j=1}^{n} U_j$: the number of tardy jobs.

In this case besides the makespan, we will also take into account the tardiness as objective function, measuring the total tardiness, the mean tardiness and the number of tardy jobs.

2.8 Online Scheduling

In an online scheduling problem the decision-maker does not know in advance how many jobs have to be processed and what the processing times are. The decision-maker becomes aware of the existence of a job only when the job is released and the processing time of a job becomes known only when the job has been completed [Pinedo (2008)]. In this section we introduce a scheduling problem which is inspired by a real world scenario, and which is an example of an online scheduling problem.

Batch chemical plants usually consist of a series of one or more processing stages with parallel processing units at each stage. A new trend in production processes is to operate flexible, adaptive multi-purpose plants. We look at an application based on the chemical production plant of Castillo and Roberts [Castillo & Roberts (2001), Peeters (2008)]. It is a two-stage process with four times two parallel machines, see Figure 2.14 for a graphical representation.

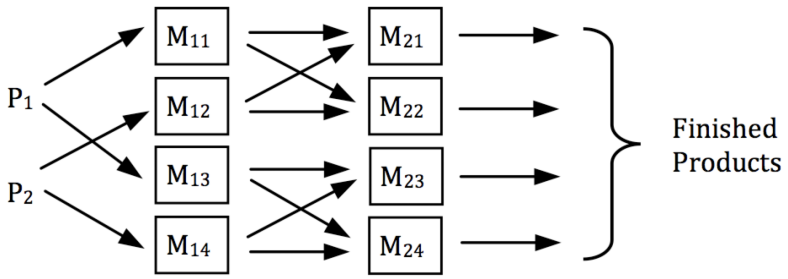

Figure 2.14: A two-stage chemical production plant. For both product types P_1 and P_2, there are two parallel machines at the first stage. At the second stage of the process, there are also two parallel machines.

Each order (created at P_1 and P_2) must be handled first by a 'stage-1' machine M_{1-} and afterwards by a 'stage-2' machine M_{2-}. At each stage, a scheduler must choose between two parallel machines. *Parallel machines* can handle the same type of tasks, but may differ in speed. The possible choice in parallel machines is depicted by the arrows in the figure. All machines have a FIFO-queue and execute jobs non-preemptively.

The length of the jobs varies according to an exponential distribution. Only the average joblength is known by the schedulers. Also, the speeds of the machines are unknown. Even the expected processing time of the jobs is unknown. However, when a job is finished, the scheduler has access to its exact processing time.

Moreover, it is not known in advance when a new order will arrive, i.e. we have an *online* scheduling problem. In an offline problem, all product orders are known in advance. An optimal algorithm will find the best feasible schedule if time and memory restrictions allow it to be computed. In an online scheduling problem, an algorithm has to make decisions based on the history (i.e. information of already released or finished jobs) and the current product request. It is obvious this makes for a more challenging problem.

This problem is particularly hard since it is stochastic, online and multi-stage at the same time. There exist heuristics for online stochastic scheduling in the single-stage scenario. But these cannot be easily mapped to a multi-stage problem, in this case we do not only need the information about the immediate available machines, but also the information about the machines of the coming stages and this, of course, increases the complexity. In Chapter 4 we propose a solution for such a problem.

2.9 Hybrid Flow Shop Scheduling

The Hybrid Flow Shop problem studied in this section is based on the scheduling of a chemical batch process. The production process is also analyzed in [Gicquel et al. (2012)] and [Borodin et al. (2011)] and it arises from a bio-process industry (real-world case).

Compared to the previous scheduling problems, this case presents several additional constraints and assumptions to be taken into account, in particular:

- it is a batch production process, meaning that a job may be processed by more than one machine and a machine can process more than one job at a time.
- it belongs to the flow shop category, which means that all the jobs move through the system following the same order (see Figure 2.15).
- machines have setup times

The optimization problem consists in scheduling production for a bio-process in which fermentation techniques are used. Bio-processes are processes that use living cells or microorganisms (bacteria, yeasts, fungi etc.) to obtain products such as antibiotics, antibodies or enzymes. Bio-processes often involve a processing step called fermentation. Fermentation consists in placing the living organisms together

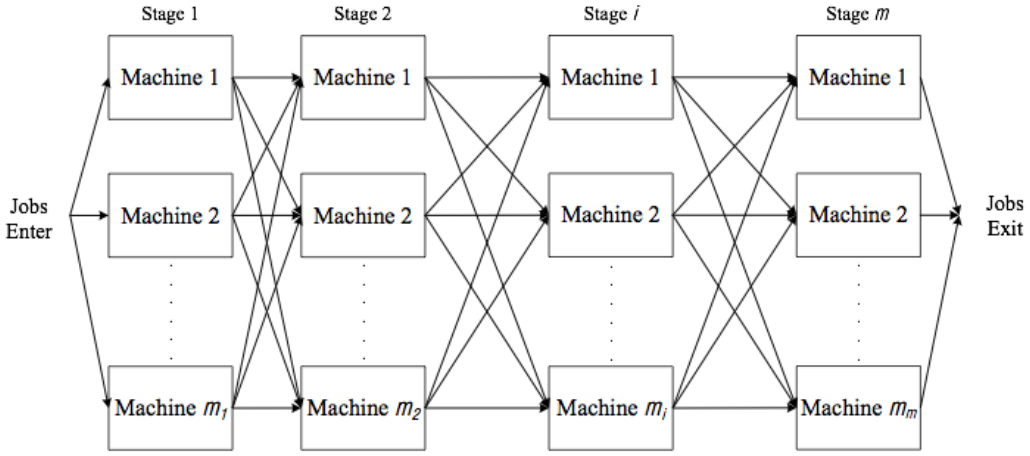

Figure 2.15: Hybrid Flow Shop Scheduling environment

with nutrients in an appropriate environment where temperature, pressure and oxygen content are controlled so that their metabolism produces the expected material. Once the fermentation is completed, the mixture of cells, nutrients and enzymes, called the broth, is transferred into a buffer tank where it is prepared for the recovery operations. This is an important phase because the biochemical reactions result in the formation of many undesired by-products. Therefore, after the fermentation is completed, it is necessary to carry out recovery operations to separate the desired end product from the other residues and isolate it in its pure form [Gicquel et al. (2012)]. The enzyme production process being addressed can be summarized in four main steps:

- seed fermentation;
- main fermentation;
- broth preparation;
- recovery.

In order to perform these operations, the following resources are available:

- two identical seed fermenters, $S = 2$;
- five identical main fermenters, $M = 5$;
- four buffer tanks: three identical small, $B^s = 3$, and one large, $B^l = 1(B^s + B^l = 4)$;

- one recovery line, $R = 1$.

Figure 2.16 provides a graphical overview of these production stages.

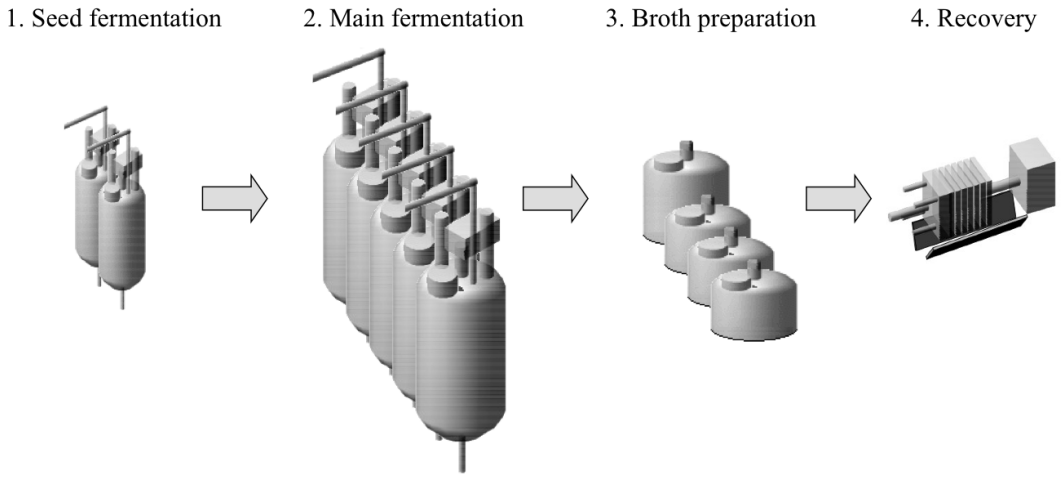

Figure 2.16: Production process stages

Several enzyme types are produced, denoted $z \in \{1...Z\}$. α_z is the number of batches per enzyme type that have to be produced, the size is given by q_z. Each batch is indexed by (z, j) consequently, where $z \in \{1...Z\}$ and $j \in \{1...\alpha_z\}$.

Both seed and main fermentation phases include three operations: 1) the setup or preparation of the fermenters; 2) the fermentation itself (duration is denoted as π_z^S and π_z^M respectively); 3) the tear down or cleaning of the fermenters. Thus, the total time required for a batch to be processed either in a seed or in a main fermenter is given by the sums of the times necessary to perform these three operations. The operational constraints are:

- a batch should be transferred without any delay to a main fermenter (which must be already setup or prepared) once the seed fermentation is finished (i.e. before starting the tear down or cleaning operations on the seed fermenter);
- a batch should be transferred to the buffer tanks without any delay, once the main fermentation is finished (i.e. before starting the tear down or cleaning operations on the main fermenter);
- the setup of a main fermenter requires a number of difficult manual operations, and because of the manpower restrictions at most one batch can be setup in main fermentation within a production shift.

Once a big or a small batch has been transferred into a buffer tank, it has to be processed in δ_z^R amount of time. The duration of the recovery phase is denoted δ_z^R. During the recovery phase, a batch is progressively transferred from a buffer tank(s) to the filtration unit. That is why the buffer tank(s) in which the batch is stored remains occupied until the filtration of the entire batch is finished.

This is the last of the problems that will be addressed in this dissertation. To conclude the chapter, some related work in scheduling will be presented.

2.10 Solution Methods for Scheduling Problems

Operations Research offers different mathematical approaches in order to solve scheduling problems, for example Linear Programming (LP) , Dynamic Programming (DP) and Branch and Bound (BB) methods. When the size of the problem is not too big, these methods can provide optimal solutions in a reasonable amount of time.

Most real world scheduling problems are NP-hard, and the size is usually not small, that is why optimization methods fail to provide optimal solutions in a reasonable time span. This is where heuristic methods become the focus of attention, these methods can obtain good solutions in an efficient way. Artificial Intelligence became an important tool to solve real world scheduling problems in the early 80s [Zhang (1996)]. Some of the methods that have been used are Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithms (GA), etc.

In the next subsections we will describe some of the related work that inspired the approach we are proposing in this dissertation.

2.10.1 Dispatching Rules

As discussed before, the complexity of scheduling problems gives raise to the search of heuristic algorithms able to provide good solutions. Dispatching rules are among the more frequently applied heuristics due to their ease of implementation and low time complexity.

A dispatching rule is a sequencing strategy by which a priority is assigned to each job waiting to be executed on a specific machine. Whenever a machine is available, a priority-based dispatching rule inspects the waiting jobs and the one with the highest priority is selected to be processed next [Nhu Binh HO (2005)].

Some of the most used dispatching rules are:

- Shortest Processing Time (SPT): The highest priority is given to the waiting operation with the shortest processing time.
- First In First Out (FIFO): The operation that arrived to the queue first receives the highest priority.
- Most Work Remaining (MWKR): Highest priority is given to the operation belonging to the job with the most total processing time remaining to be done.
- Earliest Due Date (EDD): The job due out first is processed first.

There are also some *composite* dispatching rules (CDR), which combine single dispatching rules and results have shown that a careful combination can perform better in terms of quality.

2.10.2 Reinforcement Learning approaches

One of the existing works on the application of reinforcement learning to solve the job shop scheduling problem was presented in 1995 [Zhang & Dietterich (1995)] and then extended one year later [Zhang (1996)]. This work focused on the application domain of space shuttle payload processing (SSPP) for NASA. The problem is to schedule the various tasks that must be performed to install and test the payloads that are placed in the cargo bay of the space shuttle. This study concerns a typical specification of the task, which takes the form of a job shop scheduling problem. Each shuttle mission is a job which is tied to a fixed launch date. Each job consists of a partially-ordered set of tasks with resource requirements. The goal is to schedule the set of tasks to satisfy a set of temporal and resource constraints while also seeking to minimize the makespan of the schedule.

In [Gabel & Riedmiller (2007)] and [Gabel (2009)], the authors suggested and analyzed the application of reinforcement learning techniques to solve job shop scheduling problems. They demonstrated that interpreting and solving this kind of scenarios as a multi-agent learning problem is beneficial for obtaining near-optimal solutions and can very well compete with alternative solution approaches.

Our objective is to take into account these ideas and based on that propose a generic multi-agent reinforcement learning approach that can be adapted to solve different scheduling scenarios, obtaining near-optimal solutions, but also giving the possibility to deal with unexpected events, without loosing too much in optimality.

2.11 Summary

In this chapter we presented the different scheduling problems that will be addressed in this dissertation using the approach that will be introduced in Chapter 4. We started by giving an explanation about the Job Shop Scheduling Problem and then gradually increased the number of constraints present in the problem, first by introducing the concept of Parallel Machines, then the Flexible Job Shop, and then more stochastic versions, like the online and the real-world scenarios. The different types of schedules that can be obtained were defined, and we demonstrated that looking at non-delay schedules does not always lead to optimal solutions.

Chapter 3

Multi-Agent Reinforcement Learning

Learning is any process by which a system improves performance from experience
-Herbert Simon-

Learning denotes changes in a system that enable it to do the same task or tasks drawn from the same population more efficiently and more effectively the next time [Simon & Lea (1973)]. In this chapter we start by introducing Reinforcement Learning (RL), a popular framework for designing agents that interact with their environment by executing actions and receiving feedback signals indicating how good the actions are, learning how to solve a specific task through these repeated interactions. Different solution methods are described, as well as different action selection strategies which can be used by the agents during the learning process. The last sections explain how to model a scheduling problem in order to solve it using RL methods.

3.1 Intelligent Agents

Even though agents are being used in a wide variety of applications nowadays, there is still no agreement on the definition of the term *agent*. For example, one general definition is given by Russell and Norvig in [Russell & Norvig (2003)], they define an agent as *'anything that can be viewed as perceiving its environment through sensors*

and acting upon that environment through actuators'.

Among several other definitions reported in the literature over the years, we can distinguish the one proposed in [Jennings et al. (1998)], this is a very open definition but it still summarizes the requirements that an agent needs to satisfy in the context of our work, therefore this is the one that is adopted in this dissertation.

Definition 4 (Agent)**.** *An agent is a computer system, situated in some environment, that is capable of flexible autonomous action in this environment in order to meet its design objectives [Jennings et al. (1998)].*

From this definition it is possible to notice that no specific environment is defined, and neither are the design objectives or how the agents can achieve them. Intelligent agents are defined in [Wooldridge (1999)] as agents that must operate robustly in rapidly changing, unpredictable, or open environments, where there is a significant possibility that actions can fail. According to [Wooldridge & Jennings (1995)], there are three characteristics that an intelligent agent needs to posses in order to meet its design objectives:

- *reactivity:* intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it;
- *pro-activeness:* intelligent agents are able to exhibit goal-directed behaviour by taking the initiative;
- *social ability:* intelligent agents are capable of interacting with other agents (and possibly humans).

If it is possible to guarantee that a specific environment is fixed, then it is relatively easy to design a goal-directed agent to operate in it. But the real world is not static, things are constantly changing, information is usually not complete, that is why the possibility of failure must be taken into account. A *reactive* system is one that keeps a constant interaction with its environment, and responds to changes that occur in it (in time for the response to be useful).

We want our agents to be reactive, but also to work towards long-term goals, therefore, it is important to have a good balance between reactivity and proactivity. However, some goals can only be achieved with the cooperation of other agents, and here social ability comes into play. Agents should be able to interact with other agents situated in the same environment in order to meet their design objectives.

In order to learn from experience, agents could be trained, for example, through supervised learning. In supervised learning, the agent is presented with examples of state-action pairs, along with an indication that the action was either correct or incorrect. The goal in supervised learning is to induce a general policy from the training examples, which is sufficiently general to deal with unseen examples. Thus, supervised learning requires a 'teacher' that can supply correctly labeled examples. In contrast, reinforcement learning can be applied to problems for which domain knowledge is either unavailable or costly to obtain [Moriarty et al. (1999)]. It does not require prior knowledge of correct and incorrect decisions, consequently, an agent must actively explore its environment in order to observe the effects of its actions, where for each taken action it receives a numerical signal indicating how good it was. This trial-and-error interaction with the environment is more suitable for the kind of problem we are solving in this dissertation, the next section gives a more detailed explanation about Reinforcement Learning.

3.2 Reinforcement Learning

Reinforcement learning (RL), as noted in [Kaelbling et al. (1996)], dates back to the early days of cybernetics and work in statistics, psychology, neuroscience and computer science. During the last decades it also attracted increasing interest from the machine learning and artificial intelligence communities.

RL is learning what to do (how to map situations to actions) so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the highest reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics, trial-and-error search and delayed reward, are the two most important distinguishing features of Reinforcement Learning [Sutton & Barto (1998)].

In the standard RL model, an agent is connected to its environment via perception and action, as depicted in Figure 3.1. In each interaction step, the agent perceives the current state s of its environment, and then selects an action a to change this state. This transition generates a reinforcement signal r, which is received by the agent. The task of the agent is to learn a policy for choosing actions in each state to receive the maximal long-run cumulative reward. Reinforcement

Learning methods explore the environment over time to come up with a desired policy [Zhang (1996)].

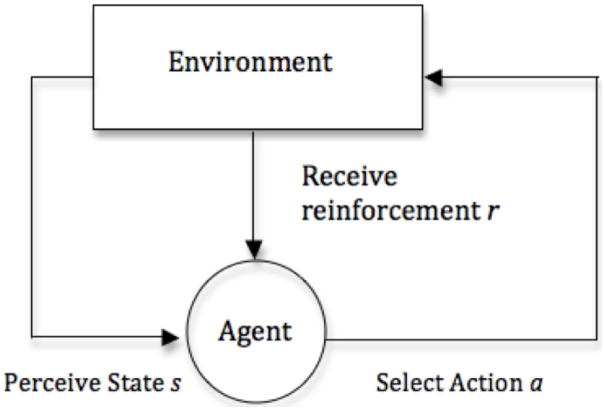

Figure 3.1: The standard reinforcement learning model.

Formally, the basic reinforcement learning model consists of:

- a set of environment states S;
- a set of actions A;
- a set of scalar 'rewards' in $\mathbb{R}$;
- a transition function T.

At each time t, the agent perceives its state $s_t \in S$ and the set of possible actions $A(s_t)$. It chooses an action $a \in A(s_t)$ and receives from the environment the new state s_{t+1} and a reward r_{t+1}, this means that the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and is denoted π_t, where $\pi_t(s, a)$ is the probability that $a_t = a$ if $s_t = s$. In words, it is the probability of selecting action a in state s at time t.

The *reward function* defines the goal in a reinforcement learning problem. It maps each perceived state (or state-action pair) of the environment to a single numerical value, a *reward*, indicating the intrinsic desirability of that action in that state. The objective of a reinforcement learning agent is to maximize the total reward it receives in the long run. The reward function defines what the good and bad events are for the agent.

In the same way that the reward function indicates what is 'immediately' good, a *value function* specifies what is good in the long run. In other words, the value of

a state is the total amount of discounted reward an agent can expect to accumulate over the future, starting from that state. For example, a state could always yield low immediate rewards, but still have a high value because it is regularly followed by other states that yield high rewards.

In order to give more details about how this works, Table 3.1 shows an example of the interaction between one agent and its environment, taken from [Kaelbling et al. (1996)].

Table 3.1: Interaction between one agent and its environment

Environment:	You are in state 65. You have 4 possible actions.
Agent:	I'll take action 2.
Environment:	You received a reinforcement of 7 units. You are now in state 15. You have 2 possible actions.
Agent:	I'll take action 1.
Environment:	You received a reinforcement of -4 units. You are now in state 65. You have 4 possible actions.
Agent:	I'll take action 2.
Environment:	You received a reinforcement of 5 units. You are now in state 44. You have 5 possible actions.
	...

This process of sequential decision making can be formalized by a Markov Decision Process (MDP) [Puterman (1994)]. MDPs provide a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision maker. This framework is described in detail in Section 3.3.

In summary, RL provides a flexible approach to the design of intelligent agents in situations for which, for example, supervised learning is impractical. RL can be applied to problems for which significant domain knowledge is either unavailable or costly to obtain. For example, a common RL task is robot control. Designers of autonomous robots often lack sufficient knowledge of the intended operational environment to use either the planning or the supervised learning regime to design a control policy for the robot. In this case, the goal of RL would be to enable the robot to generate effective decision policies as it explores its environment [Moriarty et al. (1999)].

3.3 Markov Decision Processes

A Markov Decision Process (MDP) is a model for sequential decision making when outcomes are uncertain. Emerging from operations research roots in the 1950s, MDP models have gained recognition in diverse fields like learning theory, economics and communications engineering [Goldsmith et al. (1996)]. A Markov Decision Process can be formally defined as follows:

Definition 5 (Markov Decission Process)**.** *A Markov Decision Process (MDP) is a 4-tuple [S, A, T, R] where:*

- $S = s^1, ..., s^n$ *denotes a finite set of states;*
- $A = \cup_{s \in S} A(s)$*, where* $A(s)$ *is the finite set of available actions in state* $s \in S$*;*
- $T : S \times A \times S \rightarrow [0, 1]$ *is the transition function,* $T(s, a, s')$ *specifies the probability of ending up in state* s' *when performing action* a *in state* s*;*
- $R : S \times A \times S \rightarrow \mathbb{R}$ *is the reward function,* $R(s, a, s')$ *denotes the expected reward for the transition from state* s *to state* s' *after taking action* a*.*

For MDPs, the Markov property assures that the transition from s to s' and the corresponding reward $R(s, a, s')$ depend only on the state s and the action a, and not on the history of previous states and actions.

Example 3: *L*et us assume we have a robot in an environment as depicted in Figure 3.2.

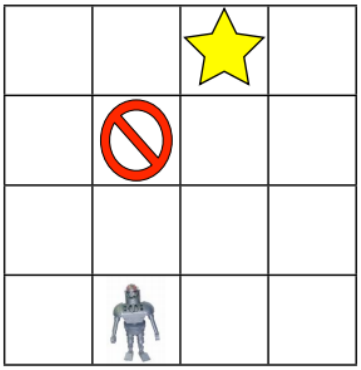

Figure 3.2: Robot navigation example.

This robot needs to get to the goal (yellow star) avoiding the stairwell (circular red sign). The possible actions are: go forward, turn left, turn right. The states are given by the specific location of the robot and its heading, it means there are 65 possible states. The rewards could be defined as follows: 1 if it reaches the goal, -1 if it ends up in the stairwell, and 0 everywhere else. The transitions are specified using probabilistic rules.

In the real world, an agent may not be able to accurately observe the current state of its environment. This may be caused by faulty or noisy sensors that disturb the true observation according to some probability distribution. Another reason may be that the observation capabilities of the agent are restricted to certain features of the world. In order to deal with such problems (uncertain observations), the model of partially observable Markov decision processes was introduced. It can be seen as an extension of the MDP model (Section 3.3), in this case observations are considered along with their probabilities of occurrence depending on the current state. This model can be formally defined as follows:

Definition 6 (Partially Observable Markov Decision Process)**.** *A Partially Observable Markov Decision Process (POMDP) is a tuple* (S, A, O, T, Ω, R)*, where:*

- S, A, T, R *are the same as in the definition of MDP (Definition 5);*
- Ω *is a finite set of observations;*
- O *is a table of observation probabilities.* $O(s, a, s', o)$ *represents the probability of observing* o *when taking action* a *in state* s *and transitioning to state* s' *as a result. Here* $s, s' \in S$*,* $a \in A$*, and* $o \in \Omega$*.*

POMDPs offer a challenging problem for RL, because the environment the agent experiences is not longer obeying the Markov property. Some more in depth study about POMDPs can be found in [Littman et al. (1995)] and in [Cassandra (1998)].

3.4 Solution Methods

In this section we will present some of the solution methods that can be used when dealing with a RL problem. The existing techniques can be classified in two main categories:

- Model based approaches: use an explicit model of the environment to derive the optimal policy.
- Model free approaches: derive the optimal policy without explicitly having the model.

In the next section we will introduce Dynamic Programming, which requires a complete and accurate model of the environment, and therefore belongs to the class of model-based approaches. Then we will present two model-free RL-algorithms, Q-Learning and Learning Automata. It is important to mention that these two model-free methods belong to different categories. Q-Learning works by learning the estimates for the values of the actions and then derives a policy, which means that it is value iteration, while Learning Automata is policy iteration, meaning that it updates the policy directly.

3.4.1 Dynamic Programming

The term Dynamic Programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov Decision Process [Sutton & Barto (1998)]. DP techniques are based on the *principle of optimality*, name due to Richard Bellman, who contributed to the popularization of this kind of techniques [Bertsekas (1995)]. The principle of optimality states that:

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. [Bellman (1957)].

The main idea behind DP is to search for optimal policies making use of the value functions of the states. As it was mentioned before, the value of a state s, given a policy π, is the total amount of reward an agent can expect to accumulate over the future, starting from that state and following the given policy.

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{R_t | s_t = s\} \\ &= E_{\pi}\Big\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \Big\} \end{aligned} \tag{3.1}$$

Where E_{π} denotes the expectation with regard to the policy π. Similarly, the value of taking action a in state s can be defined as follows:

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}\{R_t | s_t = s, a_t = a\} \\ &= E_{\pi}\Big\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \Big\} \end{aligned} \tag{3.2}$$

Once the optimal value functions have been found, it is easy to obtain optimal policies which satisfy the Bellman optimality equations:

Bellman optimality equation for V^*:

$$\begin{aligned} V^*(s) &= \max_a E_{\pi}\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} \\ &= \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \end{aligned} \tag{3.3}$$

Bellman optimality equation for Q^*:

$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')] \tag{3.4}$$

for all $s, s' \in S$, $a \in A(s)$, and given the transition function T and the reward function R. γ is the discount factor which determines the present value of future rewards. It is possible to see that DP algorithms are obtained by turning Bellman equations into update rules for iteratively improving the approximations of the optimal value functions.

The two most used dynamic programming algorithms are value iteration and policy iteration. The policy iteration approach has two main steps, policy evaluation and policy improvement. During policy evaluation, the algorithm uses the state-value function in order to evaluate the policy by calculating the value of all the states. While in the policy improvement step, the objective is to find those states in which the policy can be improved. For a more detailed explanation we refer to [Puterman (1994)].

The value iteration approach does not need a policy evaluation step, which makes it a less computationally expensive approach. It works by using a simple backup operation to iteratively calculate the optimal value function, which has been shown to converge in [Bellman (1957)].

Dynamic Programming algorithms are known for their assumption of a perfect model of the environment and also because of their computational cost, but it is important to say that they provide an essential foundation for the understanding of

the RL-methods that will be explained in the next subsections. These methods can be seen as attempts to achieve the same effect as DP, but with less computation and without assuming a perfect model of the environment [Sutton & Barto (1998)].

3.4.2 Q-Learning

A well-known reinforcement learning algorithm is Q-Learning (QL), which is based on learning an action-value function that gives the expected utility of taking a given action in a given state. The core of the algorithm is a simple value iteration update, each pair (s, a) has a Q-value associated. When the action a is selected by the agent located in state s, the Q-value for that state-action pair is updated based on the immediate reward received when selecting that action, and the best Q-value for the subsequent state s'. The update rule for the state action pair (s, a) is the following:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{3.5}$$

3.4.2.1 Learning rate and discount factor

In this expression, $\alpha \in]0, 1]$ is the learning rate and r the reward or penalty resulting from taking action a in state s. The learning rate α determines 'the degree' by which the old value is updated. For example if the learning rate $\alpha = 0$, then nothing is updated at all. If, on the other hand, $\alpha = 1$, then the old value is replaced by the new estimate. Usually a small value is used for the learning rate, for example, $\alpha = 0.1$. The discount factor (parameter γ) has range value of 0 to 1 ($0 \leq \gamma \leq 1$). If γ is closer to zero, the agent will tend to consider only immediate reward. If it is closer to one, the agent will consider future reward to be more important.

Q-Learning has the advantage that it is proven to converge to the optimal policy in MDPs under some restrictions [Tsitsiklis (1994)]. The QL algorithm can be summarized as follows:

Algorithm 1 is used by the agents to learn from experience or training. Each episode is equivalent to one training session. In each training session, the agent explores the environment and gets the rewards until it reaches the goal state. The purpose of the training is to enhance the knowledge of the agent represented by the Q-values. More training will give better values that can be used by the agent to move in a more optimal way.

Algorithm 1 Q-Learning

Initialize Q-values arbitrarily
for each episode **do**
  Initialize s
  for each episode step **do**
    Choose a from s
    Take action a, observe state s' and r
    Update Q-value, $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$
  end for
end for

3.4.3 Learning Automata

Learning Automata (LA) represent a policy iteration approach that was introduced in the 1950s in the mathematical psychology domain [Bush & Mosteller (1955)]. Some years later, in the 1960s, it started to be used in engineering research [Tsetlin (1962)].

A Learning Automaton is an adaptive decision maker that keeps track of a probability distribution over its actions [Narendra & Thathachar (1974)]. Formally, it can be described as a quadruple (A, B, p, U), where $A = \{a_1, ..., a_r\}$ is the set of possible actions the automaton can perform, p is the probability distribution over these actions, B is the response set of the environment (values between 0 and 1), and U is the learning scheme used to update p.

Figure 3.3 shows a Learning Automaton in its environment. At each timestep it selects one of its actions according to its probability distribution. After taking the chosen action i, its probability p_i is updated based on the reward $r \in \{0, 1\}$, see Equation 3.6. The other probabilities p_j (for all actions $j \neq i$) are adjusted in a way that keeps the sum of all probabilities equal to 1 ($\sum_i p_i = 1$), see Equation 3.7. This algorithm is based on the simple idea that whenever the selected action results in a favorable response, the action probability is increased; otherwise it is decreased.

$$p_i \leftarrow p_i + \alpha r(1 - p_i) - \beta(1 - r)p_i \tag{3.6}$$

$$p_j \leftarrow p_j - \alpha r p_j + \beta(1 - r)\left(\tfrac{1}{n-1} - p_j\right), \quad \forall j \neq i. \tag{3.7}$$

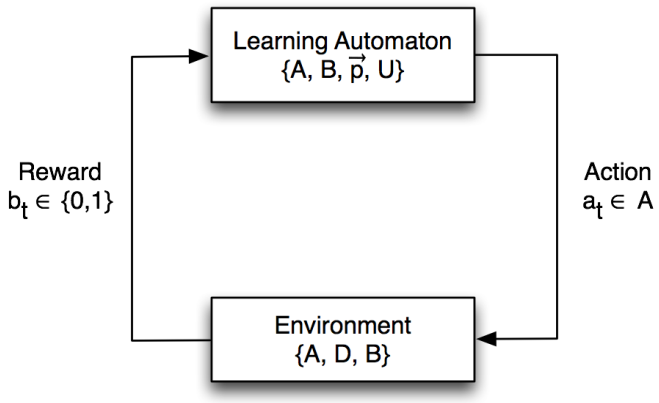

Figure 3.3: Learning Automaton in its environment.

The parameters α and β ($\alpha, \beta \in [0, 1]$) are the reward and penalty learning rate. In literature, three common update schemes are defined based on the values of α and β:

- Linear Reward-Inaction (L_{R-I}) for $\beta = 0$,
- Linear Reward-Penalty (L_{R-P}) for $\alpha = \beta$,
- Linear Reward-ϵ-Penalty ($L_{R-\epsilon P}$) for $\beta \ll \alpha$.

3.5 Action Selection Mechanisms

One of the challenges that arises in reinforcement learning is the trade-off between exploration and exploitation. To obtain a high reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task [Sutton & Barto (1998)]. The agent must try a variety of actions and progressively favor those that appear to be best. Proper control of the tradeoff between exploration and exploitation is important in order to construct an efficient learning method. We will briefly explain three commonly used action selection methods, *greedy*, *ϵ-greedy* and *softmax*.

If the agent decides to choose the best among the possible actions, then we can say that it is following a *greedy* action selection strategy. However, always choosing the best action may lead to suboptimal performance, depending on the variance of the action rewards.

An alternative to this greedy behavior is to follow the ϵ-*greedy* selection strategy. This action selection method instructs the agent to choose its best action most of the time, but sometimes, to choose an action at random (with equal probability for each possible action a in the current state s). The ϵ value determines the probability of choosing a random action. Although ϵ-greedy is an effective and popular strategy for balancing exploration and exploitation in reinforcement learning, one drawback is that when it explores it chooses equally among all actions [Sutton & Barto (1998)]. Meaning that it could choose between the worst appearing action and the next-to-best one with the same probability.

One alternative is to vary the action probabilities as a graded function of their estimated values, which is what the softmax action selection strategy does. The greedy action will still have the highest probability, but the other ones are ranked according to their value estimates. It means that the probability of selecting action a out of m possible actions is given by the Boltzmann distribution (see Equation 3.8), which is the most commonly used distribution when using this action selection mechanism.

$$Pr(a) = \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{m} e^{Q_t(b)/\tau}} \tag{3.8}$$

In this Equation τ is a positive parameter called *temperature*, which controls how greedily the agent will behave, m represents the number of available actions, and $Q_t(a)$ the estimate of action a at timestep t. High temperatures cause the actions to be all (nearly) equi-probable. Low temperatures cause a greater difference in selection probability for actions that differ in their value estimates, in other words, lower values of temperature will make the agent act more greedily.

It is possible to notice the similarities between the softmax action selection and the ϵ-greedy action selection, both methods have only one parameter to be set, but it is unclear which one could be better, as this may depend on the task being solved. It is also possible for both methods to decrease the value of their parameters over time (ϵ and τ), this means that the agents will explore more at the beginning of the learning phase, and will act more greedily (exploiting their knowledge) at the end.

Several experiments have been developed in order to determine which action selection strategy is better [Auer et al. (2002)]. The experiments have shown that different parameters for the exploration strategies result in very varying outcomes. Too little exploration leads to suboptimal results at the end of the learning process, whereas too much exploration leads to bad performance during the learning process and long learning times. The best choice is then problem dependent and there is no known way to automatically select the value of ϵ in ϵ-greedy methods.

3.6 Multi-Agent Systems

A Multi-Agent System (MAS) is a system in which several agents act in the same environment in order to accomplish a specific task. The increasing interest in using MAS for real world problems is because of their ability to solve problems that are too large for a centralized agent to solve, and also the ability to provide solutions where the expertise is distributed, like for instance, health care and manufacturing [Sycara (1998)].

The field of multi-agent systems is concerned with decentralized processes (distributed systems), as each individual agent in the system has its own perception (they can be in different locations or responsible for different parts of the system), control (different expertise), and actuation (different potential actions) [Kaminka (2004)]. All these characteristics can be summarized in the following definition:

Definition 7 (Multi-Agent System)**.** *A Multi-Agent System is a loosely coupled network of agents that work together to solve problems that are beyond the individual capabilities or knowledge of each agent [Jennings et al. (1998)].*

The main characteristics of Multi-Agent Systems are:

- each agent has incomplete information, or capabilities for solving the problem, thus each agent has a limited viewpoint;
- there is no global system control;
- data can be decentralized; and
- computation can be asynchronous.

There are two possible scenarios when dealing with multiple agents, they can either work together, trying to achieve the same common goal, or they can have

their own goals and conflicting interests, which means that we could have either joint or independent learners. When using joint learners there is the assumption that actions taken by the other agents can be observed. Independent learners do not need to observe the actions taken by other agents.

Despite the extra level of complexity that comes with the use of several agents, we can identify the following benefits [Sycara (1998)]:

- *computational efficiency*: because concurrency of computation is exploited (as long as communication is kept minimal, for example, by transmitting high-level information and results rather than low-level data);
- *extensibility*: the number and the capabilities of the agents working on a problem can be altered;
- *robustness*: the system‘s ability to tolerate uncertainty, because suitable information is exchanged among agents; besides, as there are several agents acting in the same environment, if one of these agents fails, the entire system will not fail;
- *maintainability*: a system composed of multiple components-agents is easier to maintain because of its modularity;
- *responsiveness*: modularity can handle anomalies in a local way, avoiding their propagation to the whole system;
- *flexibility*: agents with different abilities can adaptively organize to solve the current problem;
- *reuse*: specific agents can be reused in different agent teams to solve different problems.

Some examples of fields where Multi-Agent Systems have been successfully applied are:

- autonomous robots
- elevator control
- wireless collaboration and communications
- supply-chain management
- aircraft maintenance

- transportation logistics

Figure 3.4, taken from [Nowé et al. (2012)], depicts a typical model of Multi-Agent Reinforcement Learning (MARL). Multiple agents are connected to the same environment, where the next state is given by the result of the combination of the actions taken by all the agents.

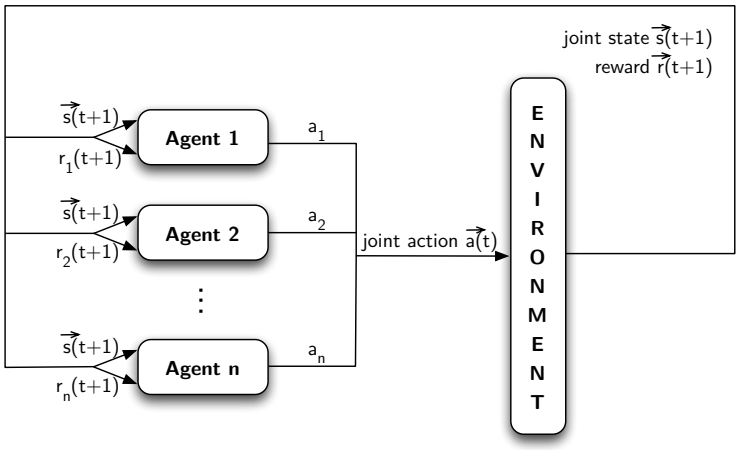

Figure 3.4: Multiple agents acting in the same environment.

An extension of the single agent MDP framework (Definition 5) to the multi-agent case can be defined as a Markov Game [Shapley (1953)]. In this case actions and states are the joint result of multiple agents choosing an action independently. While they were originally introduced in game theory and called stochastic games, Markov Games were more recently proposed as a standard framework for multi-agent reinforcement learning [Littman (1994)].

Definition 8 (Markov Game)**.** *A Markov Game is a tuple* $(m, S, A_{1,...,m}, T, R_{1,...,m})$*, where:*

- m *is the number of agents*
- $S = \{s^1, ..., s^N\}$ *is a finite set of system states*
- A_k *is the action set of agent* k
- $T : S \times A_1 \times ... \times A_m \times S \to [0, 1]$ *is the transition function*
- $R_k : S \times A_1 \times ... \times A_m \times S \to \mathbb{R}$ *is the reward function of agent* k

Note that $A_k(s^i)$ is now the action set available in state s^i for agent k, with $k : 1, ..., m$ and $i : 1, ..., N$. Transition probabilities $T(s^i, \vec{a^i}, s^j)$ and rewards $R_k(s^i, \vec{a^i}, s^j)$ now depend on a starting state s^i, ending state s^j and a joint action from state s^i, i.e. $\vec{a^i} = (a^i_1, ..., a^i_m)$ with $a^i_k \in A_k(s^i)$. The reward function is now individual to each agent k. Different agents can receive different rewards for the same state transition. Transitions in the game are again assumed to obey the Markov property.

An extension of this multi-agent framework when all the agents share the same reward function is called Multi-Agent Markov Decision Process (MMDP).

Definition 9 (Multi-Agent Markov Decision Process, MMDP)**.** *A multi-agent Markov decision process (MMDP) is a cooperative Markov Game* $(m, S, A_{1,...,m}, R, T)$ *in which all agents share the same reward function* R*.*

Until now we have assumed that each agent has full knowledge about the global system state, meaning that it is aware of the state of the other agents and the actions they are taking. However, with these assumptions we are losing an important characteristic of the MASs, the decentralized control.

There are cases where a group of agents is working together trying to maximize the utility of the team as a whole, but these agents only have a partial view of the system state and none of them is able to influence the whole system state with its actions. In order to study decentralized decision-making under the mentioned conditions, [Bernstein et al. (2002)] proposed the framework of Decentralized Partially Observable Markov Decision Processes.

Definition 10 (Decentralized Partially Observable Markov Decision Process)**.** *A Decentralized Partially Observable Markov Decision Process (DEC-POMDP) is defined as a 7-tuple* $M = [Ag, S, A, T, R, \Omega, O]$ *with:*

- $Ag = \{1, ..., m\}$ *as the set of* m *agents;*
- S *as the finite set of states;*
- $A = A_1 \times ... \times A_m$ *as the set of joint actions to be performed by the agents, where* A_i *is the set of actions available to agent* i*. Every time step, the agents take one joint action* $\vec{a} = (a_1, ..., a_m)$*, but agents do not observe each other's actions;*
- T *as the transition function,* $T(s, \vec{a}, s')$ *denotes the probability that the system will arrive at state* s' *after the execution of the joint action* $\vec{a}$ *in state* s*;*

- *R as the reward function, $R(s, \vec{a}, s')$ denotes the reward for executing $\vec{a}$ in s and transitioning to s';*
- *$\Omega = \Omega_1 \times ... \times \Omega_m$ as the set of observations of all the agents. $o = (o_1, ..., o_m) \in \Omega$ denotes a joint observation, with $o_i \in \Omega_i$ as the observation of agent i (it only observes its own component o_i);*
- *O as the observation function. $O(o|s, \vec{a}, s')$ is the probability of observing o when taking the joint action a in state s and entering the new state s'.*

In a DEC-POMDP, the process is controlled by multiple distributed agents, each with possibly different information about the state. There is also another multi-agent extension of an MDP, called a decentralized Markov Decision Process (DEC-MDP). A DEC-MDP is a DEC-POMDP with the restriction that at each time step the agents' observations together uniquely determine the state. The MDP, POMDP, and DEC-MDP can all be viewed as special cases of the DEC-POMDP [Bernstein et al. (2002)]. The relationships among the models can be seen in Figure 3.5.

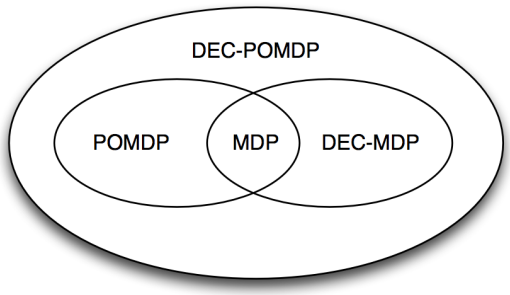

Figure 3.5: Relationship between the different MDP models.

The literature on DEC-POMDPs typically assumes that the world state can be factored into components relating to individual agents. This means that it is possible to separate features of the world state belonging to one agent from those that belong to others. Such a factorization is strict in the sense that a single feature of the global state can correspond to one agent only.

Definition 11 (Factored DEC-POMDP)**.** *A factored, m-agent DEC-POMDP is defined such that the set S of states can be factored into m agent-specific components: $S = S_1 \times ... \times S_m$.*

In [Gabel & Riedmiller (2008)], the authors identify a subclass of general DEC-MDPs that features regularities in the way the agents interact with each other. In the application domain of scheduling, more specifically job-shop scheduling (which is one of the problems being addressed in this dissertation), jobs are composed of a number of operations that have to be processed in a specific order on different machines. Here, it is natural to think of these operations as the actions and, logically, their availability depends on whether all predecessor operations have already been processed or not. Generalizing this idea, in [Gabel & Riedmiller (2008)] the authors define what it means for an agent (a machine) to dispose of changing action sets by proposing the following definition:

Definition 12 (Decentralized MDP with Changing Action Sets, DEC-MDP-CAS)**.** *An m-factored DEC-MDP with factored state space $S = S_1 \times ... \times S_m$ is said to feature changing action sets if the local state of the agent i is fully described by the set of actions currently selectable by that agent: $s_i = A_i \backslash \{a_0\}$.*
The set of all potentially executable actions of agent i is denoted by $\mathcal{A}_i = \{a_0, a_{i1}...a_{ik}\}$, such that it holds $A_i \subset \mathcal{A}_i$, and a_0 represents a null action that does not change the state and is always in A_i. Subsequently, we abbreviate $\mathcal{A}_i^r = \mathcal{A}_i \backslash \{a_0\}$, where r represents the resource to which the agent is associated.

This definition is useful for the learning methods that will be introduced in the next chapter. During the scheduling process the machines have queues associated, meaning that its corresponding agent will have to choose one operation to execute among several ones waiting to be processed, and these queued operations are the ones that will define the state of the agent.

Table 3.2 shows a comparison among all the MDP models described in this chapter, in terms of the number of agents, the knowledge they have, the number of states and the payoff received.

Figure 3.6 shows an example of two agents with their corresponding set of actions, which define their states at that specific moment. It means that, for example, for the agent associated to machine one (M_1, left hand side), its current state is:
$s_1 = \{Job_2 - Op_1, Job_4 - Op_1, Job_6 - Op_1\}$. Similarly, the status of the agent associated to machine two (M_2, right hand side) is defined as:
$s_2 = \{Job_1 - Op_1, Job_3 - Op_1, Job_5 - Op_1\}$, plus the 'remain idle' action, which is available at every time step for every agent.

Let us say that the agent associated to M_1 decides to choose $Job_4 - Op_1$, which

Table 3.2: Comparison between different representations of agents systems.

Agent	agents		knowledge		states		observability		payoff	
System	single	multiple	global	local	single	multiple	full	partial	common	individual
MDP	✓		✓			✓	✓		✓	
POMDP	✓		✓			✓		✓	✓	
DEC-MDP		✓		✓		✓	✓		✓	
DEC-POMDP		✓		✓		✓		✓	✓	
MG		✓	✓			✓	✓			✓
MMDP		✓	✓			✓	✓		✓	
DEC-MDP-CAS		✓		✓		✓	✓		✓	
DEC-MG-CAS		✓		✓		✓	✓			✓

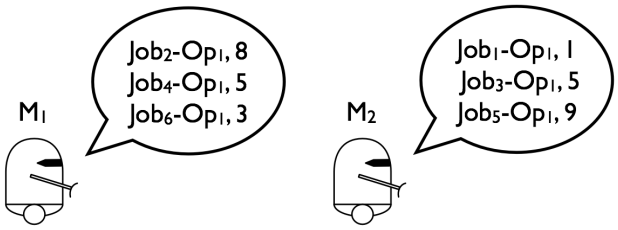

Figure 3.6: Two agents with their corresponding set of actions, which constitute their current state.

will take 5 time units, this means that after finishing this operation, it can be removed from its current set of actions, changing its state. But this also means that the next operation of the job to which that operation belongs ($Job_4 - Op_2$) can be released for execution, affecting the state of another agent. This is what is named transition dependencies [Gabel (2009)], the fact that one agent can affect the state of other agents with its actions.

Definition 13 (Transition Dependencies)**.** *A factored m-agent DEC-MDP with changing action sets is said to have partially ordered transition dependencies if there exist functions σ_i for each agent i such that:*

- $\sigma_i : A_i^r \rightarrow Ag \cup \{\emptyset\}$

Ag is the set of agents and A_i^r is the set of all the possible actions the agent in resource i can execute, with $i \in \{1, ...m\}$ (see Definition 12).
$\sigma_i(a) = j$ means that agent i takes action a, and this action affects the state of agent j, action a is then added to $A_j (A_j = A_j \cup a)$ and it is removed from $A_i (A_i = A_i \backslash a)$.

In short, the dependency functions σ_i will indicate which other agent's state will be affected when taking a specific action. These dependency functions are said to be partially ordered because it is known in which order the operations of one job have to be executed, but the order in which the agents will choose these operations is up to the policy being followed.

Using again the example shown in Figure 3.6, let us assume that the second operation of Job 4 ($Job_4 - Op_2$) has to be executed on machine two, it means that the corresponding dependency function will be $\sigma_1(Job_4 - Op_1) = 2$, and after the execution of the action, the new state of the agent associated to M_2 will be:
$s_2 = \{Job_1 - Op_1, Job_3 - Op_1, Job_5 - Op_1, Job_4 - Op_2\}$.

If we quickly think about the fact that even in simple scheduling scenarios it is possible to have more than 10 jobs and/or 10 machines, then we will see why optimal solution methods can hardly be applied. Even the case where only two agents are present has been identified as NP-hard [Garey et al. (1976)]. In this dissertation our objective is to maintain the same basic idea for solving different types of scheduling problems. By basic idea we mean the fact of having one agent per resource, each agent having a partial view of the system state and taking actions based on this information. This will then be adjusted according to specific features of the problem at hand, with the objective of finding high quality solutions, which can be in the vicinity of the optimal ones, depending on the degree of robustness wanted by the user.

3.7 Reinforcement Learning for Scheduling

After presenting the basic idea of Reinforcement Learning, the definition of the different MDP models and the relationship between them, this section will present the main concepts that have to be taken into account when solving a scheduling problem using RL.

Figure 3.7 shows an initial idea of the learning environment. Following the standard RL model presented in Figure 3.1, this is how we map agents and actions in the environment when solving a scheduling problem.

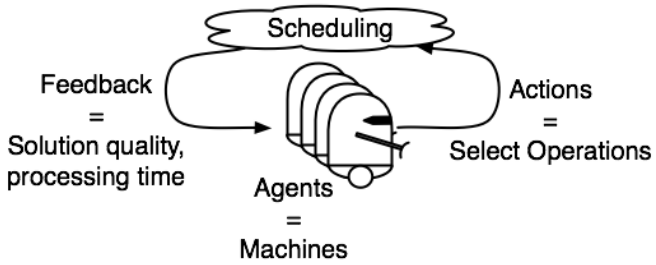

Figure 3.7: Agents in a scheduling environment.

The scheduling environment defines the number of agents in the system and the relations among them. Agents may not know the global state of the system. To achieve better performance of agents or the system, agents communicate with each other to determine their actions based on the limited information. Clearly,

the centralized approach is applicable to problems in which global information is available and agents are cooperative. Problems in which some agents want to keep their information private for competitive or other reasons call for distributed methods ranging from coordination among cooperative agents to negotiation between competitive agents [Wu et al. (2005)]. For a graphical representation see Figure 3.8.

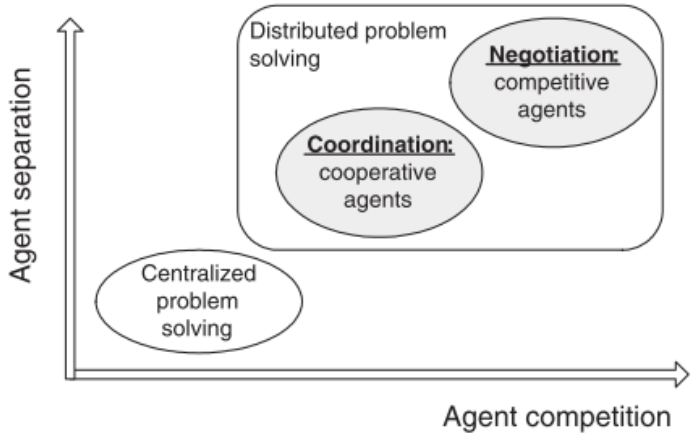

Figure 3.8: Approaches relating to the problem space, taken from [Wu et al. (2005)]

From all the different scheduling problems we will tackle in this dissertation, the job shop scheduling problem (section 2.4) will be used as the basic problem, as all the other types are mainly variants or extensions of it. Job shop scheduling problems are well suited to be modeled using factored m-agent DEC-MDPs with changing action sets and partially ordered transition dependencies, as the following information is always available at the beginning of the scheduling process:

- Number of machines
- Number of jobs
- Processing order of the operations belonging to the different jobs (ordering constraints)
- Processing time of each operation

The main characteristics that have to be considered when modeling the problem are adapted from [Gabel (2009)] and can be summarized as follows:

- **Factored World State:** The world state of a job-shop scheduling problem J can be factored: We assume that each resource has one agent i associated that observes the local state at its resource and controls its behavior. Consequently, there are as many agents as resources in the JSSP ($|Ag| = |R| = m$).
- **Local Full Observability:** The local state s_i of agent i, hence the situation of resource r_i, is fully observable. Additionally, the composition of all resources fully determines the global state of the scheduling problem. Therefore, the system is jointly observable, i.e. it is a DEC-MDP.
- **Factored Actions:** Actions correspond to the starting of jobs' operations (job dispatching). So, a local action of agent i reflects the decision to further process one particular job (more precisely, the next operation of that job) out of the set A_i of operations currently waiting at r_i.
- **Changing Action Sets:** If actions denote the dispatching of waiting operations for further processing, then the set of actions available to an agent varies over time, since the set of operations waiting at a resource changes.

  A_i^r corresponds to the set of operations $o_{j,k}$ which must be processed on resource r_i, i.e. $\varrho(o_{j,k}) = i$, where j is the job to which the operation belongs and k is the operation id. Furthermore, the local state s_i of agent i is fully described by the changing set of operations currently waiting at resource r_i for further processing, thus, $s_i = A_i$.
- **Dependency Functions:** As mentioned before, the order and the resources on which the operations belonging to a job must be processed in a job shop scheduling problem is known. These orders imply that, after one agent executes an action (processes one operation), the local state of maximally one further agent is influenced. Let $a \in A_i$ be an operation from job k, which is currently being processed by resource r_i. Once this operation is finished, the action set A_i of agent i is adapted according to $A_i = A_i \backslash \{a\}$, whereas the action set of agent $i' = \varrho(o_{k,a+1})$ is extended ($A_{i'} = A_{i'} \cup \{a\}$).

  Therefore, we can define the dependency functions $\sigma : A_i^r \rightarrow Ag \cup \{\emptyset\}$ for all the agents i (and resource r_i, respectively) as follows:

$$\sigma_i(a) = \begin{cases} \varrho(o_{k,a+1}) & \text{if } \exists a \in \{1, ...o_m - 1\} : \varrho(o_{k,a}) = i \\ \emptyset & else \end{cases} \tag{3.9}$$

where a corresponds to the operation id within job k, which has to be processed on resource r_i, i.e. k such that $\varrho(o_{k,a}) = i$, and o_m is the number of operations job k has.

There are two particular cases for which the function will return the empty set, one when the selected action is to remain idle and another one when the selected action is the last operation of the corresponding job.

1. $\sigma_i(a) = \emptyset : A_i = \{A_i \backslash a\}$: If the selected action differs from 'stay idle' (a_0), the operation is removed from the set of actions of agent i ($A_i = \{A_i \backslash a\}$), and as the action does not affect the state of any further agent, this is the only modification being performed.
2. $\sigma_i(a_0) = \emptyset : A_i = \{A_i\}$: If the agent decides to stay idle, then its set of actions will remain the same. This could also be read as $\sigma_i(a_0) = i$.

These are the basic definitions that are common for all the scheduling problems described in Chapter 2. This basic model will then be adjusted depending on the additional constraints from each particular case. All these adaptations will be explained in detail in the next chapter.

3.8 Summary

In this chapter we presented an overview of Reinforcement Learning. We first introduced the concept of *agent*, together with the main characteristics that an agent needs to posses in order to meet its design objectives. Then we explained how the agent interacts with its environment in order to learn how to solve a specific task. The underlying framework of RL, the Markov Decision Process, was also described. Different solution methods were introduced, as well as different methods for selecting actions to overcome the exploration - exploitation dilemma. In the last part of the chapter we moved from a single agent to multiple agents, the concept of *multi-agent system* was defined, as well as extensions of the single agent MDP to the multi-agent case. Section 3.7 presented the main concepts that have to be taken into account when modeling a scheduling problem as a Decentralized Markov Decision Process.

The next chapter will describe how the basic model can be adjusted in order to solve more challenging scheduling problems, in which additional constraints are present.

Chapter 4

A Generic MARL Approach for Scheduling Problems

Scheduling is a very active research field with a high practical relevance. For a long time, manufacturing environments have been known for requiring distributed solution approaches in order to find high-quality solutions, because of their intrinsic complexity and, possibly, due to an inherent distribution of the tasks that are involved [Wu et al. (2005)]. Accordingly, the naturally distributed character of multi-agent systems may be exploited in a purposive manner when addressing scheduling problems [Gabel (2009)].

In this chapter we introduce our learning approach in detail, which is initially defined for the JSSP. After this we will gradually move to more challenging scheduling problems, showing how the approach can easily be adapted in order to satisfy the extra constraints of different scheduling scenarios.

In this chapter we start by implementing a Q-Learning algorithm (introduced in Subsection 3.4.2), which is one of the most widely used RL-algorithms and which does not require an explicit model of the environment. In the first section we will define the different components of the algorithm, and its first steps will be explained through an example.

4.1 Applying QL to solve the JSSP

In the previous chapter we already defined the basic concepts to take into account when modeling a scheduling problem as a DEC-MDP, where an agent k is associated

to each of the m resources (machines). It is important to remember that an agent cannot take an action at each discrete time step t, but only after its resource has finished the execution of the current operation, because each resource can only process one operation at a time and, besides, each job can only be processed by a single machine at any given time, meaning that only one of its operations can be in execution at time t.

The problem instances (introduced in subsection 2.4.2) are defined in such a way that after an operation is completely processed on one resource, we already know to which resource the next operation of the corresponding job should be sent, what should be decided is the order in which each of the resources will execute the operations that it has assigned. Maybe an operation that arrives later in the queue of the machine should be selected first.

Using as example the instance ft06 (Figure 4.1), and looking at the first column (first operation of each job), it can be seen that initially there are only two machines which are able to execute an operation: M_2, which can select between the first operation of J_0, J_2 or J_4, and M_1, which is able to execute J_1, J_3 or J_5. The rest of the machines will remain idle waiting for these operations to be finished on their corresponding machines.

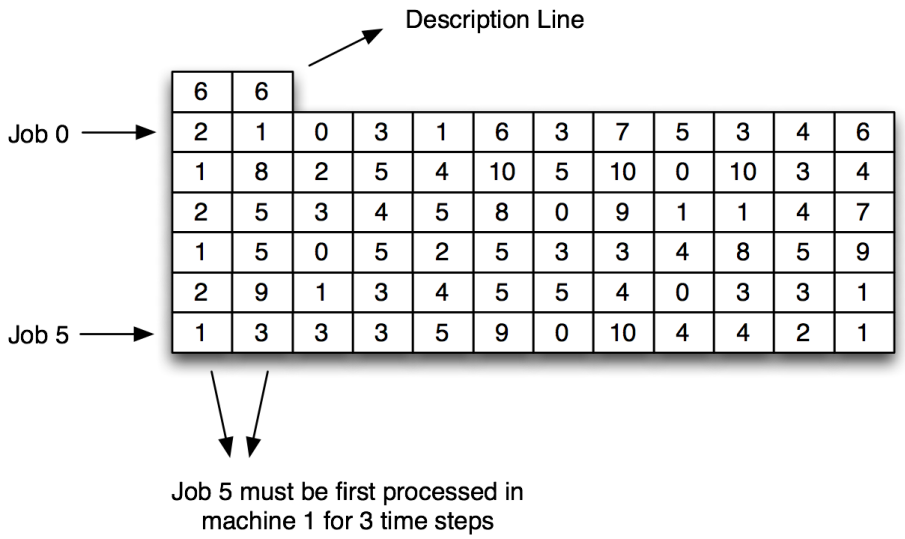

Figure 4.1: JSSP Instance ft06, composed by 6 jobs and 6 machines.

At each time step, the possible actions an agent can select take the problem constraints into account, such that only operations that can be processed at that time can be chosen. Basically, we are keeping two sets; one containing the resources

which are able to process an operation (agents able to select an action); and another one containing the operations that each agent can select as its next action, which is also what defines its current state.

These sets are continuously updated, because once an operation is finished on one resource, the status of the system changes. More specifically, the queue assigned to the resource where the next operation of the corresponding job will be executed is affected. Hence the set of operations that the agent can select is modified. This is given by the concept of transition dependencies introduced in Definition 13.

In this basic case jobs do not have release and/or due dates, that is why the objective of the agents is to minimize the makespan (C_{max}), which is the completion time or, in other words, the time it takes to complete all the operations.

When choosing the QL algorithm as solution method, there are important elements to be decided, which are summarized as follows:

- **States and actions:** As mentioned in previous sections, there is an agent associated to each resource, and this agent will make decisions about future actions. For an agent taking an action means deciding which operation to process next from the set of currently available ones (the set of currently waiting operations at the corresponding resource). Each agent has a local view, meaning that it only has information about its associated resource and the operations waiting there.
- **Action Selection Strategy:** In this thesis the action selection mechanism that will be used is the ϵ-greedy strategy (described in Section 3.5) as it has been successfully used in multi-agent environments [Rodrigues Gomes & Kowalczyk (2009)].
- **Q-Values:** Let us once again take the instance ft06 as example (Figure 4.1), in this case there are 6 agents (one per resource) and 6 jobs involved, according to the constraints of the JSSP, each agent will execute 6 actions (one operation from each of the 6 jobs). According to [Gabel (2009)], the set of states for agent i is defined as: $S_i = P(\mathcal{A}_i^r)$[1], this gives raise to $|S_i| = 2^6 = 64$ local states for every agent i, which results in an upper limit of $|S| \leq (2^6)^6 = 2^{36}$ possible system states.

  Due to the ordering constraints of the problem, many of those states can

[1] P(x) denotes the power set of x

actually never be reached. Therefore, our algorithm only stores those states that are needed, by this we mean the combinations of operations that can be at the same time in the queues of the system. These combinations are stored as they appear. For example, if the algorithm is executed only for one iteration, then only 6 states will be stored, which are the states where the agent was located in when the actions were chosen. Other executions might lead to the creation of other states.

- **Feedback Signal:** There are different possible feedback signals that can be used when solving a scheduling problem. For example, in [Gabel & Riedmiller (2007)] the authors use a cost function as feedback to the agent, this cost is defined as the number of operations present, at that point in time, in the queues of the whole system. This means that high costs are incurred when there are many operations waiting for further processing in the system and, hence, the overall utilization of the resources is poor. Note that we are using costs as reward signal, meaning that the lower the cost the better the action. In the next subsection (4.1.1) we will study different cost or reward functions that can be used in the update rule of the QL algorithm.

4.1.1 Feedback Signals

As a starting point, we use as feedback signal the cost proposed in [Gabel & Riedmiller (2007)], which is based on the idea that a makespan of a schedule is minimized if as few as possible resources with queued jobs are in the system. The cost function which is used in the Q-Learning update rule is defined as the number of jobs that are in the queues of the system. The update rule of the QL algorithm is then:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[c(s,a,s') + \gamma \min_{b \in A(s')} Q(s',b)] \tag{4.1}$$

where α is the learning rate, γ is the discount factor, and the function c is defined as follows:

$$c(s,a,s') = \sum_{k=1}^{m} \{j | j \; queued \; at \; k\} \tag{4.2}$$

The successor state s' is given by the current state of the agent (s), the action it takes (a) and the transition dependencies (based on the ordering constraints). It is important to remember that we are using costs instead of rewards, then, small

Q-Values correspond to 'good' state actions pairs, therefore we use the minimum operator in Equation 4.1.

We propose two other cost functions: the first one uses the processing times of the operations as feedback signal. In this case, the immediate reward (cost in this case) is the processing time of the operation that is selected to be processed next (based on the idea of the SPT dispatching rule, defined in Section 2.10.1). The second one is based on the cost function proposed in [Gabel & Riedmiller (2007)], but instead of using the number of operations in the queues of the whole system, we use the sum of the expected processing times of these queued operations. At some point during the scheduling process the length of the queues can be the same, but the sum of the processing times of the operations can be very different. Hence, this function better represents the overall state of the system.

One main advantage of our approach is that it is also possible to use other dispatching rules and even composite dispatching rules as feedback signal for the agents.

Besides the idea of using cost functions as feedback signals, we also propose another point of view. In this case we give a common reward to the agents based on the quality of the solution they are able to obtain. Once a solution has been found we compare it to the previous best solution, if the makespan is lower then the agents receive a reward of 1 for those state-action pairs that were involved.

$$r = \begin{cases} 0 & \text{if } C_{max} > C_{best}, \\ 1 & \text{otherwise,} \end{cases} \tag{4.3}$$

where C_{max} is the makespan, as defined in Chapter 2, and C_{best} is a variable keeping the best solution found so far.

4.1.2 Example

As mentioned before, according to the data in the instance ft06 (Figure 4.1), only machines M_1 and M_2 can perform an action during the initial step, then, the set of possible agents to select an action and the sets with the possible operations will be as follows:

Possible_Agents = $\{M_1, M_2\}$
M_1_possible_operations = $\{J_1O_0, J_3O_0, J_5O_0\}$
M_2_possible_operations = $\{J_0O_0, J_2O_0, J_4O_0\}$

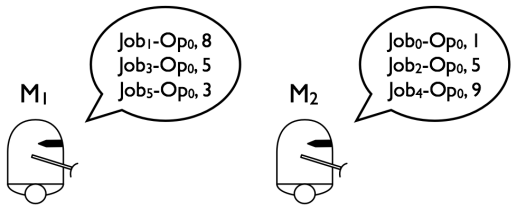

Figure 4.2: Agents choosing from their corresponding sets of currently selectable operations.

Let us say that the agent located in machine M_2 decides to select action J_0O_0, and the agent associated to machine M_1 selects J_3O_0, then, once these operations are processed, the previous sets will change according to the problem constraints. The next operations of the jobs involved in the previous process (J_3O_1 and J_0O_1) can be sent to the resources where they need to be executed. This means that current states of the agents in action will be updated as follows:

Possible_Agents = $\{M_1, M_2, M_0\}$

M_1_possible_operations = $\{J_1O_0, J_5O_0\}$ - J_3O_0 is removed from the state.

M_2_possible_operations = $\{J_2O_0, J_4O_0\}$ - J_0O_0 is removed from the state.

M_0_possible_operations = $\{J_0O_1, J_3O_1\}$ - Agent M_0 is able to start processing.

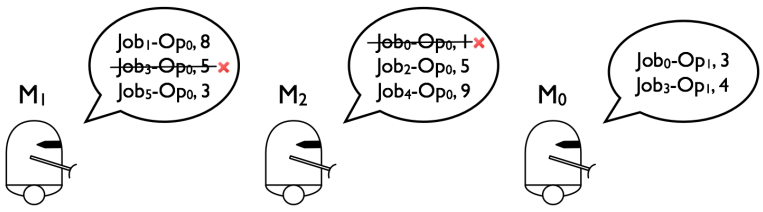

Figure 4.3: Updating the states of the agents after one action selection step.

Until this point we have been only constructing non-delay schedules, which means that machines were never kept idle if there was an action to execute. Going back to one important issue mentioned in Chapter 2, it is possible that the optimal schedule is not a non-delay one, see Figure 4.4 for example.

In that case, we will never obtain the optimal solution by only looking at non-delay schedules. That is why we include a 'stay idle' action, which will not have any effect on the state of the agent, giving the possibility to introduce some idle time

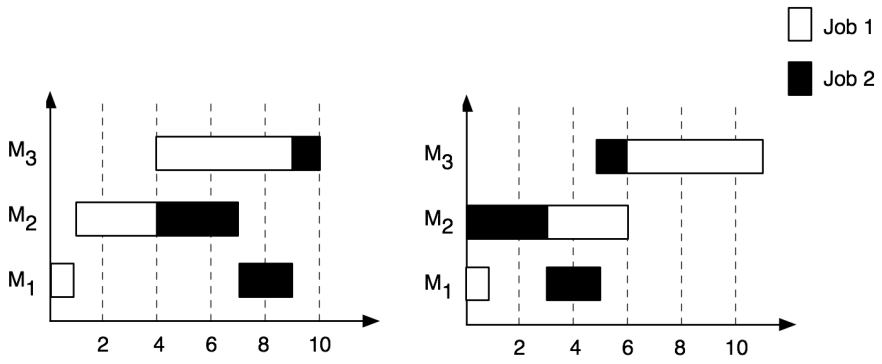

Figure 4.4: Left: Optimal schedule. Right: Non-delay schedule

and as a consequence, delay schedules will be obtained. Assuming that we have the instance shown in Table 4.1, Figures 4.5 and 4.6 show the process of constructing a non-delay and a delayed schedule, respectively.

Table 4.1: Instance with 3 resources and 2 jobs.

Job	**Op$_1$**	**Op$_2$**	**Op$_3$**
1	M_1,1	M_2,3	M_3,5
2	M_2,3	M_1,2	M_3,1

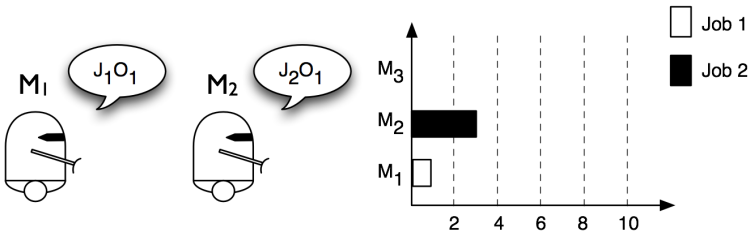

Figure 4.5: Agents acting in an environment where only non-delay schedules are obtained.

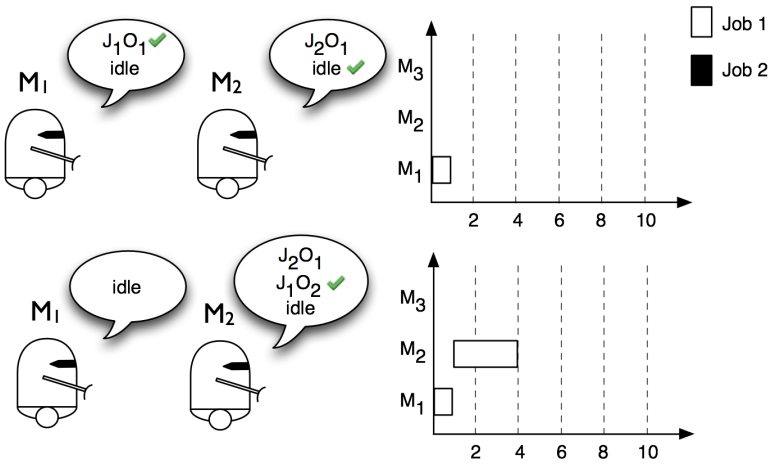

Figure 4.6: Agents acting in an environment where delay schedules can be obtained.

4.1.3 Experimental Results QL-JSSP

In order to measure the performance of the different versions of the algorithm and to compare it with the results reported by other approaches, several benchmark problems from the OR-Library [Beasley (1990)] were used. The OR-Library is a library of problem instances covering various Operations Research problems (See subsection 2.4.2). Table 4.2 summarizes the different instances that will be used as well as the best known solutions that have been reported in literature.

Some initial experiments were performed in order to analyze the learning process under the effect of different parameter values. Typical combinations for the QL algorithm are the following:

- alpha = 0.1, gamma = 0.8, epsilon = 0.2
- alpha = 0.1, gamma = 0.9, epsilon = 0.1
- alpha = 0.1, gamma = 0.8, epsilon = 0.1

After executing different experiments we decided to keep the third combination, alpha = 0.1, gamma = 0.8, epsilon = 0.1, as it was able to yield better results.

Table 4.2: Job Shop Scheduling instances and their optimal solutions, in terms of makespan.

6x6 (6 jobs and 6 machines) ft06 - Optimum 55

10x5 (10 jobs and 5 machines)	**15x5 (15 jobs and 5 machines)**
la01 - Optimum 666	la06 - Optimum 926
la02 - Optimum 660	la07 - Optimum 890
la03 - Optimum 597	la08 - Optimum 863
la04 - Optimum 590	la09 - Optimum 951
la05 - Optimum 593	la10 - Optimum 958

20x5 (20 jobs and 5 machines)	**10x10 (10 jobs and 10 machines)**
la11 - Optimum 1222	la16 - Optimum 945
la12 - Optimum 1039	la17 - Optimum 784
la13 - Optimum 1150	la18 - Optimum 848
la14 - Optimum 1292	la19 - Optimum 842
la15 - Optimum 1207	la20 - Optimum 902

For example, Figure 4.7 shows the learning process when solving the instance ft06 (6 jobs and 6 machines). Similarly, Figure 4.8 shows the learning process when solving the instance la01 (10 jobs and 5 machines). The optimal solutions for these instances are $C_{max} = 55$ and $C_{max} = 666$ respectively.

In the example shown in Figure 4.8 it is possible to see that the algorithm needed more than 700 iterations in order to reach the optimal solution.

Table 4.3 shows the results obtained by our algorithm under the different settings mentioned before. The first column shows the instances, grouped by number of jobs and number of machines, the column 'Optimum' presents the optimal solutions reported in the OR-Library. 'SPT', 'Queued jobs' and 'Sum proc. times' are columns corresponding to the use of the different cost functions described at the beginning of the section. Common Reward is the second point of view that was introduced, where agents receive the same payoff. Numbers in bold indicate that the optimal solution was reached, while underlined numbers indicate the best found solution. Part of these results were reported in [Martínez Jiménez (2008)].

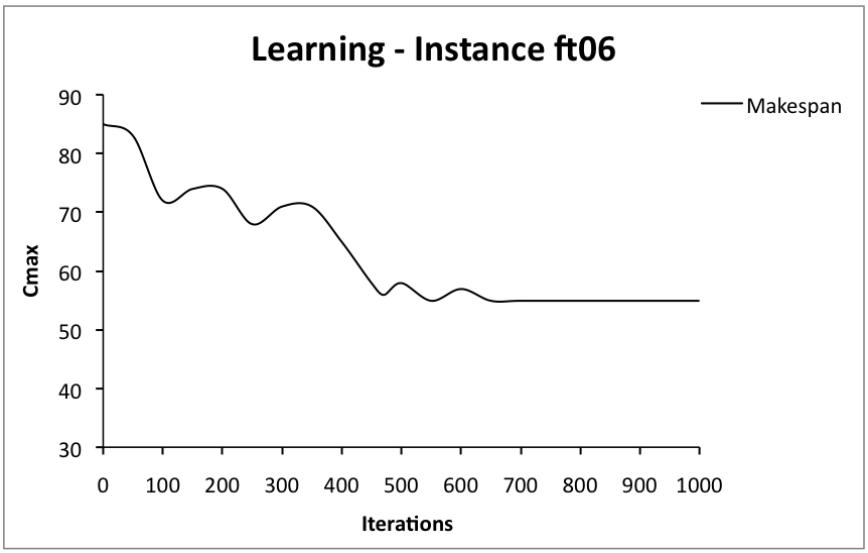

Figure 4.7: Learning - Instance ft06 - optimal solution $C_{max} = 55$

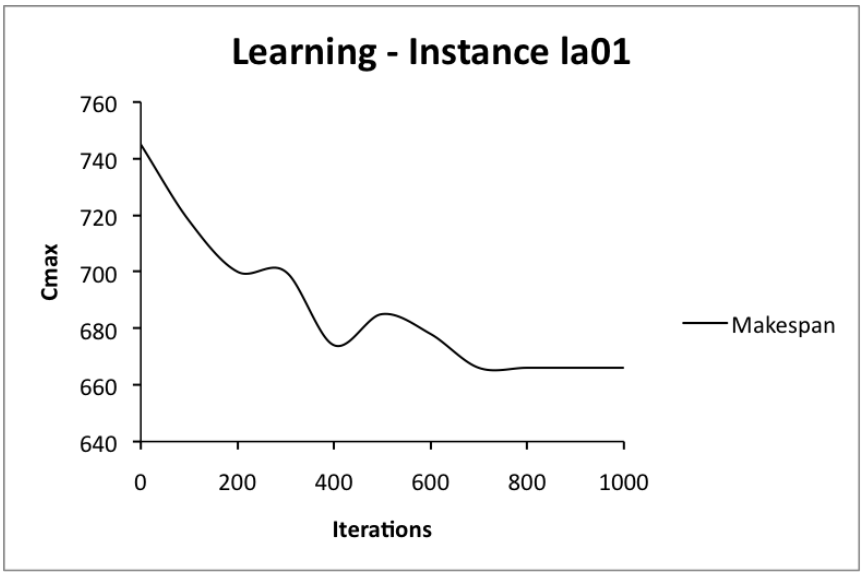

Figure 4.8: Learning process instance la01 - optimal solution $C_{max} = 666$

After these experiments we decided to select our best overall result, in order to compare with the results reported by other approaches in literature. Even though the differences were not big for the selected group of instances, in this case the best overall results were obtained by the common reward approach, as it succeeds in finding the best solutions for almost all the instances, except for the case of instance la04. Based on the experiments we can make the following observations:

When using the processing time as feedback signal, the agents are basing their decision on the time it takes to process the possible actions (operations) it can

Table 4.3: Comparison between the different variants of the algorithm.

	Instance	Optimum	SPT	Queued jobs	Sum proc. times	Common Reward
10x5	la01	666	**666**	**666**	**666**	**666**
	la02	655	673	673	667	667
	la03	597	627	610	613	610
	la04	590	611	613	611	613
	la05	593	**593**	**593**	**593**	**593**
15x5	la06	926	**926**	**926**	**926**	**926**
	la07	890	**890**	**890**	**890**	**890**
	la08	863	865	**863**	**863**	**863**
	la09	951	**951**	**951**	**951**	**951**
	la10	958	**958**	**958**	**958**	**958**
20x5	la11	1222	**1222**	**1222**	**1222**	**1222**
	la12	1039	**1039**	**1039**	**1039**	**1039**
	la13	1150	**1150**	**1150**	**1150**	**1150**
	la14	1292	**1292**	**1292**	**1292**	**1292**
	la15	1207	1271	1269	1262	1259

execute.

If the agent has multiple actions and several of them always end up in the same amount of queued jobs, the agent can not identify which one could lead to a better solution quality, as this number is not very informative.

The sum of the processing times of the queued operations makes a differentiation by telling the agent not the amount of queued jobs, but how long they will take, which better represents the real state of the system.

The last variant makes use of a common reward, which enforces cooperation. In this case the agents receive a common reward based on the solution quality, making more clear for the agent which action was leading to better solutions.

Table 4.4 shows the comparative study between our algorithm and three other approaches. The column 'TS-ACO' refers to the use of a Two Stage Ant Colony Optimization approach, which was proposed in [Puris et al. (2007)], 'GA' corresponds to a Genetic Algorithm proposed in [Hasan et al. (2007)], and 'ACO' to a classical

Ant Colony Optimization Approach, introduced in [Ventresca & Ombuki (2004)].

Instances are again grouped per number of jobs and machines and the Mean Relative Error is shown at the bottom of the results. The relative error (RE) is defined as RE =[(MK -LB)/LB * 100], where MK is the best makespan obtained by the reported algorithm and LB is the best-known lower bound. The MRE takes into account the average of the results for the whole group of instances.

Table 4.4: Comparative study for the JSSP.

	Instance	**Optimum**	**TS-ACO**	**GA**	**ACO**	**QL**
	la01	666	**666**	**666**	**666**	**666**
	la02	655	673	**655**	666	667
10x5	la03	597	627	617	617	610
	la04	590	611	607	607	613
	la05	593	**593**	**593**	**593**	**593**
	MRE	-	2.2	1.19	1.54	1.54
	la06	926	**926**	**926**	**926**	**926**
	la07	890	**890**	**890**	894	**890**
15x5	la08	863	865	**863**	**863**	**863**
	la09	951	**951**	**951**	**951**	**951**
	la10	958	**958**	**958**	**958**	**958**
	MRE	-	0.04	0.0	0.087	0.0
	la11	1222	**1222**	**1222**	**1222**	**1222**
	la12	1039	**1039**	**1039**	**1039**	**1039**
20x5	la13	1150	**1150**	**1150**	**1150**	**1150**
	la14	1292	**1292**	**1292**	**1292**	**1292**
	la15	1207	1251	**1207**	1286	1259
	MRE	-	0.98	0.0	1.57	1.19

From Table 4.4 we can see that the proposed algorithm is able to obtain good results compared to those reported by the selected approaches. We selected these approaches to compare mainly because they were using the same instances, and reporting results for all of them. Which is not the case of the approach proposed in [Gabel & Riedmiller (2007)], it was not possible to compare instance per instance, as they only present their results grouped by number of jobs and machines, and for

each group only the MRE is reported. In order to have an idea of how we compare to it, Table 4.5 shows a comparative study between the MREs of both approaches.

Table 4.5: MRE per group of instances

Instances m x n	Number of instances	Theoretical Optimum	Previous RL alg.	QL JSSP
10 x 5	5	620.2	1.9	1.54
15 x 5	5	917.6	0.0	0.0
20 x 5	5	1179.2	1.5	1.19

Table 4.5 shows both algorithms are able to obtain the optimal solutions for all the instances in the second group. In the first group, which are instances with 10 jobs and 5 machines, the proposed algorithm was outperformed by the previous RL approach, while in the last group the QL method was able to yield better solutions. From the experiments we can see that our approach was able to obtain competitive results, which was the first part of our study, to analyze whether we could obtain near-optimal solutions. In the following, we will study the extension of our approach to other types of scheduling problems to later on analyze how to increase the robustness in the proposed solutions.

4.2 Reinforcement Learning for the JSSP-PM

The approach presented in the previous section can easily be extended to the case when there are parallel machines that can execute the same type of task (See Section 2.5). We analyzed two possibilities in order to adapt the algorithm:

- **queues associated to individual machines**, which means that there is one agent per resource, as was the case in the basic approach, and each of these agents will have a queue associated (see Figure 4.9). The difference is given by the fact that when one operation is fully executed on a machine, the next operation of the corresponding job has to be released for execution, i.e. it should be assigned to a specific agent, meaning that there is an extra decision to take, which was not the case in the previous approach.
- **having m groups of k machines**, the queues can be considered to be associated to groups of similar resources instead of being associated to a single

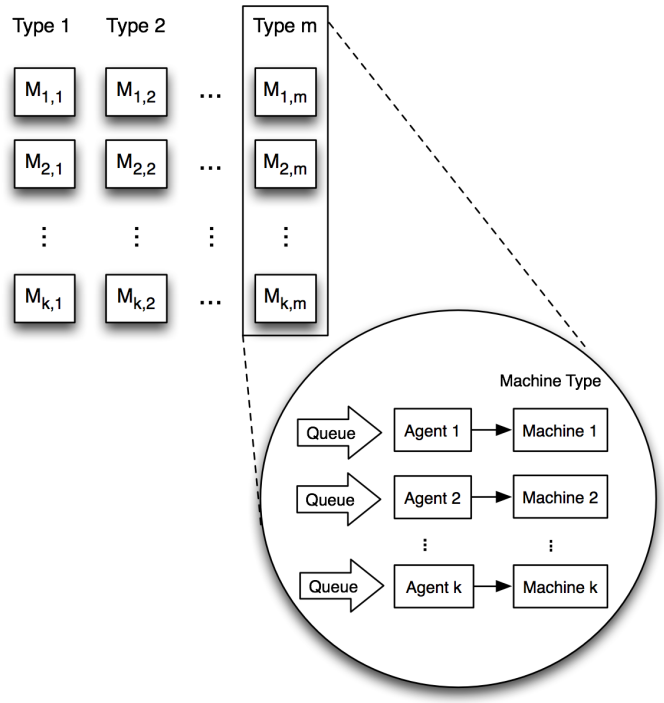

Figure 4.9: Queue per agent, and one agent per resource

resource. For example, if an operation needs to be processed by a machine of type 3, then it is placed on the queue of the group 'type 3' (machines within a group are identical). There is one agent per resource as in the basic approach, but in this case the queue is common (see Figure 4.10).

After analyzing both points of view, we decided to keep the second one, having a queue per type of resource, as it fits better in the idea of the basic approach introduced in the previous section. Nevertheless, in order to test the other point of view, we added an extra step in the algorithm where an operation is assigned to any queue at random within a group.

4.2.1 Experimental Results JSSP-PM

In order to test the algorithm we will use the benchmark instances presented in [Rossi & Boschi (2009)]. In this case the authors used the first 15 Lawrence instances from the job-shop scheduling literature. These instances can also be found in the previously mentioned OR-Library.

From these classical JSSP instances, duplicated (k=2, la01'-la15') and triplicated (k=3, la01"-la15") instances are generated. This means that we just add an extra

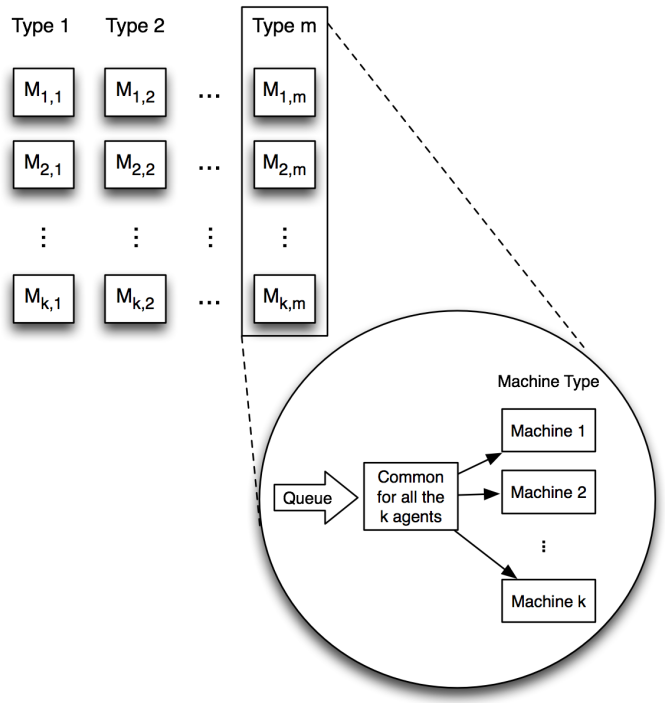

Figure 4.10: Queue per type of resource, agent per resource.

copy of all jobs once (k=2) or twice (k=3). The description line of the instance (see subsection 2.5.2) gives us the number of jobs and the number of groups of machines. The number of machines per group will be equal to k.

In Tables 4.6 and 4.7 we show a comparison of the two alternatives to solve the problem, for the experiments we only used the first 10 instances, both duplicated and triplicated.

As expected, the alternative of having a queue per type of resource was able to yield better results. This is the approach we choose to run the experiments under different setting in order to compare with other approaches.

In Table 4.8 we compare the solutions obtained by our approach to the hybrid heuristic [Rossi & Boschi (2009)] from the literature, to an ACO algorithm also proposed in the same paper, and to the results obtained by the implementation of a Learning Automaton [Wauters et al. (2010),Martínez et al. (2010)b]. The table also shows the average makespan, mean relative error in % (MRE%) and the number of optimal values found, over all 30 benchmark instances. We tested each method with 2 parameter settings. The methods and their parameters are as follows:

- LA1; alpha = 0.3, $Reward_{eq} = 0.05$, Linear Reward-Inaction

Table 4.6: Comparison of both points of view for duplicated instances

Instance	Optimal	Queue per resource type	Queue per resource
la01'	666	**666**	**666**
la02'	655	664	748
la03'	597	613	684
la04'	590	610	692
la05'	593	**593**	**593**
la06'	926	**926**	934
la07'	890	**890**	907
la08'	863	**863**	898
la09'	951	**951**	1020
la10'	958	**958**	989

Table 4.7: Comparison of both points of view for triplicated instances.

Instance	Optimal	Queue per resource type	Queue per resource
la01"	666	668	691
la02"	655	656	774
la03"	597	620	705
la04"	590	610	694
la05"	593	**593**	**593**
la06"	926	**926**	945
la07"	890	892	916
la08"	863	**863**	896
la09"	951	**951**	1017
la10"	958	**958**	989

- LA2; alpha = 0.4, $Reward_{eq}$ = 0.05, Linear Reward-Inaction
- QL1; alpha = 0.1, gamma = 0.8, epsilon = 0.2
- QL2; alpha = 0.1, gamma = 0.9, epsilon = 0.1

With QLx we refer to our method under the settings mentioned before, and with LAx to the method presented in [Wauters et al. (2010)] also with the two configurations mentioned above.

From the table we can notice that both learning methods perform slightly better than the hybrid-heuristic. On average the mean relative errors are more than 1% better. Further more, the methods are capable of finding more frequently the optimal solutions, especially for the triplicated (k=3) instances, where the learning methods succeed in finding the optimal solutions for 9 to 10 instances. In total, the learning methods succeeded in finding the optimal solution for 20 instances. The first part of the table shows the experiments for k=2 (rows la01' to la15') and the second half the experiments for k=3 (rows la01" to la15").

4.3 Reinforcement Learning for the FJSSP

After dealing with the case where multiple identical machines can execute the same type of task, we move to a more challenging kind of problem, the Flexible Job Shop. In this case, as explained in Section 2.6, there are also multiple machines available to execute the same type of tasks, but in this case, the machines are not identical, i.e. they have different speeds so more coordination is needed in order to distribute the workload.

4.3.1 The Proposed Approach: Learning / Optimization

As mentioned in Section 2.6, there are two possible ways to address the FJSSP, using hierarchical or integrated approaches. The former divides the problem in routing (assign a machine to each operation) and sequencing, while the latter considers both steps at the same time. In our case, as we already have an algorithm that can take care of the sequencing of the operations on the machines, the idea of using a hierarchical approach is more straightforward, as only the routing part needs to be incorporated.

Table 4.8: Comparative study for the JSSP-PM.

Instance	Optimal	LA1	LA2	QL1	QL2	ACO	Hybrid heuristic
la01'	666	**666**	**666**	**666**	**666**	669	**666**
la02'	655	657	657	674	664	693	688
la03'	597	612	614	613	624	642	626
la04'	590	593	594	610	610	625	611
la05'	593	**593**	**593**	**593**	**593**	**593**	**593**
la06'	926	**926**	**926**	**926**	**926**	**926**	**926**
la07'	890	**890**	892	**890**	**890**	908	894
la08'	863	**863**	**863**	**863**	**863**	865	**863**
la09'	951	**951**	**951**	**951**	**951**	**951**	**951**
la10'	958	**958**	**958**	**958**	**958**	**958**	**958**
la11'	1222	**1222**	**1222**	**1222**	**1222**	**1222**	**1222**
la12'	1039	**1039**	**1039**	**1039**	**1039**	1041	**1039**
la13'	1150	**1150**	**1150**	**1150**	**1150**	**1150**	**1150**
la14'	1292	**1292**	**1292**	**1292**	**1292**	**1292**	**1292**
la15'	1207	1208	1209	1215	1245	1249	1246
la01"	666	667	667	668	671	689	677
la02"	655	662	659	656	656	707	712
la03"	597	613	614	620	624	659	673
la04"	590	591	**590**	608	598	635	629
la05"	593	**593**	**593**	**593**	**593**	594	**593**
la06"	926	**926**	**926**	**926**	932	932	936
la07"	890	892	892	892	892	908	922
la08"	863	**863**	**863**	**863**	**863**	872	871
la09"	951	**951**	**951**	**951**	**951**	958	952
la10"	958	**958**	**958**	**958**	**958**	961	**958**
la11"	1222	**1222**	**1222**	**1222**	**1222**	1224	1239
la12"	1039	**1039**	**1039**	**1039**	**1039**	1069	1049
la13"	1150	**1150**	**1150**	**1150**	**1150**	1156	1163
la14"	1292	**1292**	**1292**	**1292**	**1292**	1293	**1292**
la15"	1207	1208	1218	1251	1245	1269	1283
Average	906.6	908.2	908.6	911.7	912.6	938.6	922.5
MRE%	0	0.26	0.29	0.70	0.77	2.29	2.04
# Optimal	30	20	20	20	19	7	13

Therefore, the learning/optimization method proposed in this section is an offline scheduling approach divided in two steps. First, a two-stage learning method is applied to obtain feasible schedules, which can then be used as initial data for an optimization procedure [Van Peteghem & Vanhoucke (2008)] during the second step.

The implemented learning method decomposes the problem following the assign-then-sequence approach. Therefore, we have two learning phases. During the first phase operations learn what is the most suitable machine and during the second phase machines learn in which order to execute the operations in order to minimize the makespan. For this, each phase has a Q-Learning algorithm associated. As the process is being divided in two, we take into account the goal of each phase in order to define the states and the possible actions.

In the first phase, where the learning takes care of the routing, we have an agent per operation being responsible for choosing a proper machine to execute its corresponding operation. This machine has to be selected from the given set of available ones, that is, the set of resources that can execute it.

This is not the case for the second phase, where the learning algorithm takes care of the sequencing and each operation already knows where it has to be executed, the main idea is to decide the order in which they will be processed on the machines. As it can be seen, this is what our basic approach (proposed in section 4.1) does, which means that in the second phase we will have an agent per resource, selecting operations based on the quality of the solution.

What still needs to be defined is how to choose a proper machine for each operation.

4.3.2 Example

Assuming that we have a small instance with 2 jobs and 3 machines, where J_1 has 2 operations and J_2 has 3 operations, and these operations can be executed by the following sets of machines, where each pair represents a possible machine and the corresponding processing time:

$$J_1O_1 \begin{cases} M_1, 10 \\ M_2, 15 \end{cases} \qquad J_1O_2 \begin{cases} M_2, 12 \\ M_3, 18 \end{cases}$$

$$J_2O_1 \begin{cases} M_1, 20 \\ M_3, 25 \end{cases} \qquad J_2O_2 \begin{cases} M_1, 25 \\ M_2, 18 \end{cases} \qquad J_2O_3 \begin{cases} M_2, 15 \\ M_3, 25 \end{cases}$$

As mentioned in the description of the algorithm, the first learning phase takes care of the routing, meaning that the first step is to choose an appropriate machine for each operation.

Let us assume that each operation can select the machine that will take care of its processing, the obvious decision will be to choose the fastest machine, which means that J_1O_1 will select M_1, because it only needs 10 times steps, while M_2 will take 15. If we repeat this process for each of the operations then the selection will be as follows:

$$J_1O_1 \begin{cases} M_1, 10 \checkmark \\ M_2, 15 \end{cases} \qquad J_1O_2 \begin{cases} M_2, 12 \checkmark \\ M_3, 18 \end{cases}$$

$$J_2O_1 \begin{cases} M_1, 20 \checkmark \\ M_3, 25 \end{cases} \qquad J_2O_2 \begin{cases} M_1, 25 \\ M_2, 18 \checkmark \end{cases} \qquad J_2O_3 \begin{cases} M_2, 15 \checkmark \\ M_3, 25 \end{cases}$$

From these operation-machine assignments it is possible to see that if all the operations go for the fastest machine, then M_1 will have to execute 2 operations, and machine M_2 the remaining three operations, while M_3 stays idle during the whole process.

After performing the sequencing, a possible solution for the operation-machine assignment is shown in Figure 4.11, which results in a makespan $C_{max} = 63$:

The optimal solution for this instance is shown in Figure 4.12, with a makespan $C_{max} = 53$.

This optimal solution responds to the following operation-machine assignment:

$$J_1O_1 \begin{cases} M_1, 10 \\ M_2, 15 \checkmark \end{cases} \qquad J_1O_2 \begin{cases} M_2, 12 \\ M_3, 18 \checkmark \end{cases}$$

$$J_2O_1 \begin{cases} M_1, 20 \checkmark \\ M_3, 25 \end{cases} \qquad J_2O_2 \begin{cases} M_1, 25 \\ M_2, 18 \checkmark \end{cases} \qquad J_2O_3 \begin{cases} M_2, 15 \checkmark \\ M_3, 25 \end{cases}$$

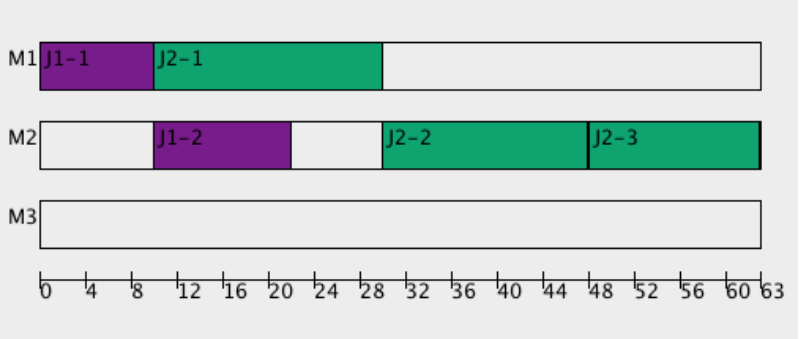

Figure 4.11: Schedule for the operation - machine assignments using the fastest machine.

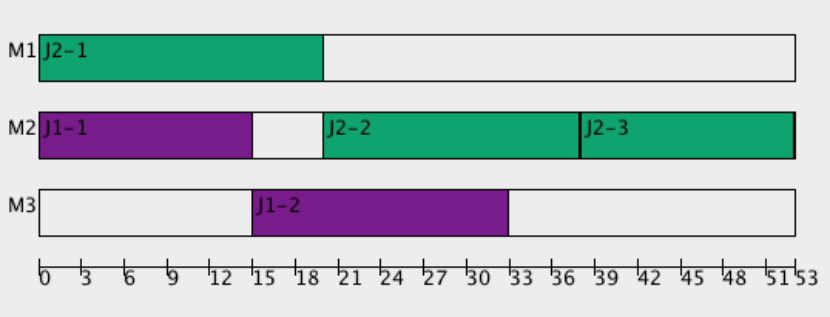

Figure 4.12: Optimal schedule for the example.

As can be seen, always taking the machine that will process the operations the fastest is not a good choice, as there are other important criteria that should be considered, for example, the workload of the machine.

That is why we use the end time of its corresponding operation on the selected machine as feedback signal for the agents. Of course its objective is to minimize the end time of the operation, that is, the time when the machine finishes executing the operation.

Once a feasible schedule has been found, the mode optimization procedure can be executed. This procedure will be described in the next subsection.

4.3.3 Mode Optimization Procedure

The mode optimization procedure [Van Peteghem & Vanhoucke (2008)] is a forward-backward procedure which tries to shift a schedule to the left (as much as possible) in order to minimize the makespan. It is executed once a feasible schedule is obtained, and we refer to it as the second step of the approach.

The forward-backward procedure can be summarized as follows:

- Order the operations according to their end times (the time when they end in the schedule received as input).
- Taking into account the previous ordering, for each operation, choose the machine that will finish it first (shortest end time, not shortest processing time). The result is a backward schedule.
- Repeat steps 1 and 2 to obtain a forward schedule.

Once the mode optimization is executed, the quality of the solution is taken into account to give feedback to the agents of the learning phases.

Example 4: *L*et us say that after executing the learning step we obtain the schedule shown in Figure 4.13.

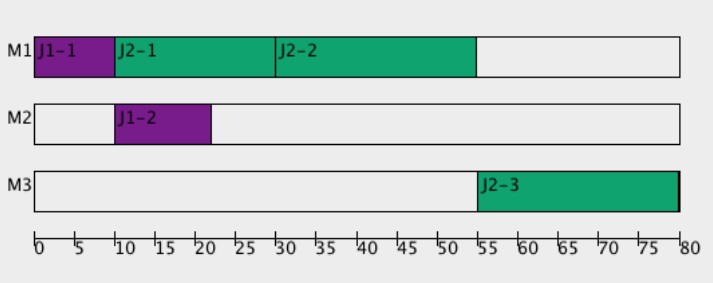

Figure 4.13: Feasible schedule obtained by the learning step.

Applying the Mode Optimization Procedure to this schedule, the first step is to order the operations according to their end times, that will give us the following ordering: $\{J_2O_3,\ J_2O_2,\ J_2O_1,\ J_1O_2$ and $J_1O_1\}$. Taking into account this ordering, the operations will choose a machine to execute it, basing the decision on the possible end time (according to the data given in subsection 4.3.2).

For example, J_2O_3 can start choosing because it was the first operation in the ordering, according to the data of the example the machines that can execute it are M_2 for 15 time steps or M_3 for 25 time steps, obviously the best choice is M_2, meaning that M_2 will be busy between time 0 and 15.

$$J_2O_3 \begin{cases} M_2, 15\ (0-15)\ \checkmark \\ M_3, 25 \end{cases}$$

Then, the next operation on the ordered list makes a choice, in this case J_2O_2 can choose between M_1 for 25 time steps and M_2 for 18. As this is the second operation of J_2 being scheduled, it can not start until the previous operation of the same job is finished, which means that its earliest starting time will be 15 (which is the time when J_2O_3 is finished), the possible end times are then 40 on M_1 (15+25) and 33 on M_2 (15+18), being the best choice M_2, which will be occupied from 15 to 33.

$$J_2O_2 \begin{cases} M_1, 25\ (15-40) \\ M_2, 18\ (15-33)\ \checkmark \end{cases}$$

When an operation from another job chooses a machine, it has to respect this busy times but it can search for an available slot of the size of the time it requires. Once all the operations are scheduled, a backward schedule is obtained, which will look as shown in Figure 4.14.

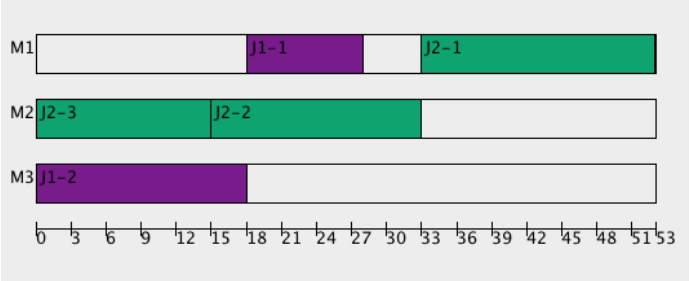

Figure 4.14: Backward schedule.

This process is repeated once again, in order to find a forward schedule, analyzing the backward schedule it will give us the following operation ordering: $\{J_2O_1, J_2O_2, J_1O1, J_1O_2, J_2O_3\}$. After repeating the choosing procedure, a forward schedule like the one shown in Figure 4.15 is obtained.

It is possible to see that we obtain a makespan $C_{max} = 53$, and initially we started with a schedule where $C_{max} = 80$.

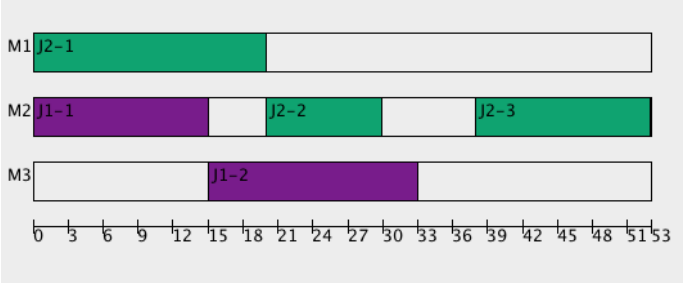

Figure 4.15: Forward schedule.

The Learning / Optimization approach can be summarized as follows:

Algorithm 2 Learning/Optimization algorithm for the FJSSP

```
Step 1 - Learning
for each operation do
    choose a machine
end for
while there are operations to execute do
    for each machine with queued operations do
        choose operation to execute
        update queues of the system
    end for
end while
Step 2 - Execute the Mode Optimization Procedure
```

4.3.4 Experimental Results FJSSP

Several experiments were performed using benchmark problems proposed in [Brandimarte (1993)] (see subsection 2.6.1). The results of this approach were reported in [Martínez et al. (2011)].

The combination of parameters used for the experiments was $\gamma = 0.8$ and $\epsilon = 0.1$, together with a discount factor $\alpha = 0.1$. The algorithm was executed for 10,000 iterations.

4.3.5 Comparative Study

Table 4.9 shows a comparative study between the proposed approach and some results reported in the literature. LB is the Lower Bound for each instance, taken from the original Brandimarte data [Brandimarte (1993)]. The algorithms used to compare our method (QL) are:

- GA: Genetic Algorithm [Pezzella et al. (2008)], algorithm integrating different strategies for generating the initial population, selecting the individuals for reproduction and reproducing new individuals.
- ACO: Ant Colony Optimization [Lining & Chen (2010)], it provides an effective integration between the Ant Colony Optimization model and knowledge model.
- GEN: Abbreviation of GENACE, an architecture proposed in [Ho et al. (2007)] where an effective integration between evolution and learning within a random search process is proposed.
- Brand: Tabu Search [Brandimarte (1993)], a hierarchical algorithm for the flexible job shop scheduling based on the tabu search metaheuristic.

Table 4.9: Experimental Results FJSSP.

Inst.	**LB**	**GA**	**ACO**	**GEN**	**Brand**	**QL**
Mk01	36	40	39	40	42	40
Mk02	24	26	29	29	32	26
Mk03	204	204	204	204	211	204
Mk04	48	60	65	67	81	66
Mk05	168	173	173	176	186	173
Mk06	33	63	67	67	86	62
Mk07	133	139	144	147	157	146
Mk08	523	523	523	523	523	523
Mk09	299	311	311	320	369	308
Mk10	165	212	229	229	296	225

Table 4.10 shows the mean relative errors in % (MRE) of the different approaches used to compare our method with respect to the best-known lower bounds.

Table 4.10: MRE: Mean relative errors

	GA	ACO	GEN	Brand	QL
MRE	17,53	22,16	23,56	41,43	19,69

From the tables (4.9 and 4.10) we see that the proposed approach is able to find the best reported value for several instances (Mk01-Mk03, Mk05, Mk08). For the instances Mk04, Mk07 and Mk10 the genetic algorithm is better. For the instances Mk06 and Mk09 the proposed algorithm is able to yield better results.

The cases in which our algorithm did not find the best solutions where instances for which a proper machine assignment was not found, probably caused by the fact that multiple machines with similar processing times can perform the same operation.

4.4 Reinforcement Learning for Online Scheduling

After dealing with three different scheduling problems where all the information was known beforehand, we switch to a completely different scenario, an online scheduling problem which is depicted in Figure 4.16 (see Section 2.8). In this case the solution method will be Learning Automata, described in subsection 3.4.3.

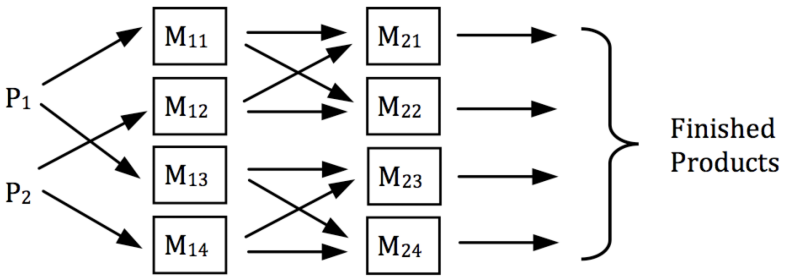

Figure 4.16: A two-stage chemical production plant. For both product types P_1 and P_2, there are two parallel machines at the first stage. At the second stage of the process, there are also two parallel machines.

4.4.1 Learning Automata for Stochastic Online Scheduling

In order to apply LA to a scheduling problem we need to define the actions of the agents and the rewards. We define an *action* of the LA as submitting a job to

one of the parallel machines. Therefore, for the problem described we will have 6 agents, one in each decision point, which means that two agents receive product orders P_1 and P_2 and decide which 'stage-1' machine will be used. The other four agents receive partially processed jobs from a 'stage-1' machine M_{1-} and send them to a 'stage-2' machine M_{2-}. Note that the agent cannot wait to submit a job and cannot stop a job preemptively. In other settings these could be added as extra actions [Martínez et al. (2010)a].

When a job j is completely finished, the two agents that decided the path of that job are notified. Based on the completion time C_j and the release times r_j for both stages a *reward* $r \in \{0, 1\}$ is created, see Equation 4.4. Note, (i) completion time is the time at which the job has finished both stage 1 *and* stage 2, and (ii) release times are the times at which the job starts stage 1 *or* stage 2 depending on the agent.

$$r = \begin{cases} 0 & \text{if } F > F_{avg}, \\ 1 & \text{otherwise,} \end{cases} \tag{4.4}$$

where the flowtime $F = C_j - r_j$ and F_{avg} is the average flowtime over the last n jobs. The larger n, the more accurate the LA's belief of the average flowtime of the jobs. The smaller n the faster the LA will adapt its belief of the average flowtime when for example a machine breaks down or the performance of a machine increases.

4.4.2 WSEPT Heuristic for Stochastic Online Scheduling

To the best of our knowledge, there is no good approximation algorithm for scheduling problems that are online, stochastic and multi-stage at the same time. For the single-stage case, there exists a very good heuristic: Weighted Shortest Expected Processing Time (WSEPT) [Megow et al. (2006)].

This heuristic works as follows: orders are arriving over time and must be processed by one out of several parallel machines. The objective is to reduce the total weighted completion time ($\sum_j w_j C_j$, for all jobs j). Each time an order arrives, the WSEPT rule submits the job to the machine that is expected to finish it first. To this end it polls each machine for its current expected makespan (including the new job). If, for example, all jobs have equal expected processing time, and each machine the same average speed, then the expected makespan is the queuelength (including the currently processed job if any). In [Megow et al. (2006)] lower bounds on the total weighted completion time ($\sum_j w_j C_j$) are given for the WSEPT heuristic.

In the next section we will compare the WSEPT and the Learning Automata in a simple single-stage scheduling task.

4.4.3 Experimental Results

4.4.3.1 WSEPT Heuristic versus Learning Automata

We ran some experiments on single-stage scheduling with $N = 4, 5$ or 6 identical machines. One scheduler receives a sequence of jobs. The joblengths are generated by an exponential distribution with average $\mu = 100$. The identical machines have unit processing speed $s_i = 1$, for $i = 1, \ldots, N$. I.e. a machine needs 100 timesteps to process an average job.

To make sure the system can actually handle the load, we set the probability of creating a job at any timestep to 95% of the total processing speed divided by the average job length: $0.95 \sum_i s_i/\mu$. In this setting, all jobs have unit weight $w_j = 1$.

We tested the following agents on the same sequence of orders:

RND: uniformly distributes the jobs over all machines,

WSEPT: uses the WSEPT heuristic as described in Section 4.4.2,

LA: a Learning Automaton as described in Section 4.4.1 with $\alpha = \beta = 0.02$.

Results: The experiments showed that the LA clearly performs better than the RND scheduler, and WSEPT outperforms LA [Martínez et al. (2010)a]. Note that the heuristic approach uses a lot more information which LA cannot access. The length of the queues over time show that WSEPT balances the load better: queues are 4 to 5 times shorter. On the other hand, the total weighted completion time ($\sum_j w_j C_j$) does not show huge differences between WSEPT and LA (in the order of 0.001 to 0.01 time units).

Although the WSEPT heuristic outperforms the Reinforcement Learning approach, it requires access to more information and only works in single-stage load-balancing problems. This is why in the next section, we test LA in the multi-stage setting described at the beginning of Section 4.4.1.

4.4.3.2 Multi-Stage Scheduling

In this case we look at two slightly different settings, which are summarized in Table 4.11. Setting 1 is adopted from [Peeters (2008)]. In both cases, the average

joblength is 100 and the jobrate is 1/45 for both product types P_1 and P_2. The performance is measured by the total flowtime of the jobs through the entire processing chain.

Table 4.11: Processing speeds of all machines for two different settings.

Machine	M_{11}	M_{12}	M_{13}	M_{14}	M_{21}	M_{22}	M_{23}	M_{24}
Speed setting 1	3.33	2	1	1	3.33	1	1	1
Speed setting 2	3.33	2	10	10	3.33	1	1	1

In order to compare the results, a random strategy for dispatching the jobs was also implemented under these settings. Figure 4.17 shows how the queues of the machines grow when following this strategy, where each color represents a machine (8 in total).

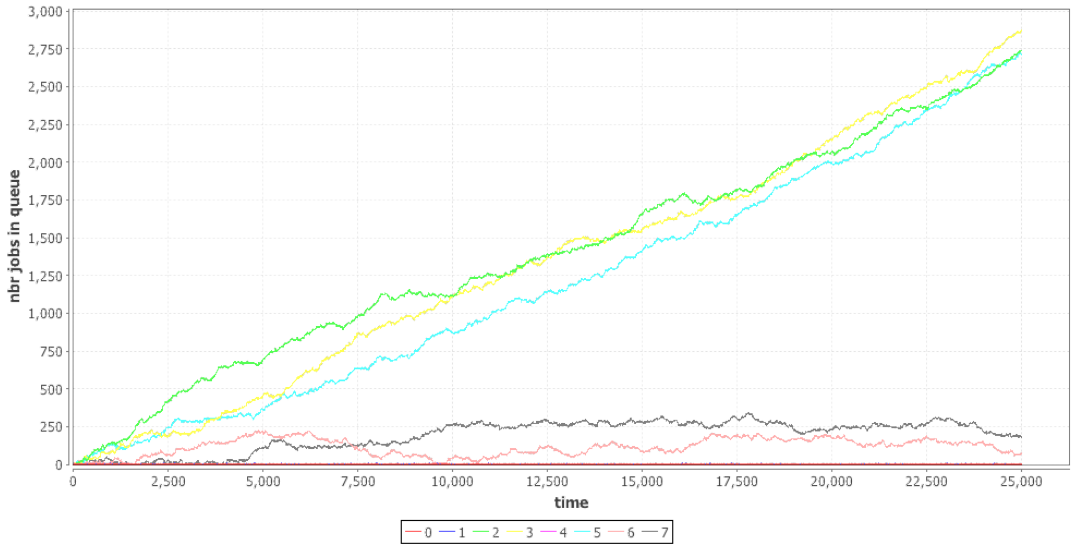

Figure 4.17: Queues of the machines when following the random strategy.

The first type of LA we tested was Linear Reward-Inaction LA (L_{R-I}). After some time, the queues started growing indefinitely. This was caused by some automata converging prematurely to a pure strategy, i.e. they end up selecting the same action forever. This is due to the fact that L_{R-I} never penalizes bad actions ($\beta = 0$). Although this may be favorable for many RL problems, it will almost never be for load-balancing. The only obvious exception is when one machine is able to process all jobs before any new order arrives.

Figure 4.18 shows the queues of the machines when using Linear Reward-Inaction Learning Automata.

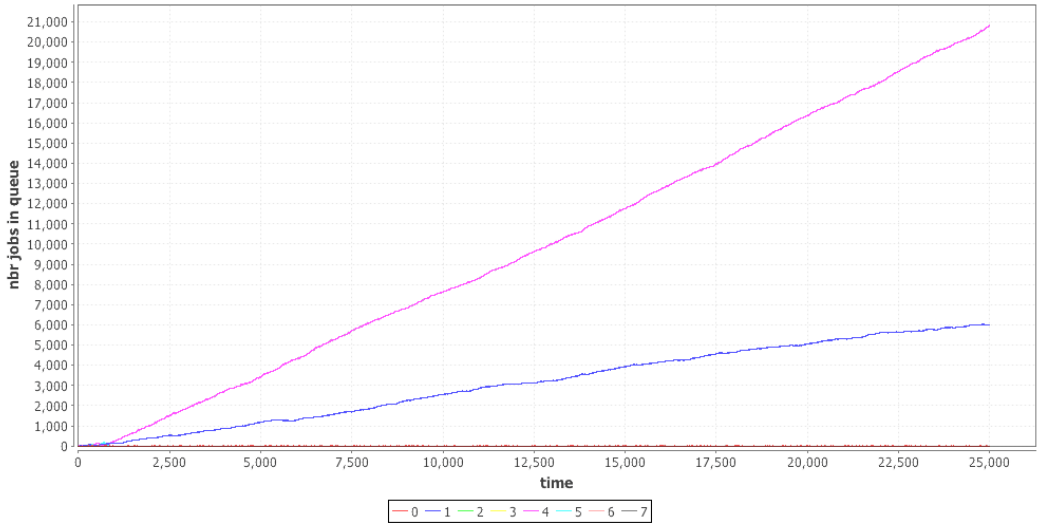

Figure 4.18: Queues of the machines when using Linear Reward-Inaction.

The $L_{R-\epsilon P}$ generated better and better results when ϵ was increased. Finally, when $\epsilon = 1$ we have L_{R-P}, where penalty and reward have an equally large influence on the probabilities. This gives the best results. Figure 4.19 shows the queues of the machines when using learning automata under the reward penalty strategy.

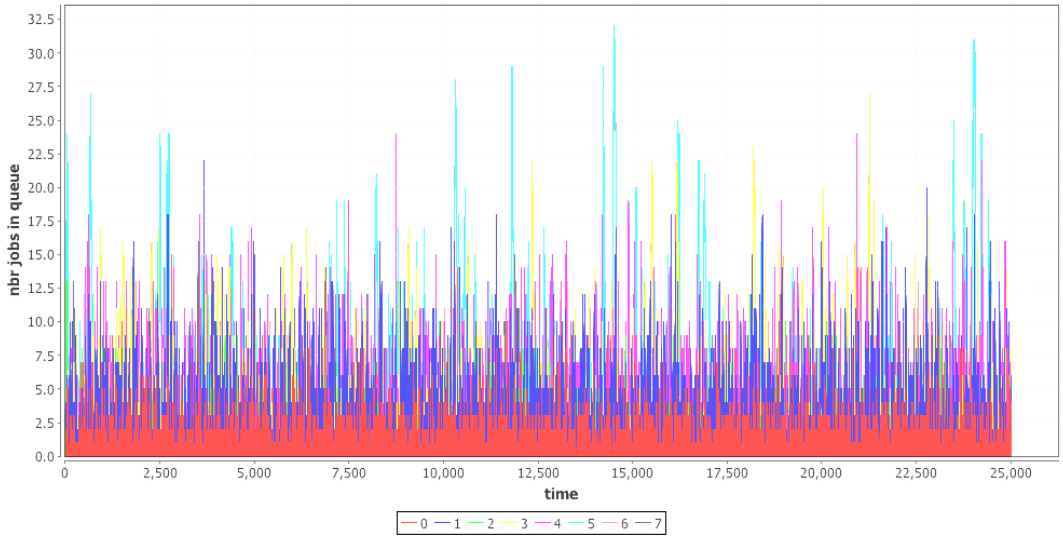

Figure 4.19: Queues of the machines when using $L_{R-\epsilon P}$.

Figure 4.20 shows the results in terms of the amount of queued jobs at the end

of the process (25000 iterations), while Figure 4.21 shows the amount of jobs that were executed also after the total number of iterations.

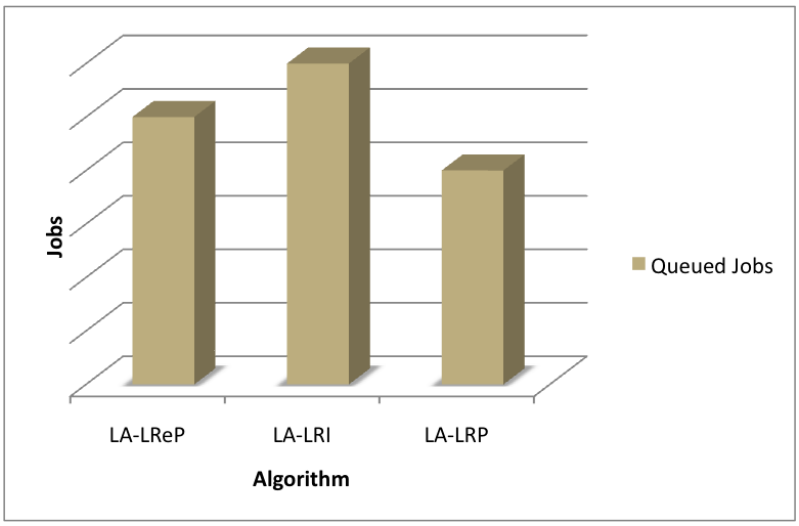

Figure 4.20: Queued jobs at the end of the process.

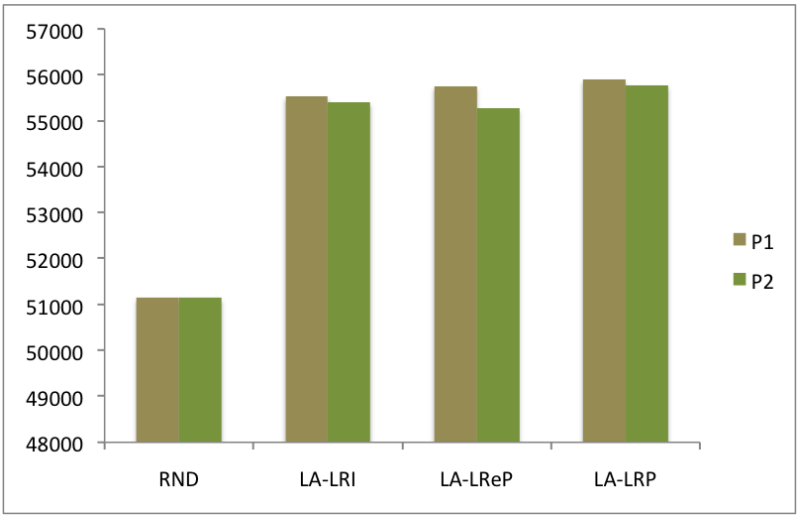

Figure 4.21: Total number of jobs executed at the end of the 25000 iterations.

Different combinations of values for the parameters α and β were studied. These parameters determine the learning speed, and the best combination for this problem was 0.01. Table 4.12 shows the average policy for each of the six agents. For example, the fourth agent receives jobs partially finished by machine M_{13} and distributes them over M_{23} and M_{24}.

The second setting shows that the agents take into account the time needed for a job to go through all stages. Machines M_{13} and M_{14} are 10 times faster as in the first setting. This does not increase the total system capacity, since the machines in the second stage would create a bottleneck. The result is that the first two agents still favor M_{11} and M_{12}, but slightly less. For example, the first agent in Table 4.12 distributes 71% of the load on M_{11} in the second setting, as opposed to 74% in the first setting.

Table 4.12 shows the average probabilities of all agents through an entire simulation.

Table 4.12: Average probabilities of all agents through an entire simulation.

Machine	M_{11}	M_{13}	M_{12}	M_{14}	M_{21}	M_{22}	M_{23}	M_{24}	M_{21}	M_{22}	M_{23}	M_{24}
Setting 1	.74	.26	.69	.31	.76	.24	.50	.50	.77	.23	.50	.50
Setting 2	.71	.29	.61	.39	.77	.23	.50	.50	.78	.22	.50	.50

4.4.4 Discussion

The following advantages of LA make them very applicable in difficult scheduling scenarios:

- They can cope with unknown joblengths, unknown future jobs and unknown machine speeds.
- The decisions based on the probability distribution and the updates of those distribution are straightforward. They can be performed in a minimum of time and require only very limited resources.
- No heuristics or additional information is needed.

The performed experiments show that LA can learn processing capacities of entire downstream chains. Note however that the rewards are delayed. While waiting for a submitted job to be finished, other jobs must already be scheduled. In our scheduling problem, this is not a problem for the LA. When more stages would be added to the system, the LA could be equipped with so-called eligibility traces [Castillo & Roberts (2001)].

Since LA are very adaptive, and will not converge to pure strategies with L_{R-P}, they are able to adapt to changes in processing speed, such as machine break downs.

Finally, when applying any randomization technique (such as LA) to balance a load, one is always better off with many short jobs than very few long ones (cf. the law of large numbers).

4.5 Summary

In this chapter we first introduced what we call the basic learning algorithm, which was initially defined for the job shop scheduling problem and later on adapted in order to solve the version with parallel machines and the flexible job shop scheduling. The learning method implemented was a Q-Learning algorithm, for which different feedback signals were studied. This QL algorithm was easily adapted and combined with other techniques in order to satisfy the extra constraints of the more challenging scenarios. In all the cases the results were compared with those reported in the literature by other approaches. For the online scheduling problem we used Learning Automata in order to learn how to distribute the incoming orders of two types of products among parallel machines. Experiments were developed using the three

possible update schemes. The results of the experiments showed that the L_{R-P} scheme was able to yield better results. In the following chapter we will present some of the more common sources of uncertainty in scheduling and introduce the concept of robustness.

Chapter 5

Uncertainty and Robustness in Scheduling

In real-world applications of Linear Programming, one cannot ignore the possibility that a small uncertainty in the data can make the usual optimal solution completely meaningless from a practical viewpoint.
- [Ben-Tal & Nemirovski (2000)]-

In real world environments scheduling can not be seen as a problem where all the information is known beforehand. For example, manufacturing scheduling is subject to constant uncertainty in its environment, where the probability of a schedule being executed exactly as it was planned is usually very low. In this chapter we summarize the more common sources of uncertainty in scheduling, and different types of approaches to deal with these uncertainties are studied. These approaches can be classified as proactive or reactive. We will go into more detail in the proactive techniques, as the approach we propose in the next chapter is based on this idea. The concept of robustness is defined and different metrics to measure how robust a schedule is are described.

5.1 Uncertainty in Scheduling

During the scheduling process machines can break down, operations can take longer than expected, new orders can arrive, the priority of some orders might change, and all these disruptions will eventually make the original schedule fail, causing delays in the delivery of the products (due dates might be violated), increasing the idle times of the machines and the staff, among many other consequences.

Therefore, in real environments it may not be useful to spend significant efforts to produce optimal solutions, since the optimality is achieved only if the solution can be executed as planned. On the contrary, as noted in [Policella (2005)], *"a sub-optimal schedule that contains some built-in flexibility for dealing with unforeseen events, might provide useful characteristics for the execution. Hence, the efficiency of the scheduling techniques employed will depend on the degree of uncertainty of the scheduling information provided at the beginning"*.

Figure 5.1 shows another graphical representation (more detailed than Figure 2.1) of the supply chain environment where scheduling operates.

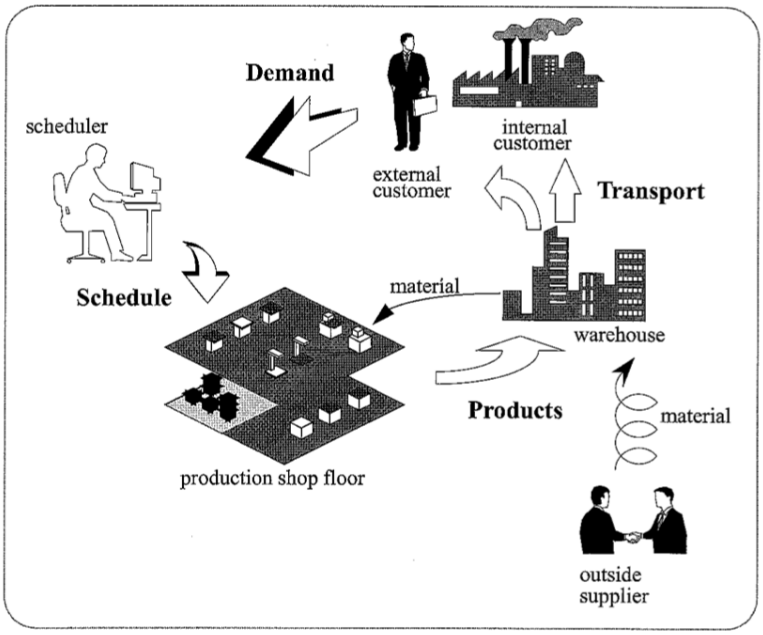

Figure 5.1: Scheduling Environment, taken from [Gao (1996)].

The input of this chain is composed of a set of demands or orders from internal and/or external customers, these demands are given to the scheduler, who provides a schedule to the shop floor for execution. The shop floor uses internally produced

materials, and also receives part of them from outside suppliers, producing the final products that are stored in a warehouse until they are delivered to the customers who ordered them [Gao (1996)].

As it can be seen, scheduling works at a very low level inside production management, but it is in this level where the shop floor strategy is planned, that is why it is very important to come up with schedules that will allow to accomplish the desired objective and satisfy the customers.

One of the major existing problems in scheduling is that plans or schedules that are good on paper often do not perform well in practice, due to the previously mentioned uncertainty in the environment. In the search of optimal solutions machines are constantly working and the operations are executed as soon as possible in order to meet the due dates and minimize objectives like the makespan or the tardiness. That is why most of the time we are faced with non-delay schedules, which are good in practice as long as unexpected events do not occur.

It is important to say that even though the presence of idle times is usually considered as bad when scheduling, it is good in performance, because it is what keeps delays from propagating [Cohn (2007)].

Some of the more common sources of uncertainty at the operational level are:

- Machine breakdowns;
- Uncertainty in the processing times of the activities;
- Unexpected arrival of new orders, some of them with high priority;
- Cancellation or modification of existing orders;
- Modification of release and/or due dates.

In this dissertation we will focus on the first two, **Machine breakdowns and Varying activity duration**. One way to deal with such disruptions is by generating robust schedules [Davenport et al. (2001)], that is, schedules with the ability to satisfy performance requirements predictably in an uncertain environment [Le Pape (1991)]. In the next section we will introduce the concept of robustness and we will answer some important questions, such as how to measure how robust a schedule is, and how to incorporate robustness in our solutions.

5.2 Robustness

In many decision processes it is common that solutions with a certain level of robustness are required in order to maintain their feasibility in settings with incomplete or imprecise data. Solutions should tolerate a certain degree of uncertainty during execution, in other words, they should be able to absorb dynamic variations in the problem.

We often talk about robust schedules, but there are several questions that have to be answered in order to truly capture the meaning of robustness, for example:

- how is robustness defined?
- how can we measure how robust a schedule is?
- how to incorporate robustness in the schedules?

The robustness of a schedule is a way to characterize its performance. In [Billaut et al. (2008)], a schedule is classified as *robust* if its performance is rather insensitive to data uncertainties. Another definition of robust schedule is given in [Leon et al. (1994)], in this case it is defined as a schedule that is insensitive to unforeseen shop floor disturbances given an assumed control policy. In other words, robust scheduling tries to protect the schedules from stochastic events.

The development of robust optimization was initiated by Soyster in 1973 [Soyster (1973)]. This approach has been extensively studied and extended, as it was considered too conservative, in the sense that it produces solutions that give up too much of optimality for the nominal problem in order to ensure robustness [Bertsimas & Sim (2004)].

After this initial work some other models were proposed, see for example [Ben-Tal & Nemirovski (2000)], [Bertsimas & Sim (2004)] and [Lui Cheng (2008)]. In all of them the main concern is how to make the schedules more robust without loosing too much in optimality.

5.2.1 Measuring Robustness

A property such as robustness is easy to define, but difficult to measure in a quantitative manner. In [Sabuncuoglu & Goren (2009)] the authors present a deep study that shows how different approaches focus on different objectives and as a consequence, the way of measuring robustness is not always the same.

In the literature, the following metrics are commonly used to measure the robustness of a schedule:

1. deviation from the original schedule;
2. the real cost incurred by the implementation of the schedule;
3. stability - relatively to a performance criterion;
4. number of changes required to 'fix' the solution.

The first two measures were proposed in [Gao (1996)], according to the author, the deviation from the original schedule can be computed by analyzing the difference between the planned makespan and the actual makespan. In the second case costs can be associated, for example, to tardiness, meaning that there is a penalty per time unit scheduled over the due date. Or costs could be associated to resource idleness, which means that there is a penalty for each time unit that the resource was kept idle.

In the case of the stability (third metric), we can say that it is quite similar to the first metric. According to its definition in [Billaut et al. (2008)], stability is a measure of the sequence difference between the two schedules, which in other words is how deviated is the actual schedule from the original one.

For the last metric, which was introduced in [Gomes (2000)], the intuition behind robustness is the following: given a set C of changes to the initial formulation of the problem instance, a solution A is more robust than a solution B with regard to the set C if the number of changes required to fix solution A is less than the number of changes required to fix solution B.

There are different approaches that can be used in order to deal with uncertainty and robustness in scheduling, either finding solutions which adapt dynamically to changes or incorporating available knowledge about possible changes to the solution [Herroelen & Leus (2005)]. These approaches will be explained in detail in the next section.

5.3 Proactive vs. Reactive Approaches

According to the literature, there are two ways to deal with uncertainty in scheduling environments, using *proactive approaches* (also called predictive) or *reactive approaches*.

The goal of proactive scheduling is to take into account possible disruptions while constructing the original predictive schedule. This allows to make the predictive schedule more robust, because it has some statistical knowledge of uncertainty [Davenport & Beck (2000)].

Proactive scheduling has also been defined as techniques that seek to produce an off-line schedule that is robust to execution time events [Beck & Wilson (2007)]. The utility of this type of approach depends on whether the uncertainty can be quantified in some way (like knowing the mean time between failure for the machines etc). If the information is available, it can be used by proactive techniques. If however, the degree of uncertainty is very high, a more reactive approach is needed.

Reactive scheduling involves revising or reoptimizing a schedule when an unexpected event occurs. Completely reactive approaches are based on up-to-date information regarding the state of the system [Davenport & Beck (2000)]. Decisions are made in real time, based on priority dispatching rules, as no predictive schedule is given to the shop floor. These approaches are used when the level of disturbances is always significant or when the data are known very late, making the computation of predictive schedules impossible [Aloulou & Portmann (2005)].

Figure 5.2 shows a graphical representation of the properties of both types of approaches.

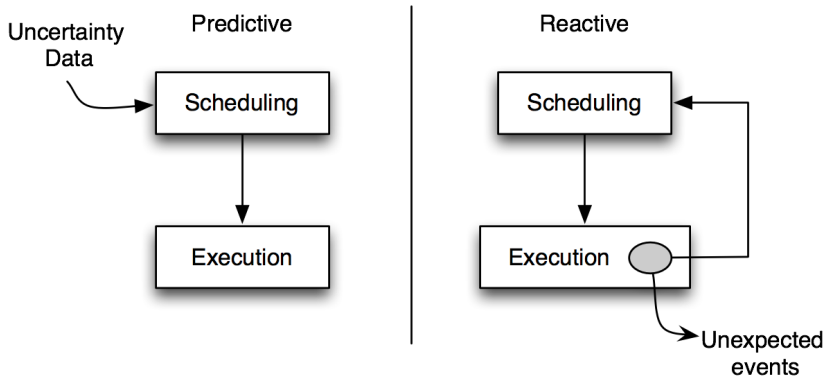

Figure 5.2: Predictive vs. Reactive Scheduling

Sometimes both points of view are combined and the combination is called predictive-reactive approaches. In this case a predictive schedule is generated without considering possible perturbations. Then, a reactive algorithm is used to maintain the feasibility of the schedule and/or improve its performances, see for exam-

ple [Church & Uzsoy (1992)], [Vieira et al. (2003)] and [Aloulou & Portmann (2005)]. For further reading and a thorough overview on these types of approaches we refer to [Sabuncuoglu & Goren (2009)].

5.3.1 Reactive Approaches

Based on the description provided in the previous section, we can conclude that reactive scheduling aims at finding approaches to (ideally) optimally react to disruptions after they occur. According to the literature, the reaction generally takes the form of either modifying the existing initial schedule (repairing), or generating a completely new schedule from scratch [Sabuncuoglu & Goren (2009)].

When the unexpected event is a breakdown, the simplest way to solve a rescheduling problem is to keep the processing order of the preschedule, but delay the processing when it becomes necessary. This kind of rescheduling is usually called 'simple rescheduling' [Jensen (2001)], and it has been used for example in [Leon et al. (1994)].

Another type of approach is the one presented in [Wu et al. (1999)]. In this work the author developed a decomposition method that partitions job operations into an ordered sequence of subsets. This decomposition identifies and resolves a 'crucial subset' of scheduling decisions through the use of a branch-and-bound algorithm. The computational experiments were conducted under a wide range of processing time perturbations.

Another reactive strategy is list scheduling, which is applied in [Lambrechts et al. (2010)]. In this type of approach a random precedence feasible priority list is used as starting point, or also a scheduled order list that allows to reschedule the activities in the order dictated by the schedule, while taking into account the new, reduced resource availabilities. More specifically, when a disruption occurs at time t a priority list L is created including the activities that are not yet completed, ordered in a non-decreasing order of their baseline starting times. The priority list is decoded into a feasible schedule using a modified serial schedule generation scheme and taking into account the known resource availabilities up to the current time period. This idea is also applied for example in [Wauters et al. (2010)].

The approaches presented in [Wauters et al. (2010)] and [Kaddoum (2011)] will be briefly explained at the beginning of next chapter, as they will be used to compare the results of our approach.

5.3.2 Proactive Approaches

In order to obtain robust schedules, an intuitive approach consists in adding some form of redundancy to the solution during the scheduling process. Figure 5.3 shows an example of two different solutions for the same scheduling problem. Schedule *a*) is the classical example of a non-delay schedule, where the objective is to optimize the resource utilization, that is why each activity starts as soon as its predecessor ends. On the other hand, schedule *b*) presents some idle intervals between the execution of the different activities. For example, activity 'b' does not start immediately after activity 'a' is finished. This approach allows to tolerate some changes in the original problem like delay in the activity processing and machine breakdowns. At the same time, by introducing redundancy the resource utilization is reduced, which means that objective functions like minimizing the makespan or the tardiness will also be affected. Therefore, the main goal of the redundancy-based approaches is to find out how much redundancy is needed and how this redundancy has to be distributed over the whole solution.

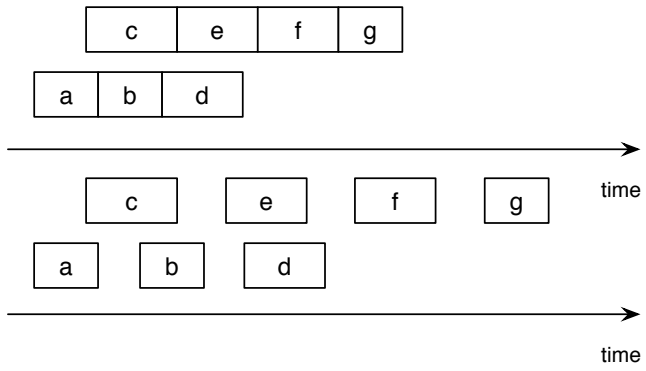

Figure 5.3: Increasing robustness by adding redundancy. Schedule b) is considered more robust than schedule a).

In this dissertation we focus on proactive approaches, more specifically on techniques that deal with uncertainty by inserting some form of redundancy (typically extra time) in the schedule. These techniques will be referred to as slack-based techniques and are explained in detail in the next section.

5.4 Slack Based Techniques

According to the literature, the main idea behind slack-based techniques is to provide each activity with some extra execution time so that some level of uncertainty can be absorbed without rescheduling. In order to do that, these techniques base their behavior on existing knowledge about failures of the machines, for example: the mean time between failures, or the mean down time of a resource. In this section we will describe three slack-based techniques: Temporal Protection, Time Windows Slack and Focused Time Windows Slack.

5.4.1 Temporal Protection

Temporal Protection, first proposed by Chiang and Fox [Chiang & Fox (1990)], is a technique that is based on the idea of predictively building a schedule taking into account previous knowledge of uncertainty.

Resources that have a non-zero probability of breakdown are identified as *breakable* resources. The duration of all the activities requiring breakable resources is extended in order to provide extra time to cope with the breakdowns and the scheduling problem with protected durations is then solved with standard scheduling techniques [Gao (1996)].

Let us show an example based on the one presented in [Davenport et al. (2001)]. Figure 5.4 depicts two activities, A and B, which are sequenced on a breakable resource. The white part of the boxes represents the original processing time of the activities, and the gray part represents the extra time added by the temporal protection technique. If the resource breaks down while activity A is being executed, the extra time can be used to absorb the breakdown, or part of it. If it does not take longer than the available protection then the rest of the schedule will remain as planned. If it lasts longer, then some adjustment is needed in order to keep executing the rest of the activities (rescheduling, right shifting, etc). If there are no breakdowns while activity A is executed then activity B can start earlier, meaning that the slack provided by the temporal protection to activity A can be used by activity B.

One important point when using the temporal protection, or any other slack-based technique is to decide the amount of extra time that will be added to each activity. Too much protection will end up with a poor quality schedule, but a highly

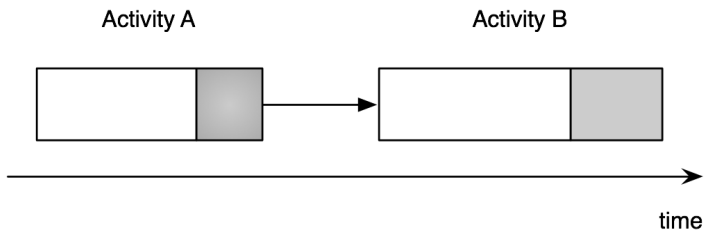

Figure 5.4: Example of two consecutive operations which are executed on a breakable resource. The white part of each box represents the original processing time of the operation and the gray part represents the extra time added by the temporal protection technique.

robust one. Too little protection will result in a poor quality schedule execution if a breakdown occurs. The approach presented in [Gao (1996)] extends the duration of each activity requiring a breakable resource in the following way:

$$p_{ij}ext = p_{ij} + \frac{p_{ij}}{F} \times D \tag{5.1}$$

Where p_{ij} is the original processing time of the activity, F represents the expected time between machine failures and D is the duration of the breakdown or interruption.

$p_{ij}ext$ is the extended processing time, by adding the total expected down time to the uninterrupted processing duration. $\frac{p_{ij}}{F}$ gives the expected number of breakdowns during the processing of an activity. $\frac{p_{ij}}{F} \times D$ represents the duration extension caused by the machine breakdowns.

Figure 5.5 shows two schedules, on top we have a non-delay schedule, where all the operations are executed as soon as the resources become available. In this case, the resource responsible for executing operations a, b and d is considered as breakable, which means that these three operations will be 'protected' by receiving some extra execution time.

It is important to remember that if breakdowns do not occur, or if they do occur, but take less time than the slack time added, operations that are scheduled after a 'protected' operation can start their processing earlier.

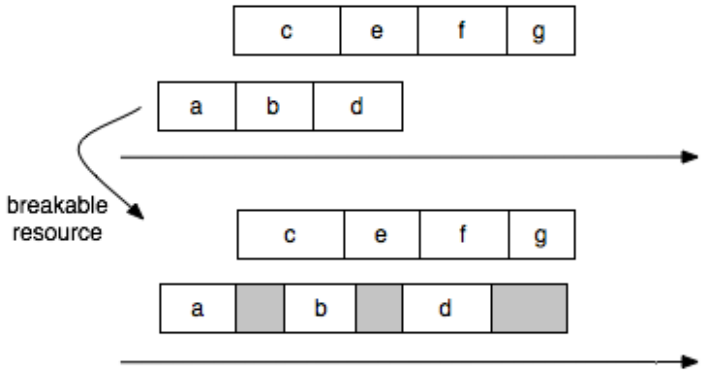

Figure 5.5: Example of schedule where a breakable resource has three operations scheduled.

5.4.2 Time Window Slack

If we analyze the way the Temporal Protection technique uses to add the slack times, we will notice that there are situations where it will not be possible to 'share' slack between activities, even if there are no breakdowns. For example, the environment shown in Figure 5.4 is extended in Figure 5.6 by adding a third operation C, which is being executed on a non-breakable resource, meaning that it does not have extra time associated. According to the ordering constraints activity B must be executed after activity C is finished, which means that the earliest start time for B is the end time of C.

In order to avoid such situations, where the slack time added can not be used by upcoming operations, the Time Window Slack (TWS) approach was proposed in [Davenport et al. (2001)]. In this approach the authors change the way of adding slack to the operations so that they make sure that the schedule has sufficient slack time for each activity.

It is important to mention that the required slack for an activity under TWS is considerably longer than the duration extension in temporal protection. More specifically, the amount of slack on each activity is equal to the sum of the durations of all the expected breakdowns of resource R, and this is mainly due to the fact that authors expect the slack of all the activities on each resource to be shared [Davenport et al. (2001)].

Equation 5.2 shows how the slack is calculated in this approach. In this equation p_{ij} represents the duration of the activity, $\mu_{tbf}(R)$ is the mean time between failures

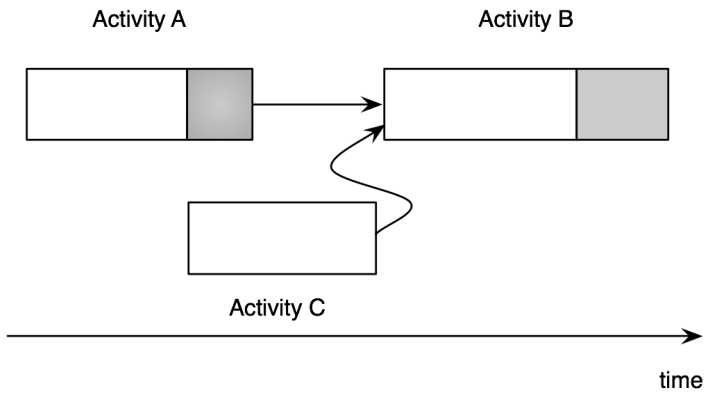

Figure 5.6: Example where the slack time added by the temporal protection method can not be used by the next activity on the same machine due to ordering constraints.

on resource R and $\mu_{dt}(R)$ represents its mean down time (breakdown duration). The set of activities that can be executed on resource R is given by $acts_R$.

$$slack_A \geq \frac{\sum_{B \in acts_R} p_{ij}}{\mu_{tbf}(R)} \times \mu_{dt}(R) \tag{5.2}$$

Figure 5.7 shows how the TWS method adds extra execution time on every operation, assuming that the slack of all the activities on each resource will be shared.

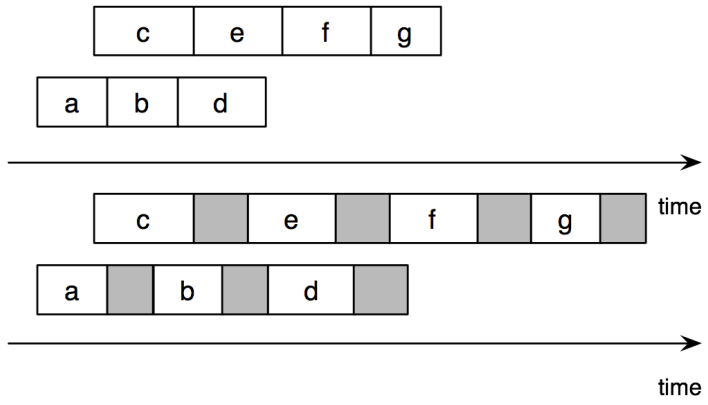

Figure 5.7: Adding slack time under the Time Windows Slack method.

In short, when using temporal protection, the expected down time of each resource is divided into the different operations and the slack times are added based

on this value. When using TWS, the slack time is equal to the sum of the durations of all the expected breakdowns on resource R, because the slack is expected to be shared.

5.4.3 Focused Time Window Slack

The Focused Time Window Slack (FTWS) approach was also proposed in [Davenport et al. (2001)]. In this approach, the placement of activities in the scheduling horizon is taken into account in order to decide if some slack time will be needed. The authors use uncertainty statistics in order to focus the slack in parts of the schedule that are more likely to need it in order to deal with an unexpected event.

For example, they consider the scenario where a new machine can arrive to the system, in this case it will not make sense to make the schedule more robust at the beginning, as this machine is new and it could take some time before it can break down.

The slack for the activities is calculated as a function of the probability that a breakdown will occur before or during its execution. Which means that the decision on whether to add slack or not is based on known distributions.

Figure 5.8 extends the example shown in Figure 5.5 by considering the use of a newly bought machine. In this case slack is added to the operations that are scheduled on this machine after a specific amount of time, for example, operation c does not receive extra execution time, as it is the first operation being executed by the new machine, and it is not supposed to fail so fast.

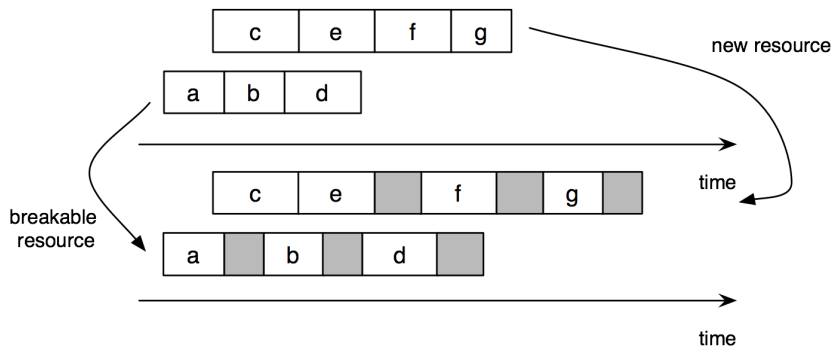

Figure 5.8: Adding slack time using the Focused Time Windows Slack approach.

5.4.4 Discussion about the slack-based techniques

In general, all these slack-based methods base their behavior on the data they receive about the uncertainty, but what if something happens on a machine where breakdowns are not expected? Consider again the example of the newly bought machine, a very fast one which is not considered as breakable, therefore, several operations are scheduled there, because it is a more reliable resource. If that machine fails, the schedule is not prepared to deal with the disruption, as it was a breakdown impossible to forecast. This leads us to the following question:

Is it possible to construct more robust schedules without looking at the probability distributions of the unexpected events as the only source of information?

In the next chapter we propose a new slack-based approach which does not rely on the probability distributions as the only source of information. It works by analyzing something we call 'criticality' of the machines and/or the jobs and the main objective is to add slack times in those parts of the schedule which are considered to be critical, or more likely to need extra execution time.

5.5 Summary

In this chapter we presented some of the more common sources of uncertainty in scheduling, as well as the concept of robustness and different methods that can be used in order to construct more robust schedules. We focused on proactive approaches, mainly slack-based techniques, as the approach we will propose in the next chapter is based on this idea. These slack-based techniques work by adding extra execution time to the operations in order to absorb some level of uncertainty without having to reschedule. The next chapter will also introduce we will explain how some new instances of the flexible job shop scheduling were generated in order to introduce some levels of uncertainty.

Chapter 6

Scheduling under Uncertainty

Robust approaches aim at building solutions able to absorb some level of uncertainty without rescheduling
- [Davenport & Beck (2000)] -

In order to address the uncertainty problems described in the previous chapter, the research described in this chapter will be divided in two parts. First, we will present a stochastic version of the flexible job shop scheduling, for which we already showed the results in the deterministic scenario. We describe the process of creating new instances and the results obtained are compared with the ones reported by other state of the art methods in terms of tardiness and makespan. In the new instances, release and due dates are added to the jobs, and perturbations are added to the machines.

In the second part we introduce a new proactive approach based on the idea of the slack-based techniques presented in Chapter 5. This approach focuses on adding slack times in the critical parts of the schedule by taking into account the criticality of the machines and the jobs, yet trying not to increase the makespan too much. It also allows the user to decide on the level of robustness to include in the schedule construction process. Some examples demonstrate how this works, together with graphical representations (gantt charts) of the solutions. The last part of the chapter is concerned with the application of the approach to a real world case.

6.1 Deterministic Case with Known Disruptions

There are cases when it is known beforehand that a specific machine will be subject to maintenance during some amount of time, and this will interrupt the execution of the production process. In this case there are two possibilities, if the environment is like the job shop scheduling, and the affected machine is the only one that can perform specific operations, then the only choice is to wait until it becomes available again. If parallel machines are available, meaning that operations can be executed by different machines, another resource can take over. It will be up to the algorithm to decide whether it is better to wait for the machine to become available or to send some operations to a parallel resource and possibly affect the schedule. Sometimes it will be better just to wait for the machine to come back from maintenance, because the other possible machines have too many jobs in the queue.

This is a particular case that can be taken into account in our approach. If there is some extra information about maintenances on the machines (which are also considered as perturbations), the user can add it to the problem instance, and the algorithm will take it into account when constructing the solution. This information can be added to the instance in the following way:

- one line with the string #Perturbations, which indicates that from that point on different perturbations will be listed.
- the second line has only one number, which represents the total amount of perturbations that will be included.
- one line per perturbation indicating:
    1. id of the perturbed machine
    2. start time of the perturbation
    3. duration of the perturbation

Figure 6.1 shows how the information will look like after being added to the scheduling instance. In this case two perturbations are included, the first one will be on machine M_1 and will happen at time step 23, with a duration of 4 time steps. The second one will occur on machine M_2 at time step 45 and will last for 8 time steps.

Once this information has been included in the problem instance, the agents will take it into account when choosing their actions. In the case where the operations do

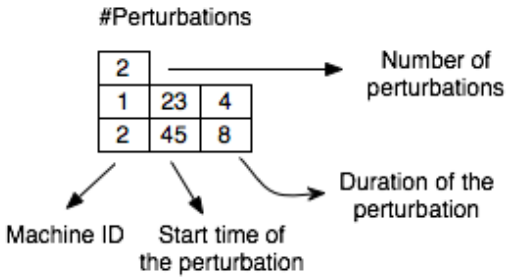

Figure 6.1: Adding scheduled maintenances to the instances.

not have execution alternatives the duration of the perturbation will just be added to the processing time of the operation being processed on the resource when the perturbation (maintenance) starts.

In the case where multiple machines can execute one particular operation, this time will be considered when choosing the machine that will execute the operation.

6.2 Stochastic Flexible Job Shop Scheduling

The Stochastic Flexible Job Shop Scheduling addressed in this section is similar to the already described Flexible Job Shop Scheduling, but in this setting the environment is subject to perturbations (see Section 2.6). In the next section we explain how the instances for the experiments are created and then we compare our Learning / Optimization approach, introduced in section 4.3, to two other approaches. The first one, called Online Forward Optimization (OFO) is a rolling time scheduling technique. The second one, SaFlexS, is a self-adaptive multi-agent system where agents interact cooperatively. Both methods will be described in Sections 6.2.2 and 6.2.3 respectively.

6.2.1 Stochastic Flexible Job Shop Instances

In order to evaluate different approaches in case of unexpected events in the environment, we generated new flexible job shop scheduling instances with release-dates, due-dates and perturbations, using classical FJSSP instances as a base problem. The selected problems were those included in the set of frequently used Brandimarte instances [Brandimarte (1993)]. This set consist of 10 problem instances, which were uniformly generated between given bounds. The number of jobs ranges from 10 to

20, and the number of machines ranges from 4 to 15. The number of operations for each job ranges from 5 to 15.

Two categories of instances were generated, 1) instances without perturbations, and 2) instances with perturbations. For each category we created 5 instances per base problem. Due-dates are uniformly generated between a lower bound LB_r and an upper bound UB_r. The lower bound is the best lower bound found in literature for the base problem [Pezzella et al. (2008)].

For the upper bound we use the makespan value of the schedule found by a greedy first available machine heuristic. This heuristic assigns operations one by one to the first available machine. Release-dates are also uniformly generated between 0 and $max(0, d_j - P_j^{max})$. Where P_j^{max} is the maximum total processing time of a job.

For the perturbations, two different distributions were used, Poisson and Erlang [Winston (2003)]. The interarrival time between perturbations was generated using a Poisson distribution with mean $\frac{UB_r}{\theta}$. Higher values of θ result in a lower mean interarrival time, and thus in more perturbations. For the duration of the perturbations we used an Erlang distribution with rate parameter R and shape parameter k, which gives us a mean perturbation duration of $\frac{k}{R}$. For the instances with perturbations we used $\theta = 5$, $R = 2$, and $k = 6$, so we have a mean perturbation time of 3 time units.

Figure 6.2 shows an example of a stochastic instance with two jobs, three resources and two perturbations.

6.2.2 Online Forward Optimization

The online forward optimization method (OFO) is an online scheduling approach which optimizes the schedule in the future, given the information known at time t. Only released operations which have not started yet at time t can be (re-)scheduled. For scheduling these operations a serial schedule generation method with a machine choice optimization procedure is used [Van Peteghem & Vanhoucke (2008)].

The procedure is started at time $t = 0$ with an empty schedule S. If at time t a job is released or a machine is perturbed, then an optimization step is applied. The optimization step will search for a good partial schedule for X iterations. At each optimization iteration, a random but feasible operation list is constructed. This operation list contains all the new operations, and all already scheduled operations

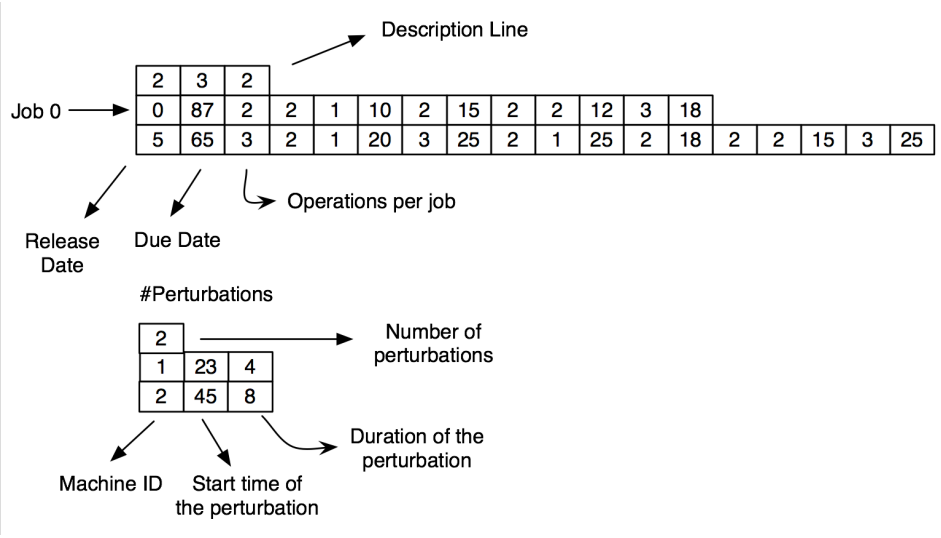

Figure 6.2: Stochastic Flexible Job Shop instance.

from the previous best partial schedule that have not been started yet at time t. Then this operation list is used in a serial schedule generation method [Hartmann (1999)], that will take into account the machine perturbations to generate a new partial schedule. The best partial schedule is kept during the X optimization iterations.

To determine if a partial schedule is better than another one some given objectives are used (e.g. partial makespan). After the optimization step, the scheduled operations are added to the schedule S. If the schedule S already contained some operation, it will use the new values. This procedure continues while there are still unhandled events left, otherwise it stops and returns the found schedule S.

6.2.3 SaFlexS

SAFlexS, which is the abbreviation of Self-Adaptive Flexible Scheduling, is a self-adaptive multi-agent system where the agents interact cooperatively. It can be considered as a real-time scheduling technique that bases its behavior on the constant communication between the agents. At each iteration, the affectations of the available operations on the available machines emerge from the local interactions of the agents respecting the problem specification and constraints. SAFlexS is based on the AMAS (Adaptive Multi-Agent System) Theory [Kaddoum (2011)].

In SAFlexS, two types of cooperative agents are present: Jobs and Machines.

The Job agent has to explore the factory searching for qualified machines for its operations. The behavior of the Machine agent consists in selecting which operation to treat among the different Job agents usage requests.

6.2.4 Learning / Optimization

The Learning/Optimization method used in this scenario is the same that was proposed for the classical FJSSP in the previous chapter, more specifically in Section 4.3. In the presence of a perturbation, a 'right shift' occurs, meaning that the processing time of the operation currently scheduled on the affected machine will be increased. Recalling the dynamics of the approach, agents will receive a feedback signal depending on the quality of the obtained solution.

6.2.5 Experimental Results

The main results obtained by the three approaches are represented using a radar view with five directions, one for the computational time (Calc. time) needed to obtain a schedule, and one for each of the four objectives introduced in Subsection 2.7. These four objectives are:

- makespan, C_{max}
- maximum tardiness, T_{max}
- mean tardiness, $\overline{T}$
- number of tardy jobs, T_n

Each colored line represents one of the approaches. The closer this line lies to the center of the radar the better the result.

We selected some random instances from the created set of problems in order to visualize the results. From the subset of instances without perturbations, but with release and due dates we selected the instances Mk01 (10 jobs, 6 machines) and Mk10 (20 jobs, 15 machines). From the subset of instances with perturbations the selected problems were Mk04 (15 jobs, 8 machines) and Mk06 (10 jobs, 15 machines).

The OFO approach used $X = 50$ as number of iterations, each iteration involves the creation of a feasible schedule and the reoptimization procedure. On the other hand, SAFlexS needs a parameter N (agent life cycle) which is fixed to 26, according

to [Kaddoum (2011)]. In the case of our approach, we keep the QL settings used when solving the traditional FJSSP: $\alpha = 0.1$, $\gamma = 0.8$ and $\epsilon = 0.1$.

Figure 6.3 shows the results for the instances without perturbations, Mk01 and Mk10. This figure shows that our approach, represented by the green line, is outperformed in terms of tardiness, which is expected, because the objective the agents were pursuing was to minimize the makespan.

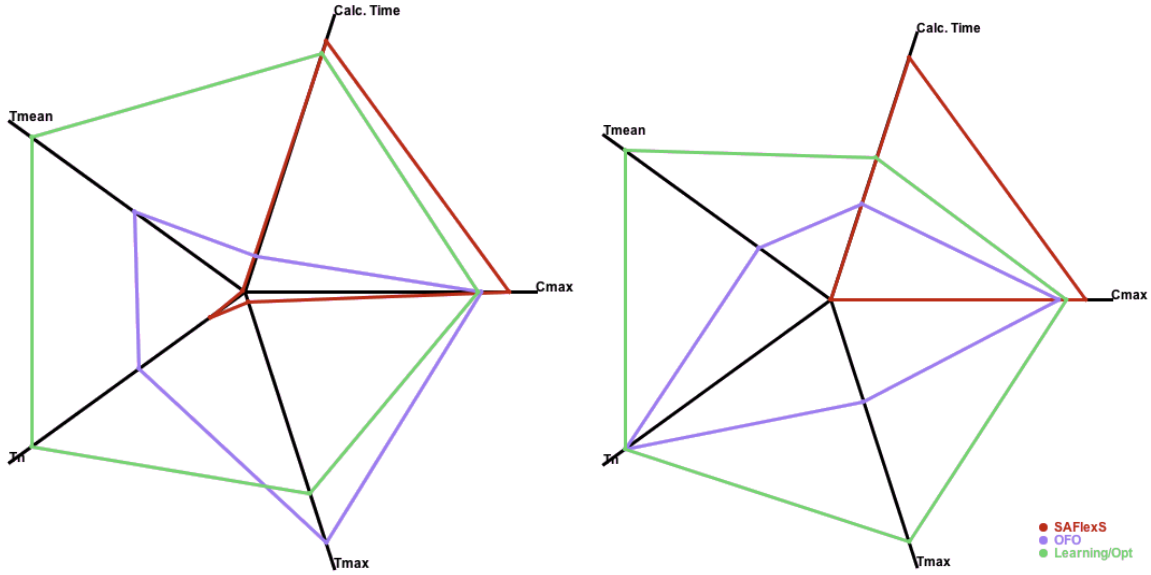

Figure 6.3: Instances Mk01 and Mk10 - No Perturbations

Figure 6.4 shows the results for the instances where some perturbations were involved (Mk04 and Mk06).

As can be seen in Figures 6.3 and 6.4, the results of the different approaches differ from one objective to another. For example, for the Learning/Optimization approach, we stressed on the makespan as objective function, which results in low values for the makespan, but higher values for the other objectives. For the OFO approach the objective was also the makespan, but the global point of view helps improving tardiness objectives. For SAFlexS, no global measure can be performed. The schedule is obtained from the local interaction and decisions of the cooperative agents that concentrate on finishing their work respecting their due dates. This results in very low values for the tardiness objectives but higher values for the makespan.

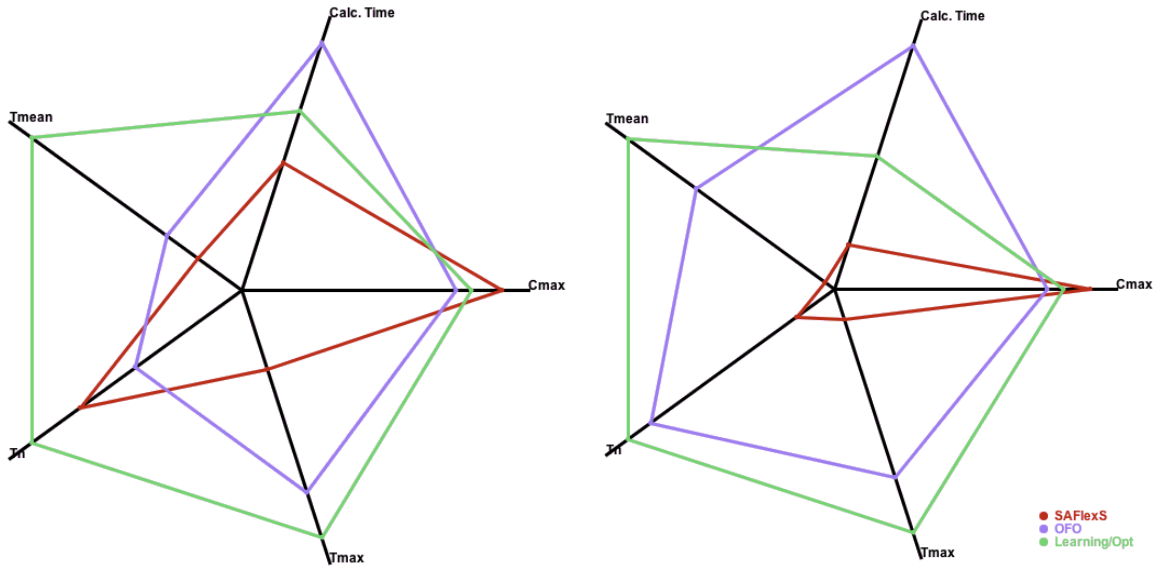

Figure 6.4: Instances Mk04 and Mk06 - With Perturbations

We also performed a last experiment in order to analyze the behavior of our algorithm when the objective is to minimize the tardiness. In this case we change the way to measure the quality of the solutions, which is the feedback sent to the agents. The reward signal is defined as follows:

$$r = \begin{cases} 0 & \text{if } T_{max} > T_{best}, \\ 1 & \text{otherwise,} \end{cases} \tag{6.1}$$

where T_{max} is the tardiness as defined in Section 2.3.3 and T_{best} is the tardiness of the best solution so far. Figure 6.5 shows the results for the instances Mk01 and Mk04 with perturbations.

It is possible to see that our approach gets closer to the center of the radar when having as objective function the minimization of the tardiness. In general, we can say that depending on the environment, and the objective(s) the user is seeking for, different approaches can be used, with possible different feedback signals. We can also conclude that with our method we are able to cover the whole range by simply adapting the feedback signal according to the objective function being specified by the user.

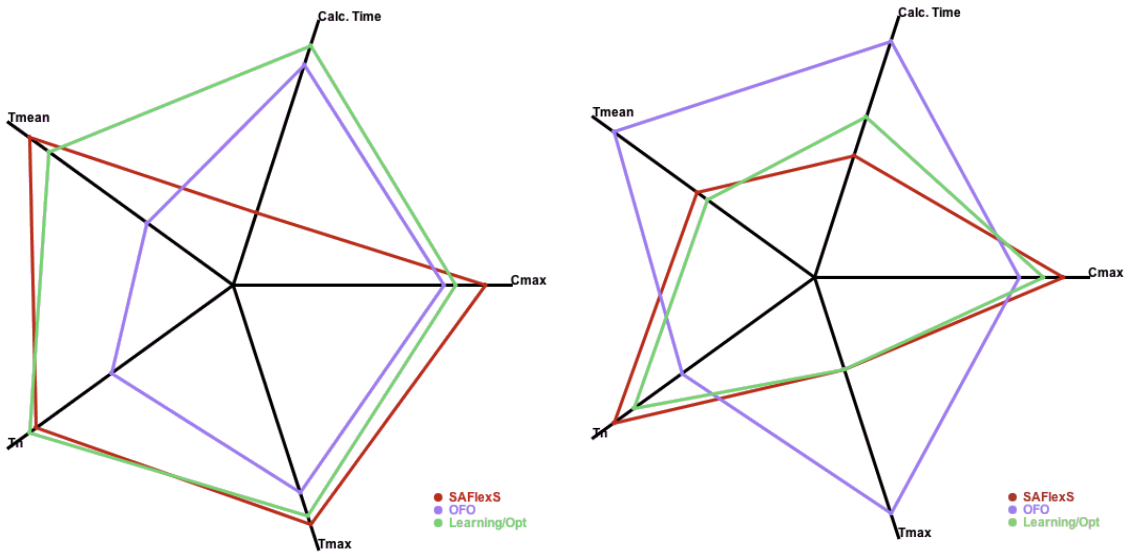

Figure 6.5: Instances Mk01 and Mk04 - With Perturbations and using tardiness as objective function.

6.3 The Proposed Slack-based Approach

In the previous chapter we discussed the importance of adding slack times in those parts of the schedule that are more likely to need them. In this section we propose a new way to incorporate slack into the schedule, the objective is to gain in robustness without loosing too much in terms of optimality. In order to do this we need to define an important concept, which is the 'criticality' of the machines.

6.3.1 'Criticality'

In this section we will use the instance ft06 as an example, this instance was already used for several experiments in previous chapters. Figure 6.6 shows the optimal solution (gantt chart) in terms of makespan for this instance, which is $C_{max} = 55$. In this figure, every job is represented by a color, each block is an operation and the number inside the block is the id of the operation within the corresponding job.

Analyzing this optimal solution, we see that there are two machines that remain busy during the initial part of the scheduling process, these are M_2 and M_3, while there are other machines, like M_5 and M_6 with a higher workload at the end.

The occurrence of an unexpected event, for example, a temporary breakdown on machine M_1 or on M_2 at the beginning of the scheduling period will not have

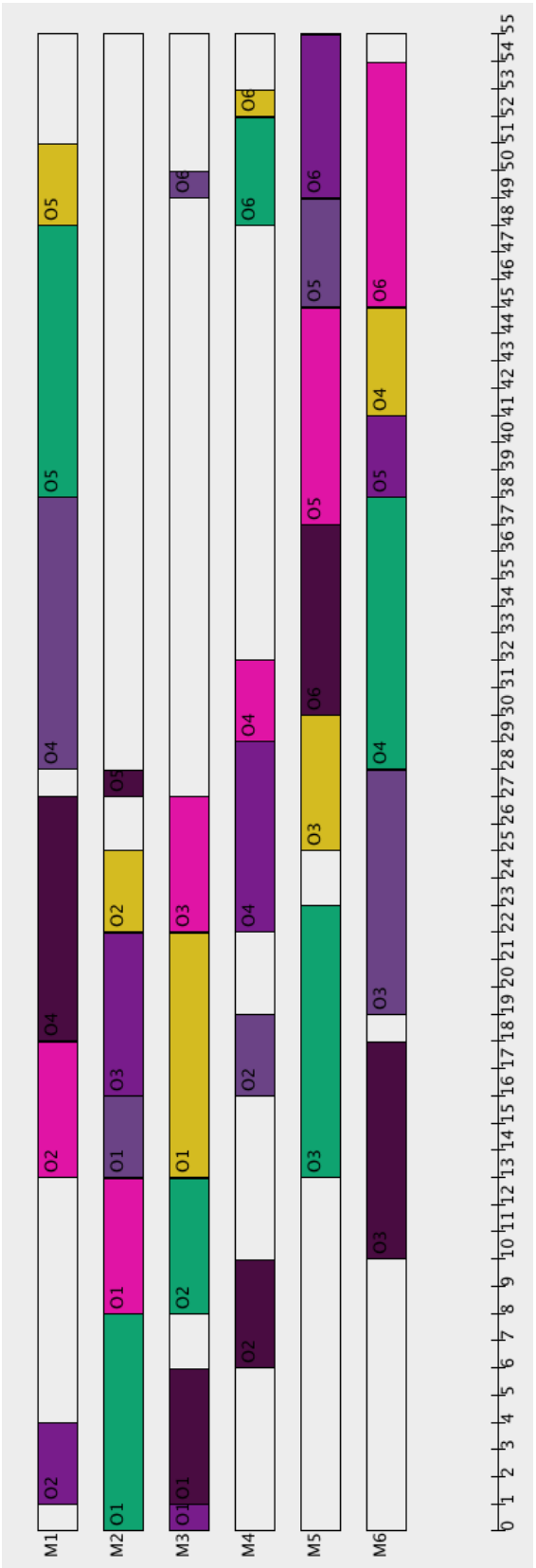

Figure 6.6: Optimal solution for the instance ft06, $C_{max} = 55$

the same impact as a breakdown on M_3. This is due to the fact that M_3 has some idle time in between the different operations, which will allow to absorb part of the breakdown. In other words, is not a critical machine. This analysis leads us to the following definition:

Definition 14 (Critical Machine)**.** *A machine M_i is said to be critical between time t_x and time t_y, if the number of consecutive operations scheduled on that interval is higher than a threshold Z.*

Of course this threshold is a sensitive parameter that will depend on the problem being solved, and which needs to be carefully selected. The user can define when a machine is considered as critical through this threshold and as such, decide on the level of robustness of the schedules generated by the algorithm. In case it is not defined by the user, the approach calculates this threshold depending on the total number of operations that have to be executed by the machine.

6.4 The Proposed Approach

Let us assume that the system for which the optimal schedule was proposed in figure 6.6 will be under the perturbations shown in Table 6.1:

Table 6.1: Set of perturbations

Machine id	Start time	Duration
0	43	4
1	9	3
1	43	5
2	45	1
3	48	3
4	39	4
5	40	3
5	77	4

Under normal circumstances, without having any previous knowledge about the problem at hand, not having any proactive or reactive technique, parallel machines or other solution method, the only choice is to wait for the perturbations to finish in order to resume the execution of the operations on the affected machines.

If we do that for the optimal solution presented in Figure 6.6, then we will obtain the schedule shown in Figure 6.7, for which the makespan is $C_{max} = 64$. In this case we performed a simple 'right shift' every time a perturbation occurred. Perturbations are represented by a block with yellow and red horizontal lines.

Let us assume that the makespan refers to the number of days that it will take to finish the execution of the whole schedule, which is the completion time of an order from an important client. If we compare the two previously shown solutions, we will see that the user will have to wait for 9 extra days in order to get his order.

If we use one of the approaches described in the previous chapter, for example, temporal protection, we could construct a more robust schedule by adding slack times to those operations that are scheduled on breakable resources, which in this case, according to Table 6.1, will be in all the operations, as all the machines will be affected. This will result in a very robust schedule, but with a very low quality. It is important to remember that in these existing slack-based approaches it is assumed that the information concerning the breakdowns, such as the mean down times of the machines, etc, is known.

Figure 6.8 shows the resulting schedule when using temporal protection for the instance ft06 under the unexpected events summarized in Table 6.1. In this case the processing of the order will be expected in 90 days, $C_{max} = 90$. Slack times are represented by a horizontal black line and are added after each block.

It is clear that it is important to properly manage the slack times in order to obtain a solution that is robust enough to deal with unexpected events, but good in terms of quality at the same time. There are two important elements to define: in which part of the schedule to add the slack times, and how long should they be. Here is where the criticality defined in the previous section plays an important role. The approach we propose is based on this definition (Definition 14), as we do not rely on the probability distributions and the information about the machines, but on their workload. If a machine is considered as critical, then the operations scheduled during the critical period will receive an extra amount of processing time.

The process goes as follows: We execute our QL algorithm in order to find a near-optimal solution, the slack-based algorithm analyzes which are the critical machines, and then adds slack times to those operations scheduled on the critical intervals of the resources. As for the amount of extra time to add, we consider that it is something problem dependent, therefore, we give the user the possibility to define it. However, the approach can also compute a value for it based on the

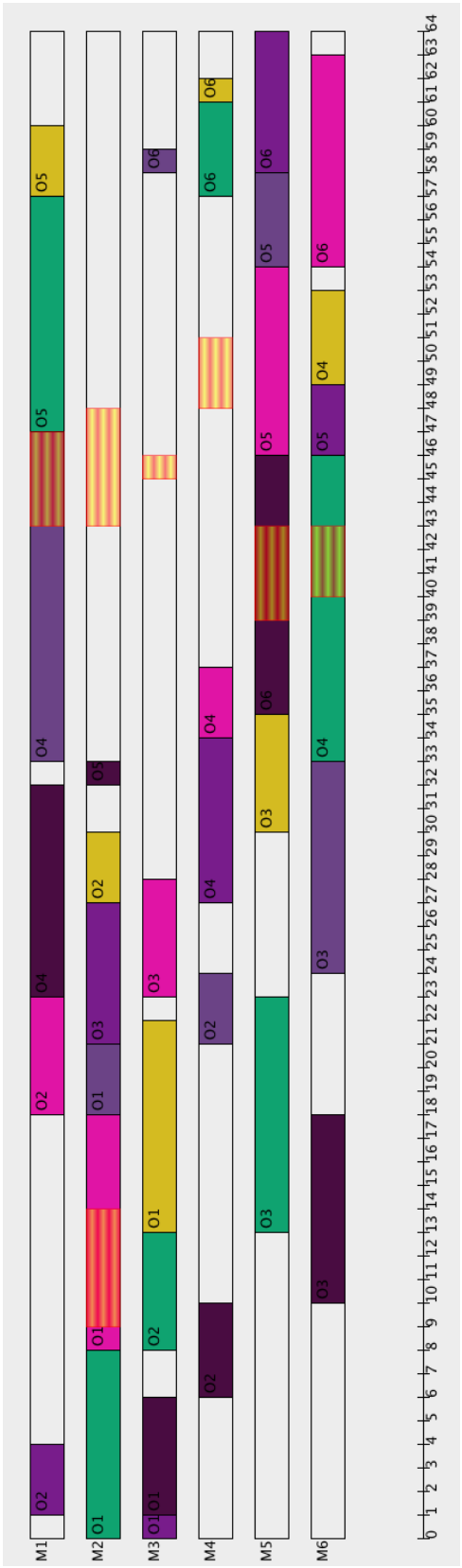

Figure 6.7: Optimal schedule affected by several perturbations, $C_{max} = 64$.

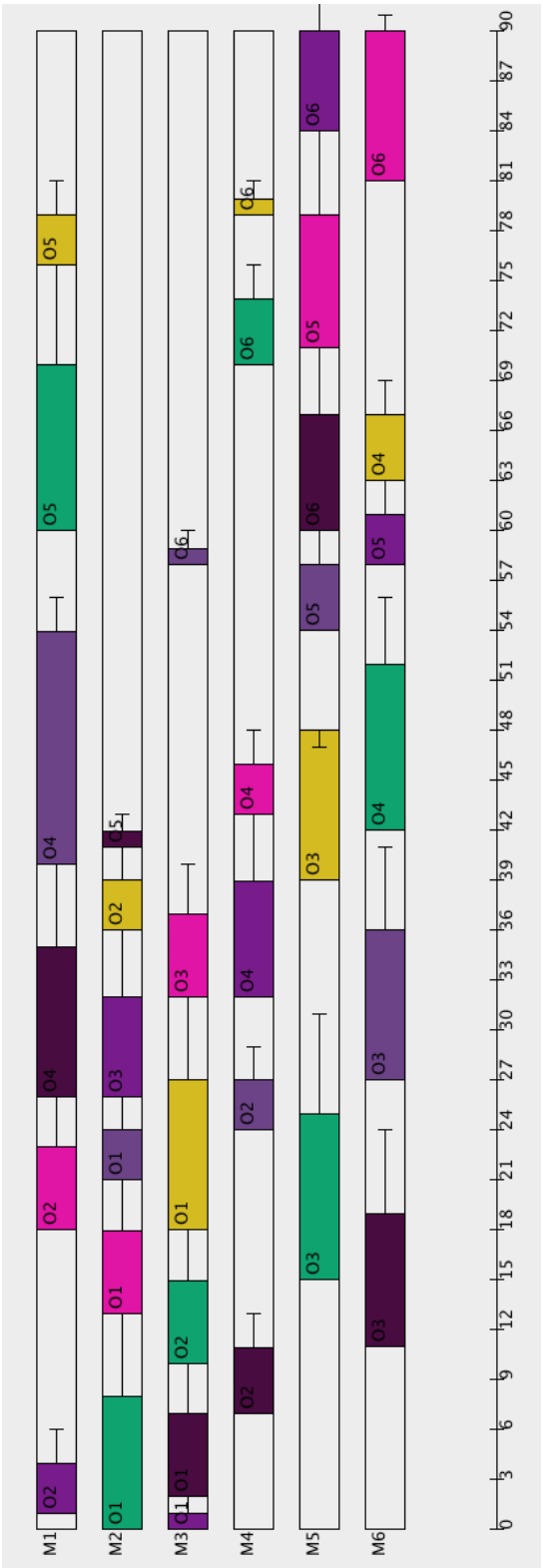

Figure 6.8: Schedule created using the temporal-protection technique.

known information about the machines. More specifically, the slack time to add is computed in the same way as the temporal protection method (see Section 5.4.1). Our objective is to change the way of defining the places where the slack is added, focusing more on critical parts of the solution. The amount of slack to add still follows the classical way, if the user does not define it.

Figure 6.9 shows an example of a schedule constructed using the proposed approach, which means that slack times are only added to the operations scheduled on critical resources. In this case a machine is considered as critical if the number of consecutive operations is higher than three (machines M_2, M_5 and M_6 in this example, see Figure 6.6 to compare). The makespan obtained after this process is $C_{max} = 68$.

An advantage of the proposed approach is that it is possible to define a starting point for the slack times to be added. For example, if the user considers that slack will only be needed after half of the process has passed, this is a constraint that can be easily satisfied.

We also implemented a second way to add slack to the operations scheduled on critical resources. This second way is also based on the criticality of the machines, but instead of taking an already constructed schedule and analyze which are the critical resources, it does the same analysis during the scheduling process. This means that if a machine is identified as critical (following the already defined criteria), the processing times of the operations scheduled on the machine will be extended, and once this new processing time is calculated, it is used to allocate the operations on the machines. This second alternative (i.e. adding slack during the scheduling process) gives the possibility of allocating the operations in a better way once the critical resources have been identified.

Figure 6.10 shows an example of a schedule constructed using this second alternative. If we compare it with the schedule obtained by the previous alternative (Figure 6.9, $C_{max} = 69$), the one depicted in this figure shows how a better schedule (in terms of makespan) can be obtained if the last two operations scheduled on M_2 are swapped. In this case the makespan is $C_{max} = 66$.

In the next section we will introduce the metric that will be used in order to measure how robust these schedules are and how they compare to each other.

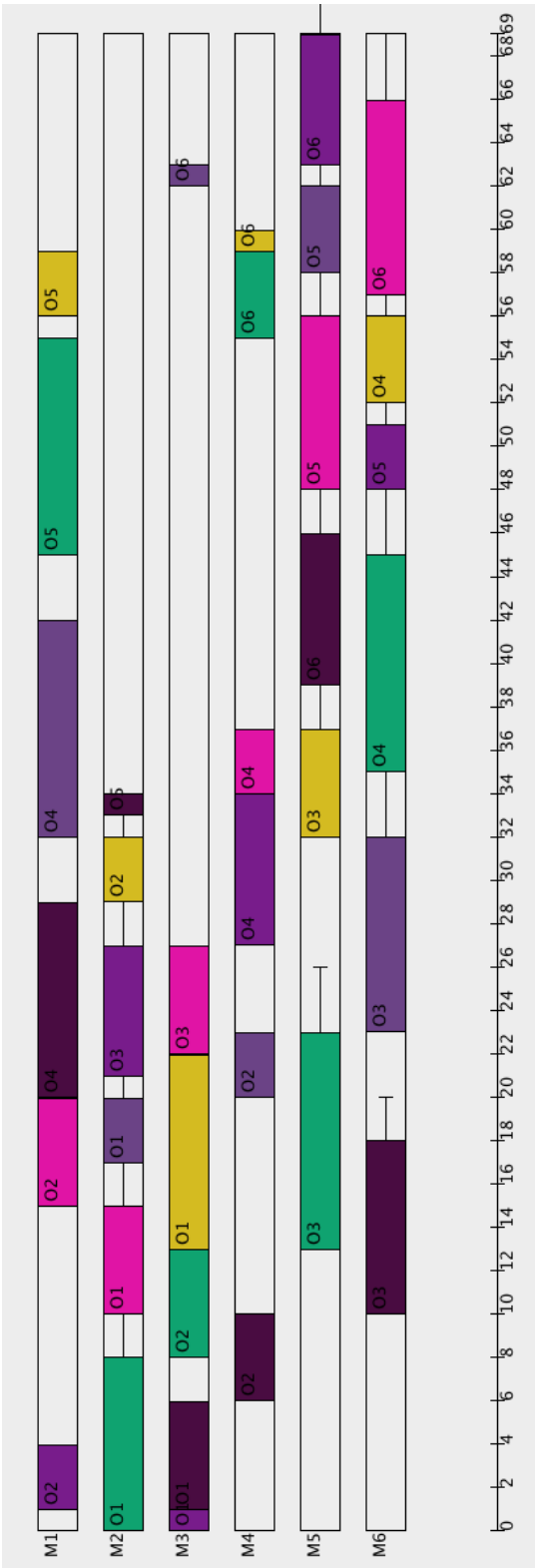

Figure 6.9: Schedule obtained by adding slack times on the critical machines after the scheduling process.

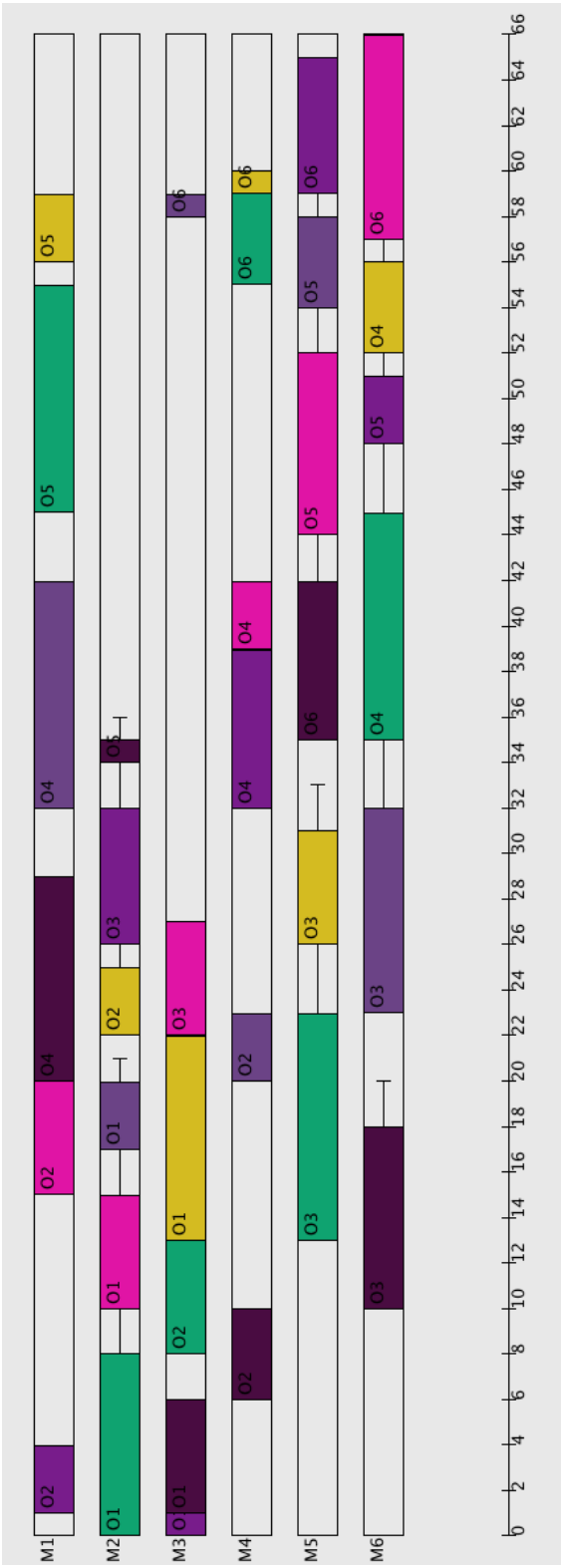

Figure 6.10: Schedule obtained by adding slack times on the critical machines during the scheduling process

6.4.1 Measuring Robustness

In Section 5.2.1 we introduced different metrics to measure how robust a schedule is. In this chapter we will use the first of the proposed metrics, which takes into account the deviation from the original schedule. Figure 6.11 shows a comparison between a non-protected schedule and two schedules generated using our approach based on criticality. The left hand side shows the comparison with a schedule where the slack was added during the scheduling process and the one in the right hand side was obtained by adding slack after generating the solution. A non-protected schedule is a standard solution obtained by any method which does not take uncertainty into account, it could even be an optimal solution. In this case we use a schedule that is constructed by our basic algorithm, described in section 4.1. These schedules are compared in terms of the deviation of its execution from the originally proposed schedule.

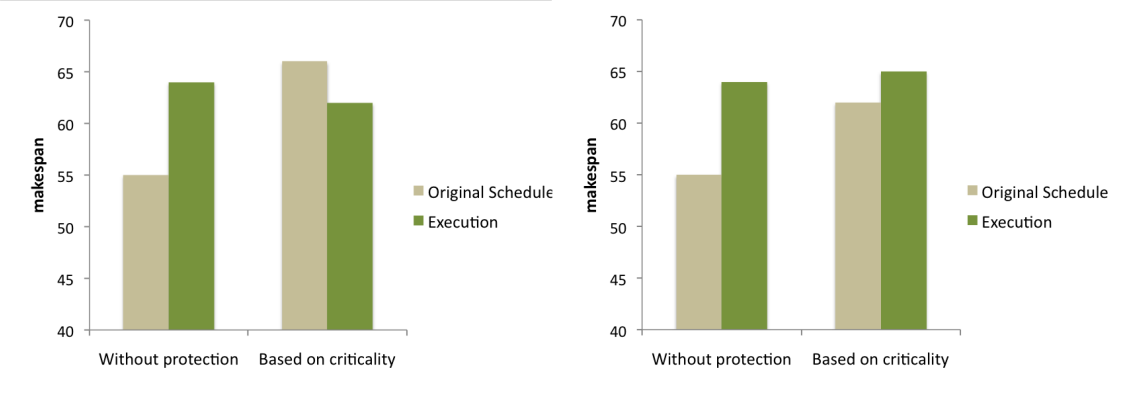

Figure 6.11: Comparing the execution of non-protected schedules and solutions constructed using our approach based on criticality, in terms of the deviation from the original schedules.

From Figure 6.11 we can see that the non-protected schedule was expecting a performance of $C_{max} = 55$, while the real execution was $C_{max} = 64$. The approach based on criticality is able to propose schedules which are closer in time to their real executions. The left hand side of the figure shows a schedule with a deviation of four time units and the right hand side a schedule with a deviation of three time units, while the deviation of the non-protected schedule was nine. We can notice that the alternative of adding slack while constructing the schedule can, in some occasions, obtain schedules with a better execution time compared to the non-

protected solutions. Let us explain with one example how this could compare for example to the temporal protection method described in the previous chapter.

Example 5: *L*et us assume that we want to solve a scheduling problem, for which we know the number of machines and the number of jobs (with their corresponding operations and processing times). The mean down time and mean time between failures of the machines are also provided. In this setting all the machines are considered as breakable, meaning that they are expected to have at least one perturbation. This is how the different algorithms will work:

- Without protection: Solves the problem with the objective of minimizing the makespan, looking for the optimal solution.
- Temporal protection: Checks for the breakable machines and adds slack times to all the operations scheduled on those resources.
- Based on criticality: Solves the problem and analyzes which are the critical machines in the solution found, then adds slack times in three possible ways:
    1. to all the operations scheduled on critical resources (option executed by default);
    2. to those operations scheduled on the critical period(s) of the resource;
    3. to those operations that are scheduled after a specific moment in time.

This could lead us to an outcome like the one shown in Figure 6.12. The temporal protection method is expecting several breakdowns, that is why its proposed schedule has a very high makespan. If we search for a solution without protection, then we are not expecting any disruption, therefore, the proposed schedule is the lowest in makespan. If we go for the criticality-based approach, then we combine a bit of these two behaviors. We add extra time but we do it based on the workload of the machines, instead of using the breakdown probabilities like the temporal protection method, which results in a better balance between optimality and robustness.

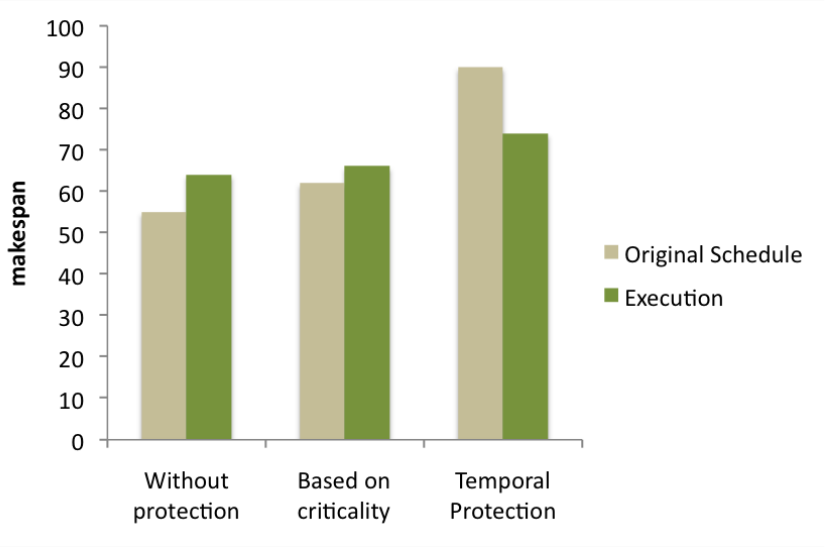

Figure 6.12: Measuring the deviation from the original schedules.

Several instances were generated with different levels of perturbations (1 to 6). By level of perturbations we mean the number of unexpected events that can occur during the scheduling period. This amount increases with the level, meaning that an instance level 6 has more perturbations or breakdowns than an instance level 1.

Figure 6.13 shows a comparison between the approaches according to their deviation from their original schedule using these six levels of perturbations.

It is possible to see that the higher the uncertainty, the worse the performance of the non-protected schedule. This is of course due to the fact that it was not prepared to deal with unexpected events. In the case of the temporal protection technique, it starts with a high deviation from the original schedule, but as the level of perturbations increases, the better results it gets, because the amount of slack time included in the solution when using this technique is usually high. Our approach (criticality) showed a more 'stable' behavior, for the first two levels of uncertainty it was able to keep the same result, but once the unexpected events started to increase, it was also affected, although it was still able to keep the lowest deviation from the original schedule.

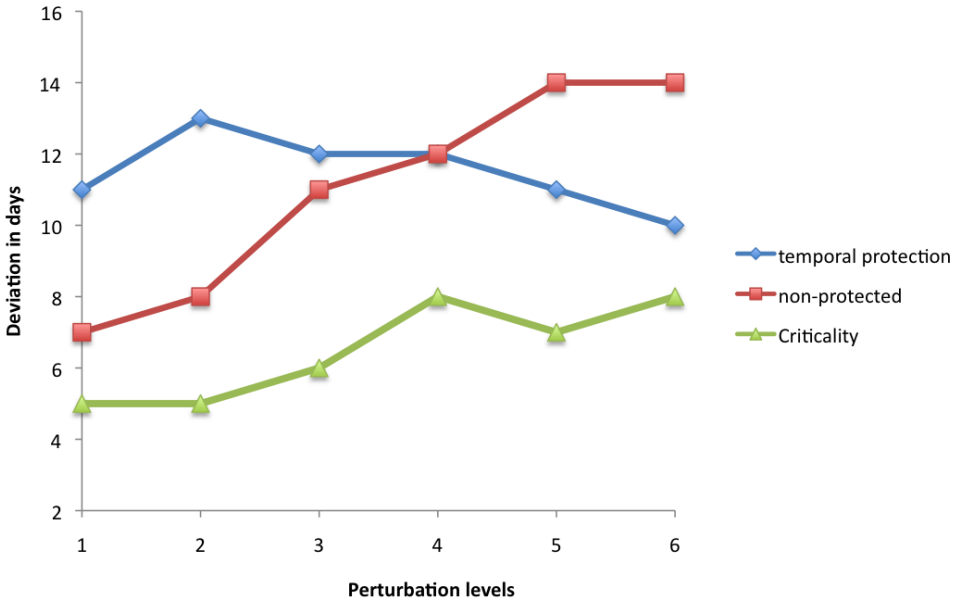

Figure 6.13: Comparison between the approaches according to their deviation from their own original schedule using six levels of perturbations.

Finally, Figure 6.14 shows a comparison of the different approaches in terms of both makespan and deviation from their originally proposed schedule. Two scheduling instances were selected at random, one of them is represented in white and the other one in gray. The numbers 1 and 6 represent the level of perturbations involved in the scheduling process. The non-protected approach was used as a reference in order to show how the slack-based techniques behave. When constructing a non-protected schedule the sole objective is to minimize the makespan, therefore, the approach obtains lower makespan values compared to the other techniques, but the execution of the proposed schedules also presents some level of deviation.

The temporal protection technique shows high makespan values and also starts with a high deviation, but once the level of perturbations increase, the deviation starts to decrease. This is due to the level of protection included in the schedule, which is typically high when using this technique.

Our approach (criticality) presents low values in terms of deviation, yet the makespan remains close to the best solutions reported by the non-protected approach. This is because the objective of the approach is to find a good balance between optimality and robustness.

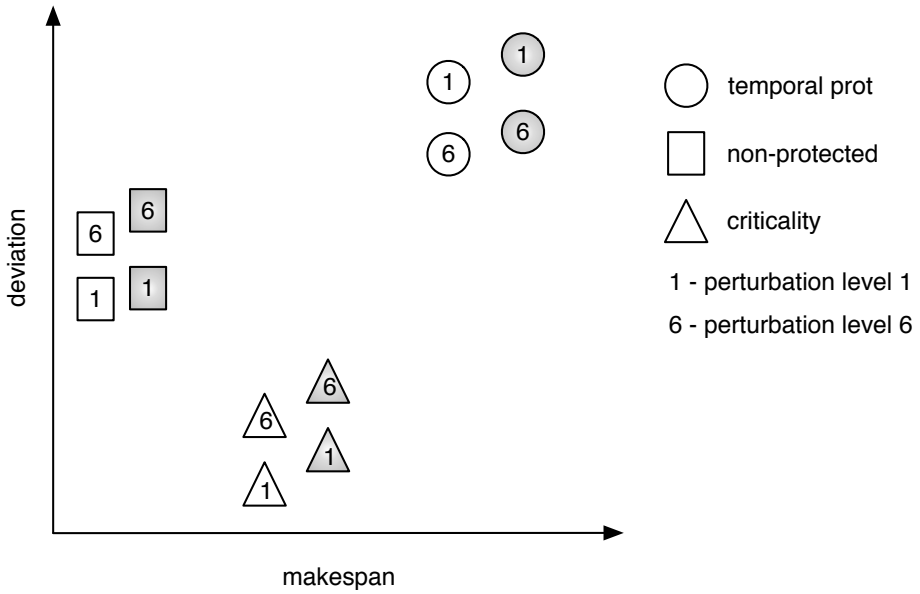

Figure 6.14: Comparison of the different approaches in terms of both makespan and deviation from their originally proposed schedule.

As a conclusion we can say that our approach is able to focus the slack times in parts of the schedule where they are more needed, by making use of the criticality of the machines. The experiments showed that the deviation from the original schedule is lower than for example a non-protected schedule, or a schedule generated using the temporal protection method.

6.5 Hybrid Flow Shop Scheduling

The last problem that will be addressed in this dissertation is the Hybrid Flow Shop Scheduling, which is based on the real-world case described in Section 2.9. This process has four phases: Seed Fermentation (SF), Main Fermentation (MF), Broth Preparation (BP) and Filtration, also called Recovery Line (RL). The proposed learning approach is based on the Q-Learning algorithm. Each resource (i.e. fermenters, tanks and recovery line) has an agent associated, which will be responsible for deciding the next batch to process on its corresponding resource. This means that the set of actions is equivalent to the set of batches that are ready to be

processed on the phase to which the agent belongs.

At the beginning of the process there is a list of orders that have to be executed, each of them with its corresponding due date. Batches start their processing on a seed fermenter, and once they have been fully processed on that phase, they become available for the next phase of the process, as described in Section 2.9.

Table 6.2 shows an example with 6 types of products (P_1 to P_6). For each of them the processing times (in hours) corresponding to the four different phases of the problem are given. Column *Sum PTimes* shows the sum of all the processing times and *Due Date* is the time when the batch is expected to be finished.

Table 6.2: Processing times of the six types of products in the different phases of the process.

Product ID	SF	MF	BP	RL	Sum PTimes	Due Date
P_1	64	232	24	32	352	400
P_2	65	183	16	16	280	310
P_3	24	64	16	16	120	170
P_4	40	160	16	16	232	270
P_5	40	144	16	32	232	290
P_6	40	160	16	16	232	290

According to the characteristics of the learning algorithm, to choose an action, each agent takes into account the Q-values associated to the set of possible actions it can execute. These Q-values are initialized arbitrarily, and they are updated according to Equation 3.5.

The objective in this problem is to minimize the tardiness. We will use the same approach as in the previous scenario, where the agents receive a reward according to the solution quality, and this reward is given by Equation 6.2.

$$r = \begin{cases} 0 & \text{if } T_{max} > T_{best}, \\ 1 & \text{otherwise,} \end{cases} \tag{6.2}$$

Where T_{max} is the tardiness of the current solution as defined in Section 2.3.3, and T_{best} is a variable that keeps the tardiness of the best solution so far.

As there was no information about previous schedule executions, or about previous breakdowns on the machines, we executed the algorithm several times in order to visualize the solutions and estimate which resources could be considered as critical. Figure 6.15 shows an example of a solution for this problem. In this case we see 12 machines, which are grouped in the following way:

- Seed Fermentation: $\{M_1, M_2\}$
- Main Fermentation: $\{M_3, ..., M_6\}$
- Broth Preparation: $\{M_7, ..., M_{11}\}$
- Recovery Line: $\{M_{12}\}$

It is important to remember that queues are not possible in this environment. If the processing of a batch is finished and there is no resource available in the next stage, the batch has to remain in the current resource until it can move forward.

When analyzing the problem, the attention could initially be focused on the recovery line, because only one batch can be processed at a given time, and there is only one resource in this stage of the process. But if we analyze the solution presented in Figure 6.15, it is possible to see that the seed fermentation is actually a red spot, as both fermenters are constantly working (five batches are scheduled one after the other). We can also notice that batches take longer to process in the main fermentation phase, and less during broth preparation, which somehow allows to make the process in the recovery line more active.

From this analysis we conclude that the resources belonging to the seed fermentation stage are the ones that can be considered as critical. Therefore, is in this part of the schedule where slack time might be needed in order to make the schedule more robust. In this case we incorporate the slack based approach proposed in Section 6.4.

6.5.1 Experimental Results

In this section, we present the results of the computational experiments for 11 problem instances involving 10, 15, 20 and 30 batches. The results are compared with the optimal solutions (presented in [Gicquel et al. (2012)] and calculated using the CPLEX solver), and with the results reported by a Genetic Algorithm proposed in [Borodin et al. (2011)].

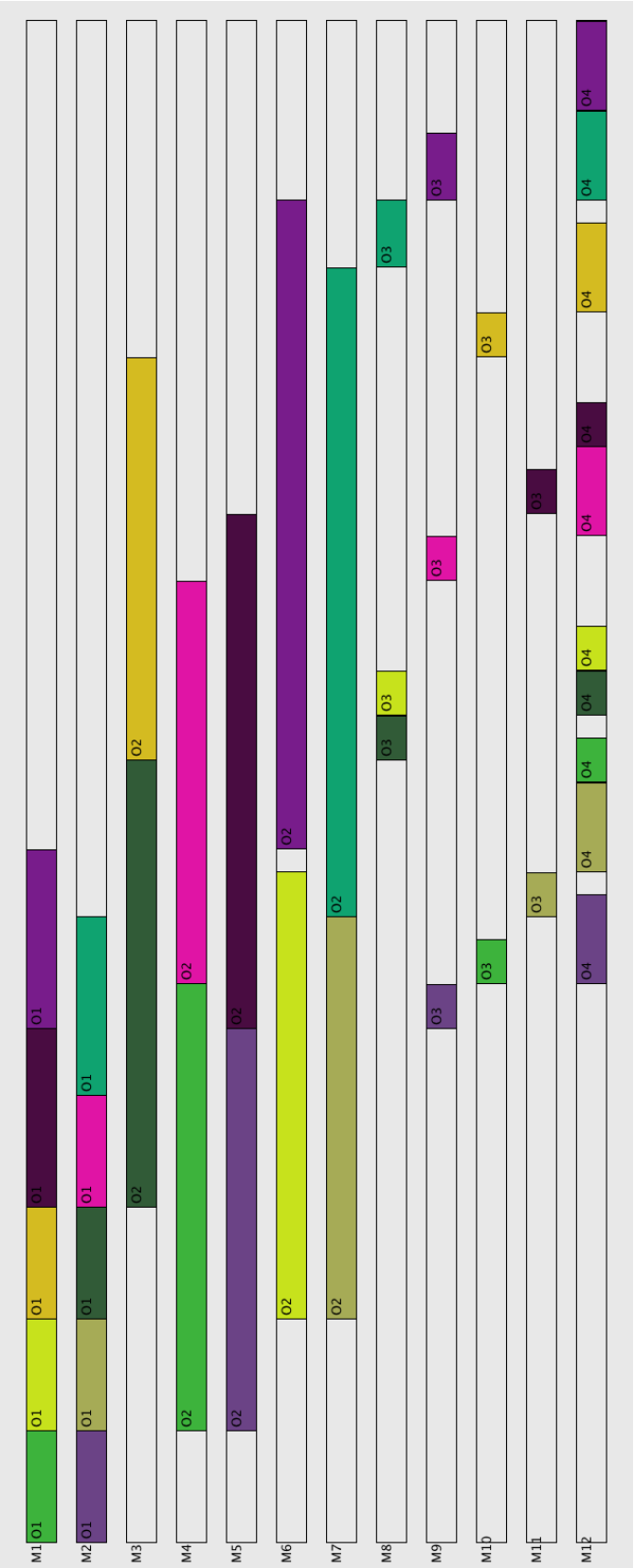

Figure 6.15: Example of a possible solution for the hybrid flow shop problem.

Table 6.3 represents the overall comparison of the results. The *CPLEX best* column contains the optimal solutions, *GA best* - the GA results, computed in *GA time*. Columns *QL best* and *QL time* show the results corresponding to the Q-Learning algorithm.

As the objective of both approaches is to minimize the tardiness, the columns showing the best solutions are reporting the weighted tardiness for each of the instances. The columns reporting the computational times show the time (in seconds) it took the algorithm to find the corresponding solution.

There are some parameters that need to be defined for the execution of the learning algorithm. The learning rate (α) used was 0.1, the discount factor (γ) 0.8 and the epsilon parameter for the action selection strategy was 0.1. The algorithm was executed for 10000 iterations, as in the previous scenarios.

From Table 6.3 it is possible to see that the GA outperforms the QL in most cases. It is important to notice that the solutions of the learning algorithm are close in quality to the best found, but the computational time is always significantly lower, also we have to remember that its main goal was to provide robust schedules that could handle some level of uncertainty.

Table 6.3: Overall comparison table in terms of tardiness.

Instance	CPLEX best	GA best	GA time	QL best	QL time
N10_1	90	93	5	98	0.7
N10_2	30	30	17	37	1.5
N10_3	42	44	6	54	0.9
N10_4	49	50	5	59	1.3
N10_5	43	45	12	58	1.5
N15_1	73	76	25	84	3.3
N15_2	43	45	34	56	3.6
N15_3	57	66	75	75	3.4
N20_1	52	54	180	61	5.4
N20_2	58	64	194	75	5.4
N30_1	-	186	1560	194	6.5

6.5.2 Conclusions about the Hybrid Flow Shop Experiments

The main conclusion is that both the GA and RL approaches are able to efficiently solve the considered batch scheduling problem. Each approach showed its own benefits. According to [Borodin et al. (2011)], the GA represented a classical approach which was based on a Mixed Integer Linear Programming (MILP) model. Solutions of high quality were obtained but a drawback of this classical approach is its low robustness - indeed, the schedule that corresponds to a high-quality solution is as tight as possible and not ready to be adapted in case an unexpected event occurs. In this case, a new schedule that takes into account the occurred events must be computed, which means that the problem will be solved from scratch.

The Reinforcement Learning approach is capable of handling this issue. Its main goal is to quickly compute a robust schedule. For instance, in real situations it often happens that a planner needs a schedule in a few seconds to report to the customer whether the new order can be processed by this or that due-date. In this approach it is also possible to start from an initial use of the machines, i.e. non-empty resources. This is useful for example in the case when orders are not finished during the day, and a new schedule has to be constructed for the next day, but some machines still have work to finish.

Computational results proved that the GA is able to obtain very good solutions (comparing to the exact solutions), whereas the learning algorithm was able to obtain robust solutions close to the optimal ones, and it reported significantly lower computational time.

In overall, this case study demonstrated that a real-life complex hybrid flow-shop scheduling problem, which is difficult to solve by exact methods in the standard MILP formulation, can be efficiently solved by a meta-heuristic technique - on the one hand; and formulated in a lighter way to be suitable for obtaining more robust schedules faster and easier by an RL method - on the other.

6.6 Summary

In this chapter we presented the results of our approach when dealing with unexpected events. We started by introducing a particular case, which we call known disruptions and then we explained the process of generating new FJSSP instances in order to incorporate extra information to the problem, such as release dates, due

dates and machine breakdowns. The concept of criticality is defined and a new slack-based approach is proposed, in order to deal with uncertainty in a proactive way. This approach uses some of the ideas behind the existing slack based techniques, but presents a new way to add the extra execution times to the operations. The main idea is to provide a solution that is robust enough to handle some level of uncertainty, without loosing too much in optimality. Finally, we presentd some results obtained from the application of a learning algorithm to the solution of a hybrid flow shop scenario, which was based on a real-world case.

Chapter 7

Conclusions

Scheduling problems are present in every situation where a given set of tasks that requires the allocation of resources to time slots has to be performed. Constructing a schedule is a common procedure that we usually perform in our daily life, but when the number of constraints that have to be met increase and the number of tasks and resources grow, then constructing a schedule that satisfies all the requirements is not so straightforward.

Scheduling is an active research field in which several important challenges arise. For example, dealing with uncertainty is a research topic that is recently receiving more attention. It is common to face unexpected events in scheduling environments, specially in manufacturing, which is the kind of problem this dissertation focuses on. In manufacturing environments machines can break down, orders can take longer than expected and all these events will make the original schedule fail. That is why it is preferable to have a solution that is robust enough to absorb some level of uncertainty, instead of a possibly optimal one, which will fail in the presence of any small disruption.

In this thesis we presented a generic multi-agent reinforcement learning approach that can easily be adapted to different scheduling settings. It is possible to increase the robustness of the solutions being constructed and also to look at different objective functions, like the tardiness or the makespan. Furthermore, the proposed approach allows the user to set, in a transparent way, certain parameters involved in the solution construction process, in order to define the balance between robustness and optimality. In the next section we give an overview of the main research results presented in this dissertation.

7.1 Contributions

We started this dissertation by presenting an overview of the scheduling theory. A classification of the different types of schedules that can be obtained was presented, together with several scheduling scenarios, in which the complexity (given by the constraints of the problem) was gradually increasing. These problems are listed below and were all addressed in this dissertation.

- Job Shop Scheduling Problem (JSSP)
- Parallel Machines Job Shop Scheduling Problem (JSSP-PM)
- Flexible Job Shop Scheduling Problem (FJSSP)
- Stochastic Flexible Job Shop Scheduling Problem
- Online Scheduling (two phases, four machines each)
- Hybrid Flow Shop Scheduling (based on a real-world problem)

In Chapter 3, besides giving an overview of Reinforcement Learning and Multi-Agent Reinforcement Learning, we introduced the main concepts that have to be taken into account when solving a scheduling problem using Reinforcement Learning. Important definitions like changing action sets and transition dependencies were adapted from literature and introduced using examples.

Chapter 4 started by introducing in detail what we call the 'basic' learning approach, which is initially defined for the job shop scheduling problem and later on adapted in order to solve the other scenarios described in Chapter 2. The learning method implemented was a Q-Learning algorithm, for which different feedback signals were studied. This QL algorithm was adapted and combined with other techniques in order to satisfy the extra constraints of the parallel machines job shop scheduling and the flexible job shop scheduling. In the latter, an optimization technique was used in order to improve the quality of the solutions. In all the cases the results were compared with state of the art techniques reported in literature and our approach was able to obtain comparable results.

In the case of the online scheduling problem we used Learning Automata to learn how to distribute the incoming orders of two types of products among parallel machines. Experiments were developed using the three possible update schemes. The results of the experiments showed that the L_{R-P} scheme was able to yield better results compared to the other two reward schemes (L_{R-I} and $L_{R-\epsilon P}$).

The more common sources of uncertainty in scheduling were summarized in Chapter 5. Important questions were answered, such as: what is robustness? and how to measure how robust a schedule is?. Furthermore, different types of approaches to deal with uncertainty were studied. These methods can be classified as reactive or proactive. We explained in detail some existing proactive approaches, mainly slack-based techniques, as this background was needed to understand the new slack-based method we proposed, which was introduced later in Chapter 6. It is also in this chapter that experiments concerning stochastic scheduling are performed. In order to measure the performance of the approach under uncertainty, several problem instances were generated. The process of generating the instances was thoroughly explained such that these could be used as a benchmark for further research.

The new slack-based method proposed in this chapter bases its behavior on something we defined as the criticality of the machines. This concept is used in order to define the parts of the schedule that are more likely to need extra execution time. Experiments in this regard are also reported, in which the main idea was to obtain solutions with a good balance between optimality and robustness. This approach was applied when solving the hybrid flow shop scheduling scenario, which is based on a real-world application. An important added value of our approach is the fact that it allows the user to set some parameters involved in the solution construction process, for example, the desired level of robustness, in a very intuitive way. The contributions are graphically summarized in Figure 7.1.

In general, we receive a scheduling problem, and additionally there can be a set of user specifications. These user specifications can be preferences, extra constraints to the problem, or preferred values for specific parameters. All this information is given to our MARL approach, which depending on the characteristics of the problem and the user specifications, will solve it on its own, or in combination with the mode optimization procedure and/or the slack-based approach. This will provide as output a near-optimal schedule, which can have different levels of robustness. It is also possible to identify which are the critical machines and provide a visualization of the resulting schedule in the form of a gantt chart. This visualization helps the user to carefully analyze the proposed solution, and maybe change some parameter settings in order to analyze what will happen if the system is under different circunstances, or just to get a better balance between robustness and optimality.

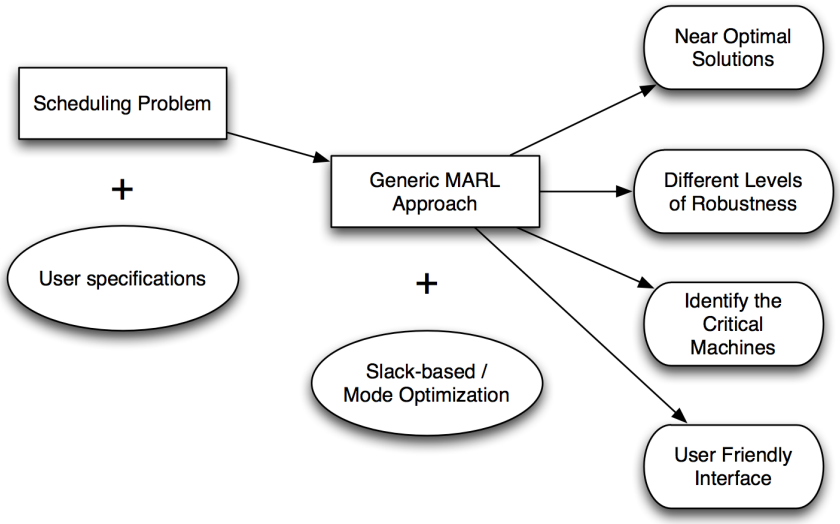

Figure 7.1: Graphical representation of the contributions of this dissertation.

7.2 Future Work

In this section we list some ideas to extend the research presented in this dissertation.

- **Integrated Approach:** In Chapter 4 we proposed a hierarchical approach for the Flexible Job Shop Scheduling Problem, which was also used to solve the stochastic version of this problem. As explained before, hierarchical approaches divide the problem in two steps: routing and sequencing, following the classical 'assign-then-sequence' rule, which means that a machine is assigned to each operation and once this machine-operation assignment has been done, the operations are sequenced on the assigned resources. It will be interesting to explore an integrated approach, which takes care of the routing and the sequencing at the same time.

- **Multiple objectives:** When solving the stochastic flexible job shop scheduling in Chapter 6, new instances were generated in order to measure the performance of the algorithm under unexpected events. These instances had release and due dates incorporated, which made it possible to report results either in terms of makespan or in terms of tardiness. The possibility of dealing with

multiple objectives at the same time would also be an interesting avenue for future research. This would mean that the feedback signals and the objective functions of the current approach should be re-defined.

- **LA for unexpected events:** In Chapter 4 we proposed a solution to an online scheduling problem using LA. In this case the information about how many jobs have to be processed and their corresponding processing times is not known in advance. The experiments performed in this scenario showed that LA can cope with unknown joblengths, unknown future jobs and unknown machine speeds, without needing heuristics or additional information. We could take advantage of this and incorporate LA in the slack-based approach proposed in this dissertation. This will allow to learn about the criticality of the machines and improve the way the robust solutions are constructed for a specific type of problem.

- **Highly constrained hybrid flow shop scheduling problem:** In the scheduling community it is common to work with benchmarks, which not always represent all aspects of real life scheduling problems. It is well known that industries need systems for optimized production scheduling that adjust exactly to the conditions in the production plant. One case of such a problem is the highly constrained hybrid flow shop scheduling scenario described in [Urlings (2010)]. This problem involves some extra constraints that were not present in the scheduling scenarios addressed in this dissertation. The application of our approach to such a complex environment is currently under study.

- **Cuban real-world application:** In the cuban context, more specifically in companies responsible for the production of spare pieces for industrial machines, there is a lack of effective methods that can contribute to the scheduling of the production process. The repair shop 'Manual Fajardo', located in Granma, a province at the east of Cuba, is responsible for the manufacturing (and repairing) of several pieces of equipments that are used by the different machines involved in the sugar production process. The production chain is structured in a way that all the tasks have to follow the same production path, which means that they are executed by the machines in the same order. This

fits the definition of flow shop scheduling.

The repair shop is experiencing delays, mainly due to the arrival of extra orders which sometimes have high priorities due to the kind of pieces being requested. This kind of requests are more common in the sugar harvesting season.

This is a potential real-world application that could benefit from the research presented in this dissertation.

Curriculum Vitae

Education

- Bachelor in Computer Science, Universidad Central 'Marta Abreu' de Las Villas (UCLV), Cuba, 2004.
- Master in Computer Sciences, Universidad Central 'Marta Abreu' de Las Villas (UCLV), Cuba, 2007.
- Master of Computer Sciences, Vrije Universiteit Brussel (VUB), Belgium, 2008.

Publications

Journals, Book Chapters and Lecture Notes

- Martínez Y., Nowé A., Suárez J., Bello R. (2011) *A Reinforcement Learning Approach for the Flexible Job Shop Scheduling Problem.* Lecture Notes in Computer Science. Proceedings of the fifth International Conference on Learning and Intelligent Optimization(LION5); Vol.6683:pp.253-262.
- Martínez Y., Van Vreckem B., Catteeuw D., Nowé A. (2010) *Application of Learning Automata for Stochastic Online Scheduling.* Recent Advances in Optimization and its Applications in Engineering. Springer Berlin Heidelberg; Eds. Diehl M., Glineur F., Jarlebring E., Michiels W.; pp.491-498.
- Puris A., Bello R., Martínez Y., Trujillo Y., Nowé A. (2008) *Two-stage ACO to solve the job shop scheduling problem.* Lecture Notes in Computer Science. Book 'Progress in Pattern Recognition, Image Analysis and Applications'; Vol.4756:pp.447-456.

- Rodríguez L., Casas G., Grau R., Martínez Y. (2008) *Fuzzy Scan Method to Detect Clusters.* International Journal of Biological and Life Sciences; Vol.3: pp.111-115.
- Puris A., Bello R., Martínez Y., Nowé A. (2007) *Two-stage ant colony optimization for solving the traveling salesman problem.* Lecture Notes in Computer Sciences. Book 'Nature Inspired Problem-Solving Methods in Knowledge Engineering'. Vol.4528(1):pp.307-316.
- Puris A., Bello R., Nowé A., Martínez Y. (2006) *Two-Stage Ant Colony System for Solving the Traveling Salesman Problem.* Journal Research in Computing Science.Vol.26(1):pp.85-92.

Conference Proceedings

- Martínez Y., Nowé A., Suárez J., Bello R. (2011) *Solution to the Flexible Job Shop Scheduling Problem using Reinforcement Learning.* In: Proceedings of the 14th International Convention and Fair Informatica 2011. Havana, Cuba.
- Kaddoum E., Martínez Y., Wauters T., (2010) *Adaptive methods for flexible job shop scheduling with due-dates , release-dates and machine perturbations.* In: Workshop on Self-tuning, self-configuring and self-generating search heuristics (Self* 2010). Krakow, Poland; 2010.
- Martínez Y., Wauters T., Nowé A., Verbeeck, K., De Causmaecker P., Bello, P. (2010) *Reinforcement Learning Approaches for the Parallel Machines Job Shop Scheduling Problem.* In: Proceedings of the Cuba-Flanders Workshop on Machine Learning and Knowledge Discovery. Santa Clara, Cuba.
- De Causmaecker P., Verbeeck K., Wauters T., Martínez Y. (2010) *Scalable decentralised approaches for job shop scheduling.* In: Proceedings of the 24th European Conference on Operational Research EURO XXIV. Lisbon, Portugal.
- Wauters T., Martínez Y., De Causmaecker P., Nowé A., Verbeeck K. (2010) *Reinforcement Learning approaches for the Parallel Machines Job Shop Scheduling Problem.* In: Proceedings of ITEC2010 - International conference on interdisciplinary research on technology, education and communication. Kortrijk, Belgium.

- Martínez Y., Van Vreckem B., Catteeuw D., Nowé A. (2009) *Multi-Stage Scheduling Problem with Parallel Machines*. In: Proceedings of BFG'09, the 14th Belgian-French-German Conference on Optimization. Leuven, Belgium.
- Martínez Y., Nowé A. (2009) *A Multi-Agent learning approach for the Job Shop Scheduling Problem*. In: Proceedings of the 23rd European Conference on Operational Research. Bonn, Germany.
- Puris A., Bello R., Nowé A., Martínez Y. (2006) Improving Ant Colony Optimization to solve the Travelling Salesman Problem. In: Proceedings of 18th Benelux Conference on Artificial Intelligence (BNAIC 2006).

Participation in Conferences and Summer Schools

- European Workshop on Reinforcement Learning. September 9-11, Athens, Greece, 2011.
- ACAI Summer School on Automated Planning and Scheduling, ACAI'11. Freiburg, Germany, June 7-10, 2011.
- Symposium 'OR problems and AI techniques', March 2011, KULAK, Kortrijk.
- Learning and Intelligent OptimizatioN, LION 5, January 17-21, 2011, Rome, Italy.
- 11th International Conference on Parallel Problem Solving from Nature (PPSN). Workshop on Self-tuning, self-configuring and self-generating search heuristics (Self* 2010), Krakow, Poland, September 11-15, 2010.
- Cuba-Flanders Workshop on Machine Learning and Knowledge Discovery, Santa Clara, Cuba, February 5-8, 2010.
- Summer School: Advanced Course in Artificial Intelligence, ACAI'09, Belfast, August 23-29, 2009.
- 14th Belgian-French-German Conference on Optimization, Leuven, September 14-18, 2009.
- 23rd European Conference on Operational Research, Bonn, Germany, July 5-8, 2009.

- Eight International Conference on Autonomous Agents and Multiagent Systems, Budapest, Hungary, May 10-15, 2009.
- Doctor Honoris Causa, Louvain-La-Neuve, Belgium, 2008.
- TAAME 07 - Theory And Applications of MEtaheuristics - Tutorial Day, KaHo Sint Lieven, Gent, Belgium, 2007.
- Biologically inspired robotics: From evolving to self replicating and self repairing machines, Iridia - ULB, Belgium, 2007.

Bibliography

[Aloulou & Portmann (2005)] Aloulou, M. A. & Portmann, M.-C. (2005). An efficient proactive-reactive scheduling approach to hedge against shop floor disturbances. In: *Multidisciplinary scheduling: theory and applications* (Kendall, G., Burke, E. K., Petrovic, S. & Gendreau, M., eds.). Springer US, pp. 223–246. Cited on pages 108 and 109.

[Auer et al. (2002)] Auer, P., Cesa-Bianchi, N. & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning* **47**, 235–256. Cited on page 54.

[Baker (1974)] Baker, K. (1974). *Introduction to sequencing and scheduling.* John Wiley & Sons. Cited on page 24.

[Beasley (1990)] Beasley, J. E. (1990). OR-Library. URL `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`. Cited on pages 26 and 74.

[Beck & Wilson (2007)] Beck, J. & Wilson, N. (2007). Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research* **28**(1), 183–232. Cited on page 108.

[Bellman (1957)] Bellman, R. (1957). *Dynamic Programming.* Princenton, NJ, USA: Princeton University Press. Cited on pages 48 and 49.

[Ben-Tal & Nemirovski (2000)] Ben-Tal, A. & Nemirovski, A. (2000). Robust solutions of Linear Programming problems contaminated with uncertain data. *Mathematical Programming* **88**, 411–424. Cited on pages 103 and 106.

[Bernstein et al. (2002)] Bernstein, D., Givan, R., Immerman, N. & Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* **27**(4), 819–840. Cited on pages 57 and 58.

[Bertsekas (1995)] Bertsekas, D. (1995). *Dynamic Programming and Optimal Control (Volume 1).* Athena Scientific, third ed. Cited on page 48.

[Bertsimas & Sim (2004)] Bertsimas, D. & Sim, M. (2004). The price of robustness. *Operations research* **52**(1), 1–34. Cited on page 106.

[Billaut et al. (2008)] Billaut, J., Moukrim, A. & Sanlaville, E. (2008). *Flexibility and robustness in scheduling.* Wiley. Cited on pages 3, 106, and 107.

[Borodin et al. (2011)] Borodin, D., Van Vreckem, B. & De Bruyn, W. (2011). Case study: an effective genetic algorithm for the chemical batch production scheduling. In: *Proceedings of the 5th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA 2011).* Phoenix, Arizona. Cited on pages 34, 140, and 143.

[Brandimarte (1993)] Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research* **41**(3), 157–183. Cited on pages 90, 91, and 119.

[Brucker (2007)] Brucker, P. (2007). *Scheduling Algorithms.* Springer. Cited on page 12.

[Bush & Mosteller (1955)] Bush, F. & Mosteller, R. R. (1955). *Stochastic Models for Learning.* Wiley. Cited on page 51.

[Cassandra (1998)] Cassandra, A. R. (1998). *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes.* Phd thesis, Brown University. Cited on page 47.

[Castillo & Roberts (2001)] Castillo, I. & Roberts, C. (2001). Real-time control/scheduling for multi-purpose batch plants. *Computers & industrial engineering* **41**(2), 211–225. Cited on pages 33 and 100.

[Charnprasitphon (2007)] Charnprasitphon, A. (2007). *Modeling and analysis of the batch production scheduling problem for perishable products with setup times.* Phd thesis, Georgia Institute of Technology. Cited on page 13.

[Chiang & Fox (1990)] Chiang, W. & Fox, M. (1990). Protection against uncertainty in a deterministic schedule. In: *Proceedings 4th International Conference*

on Expert Systems and the Leading Edge in Production and Operations Management. Cited on page 111.

[Church & Uzsoy (1992)] Church, L. K. & Uzsoy, R. (1992). Analysis of periodic and event-driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing* **5**(3), 153–163. Cited on page 109.

[Cohn (2007)] Cohn, A. (2007). Robust Schdule Planning and Recovery: The Role of Operations Research. Cited on page 105.

[Conway et al. (1967)] Conway, R., Maxwell, W. & Miller, L. W. (1967). *Theory of scheduling*. Addison-Wesley. Cited on pages 20 and 21.

[Davenport & Beck (2000)] Davenport, A. & Beck, J. (2000). A survey of techniques for scheduling with uncertainty. *Unpublished manuscript.* , 49. Cited on pages 108 and 117.

[Davenport et al. (2001)] Davenport, A., Gefflot, C. & Beck, J. C. (2001). Slack-based Techniques for Robust Schedules. In: *Proceedings of the Sixth European Conference on Planning (ECP-2001)*. Cited on pages 4, 105, 111, 113, and 115.

[Gabel (2009)] Gabel, T. (2009). *Multi-Agent Reinforcement Learning Approaches for Distributed Job-Shop Scheduling Problems*. Ph.D. thesis, Universität Osnabrück. Cited on pages 5, 38, 61, 63, 67, and 69.

[Gabel & Riedmiller (2007)] Gabel, T. & Riedmiller, M. (2007). On a Successful Application of Multi-Agent Reinforcement Learning to Operations Research Benchmarks. In: *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. Honolulu , USA. Cited on pages 38, 70, 71, and 78.

[Gabel & Riedmiller (2008)] Gabel, T. & Riedmiller, M. (2008). Reinforcement learning for DEC-MDPs with changing action sets and partially ordered dependencies. In: *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*. Estoril, Portugal. Cited on page 59.

[Gao (1996)] Gao, H. (1996). *Building Robust schedules using temporal protection-an empirical study of constraint based scheduling under machine failure uncertainty.* Master thesis, University of Toronto. Cited on pages 104, 105, 107, 111, and 112.

[Garey & Johnson (1979)] Garey, M. R. & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences).* W. H. Freeman. Cited on page 24.

[Garey et al. (1976)] Garey, M. R., Johnson, D. S. & Sethi, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research* **1**(2), 117–129. Cited on pages 12, 26, and 62.

[Garrido et al. (2000)] Garrido, A., Salido, M., Barber, F. & López, M. (2000). Heuristic methods for solving job-shop scheduling problems. In: *ECAI-2000 Workshop on New Results in Planning, Scheduling and Design (PuK2000).* Cited on page 26.

[Gicquel et al. (2012)] Gicquel, C., Hege, L., Minoux, M. & van Canneyt, W. (2012). A discrete time exact solution approach for a complex hybrid flow-shop scheduling problem with limited-wait constraints. *Computers & Operations Research* **39**(3), 629–636. Cited on pages 34, 35, and 140.

[Goldsmith et al. (1996)] Goldsmith, J., Lusena, C. & Mundhenk, M. (1996). The complexity of deterministically observable finite-horizon Markov decision processes. Tech. rep., University of Kentucky. Cited on page 46.

[Gomes (2000)] Gomes, C. P. (2000). Artificial intelligence and operations research: challenges and opportunities in planning and scheduling. *The Knowledge Engineering Review* **15**(1), 1–10. Cited on pages 3, 4, and 107.

[Graham et al. (1979)] Graham, R., Lawler, E., Lenstra, J. & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* **5**, 287–326. Cited on pages 3 and 17.

[Grossmann (2005)] Grossmann, I. (2005). Enterprise-wide optimization: A new frontier in process systems engineering. *AIChE Journal* **51**(7), 1846–1857. Cited on page 12.

[Hartmann (1999)] Hartmann, S. (1999). *Project Scheduling Under Limited Resources: Models, Methods, and Applications*. Springer. Cited on page 121.

[Hasan et al. (2007)] Hasan, S., Sarker, R. & Cornforth, D. (2007). Hybrid genetic algorithm for solving job-shop scheduling problem. In: *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)*. Cited on page 77.

[Herrmann et al. (1993)] Herrmann, J., Lee, C. & Snowdon, J. (1993). A classification of static scheduling problems. *Complexity in Numerical Optimization* , 203–253. Cited on page 17.

[Herroelen & Leus (2005)] Herroelen, W. & Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research* **165**(2), 289–306. Cited on pages 3 and 107.

[Ho et al. (2007)] Ho, N., Tay, J. & Lai, E. (2007). An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research* **179**(2), 316–333. Cited on page 91.

[Jennings et al. (1998)] Jennings, N. R., Sycara, K. & Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-agent Systems* **1**, 7–38. Cited on pages 42 and 54.

[Jensen (2001)] Jensen, M. T. (2001). Improving robustness and flexibility of tardiness and total flow-time job shops using robustness measures. *Applied Soft Computing* **1**(1), 35–52. Cited on page 109.

[Kaddoum (2011)] Kaddoum, E. (2011). *Optimization under Constraints of Distributed Complex Problems using Cooperative Self-Organization*. Phd thesis, Université Paul Sabatier - Toulouse III. Cited on pages 109, 121, and 123.

[Kaddoum et al. (2010)] Kaddoum, E., Martinez, Y., Wauters, T., Verbeeck, K., Nowe, A., De Causmaecker, P., Vanden Berghe, G., Gleizes, M.-P. & George, J.-P. (2010). Adaptive methods for flexible job shop scheduling with due-dates , release-dates and machine perturbations. In: *Workshop on Self-tuning, self-configuring and self-generating search heuristics (Self* 2010)*. Krakow, Poland. Cited on page 8.

[Kaelbling et al. (1996)] Kaelbling, L., Littman, M. & Moore, A. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research* **4**, 237–285. Cited on pages 43 and 45.

[Kaminka (2004)] Kaminka, G. A. (2004). Multi-Agent Systems. Cited on page 54.

[Lambrechts et al. (2010)] Lambrechts, O., Demeulemeester, E. & Herroelen, W. (2010). Time slack-based techniques for robust project scheduling subject to resource uncertainty. *Annals of Operations Research* **186**(1), 443–464. Cited on page 109.

[Le Pape (1991)] Le Pape, C. (1991). Constraint propagation in planning and scheduling. Tech. rep., Standford University. Cited on page 105.

[Leon et al. (1994)] Leon, V. J., Wu, S. D. & Storer, R. H. (1994). Robustness measures and robust scheduling for job shops. *IIE transactions* **26**(5), 32–43. Cited on pages 106 and 109.

[Lining & Chen (2010)] Lining, X. & Chen, Y. (2010). A Knowledge-Based Ant Colony Optimization for Flexible Job Shop Scheduling Problems. *Applied Soft Computing* **10**(3), 888–896. Cited on page 91.

[Littman (1994)] Littman, M. (1994). Markov games as a framework for multi-agent reinforcement learning. In: *Proceedings of the Eleventh International Conference on Machine Learning*. San Francisco, CA: Morgan Kaufmann. Cited on page 56.

[Littman et al. (1995)] Littman, M., Cassandra, A. & Kaelbling, L. (1995). Learning policies for partially observable environments : Scaling up. In: *Reading in Agents* (Huhns, M. N. & Singh, M. P., eds.). Morgan Kaufmann Publishers Inc., pp. 495 – 503. Cited on page 47.

[Lui Cheng (2008)] Lui Cheng, L. (2008). *Robust scheduling in forest operations planning.* Master thesis, Massachusetts Institute of Technology. Cited on page 106.

[Martínez et al. (2011)] Martínez, Y., Nowé, A., Suárez, J. & Bello, R. (2011). A Reinforcement Learning Approach for the Flexible Job Shop Schedul-

ing Problem. *Lecture Notes in Computer Science. Proceedings of the fifth International Conference on Learning and Intelligent Optimization(LION5).* **Vol. 6683**, pp. 253–262. Cited on page 90.

[Martínez et al. (2010)a] Martínez, Y., Vreckem, B., Catteeuw, D. & Nowé, A. (2010a). Application of Learning Automata for Stochastic Online Scheduling. In: *Recent Advances in Optimization and its Applications in Engineering* (Diehl, M., Glineur, F., Jarlebring, E. & Michiels, W., eds.). Springer, pp. 491–498. Cited on pages 93 and 94.

[Martínez et al. (2010)b] Martínez, Y., Wauters, T., De Causmaecker, P., Nowe, A., Verbeeck, K., Bello, R. & Suarez, J. (2010b). Reinforcement Learning Approaches for the Parallel Machines Job Shop Scheduling Problem. In: *Proceedings of the Cuba-Flanders Workshop on Machine Learning and Knowledge Discovery.* Santa Clara, Cuba. Cited on page 81.

[Martínez Jiménez (2008)] Martínez Jiménez, Y. (2008). *A Multi-Agent Learning Approach for the Job Shop Scheduling Problem.* Master thesis, Vrije Universiteit Brussel. Cited on page 75.

[Megow et al. (2006)] Megow, N., Uetz, M. & Vredeveld, T. (2006). Models and Algorithms for Stochastic Online Scheduling. *Mathematics of Operations Research* **31**(3), 513–525. Cited on page 93.

[Méndez et al. (2006)] Méndez, C., Cerdá, J., Grossmann, I., Harjunkoski, I. & Fahl, M. (2006). State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers & chemical engineering* **30**(6-7), 913–946. Cited on page 14.

[Metaxiotis et al. (2005)] Metaxiotis, K., Askounis, D. & Psarras, J. (2005). Expert systems technology in production planning and scheduling. In: *Intelligent Knowledge-Based Systems* (Leondes, C. T., ed.), vol. 3. Springer US, pp. 55–75. Cited on page 12.

[Moriarty et al. (1999)] Moriarty, D. E., Schultz, A. C. & Grefenstette, J. J. (1999). Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research* **11**, 241–276. Cited on pages 43 and 45.

[Narendra & Thathachar (1974)] Narendra, K. S. & Thathachar, M. A. L. (1974). Learning automata - A survey. *IEEE Transactions on Systems, Man, and Cybernetics,* , 323–334. Cited on page 51.

[Nhu Binh HO (2005)] Nhu Binh HO, J. C. T. (2005). Evolving Dispatching Rules for solving the Flexible Job-Shop Problem. *The 2005 IEEE Congress on Evolutionary Computation.* **3**, 2848–2855. Cited on page 37.

[Nowé et al. (2012)] Nowé, A., Vrancx, P. & De Hauwere, Y.-M. (2012). Game Theory and Multi-agent Reinforcement Learning. In: *Reinforcement Learning: State-of-the-Art* (Wiering, M. & Van Otterlo, M., eds.). Springer, pp. 441–470. Cited on page 56.

[Peeters (2008)] Peeters, M. (2008). *Solving Multi-Agent Sequential Decision Problems Using Learning Automata.* Phd thesis, Vrije Universiteit Brussel. Cited on pages 33 and 94.

[Pezzella et al. (2008)] Pezzella, F., Morganti, G. & Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research* **35**(10), 3202–3212. Cited on pages 91 and 120.

[Pinedo (2008)] Pinedo, M. (2008). *Scheduling: theory, algorithms, and systems, Third Edition.* Springer Verlag. Cited on pages 2, 11, 13, 20, and 33.

[Policella (2005)] Policella, N. (2005). *Scheduling with Uncertainty A Proactive Approach using Partial Order Schedules.* Phd thesis, Università degli Studi di Roma "La Sapienza". Cited on page 104.

[Puris et al. (2007)] Puris, A., Bello, R., Trujillo, Y., Nowe, A. & Martinez, Y. (2007). Two-stage ACO to solve the job shop scheduling problem. *Lecture Notes in Computer Science. Book "Progress in Pattern Recognition, Image Analysys and Applications"* **4756**, 447–456. Cited on page 77.

[Puterman (1994)] Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* New York, NY, USA: John Wiley & Sons, Inc. Cited on pages 45 and 49.

[Rodrigues Gomes & Kowalczyk (2009)] Rodrigues Gomes, E. & Kowalczyk, R. (2009). Dynamic Analysis of Multiagent Q-learning with Exploration e-

greedy. In: *Proceedings of the 26th International Conference on Machine Learning*. Cited on page 69.

[Rossi & Boschi (2009)] Rossi, A. & Boschi, E. (2009). A hybrid heuristic to solve the parallel machines job-shop scheduling problem. *Advances in Engineering Software* **40**(2), 118–127. Cited on pages 28, 80, and 81.

[Ruiz et al. (2008)] Ruiz, R., Şerifoğlu, F. S. & Urlings, T. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research* **35**(4), 1151–1175. Cited on page 14.

[Russell & Norvig (2003)] Russell, S. & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall. Cited on page 41.

[Sabuncuoglu & Goren (2009)] Sabuncuoglu, I. & Goren, S. (2009). Hedging production schedules against uncertainty in manufacturing environment with a review of robustness and stability research. *International Journal of Computer Integrated Manufacturing* **22**(2), 138–157. Cited on pages 106 and 109.

[Shapley (1953)] Shapley, L. S. (1953). Stochastic Games. *Proceedings of the National Academy of Sciences of* **39**, 1095–1100. Cited on page 56.

[Shen (2002)] Shen, W. (2002). Distributed Manufacturing Scheduling Using Intelligent Agents. *IEEE Intelligent Systems* **17**, 88–94. Cited on page 12.

[Simon & Lea (1973)] Simon, H. & Lea, G. (1973). Problem solving and rule induction: A unified view. *Knowledge and Cognition* , 105 – 128. Cited on page 41.

[Soyster (1973)] Soyster, A. L. (1973). Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research* **21**(5), 1154–1157. Cited on page 106.

[Sutton & Barto (1998)] Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press. Cited on pages 5, 43, 48, 50, 52, and 53.

[Sycara (1998)] Sycara, K. P. (1998). Multiagent Systems. *AI Magazine* , 79–92. Cited on pages 54 and 55.

[Tsetlin (1962)] Tsetlin, M. (1962). On the behavior of finite automata in random media. *Automation and Remote Control* **22**, 1210–1219. Cited on page 51.

[Tsitsiklis (1994)] Tsitsiklis, J. (1994). Asynchronous stochastic approximation and Q-learning. *Machine Learning* **16**, 185–202. Cited on page 50.

[Urlings (2010)] Urlings, T. (2010). *Heuristics and metaheuristics for heavily constrained hybrid flowshop problems*. Phd thesis, Universidad Politécnica de Valencia. Cited on pages 6 and 149.

[Van Peteghem & Vanhoucke (2008)] Van Peteghem, V. & Vanhoucke, M. (2008). A Genetic Algorithm for the Multi-Mode Resource-Constrained Project Scheduling Problem. Tech. Rep. January, Ghent University. Cited on pages 85, 87, and 120.

[Ventresca & Ombuki (2004)] Ventresca, M. & Ombuki, B. (2004). Ant colony optimization for job shop scheduling problem. In: *Proceeding of 8th IASTED International Conference On Artificial Intelligence and Soft Computing (ASC 2004)*, February. Cited on page 78.

[Vieira et al. (2003)] Vieira, G. E., Herrmann, J. W. & Lin, E. (2003). Rescheduling Manufacturing Systems: A Framework of Strategies, Policies, and Methods. *Journal of Scheduling* **6**(1), 39–62. Cited on page 109.

[Wang & Shen (2007)] Wang, L. & Shen, W. (eds.) (2007). *Process Planning and Scheduling for Distributed Manufacturing*. Springer. Cited on page 2.

[Wauters et al. (2010)] Wauters, T., Martinez, Y., De Causmaecker, P., Nowe, A. & Verbeeck, K. (2010). Reinforcement Learning approaches for the Parallel Machines Job Shop Scheduling Problem. In: *Proceedings of ITEC2010 - International conference on interdisciplinary research on technology, education and communication*. Kortrijk, Belgium. Cited on pages 81, 83, and 109.

[Winston (2003)] Winston, W. L. (2003). *Operations Research: Applications and Algorithms*. Duxbury Pr, 4th ed. Cited on page 120.

[Wooldridge (1999)] Wooldridge, M. (1999). Intelligent Agents. In: *Multiagent Systems* (Weiss, G., ed.). The MIT Press, p. 51. Cited on page 42.

[Wooldridge & Jennings (1995)] Wooldridge, M. & Jennings, N. (1995). Intelligent agents: theory and practice. *The Knowledge Engineering Review* **10**(2), 115–152. Cited on page 42.

[Wu et al. (1999)] Wu, S. D., Byeon, E.-S. & Storer, R. H. (1999). A Graph-Theoretic Decomposition of the Job Shop Scheduling Problem to Achieve Scheduling Robustness. *Operations Research* **47**(1), 113–124. Cited on page 109.

[Wu et al. (2005)] Wu, T., Ye, N. & Zhang, D. (2005). Comparison of distributed methods for resource allocation. *International Journal of Production Research* **43**(3), 515–536. Cited on pages 63 and 67.

[Zhang (1996)] Zhang, W. (1996). *Reinforcement Learning for Job Shop Scheduling*. Ph.D. thesis, Oregon State University. Cited on pages 25, 37, 38, and 44.

[Zhang & Dietterich (1995)] Zhang, W. & Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann. Cited on page 38.

Index