Match-up Strategies and Fuzzy Robust Scheduling for a Complex Dynamic Real World Job Shop Scheduling Problem

Patrick Moratori, BSc, MSc

Thesis submitted to The University of Nottingham
for the Degree of Doctor of Philosophy

School of Computer Science and Information Technology
The University of Nottingham
Nottingham, United Kingdom

July 2013

Abstract

This thesis investigate a complex real world job shop scheduling / rescheduling problem, in which the presence of uncertainties and the occurrence of disruptions are tackled to produce efficient and reliable solutions. New orders arrive every day in the shop floor and they have to be integrated in the existent schedule. Match-up algorithms are introduced to collect the idle time on machines and accommodate these newly arriving orders. Their aim is to obtain new schedules with good performance which are at the same time highly stable, meaning that they resemble as closely as possible the initial schedule. Subsequently, a novel approach that combines these algorithms with a fuzzy robust scheduling system is proposed. The goal is to associate an effective repairing mechanism with the production of initial robust schedules that are able to facilitate the accommodation of future disruptions. Statistical analyses reveal that match-up algorithms are effective repairing strategies for managing complex disruptions, in which high quality stable schedules are delivered. Moreover, their combination with fuzzy robust scheduling has a positive effect on responding to these disruptions leading to even more reliable solutions in a real world dynamic and uncertain shop floor.

Part of the work described in this thesis has been awarded for its contributions and also been research topics of the following publications:

Journal Articles and Lecture Notes

P. Moratori, S. Petrovic, and J. Vázquez-Rodríguez. Match-up strategies for job shop rescheduling. In N. Nguyen, L. Borzemski, A. Grzech, and M. Ali, editors, *New Frontiers in Applied Artificial Intelligence, Lecture Notes in Computer Science*, volume 5027, pages 119–128. Springer-Verlag, 2008.

P. Moratori, S. Petrovic, and J. Vázquez-Rodríguez. Integrating rush orders into existent schedules for a complex job shop. *Applied Intelligence*, 32(1):205–215, 2010.

P. Moratori, S. Petrovic, and J. Vázquez-Rodríguez. Match-up approaches for a dynamic scheduling problem. *International Journal of Production Research*, 50(1):261–276, 2012.

P. Moratori, S. Petrovic, and J. Vázquez-Rodríguez. Robust fuzzy rescheduling for a complex real world job shop problem. Journal paper currently in preparation.

Conference Papers

P. Moratori, S. Petrovic, and D. Petrovic. A match-up algorithm for job shop rescheduling. In *Annual Operational Research Conference 49 (OR49)*, Edinburgh, UK, 4-6 September 2007.

P. Moratori, S. Petrovic, and J. Vázquez-Rodríguez. Hibridisation of fuzzy robust scheduling with match-up approaches. In C. Antunes, D. Insua, and L. Dias, editors, *Proceedings of the 25th Mini-euro Conference Uncertainty and Robustness in Planning and Decision Making*, pages 1–7, Coimbra, Portugal, April 15-17 2010. ISBN 978-989-95055-3-7.

☞ **Award**

The lecture notes paper “Match-up strategies for job shop rescheduling” was awarded as the best paper at the 21st International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems, Wroclaw, Poland, June 18-20, 2008.

Acknowledgements

The support provided by the University of Nottingham, including the School of Computer Science, International Office, Rutland Hall and Language Centre. Additionally, the Engineering and Physics Science Research Council through grant GR/R95319/01 and the research partnership established with Sherwood Press Ltd - Nottingham, UK.

I would like to express my gratitude to both Federal University of Rio de Janeiro and Federal University of Juiz de Fora for supporting my academic progression.

This work is dedicated to my beloved parents, all my family and my dear friends.

Glossary

ANOVA	Analysis of variance
BB	Branch and bound algorithm
EDD	Earliest due-date first
E	Insertion of the new job at the end of the schedule
FCFS	First come first served
GA	Genetic algorithm
HP	Highest priority first
LPT	Longest processing time first
LRT	Longest remaining processing time first
Makespan (C_{max})	Completion of the latest operation on the shop floor
NP-hard problem	Problem that cannot be solved in a polynomial time
Rescheduling	Schedule rearrangement
RS	Right shift rescheduling
SFT	Same family together
SG	Satisfaction grade
SPT	Shortest processing time first
T	Total rescheduling

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

High productivity and low production costs are essential factors to describe successful businesses. Research on production scheduling has been providing many approaches to achieve this goal, in which optimisation models are proposed to allocate resources overtime.

The traditional scheduling models only consider static and deterministic future conditions, in which a finite set of jobs with deterministic processing times have to be assigned to a finite set of machines subject to certain constraints with the aim of minimising a certain cost function. However, in the real world, the presence of uncertainties and the frequent occurrence of disruptions inevitably require rescheduling of these allocations, in which initial solutions have to be coupled with reliable and effective repair mechanisms that are able to adjust schedules to reasonably respond to circumstances that often arise in the shop floor, such as the arrival of new jobs, machine breakdowns, rework of jobs, due dates changing, among others. Consequently, research on rescheduling has been attracting attention, in which new optimisation models and several techniques are proposed, analysed and employed to manage dynamic and uncertain environments. These problems are highlighted as dynamic because continuous rearrangements of current schedules are required to restore their feasibility, controlling the presence of occurring disruptions [86].

1.2 Overview of the Problem

A real world scheduling / rescheduling problem of a printing company, Sherwood Press - Nottingham, UK is considered in this thesis, which is modelled as a job shop problem with parallel machines, machine eligibility and sequence dependent setup times. The problem is dynamic since new printing orders arrive every day in the shop floor, which requires the generation of a reliable initial schedule and, more importantly, a *rescheduling* process to accommodate these newly arriving jobs. This problem is also defined as complex due to its nature of being a NP-hard problem [91]. This problem is tackled as a generalisation of possible disruptions because new orders requirements are able to compromise not only one, but many resources present in a shop floor. Consequently, the optimisation models introduced in this thesis are suitable to be replicated to other similar contexts, such as personnel scheduling and university timetabling.

1.3 Research Context

The research work present in this thesis is a build up on the investigation of the static scheduling problem presented by Sherwood Press - Nottingham. The research group has been tackling possible approaches to handle uncertainties that are present on this real world production shop floor, such as variations on processing times, due-dates and release times [32, 84, 85, 89]. The main aim of the group is to produce high quality performing schedules even in uncertain environments.

A genetic algorithm was introduced [32], in which a multiple criteria fitness function is employed to deliver schedules with high *Performance*, i.e. minimising simultaneously the average weighted tardiness of jobs, the number of tardy jobs, the total setup time, the total idle time of machines, and the total flow time. Uncertainties were managed using fuzzy sets to represent the problem parameters and, consequently, fuzzy logic to handle the required inferences, i.e. when a job has to be considered tardy or not.

Techniques such as load balancing and lot-sizing are also applied, due to their effectiveness on delivering good schedules [84,85]. The presence of parallel printing machines allow jobs to be processed more quickly since the load balancing algorithm tries to evenly distribute the processing of required jobs on them. On the other hand, a lot-sizing algorithm splits jobs into smaller lots aiming of attend possible customer demands, i.e. a smaller lot is delivered first in order to attend the customer expectations and the remaining part is subsequently produced in a more convenient time for the company.

The encouraging *Performance* results achieved for this static problem highlight the good combination of the proposed genetic algorithm, fuzzy concepts and the use of load balancing and lot-sizing [89]. These techniques are employed in this thesis to produce initial schedules and to reallocate operations on rescheduling. Further investigation on setting the fitness function is discussed in details in chapter 4 in order to appropriately address the rescheduling issues.

1.4 Aims and Scope

The aim of this thesis is to investigate optimisation models and techniques to produce reliable schedules under dynamic and uncertain environments. Solutions with high levels of *Performance* are expected, which are measured according to a certain cost function, while the *Stability* is preserved by introducing as fewer changes as possible to the current state of the shop floor. Firstly, this work argues that match-up algorithms are effective repair methods to deliver high quality stable schedules when disruptions affect multiple resources of a complex real world problem presented by a printing company in Nottingham, UK (hypothesis **H1**). Secondly, a new approach is proposed to combine match-up algorithms with initial robust schedules, in order to investigate their interaction on creating reliable high quality stable schedules (hypothesis **H2**). Note that the term robust defines schedules that aim to absorb occurring disruptions [35, 59].

1.5 Methodology

Two types of analyses are carried out to accomplish the described goals. Chapters 2-3 provide an analysis of the literature, in which dynamic scheduling and fuzzy concepts are investigated on managing the presence of uncertainties. Subsequently, chapters 4-6 describe data analysis of experiments that are done to validate the proposed hypotheses: effectiveness of mach-up algorithms for rescheduling a complex real world problem

(**H1**) and the positive effects when combining them with a fuzzy scheduling to produce reliable solutions (**H2**). All results are statistically validated, in which an analysis of variance (ANOVA) reveals the significance of the investigated problem parameters and their interactions, a pairwise comparison test using Bonferroni's correction indicates which approaches deliver superior results, and average values highlight the overall behaviour of each analysed strategy.

1.6 Structure of the Thesis

The remaining of this thesis is organised as in Figure 1.1. Chapters 2 and 3 presents the background and related work in which the investigated problem is situated. A rescheduling taxonomy is presented and match-up algorithms are highlighted as reasonable repair methods in Chapter 2. Additionally, chapter 3 identifies the application of fuzzy logic concepts as a suitable approach to help modelling possible uncertainties present in scheduling / rescheduling problems.

Both hypotheses are validated in the contribution chapters 4-6. Firstly, the investigated scheduling / rescheduling problem is discussed in detail and different match-up strategies are introduced to control this complex real world dynamic problem, as presented in chapter 4. A typical disruption that affect multiple available resources is tackled, in which new rush orders arrive everyday in the shop floor. Note that these orders define a set of jobs that have to be processed as early as possible. Statistical analyses reveal that the proposed strategies are effective repair methods to deliver high quality stable schedules (hypothesis **H1**). Subsequently, chapter 5 investigate orders with different levels of urgency in which the flexibility of the proposed strategies are verified under different scenarios. This set of jobs is identified as normal orders and the main goal is to generalise possible occurring disruptions in order to emphasise the validity of **H1**. Chapter 6 discusses a novel approach that combines match-up rescheduling algorithms with robust fuzzy scheduling. This fuzzy scheduling system inserts idle times on machines based on historical data. The main aim is to produce initial robust schedules that are able to facilitate the accommodation of the newly arriving jobs. Statistical analyses confirm that the proposed combination has a positive effect on responding to disruptions leading to reliable high quality stable

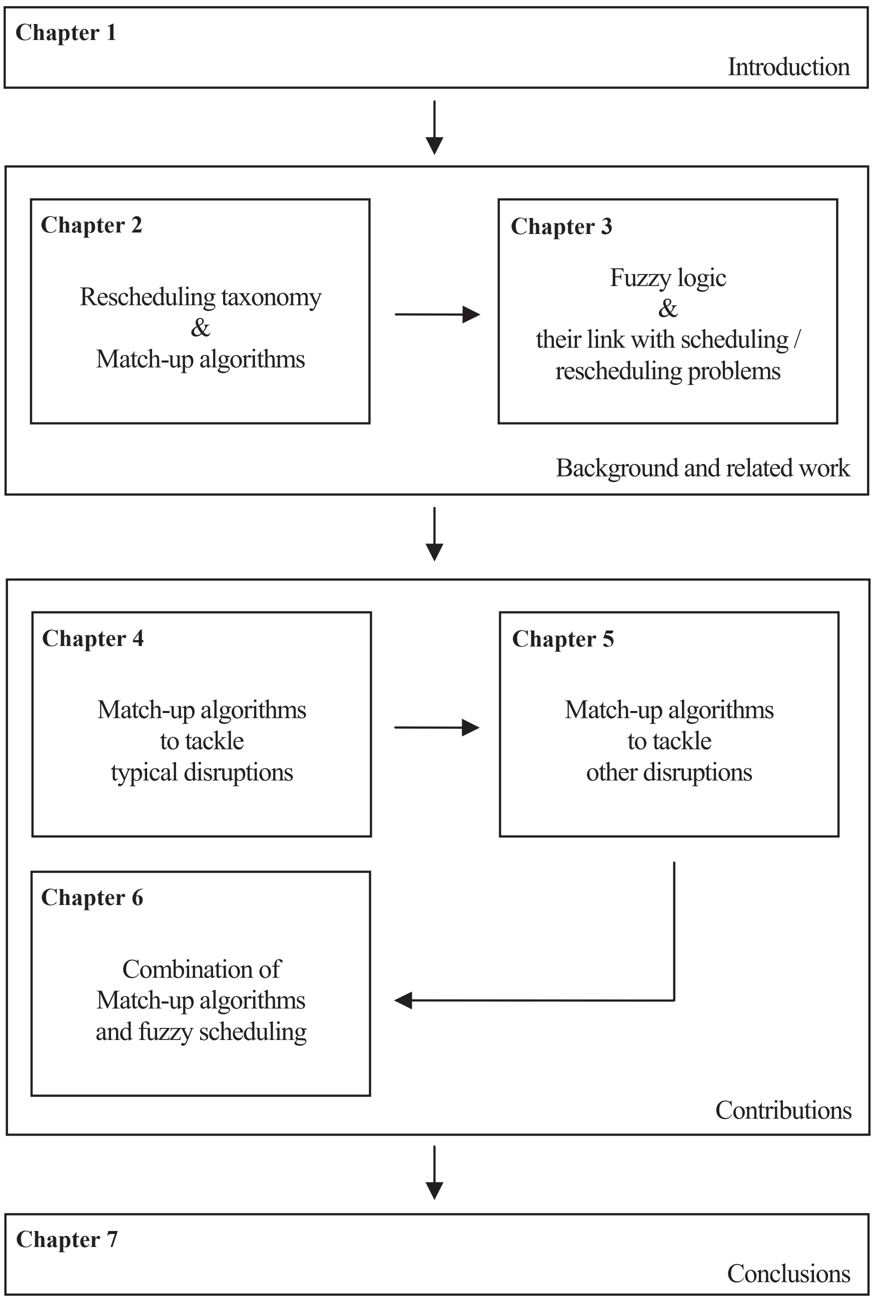


Figure 1.1: Structure of the Thesis.

schedules (hypothesis **H2**).

Finally, chapter 7 discusses and summarises the conclusions of this thesis and its possible future work.

1.7 Contributions

The main contribution of this thesis is the introduction of match-up strategies to manage uncertainties present in a complex dynamic real world job shop problem. Moreover, a novel approach that combines these repairing strategies with initial robust schedules is also discussed and validated.

The proposed approaches are described in the following chapters:

- Chapter 4 describes developed match-up algorithms for a real world problem presented by a printing company in Nottingham, UK, in which a typical disruption affects multiple resources available in the shop floor, as in [69, 70, 73];
- Chapter 5 does a further investigation of match-up algorithms, in which improvements are applied to the genetic algorithm responsible for optimising the job allocations on machines. Additionally, a more general type of disruption is analysed in order to check the flexibility of the proposed strategies under different scenarios, as in [74];
- Finally, chapter 6 introduces a new approach to combine the repairing mechanism provided by match-up strategies with the generation of initial robust schedules. A fuzzy control system is designed to produce these schedules, in which historical data from the investigated company provides information about good practices. The aim is to facilitate the accommodation of future disruptions by inserting idle times on machines and, consequently, produce more reliable and effective solutions, as in [71, 72];

The flexibility of the proposed approaches on managing disruptions in a complex real world dynamic scenario highlights the suitability of replicating them to other similar contexts.

Shaded cells in Table 1.1 identify which approaches are investigated in this thesis. The aim is to highlight the previously mentioned contributions within a rescheduling taxonomy [8, 42, 79, 114].

The investigated scheduling / rescheduling problem has a high variability on arrival of new jobs, which sets it as a dynamic environment. These arriving jobs often affect multi-

Table 1.1: Investigated approaches within a rescheduling taxonomy

Environment

Static (finite set of jobs)		Dynamic (infinite set of jobs)		
Deterministic (all information is given)	Stochastic (some information is uncertain)	Cyclic production (no arrival variability)	Medium variability (some arrival variability)	High variability (high arrival variability)

Approach

Reactive	Predictive (robust)	Predictive-reactive

Frequency

Periodic	Continuous	Event-driven	Hybrid

Method

Schedule generation		Schedule repair		
Nominal	Robust	Right / left-shift	Complete	Partial

ple resources and they are classified as important disruptions. As a result, a rescheduling process has to be started whenever a new job is required to be processed (event-driven frequency).

Firstly, initial schedules are generated only optimising the current state of the shop floor (nominal generation), as in chapters 4-5. Subsequently, a newly proposed approach generates schedules that aim to predict future disturbances (robust generation), as in chapter 6. Match-up strategies are used as a repair method, modifying only required parts of the current allocations (partial repair). This configurations set two explicit scheduling / rescheduling approaches: (1) predictive-reactive, which creates a nominal schedule and react when a disruption occurs and, (2) robust, which creates a robust schedule that helps to absorb occurring disruptions during the repairing process.

The following chapter provides a more detailed discussion about these approaches and all the remaining ones presented in Table 1.1, in which their strengths and limitations are extensively discussed.

Chapter 2

Survey of Dynamic Scheduling - Rescheduling

2.1 Introduction

This chapter presents a literature review of rescheduling algorithms, in which a taxonomy of possible environments, approaches, frequency and methods is described and discussed. The aim is to provide a guideline to understand related terminologies, applied strategies and their limitations. This taxonomy is subsequently linked with match-up algorithms and their possible combination with robust schedules, which are the main research topics investigated in the following chapters of this thesis.

In a competitive world, high productivity and low production costs are very important factors to guarantee successful businesses. Research on production scheduling has been providing many approaches to achieve these factors, in which optimisation models are proposed to allocate resources to jobs (tasks) over time. Literature on these models has been mainly focused on the problem of generating efficient schedules under a given static scenario. Typically, a fixed number of jobs with deterministic processing times have to be assigned to a given number of machines minimising a certain cost function. However, when the possibility of disruptions, and uncertainty in the broad sense, is taken into account, such deterministic models, and their corresponding solution approaches, have to be coupled with repair mechanisms that adjust the initial schedule to respond to the new circumstances that may arise from an unexpected event. The types of disruptions that may occur can be either (1) related to jobs, such as changes to production orders, including the insertion [13, 17, 30, 93] and removal of jobs [89], rework [103], changes to processing times and due dates [24, 52]; or (2) related to the shop floor, such as changes to manufacturing resources, including substitution or breakdowns of machines [4, 67, 90, 115], sickness of workers [106], tool unavailability [2, 12], delay or shortage on material supply [53, 97], etc. For all these cases, a rescheduling of the previously allocated jobs is required in order to restore the feasibility of the schedule and keep its optimal performance results. More details about each step of this process are described in the remaining sections of this chapter.

In order to understand the terminology present in the literature, some terms commonly used by different researchers are described below:

- *Rescheduling point*: when a schedule is repaired;
- *Rescheduling period*: time between two consecutive rescheduling points;
- *Rescheduling frequency*: how many times a rescheduling process is required;
- *Rescheduling horizon*: selected allocations within a time horizon that must be rescheduled;
- *Scheduling nervousness*: associated with the repairing times required during the schedule execution;
- *Scheduling stability*: inverse of scheduling nervousness;
- *Scheduling robustness*: how much the repairing process does affect the schedule performance.

Once values are given to these terms, it is possible to have an overall idea about how a rescheduling problem has been tackled. Moreover, the impact of this process is evaluated using the following metrics to check the quality of the generated schedules:

- *Performance*: metric commonly used in scheduling and rescheduling problems, in which tardiness, lateness, makespan, number of tardy jobs, setup times, idle times and flow times of schedules are evaluated;

- *Stability*: metric only used in rescheduling problems, which measures the difference between the initially planned schedule and the executed one. This metric mostly check changes on start/end times of operations (time deviation), sequence of operations on machines (sequence deviation) and operations switching between parallel resources (machine deviation). Note that this measure is not applicable to check initial schedules due to the absence of changes;
- *Efficiency*: measures how quickly a disruption is managed;
- *Cost*: measures the computational burden, setup and transportation costs involved during the process.

These metrics provide crucial information because they allow the selection of the most appropriated rescheduling method to be applied to a specific problem. Note that more than one metric can be used to evaluate the quality of a generated schedule. In practice, *Performance* and *Stability* are the most commonly used metrics because they can give an overall picture of the production process [1, 21, 90, 93, 116].

The remaining of this chapter is organised as follows. Section 2.2 introduces a taxonomy for rescheduling algorithms, in which possible environments, approaches, frequency and methods are described. Section 2.3 discuss match-up algorithms, presenting their current applications, limitations, possibilities and their possible combination with robust schedules. Finally, sections 2.4 and 2.5 conclude this chapter.

2.2 Rescheduling Taxonomy

Research on rescheduling has been exploring the potential of optimisation models applied to dynamic contexts, in which expected and/or unexpected disruptions have to be managed in order to guarantee the quality of the planned schedules. Several authors have been investigating this rescheduling process applied to different contexts such as single machine problems [24, 35, 36, 58], flow shop [3, 22, 105, 120], job shop problems [13, 28, 33, 100] and the use of parallel machines [12, 18, 60, 103]. A taxonomy for possible rescheduling approaches is shown in Table 2.1, based on features present in these studies combined with literature reviews presented by [8, 42, 79, 114].

Table 2.1: Rescheduling taxonomy

Environment				
Static (finite set of jobs)		Dynamic (infinite set of jobs)		
Deterministic (all information is given)	Stochastic (some information is uncertain)	Cyclic production (no arrival variability)	Medium variability (some arrival variability)	High variability (high arrival variability)

Approach		
Reactive	Predictive (robust)	Predictive-reactive

Frequency			
Periodic	Continuous	Event-driven	Hybrid

Method				
Schedule generation		Schedule repair		
Nominal	Robust	Right / left-shift	Complete	Partial

A scheduling / rescheduling problem is defined based on the following aspects: (1) environment, which is related to the number of jobs that have to be scheduled; (2) approach, to set how jobs are allocated; (3) frequency, which defines when to reschedule; and (4) method, which describes how to generate and update the schedule. The following subsections provide more detailed information for each of these aspects.

2.2.1 Rescheduling Environments

The rescheduling process may happen either in a *static environment*, in which the number of jobs is finite and known in advance, or in a *dynamic environment*, in which jobs arrive in the shop floor continuously. A job is used as reference to represent possible disruptions because its requirements are able to compromise not only one, but many resources present in a shop floor.

In the static case, one can differentiate between *static-deterministic environments*, where there is no uncertainty in problem data [91], or *static-stochastic environments*, in which processing times, due dates and other problem data are subject to minor changes [86]. Note that there is no rescheduling in static environments and the presence of uncertainty in problem data does not require changes on planned schedules. These environments define the category of classical scheduling problems.

In the dynamic case, the presence of unpredictability is mostly concerned with time of the arrival of jobs. These environments reflect a better representation of real world problems and they are classified as *dynamic cyclic environments*, in which jobs arrive in the shop floor in regular and perfectly predictable intervals of time [11, 14, 15]; *dynamic medium variability environments* [21, 26, 47] which also have somehow predictable job arrival patterns and consider some level of uncertainty in other problem parameters; and *dynamic high variability environments* [27, 31] in which arrivals of jobs is highly unpredictable and certain events, such as machine breakdowns and tools unavailability are also taken into account.

Table 2.2 summarises relevant references for these environments considering different types of disruption and proposed optimisation methods. Both GA and BB algorithms are highlighted as commonly used techniques because they are able to approximate optimal solutions for NP-hard problems within reasonable computation time [92, 111]. Match-up algorithms are also identified as common solving techniques and a detailed discussion about their application is presented in the following section of this chapter. Other promising method is the use of hyper-heuristics in dynamic environments, in which proposed search techniques are able to select, combine, generate and adapt several simpler heuristics to solve scheduling / rescheduling problems [16,37]. Note that no references are given to static determinist problems because they belong to the group of classical scheduling problems, in which disruptions are not taken into account.

2.2.2 Rescheduling Approaches

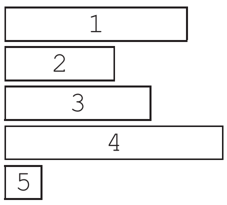
Approaches which deal with disturbances in the production shop can be classified into three main groups: (1) *reactive* scheduling; (2) *predictive* scheduling; and (3) *predictive-reactive* scheduling algorithms.

The main feature of reactive scheduling is that no initial schedules are generated and real time control actions are applied to allocate the available resources over time. This approach is also known as “online scheduling” given that it sets a passive method that react to unforseen events as they occur. Dispatching rules [27, 33, 80], pull mechanisms such as Kanban cards [44] and idling policies [19] are mostly used to prioritise jobs that

Table 2.2: Rescheduling relevant references

Environment	Disruption	Technique	Reference
Static			
Stochastic information			
	Machine breakdown	Genetic algorithm	[59]
		Match-up algorithm	[3]
	Uncertain processing times, release and due-dates	Branch and bound algorithm, fuzzy variables	[105]
		Genetic algorithm, fuzzy variables	[32, 84, 85, 87, 88]
Dynamic			
Cyclic production			
	Machine breakdown	Match-up algorithm	[11, 14, 15]
Medium variability			
	Machine breakdown	Mathematical programming, expert system	[26, 35, 36]
		Branch and bound algorithm	[67]
		Fuzzy variables	[21]
		Genetic algorithm	[47]
	New jobs	Genetic algorithm	[93]
High variability			
	Machine breakdown	Match-up algorithm	[12, 116]
		Branch and bound algorithm	[61]
		Genetic algorithm	[90]
		Heuristics	[1]
	Uncertain processing times, release and due-dates	Simulation	[24]
	New jobs	Heuristics	[31]
		Genetic algorithm	[13]
	Machine breakdown, new jobs and order cancelation	Simulation	[33]
	Machine breakdown and quality control	Knowledge-based system	[103, 104]
		Expert system	[60]
	Order changes	Match-up algorithm	[106]
	Resource availability	Tabu search	[28]
	Examination Timetabling	Hyper-heuristics	[16]

need to be processed next. These control actions can be combined with machine learning techniques, which are useful to select the most appropriate response to a disruption based on decision trees [6]. Alternatively, artificial neural networks can be used to predict an adequate control action [7] and genetic algorithms can be applied to choose a population of suitable actions [20]. A low computational burden is usually required for this approach. However, a *Performance* value is difficult to predict since no schedules are generated. Figure 2.1 shows an example of control actions using dispatching rules. A single machine problem with 5 jobs to be allocated is illustrated in Figure 2.1 (a), in which the rules shortest processing time first (SPT), longest processing time first (LPT), first come first served (FCFS) and earliest due-date first (EDD) are applied. The application of each rule prioritises jobs that will be processed next. Note that these priorities have to be changed if a new job arises in the shop floor, i.e. job 6 with release time $r_j = 0$, due-date $d_j = 12$ and processing time $p_j = 2$. All jobs are reconsidered because, hypothetically, the current time is 0 and they all have been already released at this time. The updates are illustrated in Figure 2.1 (b). The other described control actions follow the same pattern used here, in which a rule gives directions about how to prioritise the allocation of jobs on the available resources over time.



r_j	d_j	p_j
0	21	5
0	15	3
0	5	4
0	10	6
0	6	1

SPT	5	2	3	1	4
LPT	4	1	3	2	5
FCFS	1	2	3	4	5
EDD	3	5	4	2	1

(a)

r_j	d_j	p_j
0	12	2

SPT	5	6	2	3	1	4
LPT	4	1	3	2	6	5
FCFS	1	2	3	4	5	6
EDD	3	5	6	4	2	1

(b)

Figure 2.1: Example of prioritising jobs using the following dispatching rules: shortest processing time first (SPT), longest processing time first (LPT), first come first served (FCFS) and earliest due-date first (EDD). Note that each job has a release time r_j, due-date d_j and processing time p_j. Initially, the priorities are set between (a) 5 jobs and, subsequently, between (b) 6 jobs.

Predictive approaches, also known as "robust scheduling", generate schedules with a hope that it would be able to absorb any disruptions without compromising the schedule *Performance*. Idle times are allocated on machines in such a way that future disturbances can be accommodated. The main idea is to preserve the initially produced schedule in order to avoid extra production costs. Genetic algorithms [47, 59], fuzzy systems [10, 21, 26, 28, 35, 36], branch and bound algorithms and simulated annealing [61, 62] have been used as components of predictive scheduling systems. A robust schedule is presented in Figure 2.2, in which idle times were inserted on available machines. For instance, a new job 8 requires processing on machines M1 and M2, as in Figure 2.2 (a) and it can be inserted in the current schedule without changing the current allocations, as in Figure 2.2 (b) and (c), respectively. As a matter of simplifying this example, the problem parameters release time and due-dates are not considered.

Pure predictive and pure reactive approaches have their limitations. A pure predictive approach can only absorb a limited number of disruptions, these of a relatively low mag-

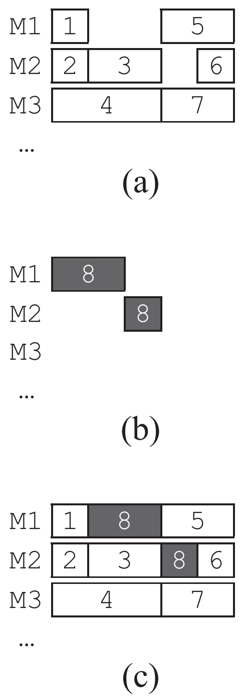


Figure 2.2: Example of (a) a robust schedule, with (b) a new job requirements and its (c) resultant schedule.

nitude, before requiring a complete reallocation of the jobs in the shop floor. A purely reactive approach is concerned with keeping neither the schedule *Performance* nor the *Stability* and it often generates highly suboptimal schedules. Predictive-reactive approaches are an alternative to overcome these drawbacks [8, 114]. In predictive-reactive rescheduling, an initial high quality schedule is constructed, and when a high impact disruption occurs, it is modified using an appropriate repair method. This process is subdivided into three phases: (1) planning - to delimitate the initial schedule; (2) controlling - to check the production process; and (3) reacting - to set a response to unexpected events. Note that all changes are done during the execution of the schedule. Genetic algorithms [93], expert systems [106], simulation models [60], knowledge based models [103,104], heuristics [31] and match-up strategies [3, 12] have been used in predictive-reactive approaches. They constitute the strategy that has been mostly used in practice for rescheduling real world dynamic manufacturing problems [13]. Figure 2.3 (a) shows an initial schedule produced to allocate 7 jobs on machines M1-M3. A new job 8 arises and this initial solution has to be changed to accommodate the new requirements and keep the feasibility of the schedule, as in Figure 2.3 (b). Two different alternatives are shown in Figure 2.3 (c)

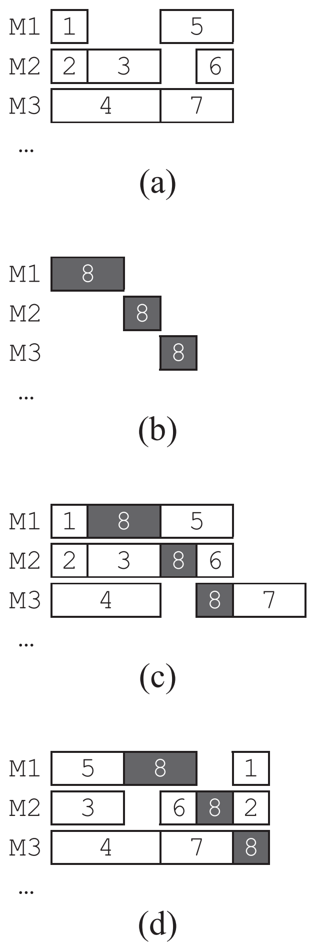


Figure 2.3: Example of (a) a predictive-reactive schedule, with (b) a new job requirements, and its (c) resultant schedule with minor and (d) major changes on the initial allocations.

and (d) to insert this new job, in which minor and major changes are done in the current schedule, respectively. The most appropriated method is chosen based on requirements of each problem. Further details about repair methods are discussed in subsection 2.2.4.

2.2.3 Rescheduling Frequency

Rescheduling approaches are also classified according to the frequency with which rescheduling occurs. They are subdivided into *periodic*, *continuous*, *event-driven* and *hybrid* approaches.

In periodic approaches, the rescheduling follows predefined time intervals [90, 93]. This means that the current schedule is kept unchanged, even if a recent disruption has occurred, until the next predefined rescheduling point is reached. In periodic approaches, schedules are changed relatively infrequently. Consequently, the *Stability* of the schedule is maintained, which makes these approaches popular in the industry [8]. The drawbacks, however, are that the *Performance* of the schedule may deteriorate when the rescheduling frequency is too low and the fact that it may be difficult to define appropriate rescheduling points. Near optimal *Performance* can be obtained when one disruption occurs on a regular basis, i.e. a certain group of jobs that has to be processed every month in a shop floor.

In the continuous approach the schedule is modified whenever a new disruption occurs, regardless of its relevance [23, 75]. Because of this, the *Stability* of the shop floor may be compromised when a large number of modifications, some of them unnecessary, are done. Reactive and predictive-reactive scheduling, previously discussed in subsection 2.2.2, are usually associated with this approach due to their feature of low predictability. Note that schedules with high *Performance* can be produced since a scheduling problem can be continuously re-optimised at a price of a higher computational burden.

The event-driven approach modifies the schedule only when important event occur [31,106]. This feature helps to overcome the drawbacks present in continuous approaches, in which highly stable and good quality schedule are obtained. Note that the schedule *Stability* may still be sacrificed when a large number of modifications is required. In practice, this approach is usually combined with repair methods that control shop floor *Stability*. More details about these methods are given in the following subsection.

The fourth type of approach, so-called hybrid, reschedules at predefined points in time and whenever a critical event occurs [28, 60, 90]. Critical events are set accordingly to the scheduler preferences and they identify the disturbances that have to be tackled during the schedule execution, such as machine breakdowns, rush orders, job cancellation, priority changes, among others. This approach combines the good *Stability* and the good *Performance* from the periodic and event-driven approaches, respectively. Note that it is possible to set alternative options mixing the other available approaches, i.e. periodic in the first moment and continuous in the next period. The aim is to consider the specificity of each scheduling problem to define the best rescheduling frequency.

Figure 2.4 illustrates a match between rescheduling frequencies and possible approaches. Periodic rescheduling is usually done on robust schedules due to their high predictability

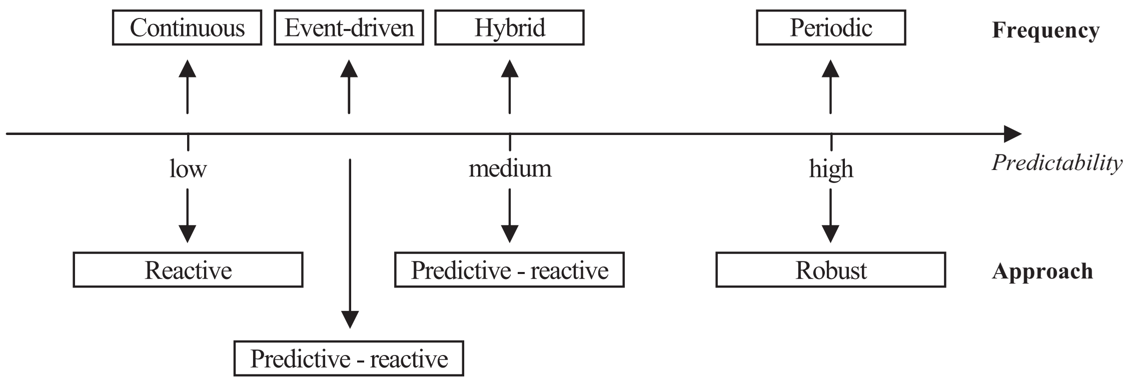


Figure 2.4: Match between rescheduling frequencies and possible approaches, based on predictability of the produced schedules.

of upcoming events. Contrary, continuous approaches are frequently applied in reactive schedules since a low predictability is presented by them. Event-driven approaches are usually associated with predictive-reactive schedules because their low predictability are often managed by applying some repair methods. Alternatively, a hybrid approach can be applied to these schedules due to the possibility of a medium predictability, i.e. events occurring on a regular basis coupled with other unexpected ones. In practice, good quality schedules are produced when the requirements of a scheduling problem are combined with a moderate rescheduling frequency.

2.2.4 Rescheduling Methods

The rescheduling methods are subdivided into two independent phases: (1) *schedule generation*, which determines how an initial schedule is produced; and (2) *schedule repair*, which establishes how a current schedule recovers from a disruption in order to restore its feasibility.

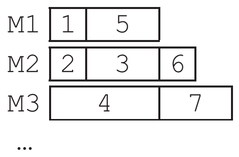
The initial schedule may be *nominal*, which is a schedule generated with the only focus on optimising *Performance* [60, 90]. The problem with nominal schedules is that they are highly sensitive to the problem data, which means that if the problem data changes, due to unpredictable circumstances, both *Performance* and *Stability* of the initial schedule are usually badly deteriorated. Despite this problem, most of the literature on scheduling is concerned with generating nominal schedules [91]. In practice, they have to be combined with appropriate repair methods in order to deliver high quality schedules [70, 73, 74, 116].

The initial schedule may also be *robust* [35], in which case it is generated with a protection against unforeseeable events. This protection takes the form of a certain amount of idle time that is inserted on the machines, between jobs, whose purpose is to absorb a number of disruptions without severely compromising both *Performance* and *Stability* of the schedule. The main drawback with the generation of robust schedules lies in the difficulty of defining the size of the temporal protection; too much inserted time inevitably deteriorates the schedule *Performance*, too little and the protection is useless.

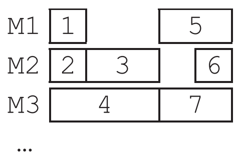
An example of a nominal and a robust schedule is shown in Figure 2.5 (a) and (b), respectively. The nominal schedule only prioritises its *Performance*, in which no idle times are inserted on machines and all required operations are processed as soon as possible in the shop floor. Contrary, the robust schedule allows a certain level of flexibility since idle times are present on both machines M1 and M2, which aim to manage possible disruptions and keep a good quality stable schedule. Note that both schedules finish the processing of their operations at the same time and the temporal protection present in Figure 2.5 (b) does not compromise the schedule *Performance*, since, hypothetically, the makespan is used to check its quality.

The repair methods restore the feasibility of the schedule when disruptions occur in the shop floor and they are subdivided into *right / left shift*, *complete* and *partial* rescheduling.

Basic methods such as right and left shift are commonly used in practice because they produce stable schedules [1, 59, 67, 90]. When a disruption occurs, assigned operations



(a)



(b)

Figure 2.5: Example of (a) a nominal and (b) a robust schedule.

may be either postponed or executed in advance depending on the new requirements. For instance, the insertion of a new job may require postponing, i.e. shifting to the right, a number of operations, whereas the removal of an assigned job may cause other operations to be shifted to the left. Such rules deliver stable schedules because the sequence of operations on machines is kept unchanged. However, the *Performance* is usually compromised due to the absence of optimisation methods during the pushing and pulling processes. An example of an initial schedule is presented in Figure 2.6 (a), in which a new job has to be inserted at the highlighted *rescheduling point*. The new job requirements and the resultant schedule applying right shift are shown in Figure 2.6 (b) and (c), respectively. Subsequently, job 4 is removed, in which the left shift method is applied. Note that only job 7 is moved backward because job 8 sets a precedence constraint to execute its operations in a predefined sequence, as in Figure 2.6 (d). The schedule *Performance* can be mainly affected because some jobs may become tardy after right shifting their allocations. Similarly, tardy jobs that were allocated in the initial schedule are not rearranged to use the extra space provided by the removal of some jobs, which may affect its overall quality.

The complete repair method, also known as “total rescheduling”, reallocates all the remaining operations present in the shop floor considering the new requirements presented by disruptions. This method often leads to high *Performance* values because the same scheduling problem is continuously optimised [13, 31, 93]. As a result of these several changes, a very low *Stability* is frequently associated with this method. In practice, total rescheduling is usually avoided because it generates additional manufacturing costs related to the holding of raw material, machine setups, and others, and is computationally expensive. A reasonable solution to use total rescheduling is to combine it with optimisation functions that aim to maximise both *Performance* and *Stability* during the repairing process [74]. Figure 2.7 (a) shows a schedule, in which a job has to be inserted at the highlighted *rescheduling point*. The new job requirements is shown in Figure 2.7 (b). The schedule is repaired and all the remaining operations after the *rescheduling point* are reallocated to accommodate the requirements of the new job 16, as in Figure 2.7 (c).

In partial rescheduling, only those operations affected by disruptions are reallocated. The aim is to preserve as much as possible the current schedule since it hypothetically sets an optimal solution. As a consequence, schedules are more stable than with to-

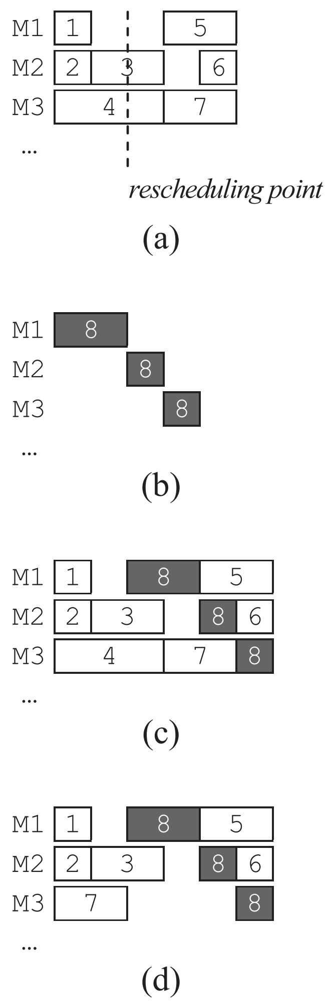


Figure 2.6: Example of (a) an initial schedule, with (b) a new job requirements, and its (c) resultant schedule when job 8 is inserted using right shift and, subsequently, (d) the removal of job 4 applying the left shit method.

tal rescheduling. Moreover, partial rescheduling often delivers schedules with similar *Performance* values as with total rescheduling, hence their popularity in practice. Partial rescheduling may use (1) match-up algorithms, in which modified schedules try to match-up its optimal initial solution as soon as possible; (2) knowledge-based models, in which the most constrained area of the scheduling problem is prioritised to be resolved first; and (3) robust scheduling, in which minor changes may be required in order to use idle times to absorb new disruptions. A general example is shown in Figure 2.8 (a), in which the same job 16, introduced in the previous example, has to be integrated in the current schedule at the highlighted *rescheduling point*. Note that only a part of the current schedule has to be changed to insert this new job, i.e. shaded operations. Consequently, a

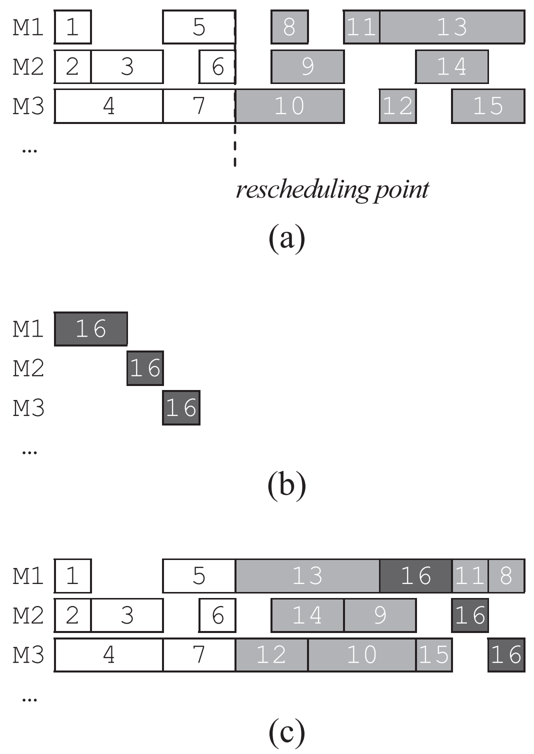


Figure 2.7: Example of (a) an initial schedule, with (b) a new job requirements, and its (c) resultant schedule when a complete repair method is applied.

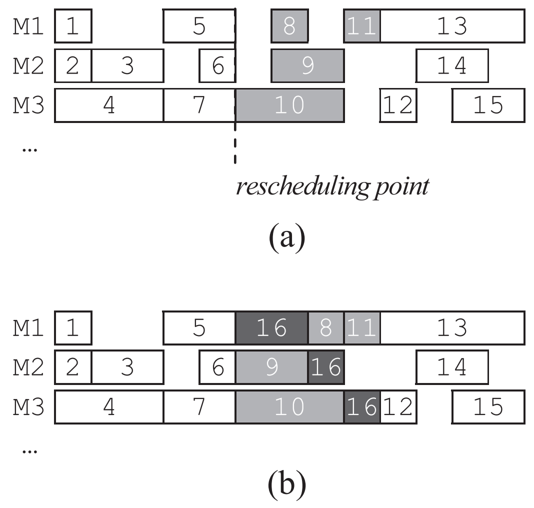


Figure 2.8: Example of (a) an initial schedule and its (b) resultant allocations when a partial repair method is applied.

more stable schedule is delivered in Figure 2.8 (b), when it is compared with a complete reallocation previously shown in Figure 2.7 (c).

Figure 2.9 summarises the rescheduling methods matching both scheduling generation and repair with possible rescheduling approaches. Nominal schedules are associated

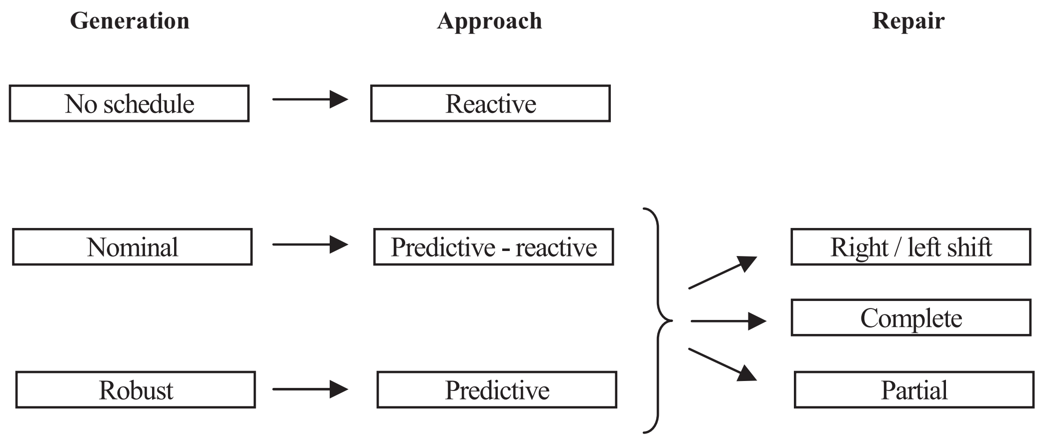


Figure 2.9: Match between generating schedules and possible approaches, together with their applicable rescheduling repair methods.

with predictive-reactive approaches because the absence of disruption prediction always requires some repair. Contrary, robust schedules are mostly applied with predictive approaches since they aim to absorb disruptions inserting idle times on machines. Note that both nominal and robust schedules may use the same rescheduling repair methods if the available idle times are not enough to accommodate the requirements of the new disruptions. Consequently, all repair methods can be associated with either predictive or predictive-reactive approaches. As a matter of completeness, reactive schedules do not generate an initial solution, hence no repair has to be done.

2.3 Match-up Approaches

Match-up algorithms start with an initial schedule, and whenever a disruption occurs, a time window within the schedule is defined, re-optimised taking into account the new disruption(s), and put back into the initial schedule. This repair is achieved by collecting available idle times on machines and changing a part of the current allocations to accommodate the unexpected event(s). These algorithms are originally inspired by the “turnpike theory” [66], in which an initial patch between two points has to be restored as soon as possible, since it already defines an optimal solution.

Match-up algorithms may belong either to the class of predictive-reactive approaches or to the class of predictive approaches depending on whether they start with an initial nominal schedule or a robust schedule, respectively. They also belong either to the class of continuous approaches or to the class of event-driven approaches, depending on whether the rescheduling is triggered at every disruption or only after the occurrence of what may be considered a relevant event. Additionally, match-up algorithms belong to the class of partial repair methods since they only modify a part of the schedule when accommodating occurring disruptions. The match-up approaches proposed in this thesis initially investigate predictive-reactive and event-driven approaches, since the initial schedule is a nominal one and the rescheduling process is triggered when a new job enters the system. Note that job arrivals are relevant disruptions because they often compromise multiple resources in the shop floor. Further details are presented and discussed in chapters 4 and 5. Subsequently, predictive approaches are also investigated because their strategy of inserting idle times on machines could possibly contribute to the effectiveness of match-up approaches, as described in chapter 6.

Match-up approaches are attractive given that they are easy to conceptualise and because they provide good results not only with respect to schedule *Performance*, but also *Stability*. Nevertheless their application has been limited only to a small variety of problems, most of which are of a more theoretical than practical importance. For instance, match-up algorithms have been used in predictive-reactive approaches to repair single machine shop floors [11, 14, 15, 116] and single stage with parallel machine shop floors [12]. Flow shop models have been considered in [3] and job shop problems in [1, 97, 103, 104, 106]. A detailed description of these scheduling models can be found in [91]. This thesis and its resultant papers in [70, 72–74] are the only attempts to use match-up algorithms in a complex production shop floor which includes multiple criteria, setup times and parallel machines.

An essential part of match-up approaches is the algorithm in charge of re-optimising the rescheduling horizon. In most cases, re-optimisation algorithms have been relatively simple scheduling heuristics; however, in [1, 97, 103, 104, 106, 107] considerably complex knowledge based systems have been investigated. In these systems, previously stored knowledge, obtained after experience (training), is used to select a rescheduling strategy which is expected to be appropriate for the current rescheduling problem. These methods select the most constrained part of the schedule as the rescheduling horizon. An important drawback of this is that constraint violations may propagate to a large part of the schedule, requiring multiple repairing iterations that compromise both *Performance* and *Stability* of the schedule. The investigation presented in this thesis and the research presented by [116] are the only ones using genetic algorithms as re-optimisation engines of match-up algorithms.

As previously mentioned, another important feature that may contribute to the match-up approaches effectiveness is the algorithm in charge of generating its initial predictive schedule. Robust scheduling has been mostly investigated in machine breakdowns problems, in which a single resource is usually compromised by disruptions on the shop floor. These problems have been using fuzzy processing time and release time to manage temporal uncertainties [21, 35]. Alternatively, branch and bound heuristics [61, 67], genetic algorithms [47, 59] and temporal protection based on historical data of the resources allocation [26] have been used to produce schedules that aim to absorb occurring disruptions. Jobs with changing processing times are investigated in [28], which also applies fuzzy variables to set durations of operations. The research present in this thesis and its resultant paper in [73] are the only applications of match-up algorithms with robust scheduling to a complex real world job shop problem.

2.4 Discussion

All rescheduling features, presented in the previous sections, have their strengths and limitations. For instance, static environments set desirable hypothetical problems because all information is always given in advance, which allows an optimal schedule to be mostly executed as initially planned. Unfortunately, real world situations are not that predictable and unexpected events often occur in the shop floor, which usually require rescheduling. These problems belong to the class of dynamic environments.

Different algorithms can be applied to manage uncertainty present in dynamic environments. For instance, reactive approaches do not create a schedule and real time control actions are applied to allocate the available resources over time. However, a *Performance* value is difficult to predict and the shop floor productivity can be easily affected. Alternatively, predictive algorithms produce robust schedules, which aim to absorb some disruptions using extra idle times that were inserted on machines during the schedule generation.

The main issue presented by this approach is how to define the amount of this temporal protection without affecting the overall quality of the schedules. Predictive-reactive algorithms aim to overcome those drawbacks, in which an initial optimal schedule is produced and it is subsequently changed when a disruption occurs. This approach, however, may easily compromise the schedule *Stability* and an effective repair method, such as *partial rescheduling*, must be applied in order to produce a high quality and stable solution. Note that complete rescheduling and right / left shift usually generate suboptimal repaired solutions, because they either optimise the schedule *Performance* or *Stability*, respectively.

The rescheduling frequency is also an important factor which controls the quality schedules. Periodic approaches guarantee stable solutions, because the rescheduling is done only at predefined rescheduling points. The main difficulty is to define these points in order to avoid the *Performance* being deteriorated. Contrary, a continuous approach sets schedules with good *Performance* and poor *Stability* because a problem is continuously optimised whenever a disruption occurs. Event-driven approaches aim to overcome those drawbacks requiring rescheduling only when a critical event arises in the shop floor. Alternatively, hybrid approaches can be applied to combine the previously described options, i.e. rescheduling at predefined points and whenever an important event occurs. The aim is to consider the specificity of each scheduling problem in order to define the best rescheduling frequency.

In summary, a reasonable approach to manage real world problems is to consider production scheduling as a dynamic environment, in which disruptions occur and a rescheduling process may be required. Predictive-reactive and predictive approaches are suitable strategies to model these problems whenever *Performance* and *Stability* are considered to be relevant factors during the schedule execution. Moreover, event-driven or hybrid approaches coupled with partial rescheduling are highlighted as good repairing methods because they aim to deliver high quality and stable schedules.

This thesis and its resultant papers investigate a real world dynamic environment, in which new jobs have to be integrated in a current schedule. This problem is a generalisation of possible disruptions because its requirements are able to compromise not only one, but many resources present in a shop floor. A partial repair method called match-up is responsible to accommodate these disruptions with the aim of keeping good schedule

Performance and *Stability*. This study represent the only attempts to employ match-up algorithms in a complex production shop floor which includes multiple criteria, setup times and disruptions affecting multiple resources. These algorithms are initially applied following a combination of predictive-reactive and event-driven approaches, since initial optimal schedules are changed when relevant disruptions enters on the system, i.e. the arrival of new jobs. Subsequently, predictive approaches are also investigated because their strategy of inserting idle times on machines could affect positively the rescheduling process. More details about these investigations are discussed in the following chapters.

2.5 Summary

This chapter describes a literature review of rescheduling algorithms, in which a taxonomy of possible environments, approaches, frequency and methods is presented and discussed. The aim is to provide a guideline to understand related terminologies, applied strategies and their limitations. This taxonomy is subsequently linked with match-up algorithms and their possible combination with robust schedules, which are the main research topics investigated in the following chapters of this thesis.

A reasonable approach to manage real world problems is to consider production scheduling as a dynamic environment, in which disruptions occur and a rescheduling process may be required. Both predictive-reactive and predictive approaches are suitable strategies to model these problems because they either re-optimise a current solution or try to absorb unexpected events, respectively. Event-driven or hybrid approaches are suggested as good rescheduling frequencies because they are able to prioritise only relevant disruptions, delivering high quality solutions. Match-up algorithms are recommended as repair methods due to their ability to keep as much as possible an original optimal solution, which positively affect both *Performance* and *Stability* of schedules.

The match-up algorithms proposed in this thesis are combined either with predictive-reactive or predictive approaches, at a event-driven frequency. Their application has been limited only to a small variety of problems, most of which are of a more theoretical than practical importance. The research present here and its resultant papers represent the only attempts to employ match-up algorithms in a complex real world shop floor which includes multiple criteria, setup times, parallel machines and disruptions affecting multiple resources.

Fuzzy logic concepts are employed to manage the uncertainties that are present in the analysed shop floor. The main aim is to control ongoing variations on processing times, release and due-dates; and minimise possible effects of occurring disruptions. Consequently, the following chapter provides an overview of fuzzy systems and their link with scheduling / rescheduling problems.

Chapter 3

Fuzzy Systems and Scheduling

3.1 Introduction

This chapter describes an introduction to fuzzy systems, which represents an effective means to manage uncertainties that are always present in real world problems. The aim is to introduce their essential concepts and show an example how to create them. Additionally, these concepts are linked with requirements usually present in scheduling problems, such as uncertainty and flexibility for making decisions on dynamic problems. Further investigation about applying these concepts to a real world problem is described in the following chapters of this thesis.

The human being has the ability to handle complex processes on its daily routine, which often involve approximate reasoning. The ways adopted by human operators to manage such situations has also inaccurate sources, due to the fact that people commonly use linguistic terms in their decision making, using words such as "high", "low", "very", "little", among others.

The classical logic described by Aristotle, also known as standard logic, classifies objects in well-defined categories, in which "everything" has to be or not to be "something", either now or in the future. Although this binary logic has the ability to solve an extraordinary range of problems, it is necessary to fulfil remaining gaps that are not adequately addressed by these traditional methods. Fuzzy logic concepts bring more flexibility to this binary classifications, in which new "degrees of truth" are available between "yes" and "no". These degrees can be compared as shades of gray between black and white, which gives a generalisation of the Aristotelian logic.

Important philosophers, such as Bertrand Russell and Albert Einstein, highlighted the inability of standard logic to manage real world problems. The following thoughts are attributed to Russell showing his position: “Every language is vague”, “All traditional logic habitually assumes that precise symbols are being employed”. “Therefore, this is not applicable to terrestrial life, but only to an imaginary heavenly existence” and “... you cannot imagine how it is vague until you try to do it accurately”. The following statement is attributed to Einstein : “When the laws of mathematics refer to the reality, they are not correct. But, when these laws are correct, they do not refer to the reality”. A Polish mathematician called Jan Lukasiewicz developed a multi-valued logic in 1920 [63], discussing mainly the law of contradiction, in which a statement such as “X and Y can be and not be something at the same time” is perfectly plausible, in mathematical terms, since the degrees of truth are not only bivalent as true and false.

But it was in 1965 that the fuzzy set theory was conceived by Professor Lotfi Zadeh at the University of California, Berkeley. The aim was to introduce a more flexible logic, called Fuzzy Logic, creating a method to translate verbal expressions (vague, imprecise and/or qualitative) to tractable numerical values [118]. Professor Zadeh also formulated the principle of incompatibility in 1973, stating that: “As the complexity of a system increases, our ability to make accurate statements and that are significant about this system decreases until a threshold is reached, beyond which precision and significance (or relevance) become almost mutually exclusive characteristics” [119]. Additionally, there is an inconsistency between the human creativity and the possibilities offered by binary machines. Therefore, the concepts presented by Zadeh eliminate those restrictions by providing a mathematical tool for handling properly the vagueness present in real world.

Fuzzy logic systems were firstly explored in commercial market contexts due to the resistance of scientists. However, they have been designed and enhanced in academic contexts after their effectiveness has been proved [109]. The first commercial applications were in the control area, both in process automation and supervision. Since then, there has been an increasing use in various scientific fields such as classification, series forecasting, data mining, planning and optimisation. Some successful examples are: speed control, acceleration and braking of the trains in Sendai subway (Japan), ultra-fast chargers for

NiCd battery of Bosh, smoke detectors Cerberus (Switzerland), image adjustments for Sony Tvs, auto-focus video camera for Canon, Hitachi elevators optimisation, among others [101]. Consequently, fuzzy logic represent a more realistic way to model real world problems, allowing binary machines to work closely to human thinking, which is inherently “fuzzy”.

The remaining of this chapter is organised as follows. Section 3.2 introduces the concepts of fuzzy variables and sets. Section 3.3 describes how to set fuzzy rules and make implications. Section 3.4 presents the structure of a general fuzzy controller. Section 3.5 combines the previous concepts describing a detailed example. Section 3.6 discusses the application of fuzzy logic to scheduling problems. Finally, sections 3.7 and 3.8 conclude this chapter.

3.2 Variables and Sets

The concept of membership of an element to a particular set is well defined when classical Aristotelian logic is used, which means that using the attribute of bivalence it is possible to set a function to identify whether an element belongs or not to a specific group. For example, given a set A in a universe of discourse X, the characteristic function $f_A(x) = 1$ defines when $x \in A$ and, consequently, $f_A(x) = 0$ when $x \notin A$, where x is an element of the universe of discourse X.

However, there is a mismatch between the real world and such bivalent approach, i.e. how to define correctly when a person is *young*, or when the weather is *hot*. In the real world, everything is a matter of perspective and very strict definitions may certainly lead to loss of information. Therefore, a multivalent approach is required to define gradations between true and false, in which possibilities of interpretation are extended. The concepts of fuzzy logic allow to capture such degrees of truth of statements, working with the uncertainty and partial truth of natural phenomena in a systematic and accurate fashion [101]. Consequently, the characteristic function can now be defined as a real number belonging to the interval $[0,1]$, eliminating the restriction of the values being described as only 0 or 1. The membership function $\mu_A(x)$ indicates the membership degree (or compatibility) of an element x to set A within the universe of discourse X, with:

- $\mu_A(x) = 1$ when x is fully compatible with A;
- $\mu_A(x) = 0$ when x is completely incompatible with A;
- $0 < \mu_A(x) < 1$ when x is partially compatible with A, assuming the value $\mu_A(x)$.

Figure 3.1 shows a comparative example of the set “hot” using the boolean (a) and the fuzzy (b) approaches. In the boolean approach, temperatures up to 25^{o}C are not considered to be hot and this status abruptly changes to hot when values exceed this point. This definition is rather restrictive because there is no space for different perspectives regarding the feature temperature. On the other hand, the fuzzy approach sets that elements with values greater than 20^{o}C become part of the set “hot” with an increasing membership degree, with its minimum and maximum value at 20^{o}C and 25^{o}C, respectively. This definition brings flexibility to identify degrees of representativeness that a value can assume within a certain set.

A representative fuzzy variable can be built when its universe of discourse is subdivided into different fuzzy sets, in which each set has an identification label. Figure 3.2 shows a graphical representation of the fuzzy variable *temperature* with these subdivisions. The universe of discourse is delimited by temperatures with values between 0^{o}C

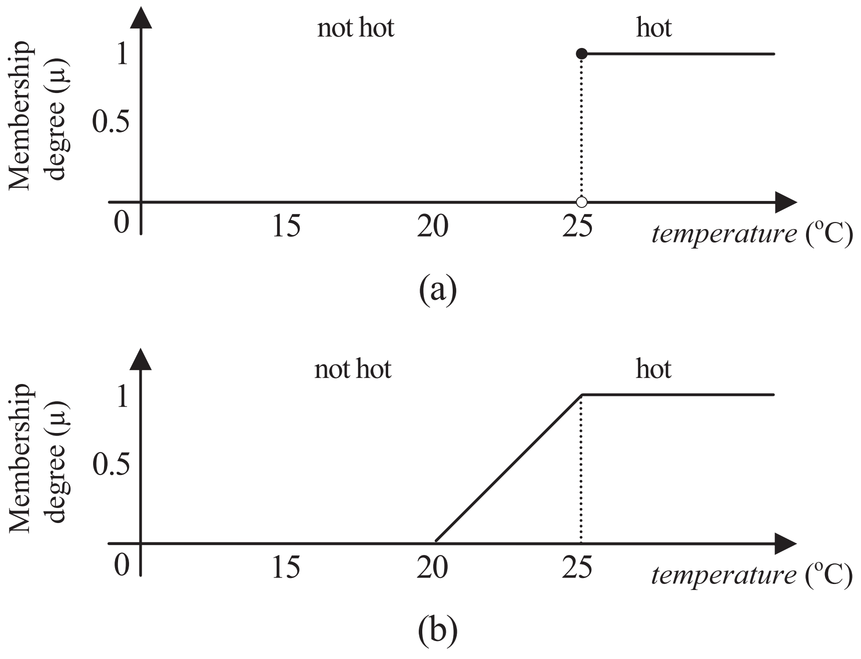


Figure 3.1: Comparative example of the set “hot” using the boolean (a) and the fuzzy (b) approach for the variable *temperature*

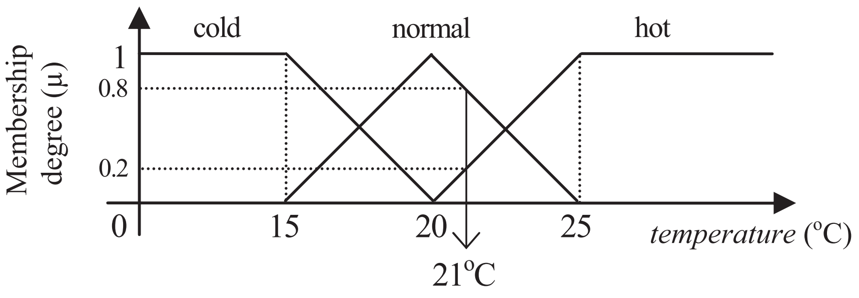


Figure 3.2: Fuzzy variable *temperature*

and 50°C. The variable *temperature* is then subdivided into 3 fuzzy sets: cold, normal and hot. Each set has an interval to describe the related feature, i.e. cold, normal and hot have the following intervals $[0,20]$, $[15,25]$ and $[20,50]$, respectively. Note that shapes and position of each set within the universe of the discourse will depend on the expert preference, which takes into consideration the complexity of the model and the required computational costs. In practice, simple functions, such as triangular, trapezoidal and Gaussian, are the most commonly used to describe fuzzy sets because they simplify the computation and produce good results. These sets quite often require some tuning before becoming good representatives of a variable.

A temperature of 21°C is highlighted by a thin arrow in Figure 3.2. This temperature belongs to both sets normal and hot with the following membership degrees 0.8 and 0.2, respectively. Consequently, the same element x can simultaneously assume different membership degrees to different sets, which is represented by $\mu_A(x)$, in which $\mu_{normal}(21) = 0.8$ and $\mu_{hot}(21) = 0.2$. The flexibility is a important feature present in fuzzy variables, because their labels are not necessarily exclusive. This kind of definition also allows the identification of elements that are more representative of a general idea of a specific set, i.e. as closer the value $\mu_A(x)$ is to 1.0, the greater is the degree of representativeness of the linguistic term applied.

Operations between fuzzy sets are calculated based on the applied membership functions. According to Zadeh [118], the inclusion function for the union U of two sets A and B (U= A ∪ B) is defined as $\mu_U(x) = \max(\mu_A(x), \mu_B(x))$, for an element x within the universe of discourse X. The intersection I between the same sets A and B (I = A ∩ B) is defined as $\mu_I(x) = \min(\mu_A(x), \mu_B(x))$ with $x \in X$. Finally, the complement function C of a set A is $\mu_C(x) = 1 - \mu_A(x))$ with $x \in X$. These configurations are equivalent to operations described in the classical set theory, in which possible values are described between the interval [0,1] and not only 0 or 1 anymore. Alternatively, other definitions for the union and intersection operators has been investigated by other researchers [25, 117].

Note that the "Law of Non-Contradiction" ($A \cap \neg A = \varnothing$) and the "Law of Exclusion" ($A \cup \neg A = E$) are not included in the fuzzy approach. The classical logic would identify as a contradiction elements belonging to a set and its complement simultaneously. For instance, a temperature would not be able to be part of the sets "not hot" and "hot" at the same time. The fuzzy variable *temperature* is illustrated again in Figure 3.3 (a), in which the intersection between the sets "not hot" and "hot" are not empty. Note that a day with a temperature of 22.5^{o}C is considered to be "hot" and "not hot" with the same membership degree of 0.5 to both sets. Similarly, Figure 3.3 (b) shows that the union between these sets does not cover the entire universe of discourse of variable *temperature*, which means that there is an "uncertainty" factor for values between 20^{o}C and 25^{o}C.

3.2.1 Hedges

Fuzzy variables can have also their meaning intensified (or attenuated) by hedges, which act like adverbs and adjectives to modify the meaning of nouns, such as the temperature

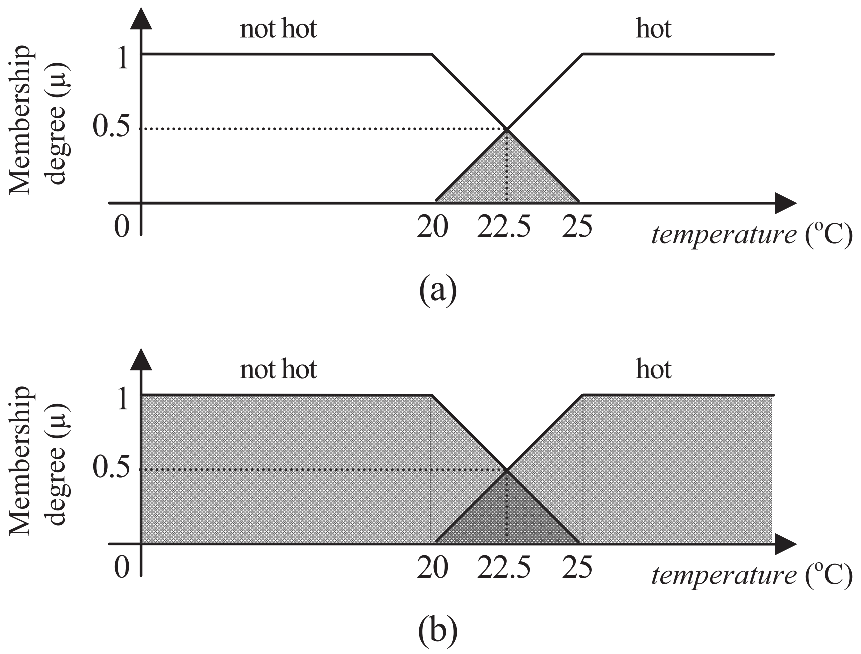


Figure 3.3: Intersection (a) and union (b) between the fuzzy sets "not hot" and "hot" for the fuzzy variable *temperature*

today is “very” cold, and the water yesterday was “somewhat” cold. The main idea is to intensify (or attenuate) membership functions in such a way that fuzzy variable representatives assume higher (or smaller) values between the interval [0,1]. A graphical example from the previous sentences are shown in Figure 3.4 (a) and (b), respectively.

Note that “very” cold defines a more concentrated representation for the variable *temperature*, while “somewhat” cold sets a more dilated area for the variable *water*. A temperature of 16°C is definitely a member of cold, but less of a member of “very” cold. Similarly, the water at 18°C is a member of cold, but more a member of “somewhat” cold.

The modifier “very” will be used on Chapter 6 to describe a fuzzy variable of the investigated scheduling problem.

3.3 Logical Implications and Inference Rules

Logical implications are commonly used by human beings to formulate connections between causes and effects, in which inference rules are consciously or unconsciously created in the following format: *if* (antecedents) *then* (consequents).

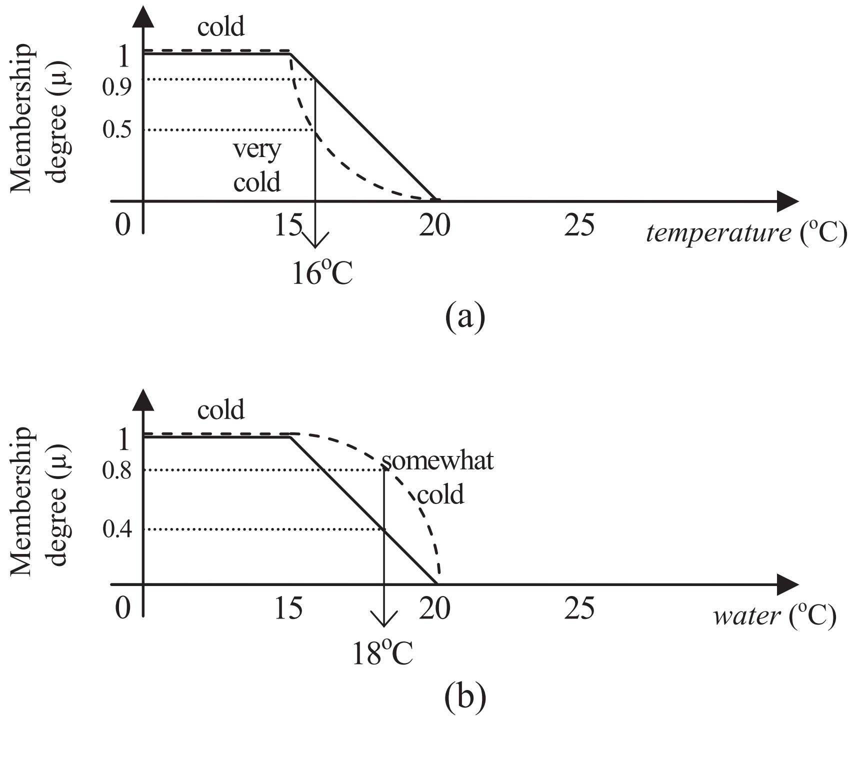


Figure 3.4: Hedges “very” and “somewhat” applied to the fuzzy set cold from variables *temperature* and *water*, respectively.

These rules can combine several antecedents (premises) and consequents (conclusions) by using logical operators such as “and” and “or”. The structure of a fuzzy conditional proposition is similar to the boolean logic, in which signs such as $<$, $>$ and $=$ can be easily replaced by linguistic terms as “lower”, “larger” and “equivalent”. However, the interpretation of a fuzzy rule is rather different when compared with a traditional rule.

In the boolean logic, a conclusion is inferred only if the statement of the antecedents is considered to be true. For instance, a rule having only connectives “and” must have all premises as positive to validate its conclusions. On the other hand, rules having only connectives “or” must have at least one of its premises true to infer the conclusions.

In the fuzzy logic, the premises may take degrees of truth in an interval between completely false and entirely true. Therefore, evaluations of the antecedents can be analysed through the operations defined by Lotfi Zadeh, in which the operators *max* and *min* are representations for the classical operators “or” as union and “and” as intersection, respectively [118]. For instance, consider the rule “if $(a$ is $A)$ and $(b$ is B or c is $C)$ $then$ $(d$ is $D)$”, with the following degrees of inclusion $\mu_A(a) = 0.7$, $\mu_B(b) = 0.3$ and $\mu_C(c) = 0.5$, then the assessment would generate the following result $\mu_A(a) \cap (\mu_B(b) \cup \mu_C(c)) = min(0.7, max(0.3, 0.5)) = min(0.7, 0.5) = 0.5$. This result reflects the membership degree of the conclusion D, i.e. the degree of relevance that the consequent has over the set D. Note that multiples rules can be activated during the inference process. A more detailed and graphical example is described in section 3.5.

It is important to highlight that several rule-based systems can be created based on interviews with experts, which usually have a solid experience about the context of the proposed application. Therefore, the freedom of a system designer to change the structure of inference system is related with their understanding about the descriptions provided by the specialist. On the other hand, much less adjusting time is expected for this type of system, since the experience of the expert will be embedded in the rules, which often contains the best performance.

The knowledge base of a controller contains the combination of all inference rules, which will perform all the desired control actions under specified conditions. Table 3.1 shows an example of inference rules applied to control the level of an air conditioner, in which the input variables $temperature$ and $humidity$ are combined to generate the ap-

Table 3.1: Fuzzy rules r_i for the inputs *temperature* and *humidity* to decide the appropriate *level* of the fan

		Fuzzy rules r_i					
		r_1	r_2	r_3	r_4	r_5	r_6
if	*temperature*	low	low	medium	medium	high	high
and	*humidity*	low	high	low	high	low	high
then	*level*	very low	low	medium	medium-high	high	very high

propriate *level* of the fan. For instance, a day with low *humidity* and high *temperature* activates the fuzzy rule r_5, which sets the fan to *level* high. A more extensive discussion about setting rules are presented in section 3.5.

It is important to have as many rules as necessary to map all combinations of input variables in order to create a complete knowledge base, which triggers at least one rule independently of the input. The consistency between rules are also essential since contradictions and cyclic situations must be avoided [99]. Note that rules can have multiple inputs and outputs. However, they do not accept the connective “or” in conclusions [48].

3.4 Fuzzy Systems

Computer science is based on the principle of bivalence, in which bits assume either value 0 or 1. Regular computational procedures do not have the ability to recognise linguistic terms, which are commonly used in the human communication. The fuzzy logic concepts aim to fulfill this gap setting degrees of truth for statements in such a way that machines can successfully process such information.

There are many types of fuzzy systems presented in the literature. The most commonly used are the classic ones described by Mamdani [65] and Larsen [56]. Alternatively, other approaches were proposed by Takagi-Sugeno [108] and Tsukamoto [113], in which interpolation techniques are added to describe their models. These types are discussed in details in section 3.4.3. All of them have the same basic representational structure shown in Figure 3.5, which is an adaptation of the description present in [57]. Note that different systems may have different requirements depending on their specification. Figure 3.5 shows a general model to identify how the information flows in a typical fuzzy system. All inputs and outputs are crisp values and the fuzzy controller defines three

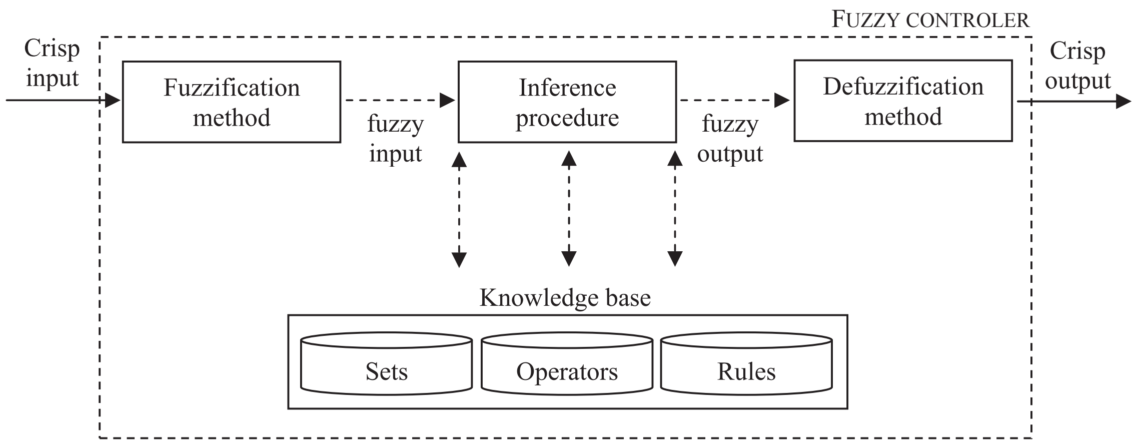


Figure 3.5: Typical structure of a fuzzy controller

main processes in which the inputs are fuzzified, and then an inference procedure uses a knowledge-base containing sets, operators and rules to generate a control action, and finally, the outputs are defuzzified. A brief description of each step and their associated modules are described in the following subsections. Subsequently, section 3.5 presents a complete example of the whole process.

3.4.1 Fuzzification Method

The fuzzification method evaluates all the input values and map them into fuzzy sets. In other words, this process converts a crisp number into a fuzzy one in such a way that a numerical number becomes an instance of a linguistic variable.

3.4.2 Knowledge Base

The knowledge base stores all the information about the sets and operators of the fuzzy model, describing the universe of discourse of each variable, their membership functions and their respective linguistic terms. Additionally, it has the inference rules, which are responsible to configure a control strategy and its goals.

3.4.3 Inference Procedure

The inference procedure combine the system rules, described in the knowledge base, with the input data transformed into fuzzy variables. As a result, control actions are regenerated based on the current state of the system, in which implication operators such as if and $then$ are applied. This process is described by the following steps, as in [99]:

1. Check the membership degrees of the inputs;
2. Determine an overall degree for each activated rule;
3. Determine a conclusion value, based on the membership degree of each activated rule, which can be a crisp or a fuzzy number;
4. Combine all the values obtained by all activated rules in order to generate an output with a global control action.

In a classical fuzzy model, the conclusion of each rule specifies a fuzzy set. Consequently, it is necessary to apply an aggregation technique on the antecedent sets for each rule in order to generate a consequent set. The following models will be described: Mamdani, Larsen, Takagi-Sugeno and Tsukamoto.

In the Mamdani model, this aggregation is done by applying the operator “intersection” (minimum), in which the consequent is cut horizontally in the lower level of inclusion activated by applied rules [65]. Figure 3.6 shows an example of two inputs x_a^* and x_b^* simultaneously activating two fuzzy sets, A_1–A_2 and B_1–B_2, respectively. The combination of antecedents A_1 and B_1, and A_2 and B_2 generate the conclusions C_1 and C_2, respectively. Note that the consequent sets C_1 and C_2 were cut in the minimum degree of inclusion of the antecedents A_1 and B_1, and A_2 and B_2, respectively. The combination of the antecedents A_1 and B_2, and A_2 and B_1 were not considered just as a matter of simplifying this example.

In the Larsen model, this aggregation is done by the operator “product”, which has a flattening effect on the consequents [56]. Figure 3.7 shows the same example previously presented in Figure 3.6, but now using the Larsen approach. The inferences obtained in $C1$ and $C2$ are results of a proportional reduction when the antecedents A_1 and B_1, and A_2 and B_2 are combined, respectively.

Lets consider that the only generated consequents are $C1$ and $C2$, as shown in both Figure 3.6 and Figure 3.7. The next step is to combine these consequents into C' by using

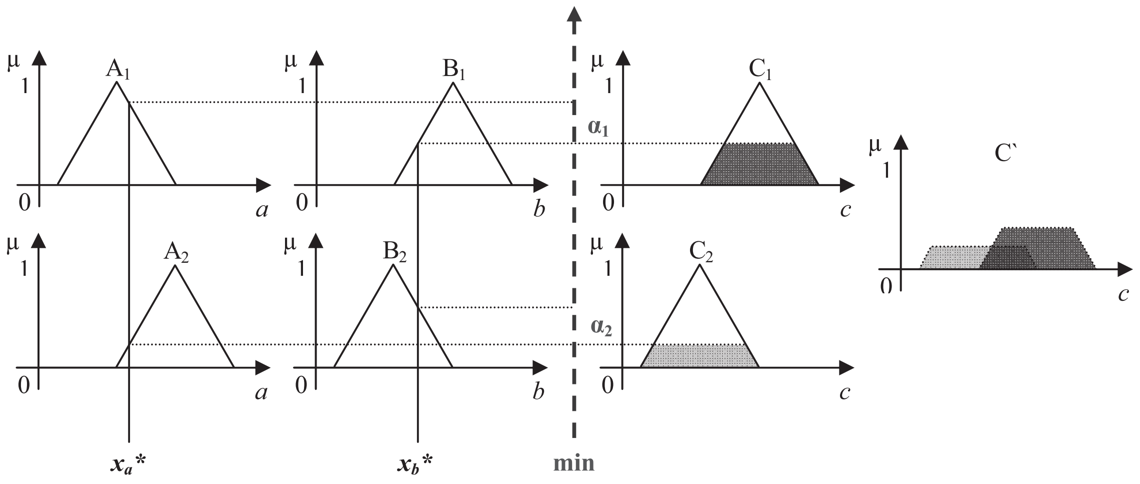


Figure 3.6: Example of a classic Mamdani model

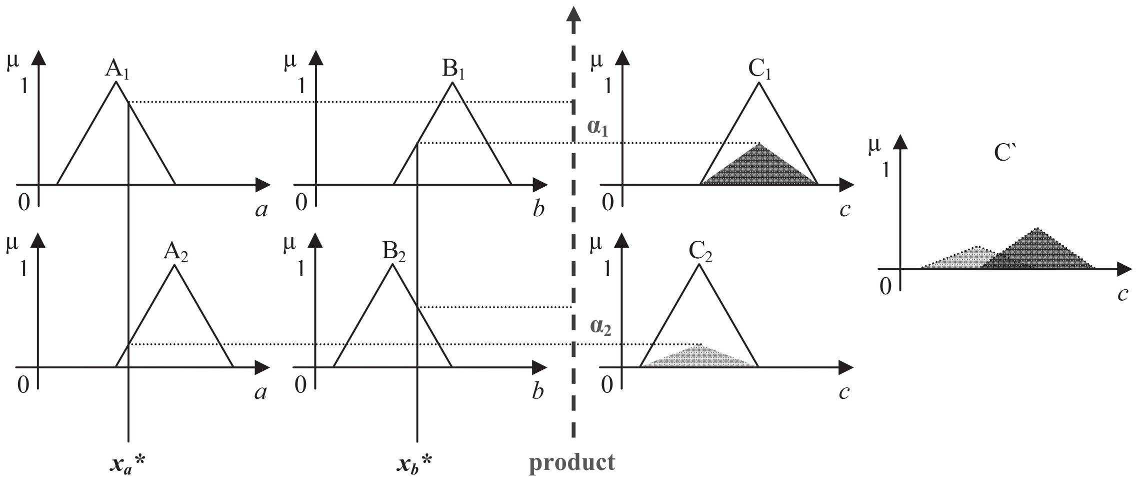


Figure 3.7: Example of a classic Larsen model

the aggregation operator “union” (maximum), which are highlighted in the previously mentioned Figures for both Mamdani and Larsen models.

In the fuzzy interpolation models, each consequent is given by a monotonic function, which is usually unique for each activated rule. These functions are generated using training and validation samples, in which weights are adjusted in order to set control actions. This process follows the same principle used by neural networks [96], in which the use of historical data allows the prediction of expected actions.

In the Takagi-Sugeno model, this function is a linear combination of inputs, in which parameters are defined as a set of constants [108]. Figure 3.8 illustrates this approach, using the same example described for classical models. The antecedents aggregation is done by applying the operator “intersection” (minimum) and each activated rule defines

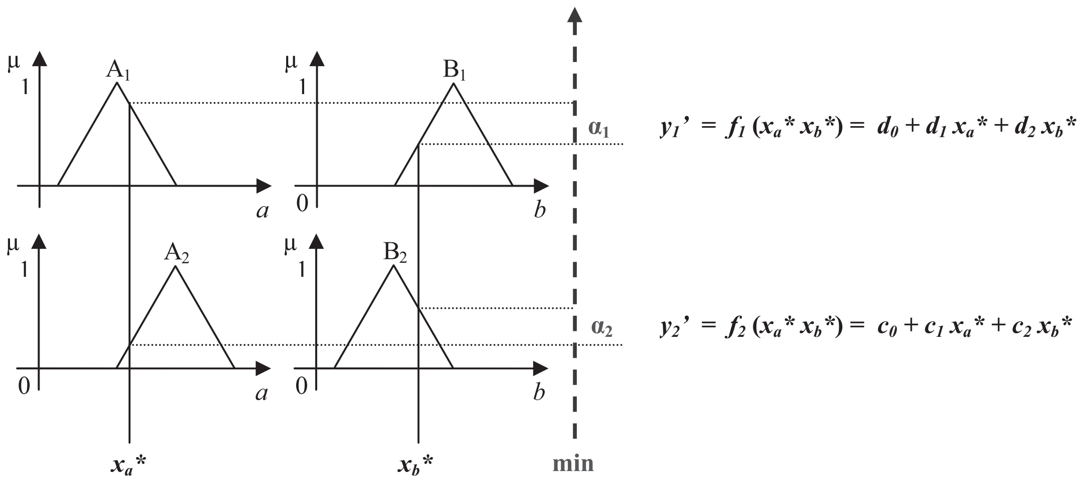

μ
1
A1
a
μ
1
A2
0
a
μ
1
B1
0
b
μ
1
B2
0
b
α1
α2
x_a*
x_b*
min
$y_1' = f_1(x_a^* x_b^*) = d_0 + d_1 x_a^* + d_2 x_b^*$
$y_2' = f_2(x_a^* x_b^*) = c_0 + c_1 x_a^* + c_2 x_b^*$

Figure 3.8: Example of a Takagi-Sugeno interpolation model

a monotonic function. For instance, A1 with B1 activates the rule $y_1' = f_1(x_a^* x_b^*) = d_0 + d_1 x_a^* + d_2 x_b^*$, in which d_0, d_1 and d_2 are weights for the monotonic function. Subsequently, a crisp value y_1' is obtained, since x_a^* and x_b^* are substituted in the function f_1 together with the previously defined constants d_0, d_1 and d_2. The same procedure is followed for the second rule, in which A2 and B2 are combined.

In the Tsukamoto model, the function is usually nonlinear [113]. Figure 3.9 shows this approach applied to the previous example. A reference to the minimum membership degree is still applied, but the consequent is now set with a pre-defined function. As in the Takagi-Sugeno model, crisp values are generated for y_1' and y_2' combining A1 and B1, and A2 and B2, respectively.

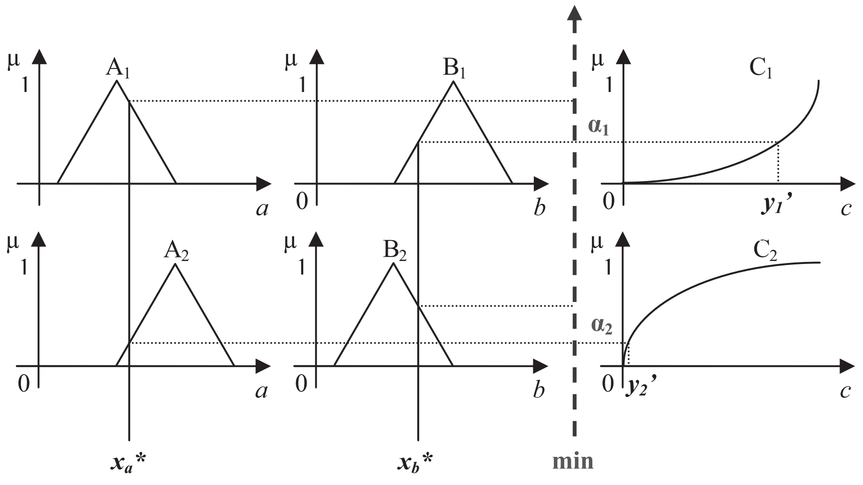


Figure 3.9: Example of a Tsukamoto interpolation model

Note that each rule sets a consequent value when interpolation models are applied, and consequently crisp conclusions are defined. An overall control action is then obtained when a weighted average of these individual conclusions is calculated, i.e. y'_1 and y'_2, from both Figures 3.8 and 3.9, where the weights are the membership degrees of the inputs x^*_a and x^*_b [29].

3.4.4 Defuzzification Method

The defuzzification method is responsible to create a control action based on results provided by the inference procedure. In other words, it transforms the consequent fuzzy sets into a “crisp” output value. Note that only classic fuzzy models require this procedure, since mathematical functions used in interpolation models already set accurate outputs. The defuzzification methods most commonly used are described below:

- First maximum value: the curve generated by the consequent fuzzy sets is analysed and the first point of maximum of this curve defines the output;
- Average between maximum values: same idea as the previous method, but all maximum values are considered and an average point among them is calculated in order to set the output;
- Centre of gravity: the area defined by the consequent fuzzy sets is evenly subdivided by a centre point, which represents the required output.

The selection of the defuzzification method is done by taking into consideration the expected behaviour of the control system. For instance, both methods “first maximum” and “average between maximum values” are not suitable to set machine operation modes, because abrupt changes will be often inferred by the system and these bumps could easily damage the involved equipments. For this problem, the method “centre of gravity” would be recommended, since the generated control actions are smoother.

There are other defuzzification methods, in which different factors, as speed and efficiency, are considered [29, 38, 40, 78, 81, 99]. Note that the most appropriated method for a system depends on the specificity of each problem.

3.5 Example

A classic problem of parking a truck [34, 50, 51] is described in this section in order to demonstrate the steps previously described in section 3.4. This illustrative example is selected because it provides clear details about generating fuzzy inferences and, more importantly, it is quite simple to be understood. Note that a fuzzy scheduling example is subsequently described in section 3.6.

The problem starts with a truck parked in a random position (x,y) with an angle ϕ with the horizontal line. The pair (x,y) specifies the central position of the truck's back and the goal is to define control actions to allow the truck to reach the final parking position (x_f,y_f) with angle $\phi = 90^o$, in which maneuvers are only made when the vehicle is reversing. Figure 3.10 identifies the truck, in the position (x,y) with its respective angle ϕ with the horizontal line, and the desired final parking position (x_f,y_f).

At each step of the simulation, the fuzzy system has to produce a rotation angle θ, which updates the position of the steering wheel, allowing the truck to develop a patch toward his goal on the position (x_f,y_f). The angle θ is initially set to zero, in which wheels are considered to be parallel to the side of the vehicle. It is also assumed that there is enough space for the truck to make several moves with a constant speed r. The following equations describe the movement between the positions (x,y) and (x',y'):

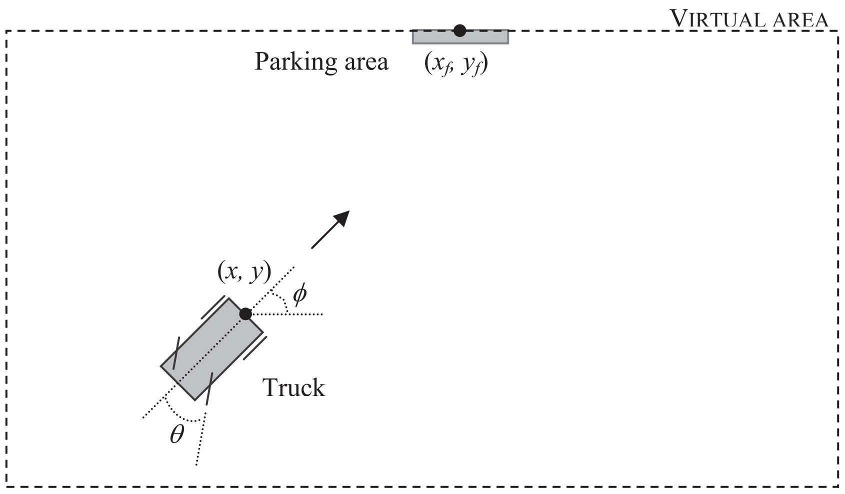


Figure 3.10: Representation of a virtual area with the truck and its parking area

$$\begin{cases} \phi' = \phi + \theta \\ x' = x + r(\cos\phi') \\ y' = y + r(\sin\phi') \end{cases}$$

At each iteration, the fuzzy system is responsible to set the output θ based on the inputs x and ϕ, as in Figure 3.11. Note that the parameter y is not involved in the decision making due to its effectiveness while using few parameters, as described in [51]. Subsequently, the position of the wheels are updated, in which the current angle ϕ is incremented by the newly generated θ. Additionally, the overall position of the vehicle is updated, since the speed parameter r is applied and a reversing movement is done at each step of the simulation, respectively.

The universe of discourse of each variable is described by the intervals below, in which positive and negative angles represent clockwise and counterclockwise rotations.

$$\begin{cases} 0 \leq x \leq 100 \\ -90^o \leq \phi \leq 270^o \\ -30^o \leq \theta \leq 30^o \end{cases}$$

The three fuzzy variables x, ϕ and θ are subdivided in the following linguistic sets:

- Position x: LE (left), LC (left centre), CE (centre), RC (right centre) and RI (right);
- Angle ϕ: RB (right below), RU (right upper), RV (right vertical), VE (vertical), LV (left vertical), LU (left upper) and LB (left below);
- Angle θ: NB (negative big), NM (negative medium), NS (negative small), ZE (zero), PS (positive small), PM (positive medium) and PB (positive big).

The rule base, which represents the strategy to update the wheels of the truck, is

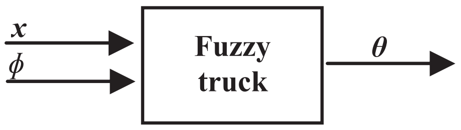


Figure 3.11: Fuzzy truck with the inputs x and ϕ, and the output θ

represented by the matrix shown in Table 3.2. The fuzzy sets from both inputs x and ϕ are combined among themselves in order to define on each cell the possible outputs for θ. For instance, row 4 and column 3 corresponds to the rule $if\ (x \text{ is } CE)$ and $(\phi \text{ is } VE)$ then $(\theta \text{ is } ZE)$, highlighted in bold in Table 3.2.

This control system is implemented as a classic Mamdani model, in which the operator intersection (“min”) combines the antecedents of each rule and the operator union (“max”) generates the output set. The fuzzy sets for both inputs and output are graphically shown in Figure 3.12. Details about shapes and intervals for each fuzzy set are described in Table 3.3 and a more extensive discussion about their design can be found in [34].

As an example of iteration, the inputs $x = 68$ and $\phi = 113^o$ are used as current state of the virtual world to generate the output angle θ, which will be responsible to update

Table 3.2: Fuzzy rules for the inputs x and ϕ to produce a rotation angle θ

		x				
		LE	LC	CE	RC	RI
ϕ	RB	PS	PM	PM	PB	PB
	RU	NS	OS	PM	PB	PB
	RV	NM	NS	OS	PM	PB
	VE	NM	NM	**ZE**	PM	PM
	LV	NB	NM	NS	NS	PM
	LU	NB	NB	NM	NS	OS
	LB	NB	NB	NM	NM	NS

Table 3.3: Fuzzy sets, shapes and intervals defined for the fuzzy truck

Variable	Fuzzy Set	Shape	Interval
x	LE	Trapezoidal	[0 0 15 35]
	LC	Triangular	[10 40 50]
	CE	Triangular	[40 50 60]
	RC	Triangular	[50 60 90]
	RI	Trapezoidal	[65 85 100 100]
ϕ	RB	Triangular	[-90 -30 0]
	RU	Triangular	[-45 0 45]
	RV	Triangular	[0 60 90]
	VE	Triangular	[45 90 135]
	LV	Triangular	[90 120 180]
	LU	Triangular	[135 180 225]
	LB	Triangular	[180 210 270]
θ	NB	Triangular	[-30 -30 -15]
	NM	Triangular	[-25 -15 -5]
	NS	Triangular	[-15 -5 0]
	ZE	Triangular	[-5 0 5]
	PS	Triangular	[0 5 15]
	PM	Triangular	[5 15 25]
	PB	Triangular	[15 30 30]

the steering wheel position. Note that each input parameter enables two fuzzy sets with different degrees of membership, i.e. $x = 68$ activates RC and RI with degrees 0.7 and 0.2, respectively; and $\phi = 113^o$ activates VE and LV with values 0.5 and 0.9, respectively, as pointed out by the thin vertical arrows in Figure 3.12 (a) and (b). Consequently, four different rules, highlighted by shaded cells in Table 3.2, will be responsible to deliver the

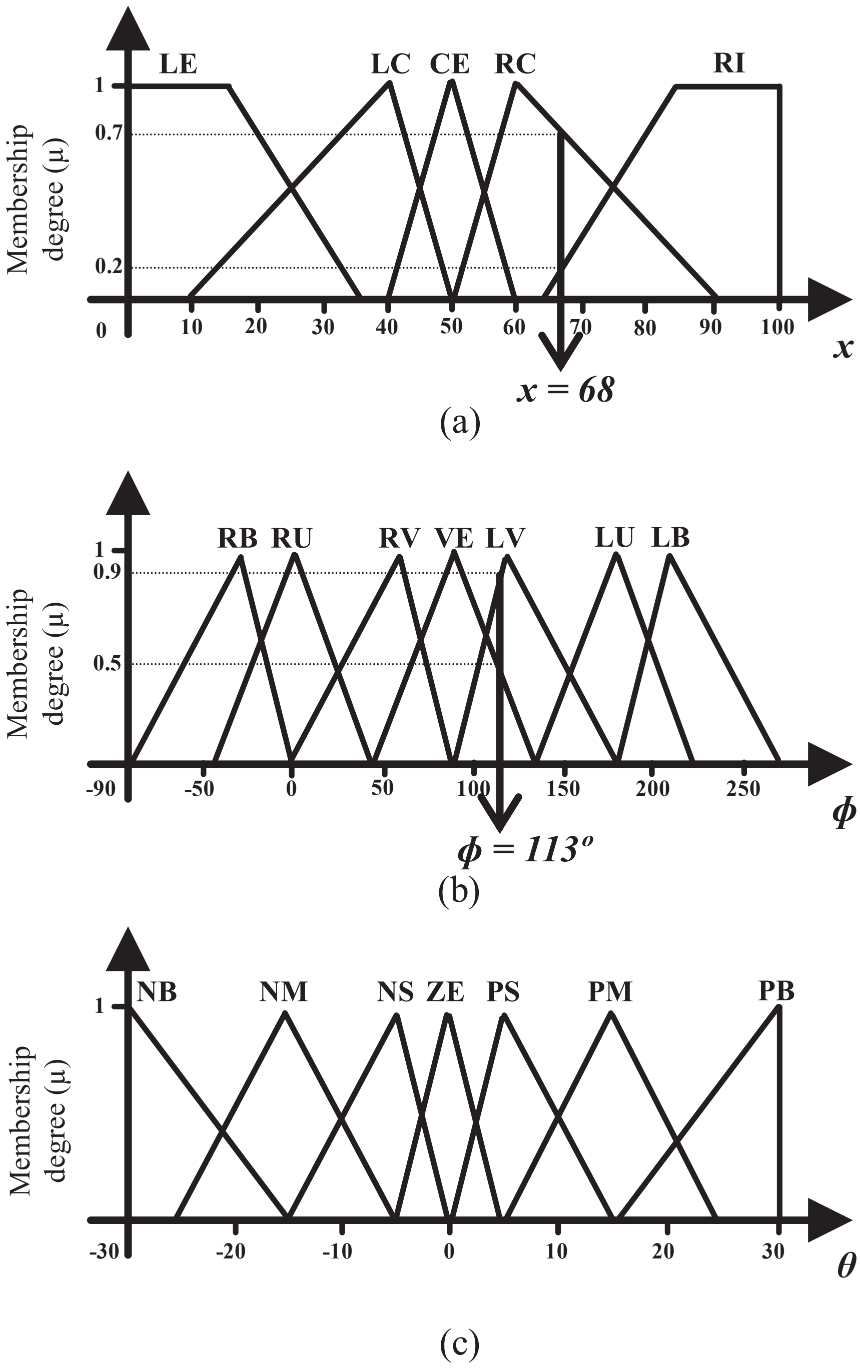


Figure 3.12: Fuzzy sets for the inputs x and ϕ, and the output θ

output θ, since they are a combination of the sets RI and RC with VE and LV.

Each rule has to be analysed, in which the operator intersection takes the minimum degree of membership between the two activated sets. Equation 3.1 illustrates this operation, in which a resultant set B^* is a combination of the input variables x and ϕ with their respective activated sets $A1$ and $A2$, over the set B of the output variable θ. Note that the resultant set B^* is not necessarily a specified fuzzy set, since it represents the combination of other sets.

$$\mu_{B^*}(\theta) = (\mu_{A1}(x) \wedge \mu_{A2}(\phi)) \wedge \mu_B(\theta) \tag{3.1}$$

Each activated rule deliver the following results:

$$\begin{aligned}
\mu_{PM^*}(\theta) &= (\mu_{RC}(x) \wedge \mu_{VE}(\phi)) \wedge \mu_B(\theta) = (0.7 \wedge 0.5) \wedge \mu_B(\theta) = 0.5 \wedge \mu_B(\theta) \\
\mu_{PS^*}(\theta) &= (\mu_{RC}(x) \wedge \mu_{LV}(\phi)) \wedge \mu_B(\theta) = (0.7 \wedge 0.9) \wedge \mu_B(\theta) = 0.7 \wedge \mu_B(\theta) \\
\mu_{PM^*}(\theta) &= (\mu_{RI}(x) \wedge \mu_{VE}(\phi)) \wedge \mu_B(\theta) = (0.2 \wedge 0.5) \wedge \mu_B(\theta) = 0.2 \wedge \mu_B(\theta) \\
\mu_{PM^*}(\theta) &= (\mu_{RI}(x) \wedge \mu_{LV}(\phi)) \wedge \mu_B(\theta) = (0.2 \wedge 0.9) \wedge \mu_B(\theta) = 0.2 \wedge \mu_B(\theta)
\end{aligned}$$

Subsequently, these rules are combined using the operator union, which takes a maximum value for each activated output set. Note that both PM and PS sets are activated for the output θ, but only the maximum one must be kept, as highlighted by the following equations:

$$\begin{aligned}
\mu_{PM^*}(\theta) &= 0.5 \wedge \mu_B(\theta) \\
\mu_{PS^*}(\theta) &= 0.7 \wedge \mu_B(\theta)
\end{aligned}$$

The interpretation of this example follows the same reasoning described in Figure 3.6 in section 3.4.3 for the Mamdani inference model. First, the inputs x and ϕ activate four rules and their respective fuzzy sets. The operator intersection selects a minimum degree of membership between the antecedents in order to generate a degree of membership for the consequent, i.e. for the rule if $(x$ is $RC)$ and $(\phi$ is $VE)$ then $(\theta$ is $PM)$ with

$\mu_{RC}(x) = 0.7$ and $\mu_{RC}(\phi) = 0.5$, the degree $\mu_{PM}(\theta) = 0.5$ is calculated as a partial output. This process is then repeated for each activated rule. Subsequently, the four calculated output sets are combined using the operator union, which aggregates fuzzy sets with the same label selecting its maximum degree of membership, i.e. three rules activate the same output set PM as $\mu_{PM}(\theta) = 0.5$, $\mu_{PM}(\theta) = 0.2$, $\mu_{PM}(\theta) = 0.2$ and the degree $\mu_{PM}(\theta) = 0.5$ is selected. Note that no aggregation was necessary for the remaining rule because only one degree of membership $\mu_{PS}(\theta) = 0.7$ is calculated for the output set PS. These results are graphically shown in Figure 3.13.

A final output θ is calculated transforming the obtained fuzzy area into a crisp number, in which the defuzzification method "centre of gravity" is applied. The output value $\theta = 9.7^o$ is then inferred, since it subdivides the obtained area into two equal parts, as highlighted in Figure 3.13.

New updated inputs are used at each step of the simulation, since the truck keeps moving towards its goal. Note that this rule-based fuzzy control system is responsible

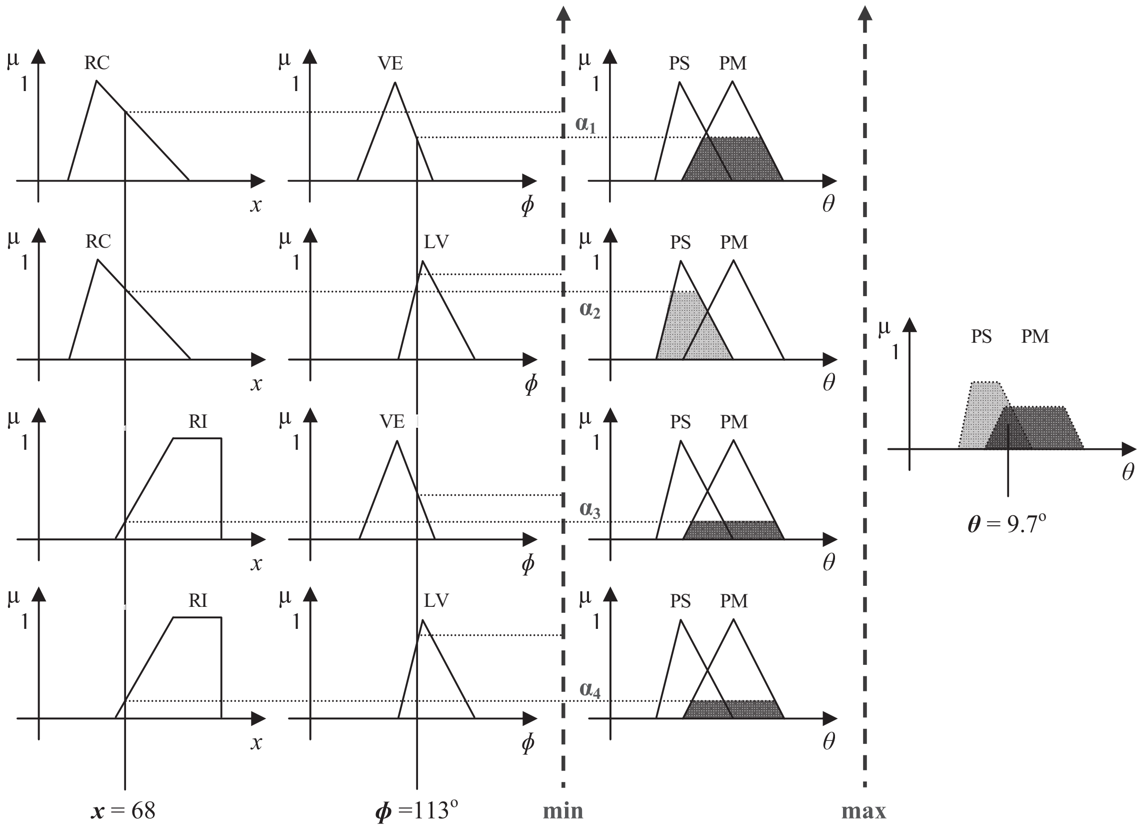


Figure 3.13: Inference procedure to calculate the output θ based on the inputs x and ϕ using a classic Mamdani model

for generating successive decisions to park a truck into a specified parking area. Consequently, this process has to be repeated until the moment that the goal is reached.

3.6 Fuzzy Scheduling

As previously mentioned in section 3.1, fuzzy logic concepts have been successfully applied to many industry contexts, providing a realistic way to model, control and optimise real world problems. Their effectiveness on managing uncertainties and flexibility on handling human thinking have been attracting the attention of many scheduling and rescheduling researchers [41, 64, 110].

Different sources of uncertainty are present in scheduling problems such as allocation changes, delay on raw material delivery, last minute absence of employees, changing on order details, order cancellations, new orders, machine breakdowns, unexpected maintenance, among others. For all these cases, it is necessary to create flexible optimisation models which are able to minimise or even absorb the negative effects of such disruptions. These problems has been mostly tackled by using fuzzy numbers to describe scheduling parameters and constraints, such as release and processing times [5, 45, 112], due-dates [43, 76, 77, 98], completion and setup times [55, 86], precedence constraints [46], among others [83, 102]. Additionally, decision support systems using fuzzy control allow managing uncertainties based on historical data or expertise, i.e. how to split jobs into smaller lots to guarantee customer satisfaction [85], how to combine dispatching rules [39], how to optimise family assignments to reduce setup times [54], and so on.

Figure 3.14 (a)-(c) shows the scheduling parameters release r_j, processing time p_j and due-date d_j of a generic job j using the fuzzy numbers $\tilde{r}_j$, $\tilde{p}_j$ and $\tilde{d}_j$, respectively. For this example, both release and processing times have a triangular shape because, hypothetically, changes often occur in raw material deliveries and maintenances are required during the processing of some operations. Consequently, triplets such as $\tilde{r}_j^1$, $\tilde{r}_j^2$ and $\tilde{r}_j^3$ transforms the crisp parameter r_j into a fuzzy one $\tilde{r}_j$, in which $\tilde{r}_j^1$ and $\tilde{r}_j^3$ set a time window for the release and $\tilde{r}_j^2$ is set as r_j, since its crisp value is the best representative of the original release time, i.e. $\mu_{\tilde{r}_j^2} = 1$. The same pattern is followed to define the fuzzy processing time $\tilde{p}_j$. Note that $\tilde{r}_j$ and $\tilde{p}_j$ will generate a fuzzy completion time $\tilde{C}_j$, as in

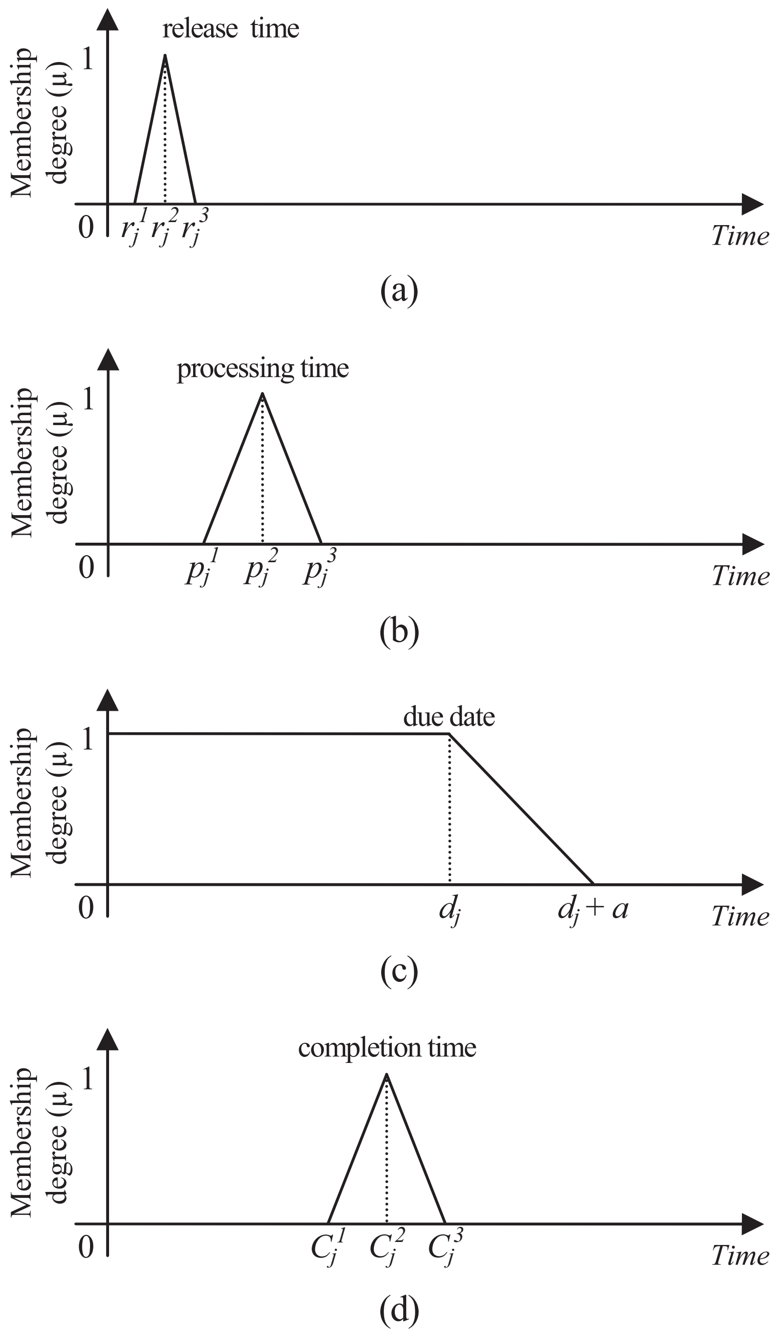


Figure 3.14: Fuzzy sets representing the scheduling parameters release (a), processing time (b), due-date (c) and completion time (d)

Figure 3.14 (d). On the other hand, the due-date $\tilde{d}_j$ have a trapezoidal shape, in which, hypothetically, the original d_j may be slightly extended without compromising the customer satisfaction, i.e. a job with lower urgency. The parameter "a" extends d_j and the membership grade linearly declines from 1 to 0, when the current time $\in [d_j, d_j + a]$, as illustrated in Figure 3.14 (c). There are several objective functions to evaluate the quality of the schedule and some of them are discussed in the following chapters of this thesis. As an example, the tardiness of the job j can be defined as a crisp number within the interval [0,1] when the intersection area of $\tilde{C}_j$ with $\tilde{d}_j$ is divided by $\tilde{C}_j$ [98]. Figure 3.15 illustrate the job j meeting its due-date d_j (a), partially meeting d_j (b) and when the job is considered to be tardy (c).

Figure 3.16 shows an example of a decision support system using fuzzy control described by [82]. The three inputs, time of occurrence TO, importance of efficiency EF and importance of stability ST, define the current state of the shop floor and the output determines the best rescheduling method Re_i to be applied, which is left shift rescheduling Re_1 or rebuild a new schedule from scratch Re_2. This problem is modelled as a modified Sugeno type, in which inputs are represented by fuzzy sets and the output is a crisp action

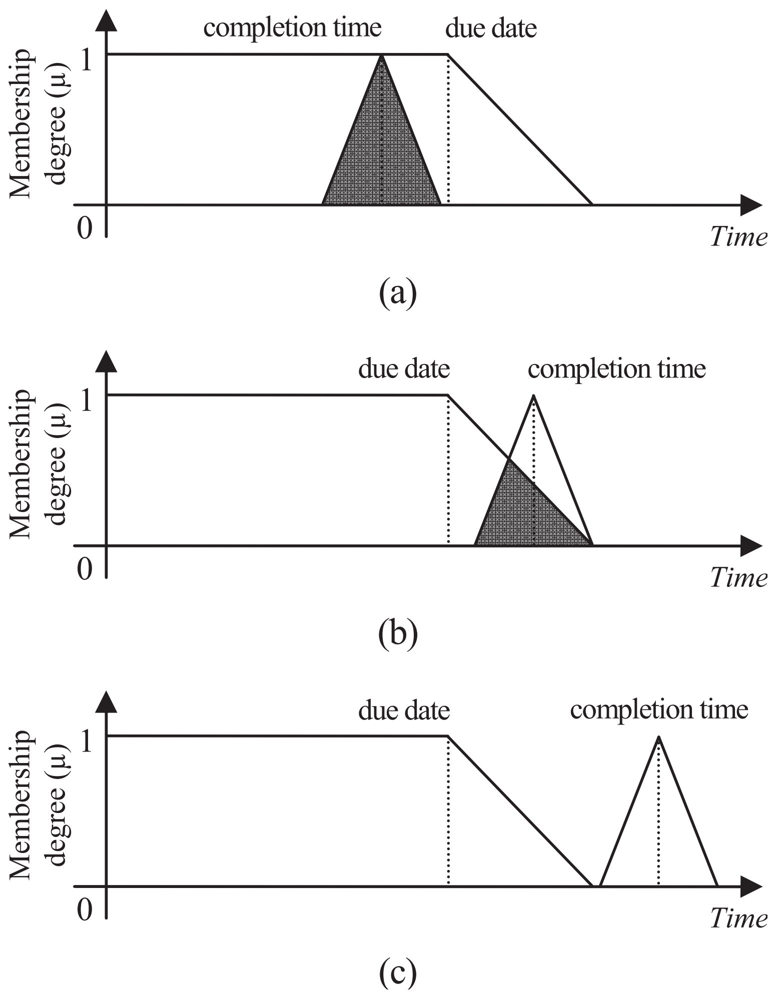


Figure 3.15: A generic job j meeting its due-date d_j (a), partially meeting d_j (b) and when it is tardy (c) as proposed by [98]

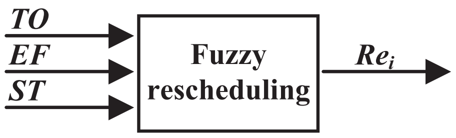


Figure 3.16: Fuzzy rescheduling with the inputs TO, EF, ST, and the output Re_i

with associated weights (more details can be found in [121]). Details about the inputs fuzzy sets are presented in Figure 3.17. The combination of these three inputs generates 12 fuzzy rules r_i, as described in Table 3.4. Note that all rules simultaneously activate both rescheduling methods R_1 and R_2 with their respective weights a_r and b_r, which are responsible to set their priorities. More details about tuning a_r and b_r are discussed in [82]. The main idea is to use Re_1 or Re_2 when the priority is stability or efficiency, respectively. The antecedents of the rules are combined using the operator "and" in which a degree of match is calculated as $\alpha_r = min(\mu_{TO}, \mu_{EF}, \mu_{ST})$. Additionally, activation rates are calculated for both rescheduling method as $\alpha_{R_1} = \sum_{r=1}^{12} a_r \alpha_r$ and $\alpha_{R_2} = \sum_{r=1}^{12} b_r \alpha_r$. A crisp decision is generated when the values α_{Re_1} and α_{Re_2} are compared, in which the larger one defines the rescheduling method to be applied. Re_1 is always preferred in cases of a tie.

Surprisingly, most of the literature on scheduling has been considering only static problems, in which the previously mentioned disruptions has not been extensively investigated. Fuzzy logic is an effective approach to manage various types of uncertainties, including the ones present in scheduling and rescheduling problems. The work present in this thesis make use the strengths presented by fuzzy logic concepts applied to a dynamic and complex real world job shop problem, in which uncertainties are often present in the

Table 3.4: Fuzzy rules r_i for the inputs TO, EF and ST to decide between the rescheduling methods left shift Re_1 or rebuild a new schedule from scratch Re_2

		Fuzzy rules r_i											
		r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}
if	TO	early	early	early	early	middle	middle	middle	middle	late	late	late	late
and	EF	low	low	high	high	low	low	high	high	low	low	high	high
and	ST	low	high	low	high	low	high	low	high	low	high	low	high
then		a_1Re_1	a_2Re_1	a_3Re_1	a_4Re_1	a_5Re_1	a_6Re_1	a_7Re_1	a_8Re_1	a_9Re_1	$a_{10}Re_1$	$a_{11}Re_1$	$a_{12}Re_1$
		b_1Re_2	b_2Re_2	b_3Re_2	b_4Re_2	b_5Re_2	b_6Re_2	b_7Re_2	b_8Re_2	b_9Re_2	$b_{10}Re_2$	$b_{11}Re_2$	$b_{12}Re_2$
with	a_r	0.3	1	0	0.5	0.4	1	0	0.5	1	1	0	1
	b_r	0.7	0	1	0.5	0.6	0	1	0.5	0	0	1	0

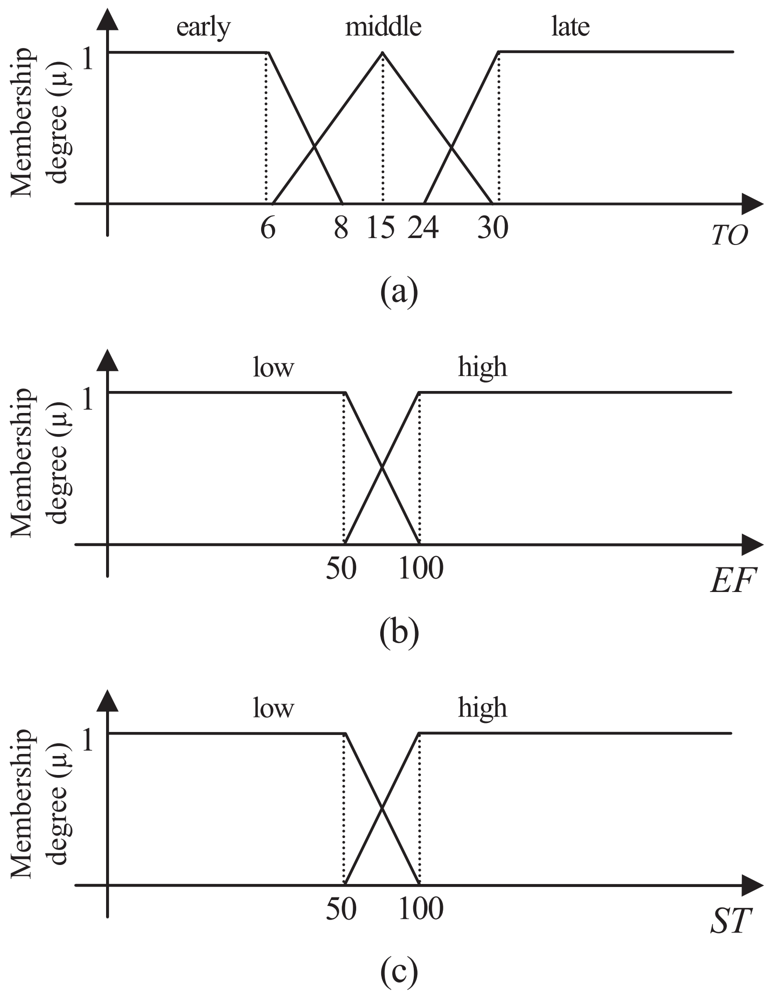


Figure 3.17: Fuzzy sets for the inputs TO, EF and ST

shop floor. The aim is to produce reliable schedules, combining a robust fuzzy scheduling system with match-up rescheduling algorithms when disruptions occur in the shop floor.

3.7 Discussion

Fuzzy logic concepts bring an effective approach to represent real world problems, since they are able to manage uncertainties and create more flexible optimisation models. The reasoning using fuzzy logic follows the same pattern of statements commonly used by human beings, in which linguistic and vague terms are always present. The model design is quick and few adjustments are expected, since rules are able to embed the knowledge provided by experts.

The design of a fuzzy system to park a truck in a virtual area confirms the strengths previously mentioned. The vagueness present in the problem parameters, such as positions and angles of the truck, where easily represented by fuzzy sets and the designer expertise allowed the definition of good representative rules to produce the expected control actions.

Several industry contexts have been successfully using fuzzy logic systems, including home appliances, public transports, safety systems, among others. This effectiveness has extending their application to academic areas, including research on scheduling and rescheduling. The uncertainties in this areas can be mostly tacked using fuzzy numbers to represent problem parameters and constraints, and fuzzy decision support systems to generate control actions based on historical data or expertise.

Unfortunately, most of the literature on scheduling has been considering only static problems and commonly present disruptions such as allocation changes, absences of employees, among others, has not been extensively investigated. The work present in this thesis aim to use the strengths presented by fuzzy logic concepts applied to a dynamic and complex real world job shop problem, in which uncertainties are often present in the shop floor.

3.8 Summary

This chapter presents an introduction to fuzzy systems, which represents an effective strategy to manage uncertainties that are always present in real world problems. Their essential concepts are described and an example is discussed in order to illustrate how to create them step-by-step and also to identify the flow of information within these systems. A classic problem of parking a truck using a Mamdani fuzzy control system is discussed, in which the vagueness on the problem parameters are easily tackled by using fuzzy sets and fuzzy rules.

The model design is quick and easy to understand, since the reasoning on fuzzy logic follows the same pattern of human thinking, in which vague and linguistic terms are always present in decision making, having words such as “low”, “high”, among others. Additionally, these systems usually require few adjustments on their design because their rules are capable to embed the knowledge provided by experts and/or information provided by historical data.

This thesis aims to use the strengths presented by fuzzy logic concepts applied to a dynamic and complex real world job shop problem, in which uncertainties are often present in the shop floor. The following chapter describes this problem and the application of fuzzy numbers to manage scheduling variables. Additionally, fuzzy scheduling systems are presented in Chapter 6, in which their combination with mach-up algorithms are able to produce reliable solutions.

Chapter 4

Match-up Strategies for a Complex Real World Job Shop Problem

4.1 Introduction

This chapter investigates the problem of inserting newly arriving jobs into an existing schedule of a real world manufacturer. These type of disruption occurs on a daily basis and requires rescheduling. A number of match-up strategies, which collect the idle time on machines of a current schedule for the insertion of new jobs, are proposed. Their aim is to obtain new schedules with a good performance which are at the same time highly stable, meaning that they resemble as closely as possible to the initial schedule and avoid additional production costs. Other rescheduling strategies such as “total rescheduling”, “right shift” and “insertion in the end” deliver either good performance or stability, but not both. Contrary, experimentations and statistical analysis reveal that the proposed match-up strategies deliver high performing schedules with a high stability, validating hypothesis 1 from Chapter 1.

This chapter is concerned with the scheduling/rescheduling problem presented by Sherwood Press - Nottingham, UK, which is a job shop problem with parallel machines, machine eligibility and sequence dependent setup times. More details of these features are described in the following section. The problem is dynamic since new jobs with different levels of urgency arrive everyday in the shop floor and they have to be integrated into the existent schedule. Typical arriving jobs are rush orders, which means that they have to be processed as early as possible on the current schedule. This type of disruption is tackled first and the goal is to find appropriate rescheduling approaches to achieve high quality schedules. Additionally, orders with different levels of urgency are investigated in the following chapter in order to check the flexibility of these approaches under various scenarios.

There are two important criteria to consider when evaluating a rescheduling strategy: (1) the *Performance* of the resultant schedule, which is measured with the same objective functions used to evaluate the initial schedule and (2) the *Stability* of the resultant schedule, which refers to how closely the new schedule resembles the initial one. Match-up algorithms are concerned with both of these criteria, and are therefore appropriate for a large variety of rescheduling problems, including the one present in Sherwood Press.

Match-up algorithms aim to maintain both *Performance* and *Stability* by modifying only a part of the initial schedule when a disruption occurs. Their motivation is that once having an initial optimum schedule the best is to return to such optimum schedule as quickly as possible after repairing it. In other words, the idea is to "match-up" the disturbed schedule to the initial one, as quickly as possible. This goal is achieved by only modifying the schedule within a defined rescheduling time window, keeping unchanged the schedule before and after this interval. The research presented in this thesis and the resultant papers in [69, 70, 73, 74] describe the only applications of match-up algorithms to a complex real world job shop problem, which includes multiple criteria, setup times, parallel machines and disruptions affecting multiple resources.

The remaining of this chapter is organised as follows. Section 4.2 introduces the problem present in Sherwood Press and formally defines the *Performance* and *Stability* measures. Section 4.3 describes the match-up algorithm for rush orders. Section 4.4 presents the problem instances used to test the proposed algorithms, presents the results of the experimentation including the adequate statistical tests, and gives an analysis of the problem parameters that have an effect on algorithm behaviour. Sections 4.5 and 4.6 conclude this chapter.

4.2 Problem Statement

The job shop scheduling problem in Sherwood Press requires the allocation of a variable number of jobs onto 18 machines, which are grouped into 7 work centres for printing, cutting, embossing / debossing, folding, card-inserting, gathering and finishing. Some printing machines are identical and are treated as parallel machines. Each job $j = 1, \ldots, n$ is subject to precedence constraints, meaning that it has to visit the required machines following a predefined order. Possible routes on machines are shown in Figure 4.1. However, jobs are mostly processed by 3 to 5 machines in the shop floor and they follow one of the routes given in Figure 4.2, as described in [87]. Each job j has a release time r_j and a due date d_j which are the earliest time when job j can start its processing and the time when job j is required to be completed, respectively. The processing of job j on machine $i = 1, \ldots, 18$ is referred to as operation o_{ij} and each operation requires a certain amount of processing time p_{ij}. Each job has also a priority w_j which indicates how strict its due date is, i.e. jobs with $w_j = 1$ must be completed by their due date d_j, jobs with $w_j = 2$ are given two days tolerance period after d_j and jobs with $w_j = 3$ are given up to one week of tolerance. Each job j has a family f which identifies its colouring requirements. Setup times are then considered when operations requiring different colours are processed one after the other on printing machines.

The static version of the investigated problem is known as the job shop problem with parallel machines, release times, job weights and sequence dependent setup times. The scheduling problem of Sherwood Press, however, is not static but dynamic; every day a

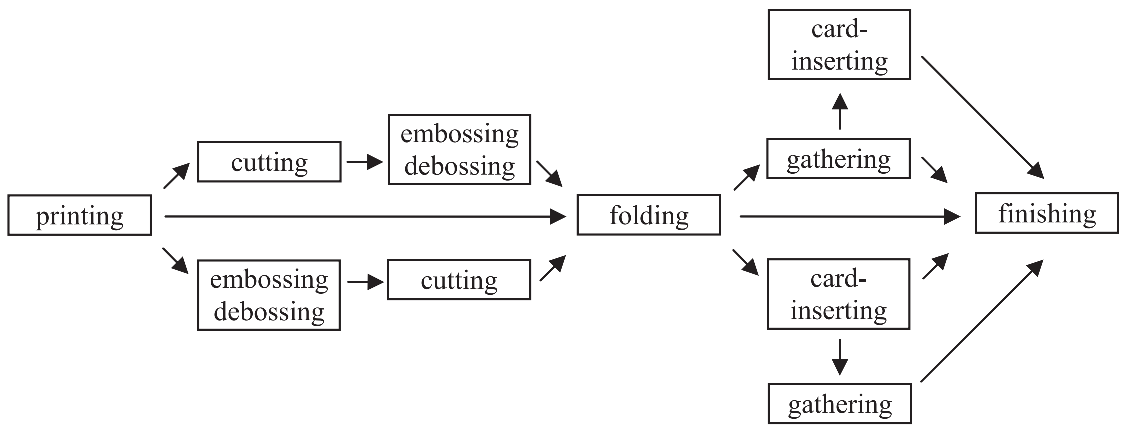


Figure 4.1: Possible machine routing for jobs

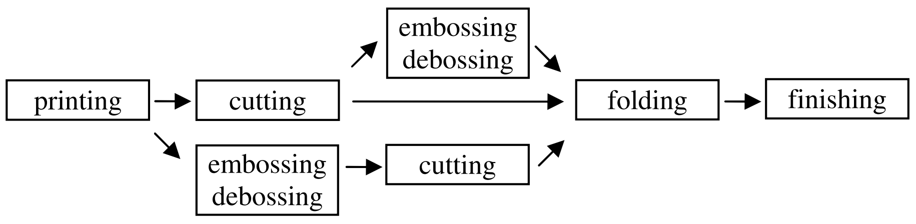


Figure 4.2: Typical machine routing for jobs

number of new jobs arrive in the system and these have to be incorporated into the existing schedule. These jobs are classified as complex disruptions because multiple resources are usually affected, i.e. changing the allocation of operations on multiple machines. Newly arriving jobs are of two types: “rush” orders, which have a high priority and must be inserted as early as possible, and “normal” jobs, which have the same priority as most of the jobs. “Rush” orders are the most common disruption present in this job shop scheduling problem and they are investigated first.

Rescheduling algorithms must produce schedules that include the newly arrived jobs and are of a good quality with regards to the $Performance$ function and be as similar as possible to the initial schedule, refereed here as *Stability*. The rescheduling problem is, then, a bi-objective problem in which *Performance* and *Stability* have to be maximised simultaneously. Both measures are formally introduced next.

4.2.1 Performance

The *Performance* measure considers five scheduling objective functions that have been previously applied to Sherwood Press problem [32,84,85]: the average weighted tardiness of jobs, the number of tardy jobs, the total setup time, the total idle time of machines and the total flow time. Given a schedule, each of these functions is evaluated and mapped into a satisfaction grade within the $[0,1]$ range. The $Performance$ measure is the average of the five satisfaction grades.

It is important to highlight that this multiple criteria decision making was originally introduced by Fayad and Petrovic [32] using a genetic algorithm for the static version of the problem presented by Sherwood Press. This thesis is a build on this work, in which disruptions are now taken into account. As a matter of providing a self-content package, a short description of satisfaction grades is presented next, while more details can be found in [86]. Additionally, a detailed discussion of multi-objective scheduling using GA is described by Bagchi in [9].

Satisfaction grades SG_i, $i = 1, \ldots, 5$, are applied to this job shop scheduling problem because of two main reasons. First, they allow to handle simultaneously objective functions that are measured in different units. Second, they enable the production manager to express his/her preferences with respect to the objective functions by assigning weights to the different objectives.

SG_1 - Average Weighted Tardiness

In order to address the uncertainties inherent in real world scheduling, the processing times of jobs and due dates were modelled using fuzzy numbers. A crisp number is mapped into a fuzzy number through a membership function. Note that the use of a fuzzy processing time lead to a fuzzy completion time. Figure 4.3 (a) and (b) show the membership functions for the processing time p_j and a intersection between the due date d_j and completion time C_j of job j, respectively.

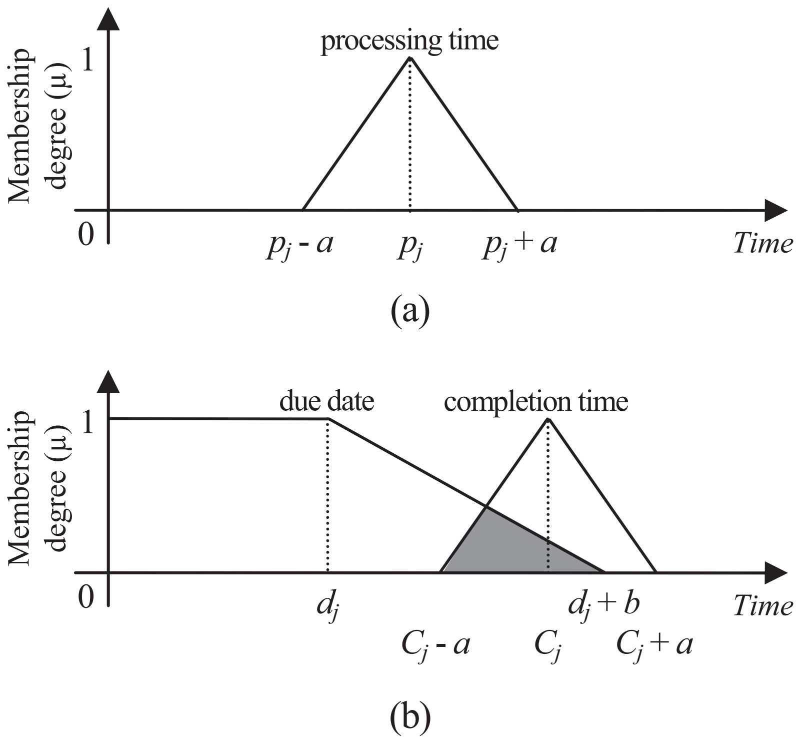


Figure 4.3: Example of (a) membership functions for the processing time p_j and (b) a intersection between the due date d_j and completion time C_j of job j

An interval between $p_j - a$ and $p_j + a$ represent the p_j uncertainties, when the processing time is either early or late, respectively. The membership degree for p_j is 1 when the original crisp processing time is executed, and it declines linearly to 0 within the interval for both $p_j - a$ and $p_j + a$. The same pattern is followed to set the completion time C_j. Similarly, the uncertainties of d_j are set using $d_j + b$, which determines the flexibility of a job on meeting its due date d_j. Note that his flexibility is determined by the parameter priority w_j previously defined in section 4.2. For instance, the membership degree for d_j is 1 when the completion time of job $j \in [0, d_j]$, and it declines linearly to 0 when the completion time is increasing within the interval $[d_j, d_j + b]$. Both parameters "*a*" and "*b*"are specified by the production manager. In this work "*a*" is 10% of the original processing time p_j, and "*b*" may take the value 0, 2 or 5 working days, depending on how urgent a job is. In other words, his priority w_j.

A triangular membership function is used to represent both processing and completion times because although a job is in theory executed and completed at p_j and C_j, respectively, in practice it may be done within a time period "*a*" either before or after these values. For the due date of jobs, the membership function is trapezoidal meaning that jobs are desired to be completed between time 0 and d_j, but there is a time window after d_j when jobs are still considered to be on time. Completions after $d_j + b$ are considered late and the membership degree is 0.

A satisfaction grade on the job's completion time is calculated to identify the tardiness of a job j. This grade is obtained as the area described by the intersection of the due date and completion time membership functions divided by the area described by the membership function of the completion time [98], as previously seen in Chapter 3. In the example from Figure 4.3, the satisfaction grade is calculated as the size of the shaded area divided by the area of the triangle labelled "completion time". The satisfaction grade of the average weighted tardiness is calculated as the average of the weighted satisfaction grades of the completion times of all jobs.

A hypothetical resultant schedule is shown in Figure 4.4, in which 11 jobs are allocated on available machines. Jobs are graphically represented by rectangles, and rectangles with the same number mean that a job has more than one operation, i.e. job 3 have 2 operations represented by the rectangles on M3 and M4. Additionally, jobs are subject

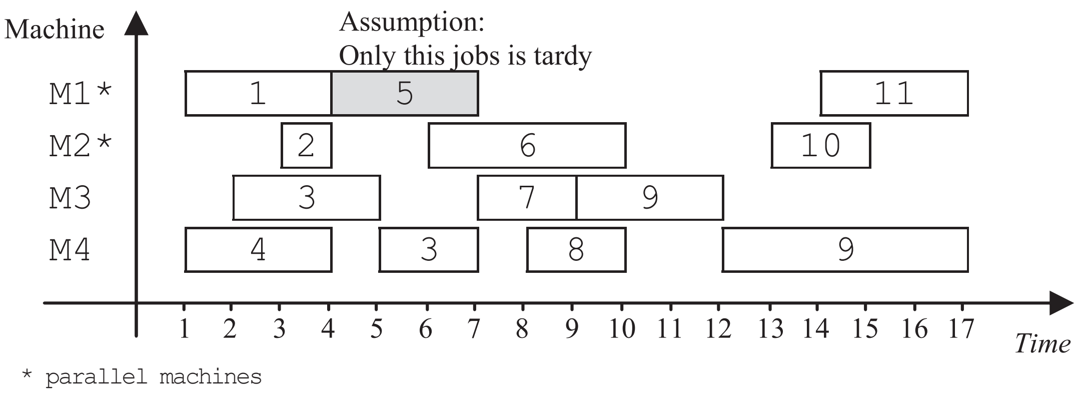


Figure 4.4: Example of a resultant schedule with 11 jobs, considering that only job 5 is tardy

to precedence constraints, in which a sequence of operations is predefined when a job j has to be processed on more than one machine. Note that each operation of a job j has to be completed before the next one can be started. For instance, job 9 has to be processed first on M3 and when the execution is completed its next operation can be started on M4. Parallel machines are represented by M1 and M2. Suppose that only job 5 is tardy, i.e. job 5 has a fuzzy completion time C_5 within the interval $[5,9]$ and its due date $d_5 = 2$ has a tolerance "a" of 5 time units, as in Figure 4.5. A point k between these two fuzzy sets is then calculated as an intersection of two lines, i.e. $k = (5.57, 0.28)$. Subsequently, the correspondent areas for the shaded triangle and the completion time membership function are 0.28 and 2, respectively. The resultant satisfaction grade for job 5 is 0.14, as the shaded area is divided by completion time area. Since the other 10 non-tardy jobs have 1 as their satisfaction grade, the final average for weighted tardiness of jobs is $SG_1 = 0.92$. Note that for this hypothetical example all jobs have their satisfaction grade with weight 1, which means that they are equally important.

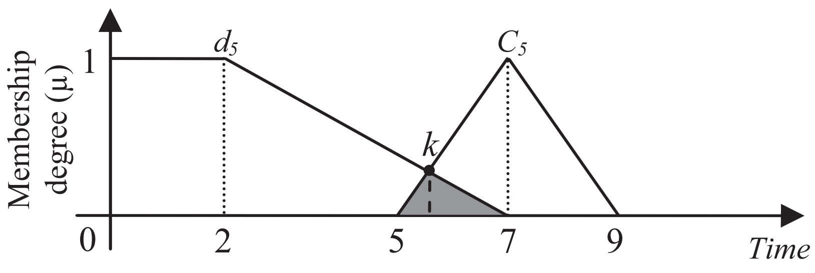


Figure 4.5: Intersection area between the fuzzy sets for completion time C_5 and the due date d_5 of job 5

SG_2 **- Number of Tardy Jobs**

A job j is considered to be tardy when its satisfaction grade for tardiness does not exceed a certain threshold λ. All jobs not achieving λ are counted. After investigating several values, Fayad and Petrovic predefined $\lambda = 0.3$ [32]. Consequently, job 5 previously presented is counted as tardy because its tardiness satisfaction grade 0.14 is smaller then the predefined λ. Note that again, job 5 is the only one considered to be tardy on this hypothetical example.

The final satisfaction grade for number of tardy jobs SG_2 is calculated from a decreasing linear function, following the production manager preferences regarding a maximum number α of jobs allowed to be tardy. For instance, the schedule in Figure 4.4 has an allowance $\alpha = 20\%$, which means that 20% of all 11 jobs are accepted to be tardy, i.e. a threshold of 2.2 is defined as shown in Figure 4.6. A maximum satisfaction of 1 is obtained when none of the jobs are tardy, and a minimum of 0 occurs when the number of tardy jobs is equal or exceeds this threshold value. Since only job 5 is tardy, $SG_2 = 0.45$ is obtained from the delimitated linear function.

SG_3 **- Total Setup Time**

Jobs having different colouring families may require setup time between their operations on printing machines. Additionally, these machines are initially cleaned before starting the processing of any job j. Suppose that M1 and M2 from Figure 4.7 are printing machines and jobs $j = 1, \ldots, 10$ belong to the same colouring family. Therefore, job 11 is the only one which needs a setup before its operation can be processed on M1. Setup times are highlighted by shaded cells labelled by "S" in Figure 4.7. Note that this resul-

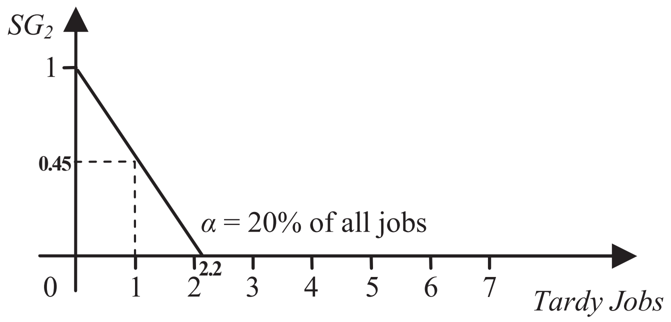


Figure 4.6: Example of decreasing linear function to calculate SG_2

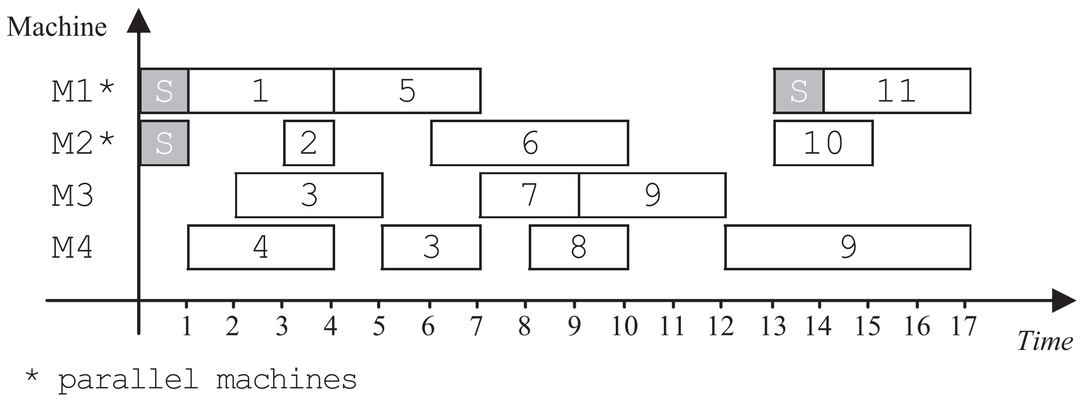


Figure 4.7: Resultant schedule with 11 jobs and the required setup times highlighted by shaded cells labelled by "S" on the printing machines M1 and M2

tant schedule is the same one presented in Figure 4.4. A final satisfaction grade SG_3 is obtained comparing the total setup time with a maximum value for setups, the latter referring to the situation when all operations on printing machines require setup times before their processing. These values are mapped following the same idea of the decreasing linear function presented for SG_2, in which a maximum satisfaction is obtained when none setups are required, and a minimum value when all operations demand setups. Consequently, $SG_3 = 0.5$ is obtained for the schedule, since a maximum of 6 setup times is set when all operations on printing machines require setup, while only 3 setups are required, i.e. the processing of job 11 added to the initial cleaning of M1 and M2.

SG_4 - **Total Idle Time**

Idle times are present on machines when no jobs are being processed or setups are performed. The blank spaces between operations and setups in Figure 4.4 represent idle times occurring in the analysed schedule. The completion time of the last operation on each machine defines a reference point to calculate idle periods. The sum of these completion times sets a maximum value for possible idle times. A decreasing linear function maps SG_4 within the interval $[0,1]$, in which a maximum satisfaction is obtained when no idle times are present, and a minimum value when all machines are idle. For instance, M2, in Figure 4.4, have a total processing time, setup and idle of 7, 1 and 7 time units, respectively. For all machines, the sum of completion times sets a maximum idle time of 63 units and an idle period of 22 time units, which results in $SG_4 = 0.34$.

SG_5 - Total Flow Time

The flow time measures how long a job j remains in the shop floor until its completion, i.e. time window between its release r_j and completion time C_j. The total flow time aggregates these measures for all jobs. The decreasing linear function for SG_5, sets a maximum satisfaction when flow time is equal to 0 present and minimum when all jobs complete their operations at C_{max}. Note that flow time equals 0 means that all jobs are being cancelled. Suppose that jobs 1-7 and 8-11 from Figure 4.4 have release time at 0 and 8 times units, respectively. Therefore, $SG_5 = 0.45$, since the total and the maximum flow times are $\sum_{j=1}^{11} C_j - r_j = 70$ and $\sum_{j=1}^{11} C_{max} - r_j = 155$, respectively.

Once the satisfaction grades $SG_1 - SG_5$ have been calculated, an overall *Performance* of the schedule is defined as:

$$Performance = \sum_{i=1}^{5} SG_i/5. \tag{4.1}$$

Consequently, the resultant schedule from Figure 4.4 delivers $Performance = 0.53$.

SG_{Make} - Makespan

Alternatively, the *Performance* measure may consider only one specific scheduling objective function depending on the rescheduling process. The makespan, previously defined as C_{max}, is a criterion commonly used for job shop scheduling problems [1, 90, 106, 116]. The production manager sets a threshold ψ of maximum acceptable makespan. Following the same idea presented for satisfaction grades $SG_2 - SG_5$, a decreasing linear function sets a maximum satisfaction when the makespan is *null* and a minimum when the threshold ψ is achieved. Note that $C_{max} = null$ is just a reference point, meaning that all jobs have been cancelled in the shop floor. Suppose that $\psi = 33$ for the schedule in Figure 4.4. Therefore, $SG_{Make} = 0.51$, because the completion time of both jobs 9 and 11 leads to $C_{max} = 17$. This alternative *Performance* measure is defined as:

$$Performance = SG_{Make} = C_{max}/\psi. \tag{4.2}$$

Details about preferences and decisions for appropriate scheduling and rescheduling objective functions are formally discussed in section 4.4 and subsection 4.4.3.

4.2.2 Stability

The *Stability* of a new schedule is measured with respect to an initial schedule using two components, Sta_1 and Sta_2, adapted from [1] and [95], respectively, to consider parallel machines.

Sta_1 - Sequence Deviation

The first *Stability* measure, Sta_1, considers changes to the relative order of operations in the initial and new schedule sequences. Let M be the number of machines in the shop floor and let O_i be the number of operations that have to be processed on machine $i = 1,\dots,M$. The following measure of sequence similarity $Reward_i \in [0,1]$ is assigned to each machine i:

$$Reward_i = \sum_{j=1}^{O_i-1} \frac{Reward_{ij}}{O_i - 1} \text{ , where}$$

$$Reward_{ij} = \begin{cases} 1 & \text{if operation } j+1 \text{ remains successor} \\ & \text{of operation } j \text{ on machine } i; \\ 0 & \text{otherwise.} \end{cases}$$

There are certain situations in parallel machine environments that require special treatment in the calculation of the $Reward_i$ value. For instance, the case in which a machine is in operation in the initial schedule and it becomes idle in the new schedule and the case in which a machine is idle in the initial schedule and it has to process any number of jobs in the new one, should be heavily penalised. On the contrary, the cases in which a machine is idle in the initial schedule and it remains idle in the new one and when a machine has assigned only one operation in the initial schedule and any number N in the new one, have to be highly rewarded. Therefore, the following four cases R1-R4 are considered in the calculation of $Reward_i$:

R1 If machine i is empty and stays empty after rescheduling, then $Reward_i = 1$;

R2 If machine i has originally only one operation and in the new schedule any other number N, then $Reward_i = 1$;

R3 If machine i is empty and any number of operations are assigned to it in the new schedule, then $Reward_i = 0$;

R4 If machine i has N operations and becomes empty in the new schedule, then $Reward_i = 0$.

In order to keep the sequence of operations on each machine as unchanged as possible, the sum of the rewards of the machines has to be maximised. This first *Stability* measure is defined as:

$$Sta_1 = \frac{1}{M}\sum_{i=1}^{M} Reward_i. \tag{4.3}$$

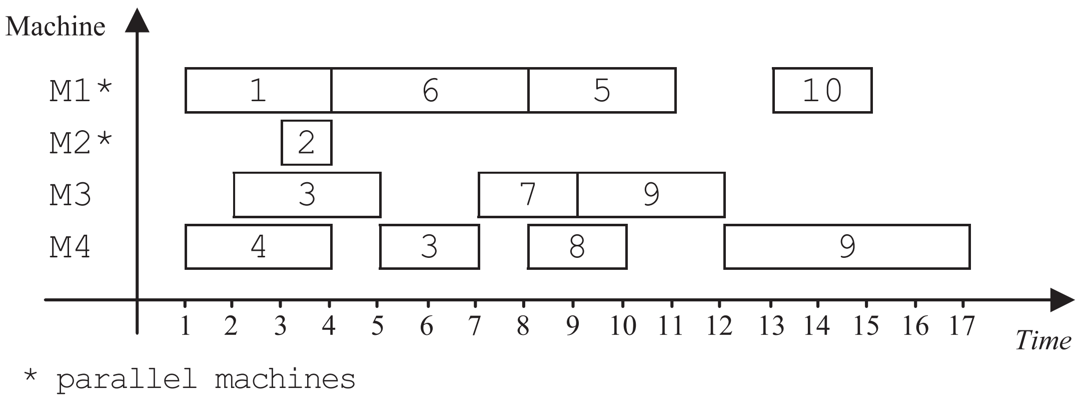
Suppose that the initial schedule present in Figure 4.8 (a) has a new job 11 to be inserted. The resultant schedule is shown in Figure 4.8 (b). Note that jobs 6 and 10 were initially allocated to M1, but they swap to the parallel machine M2 after the insertion of this new job. $Reward_1 = 0$, because no successor operation $j+1$ remains the same for operation j on M1, i.e. operations 6, 5 and 10 are not successors anymore for 1, 6 and 5, respectively. Operation 5 is not considered to remain successor of operation 1 on M1 because only the immediate successor of each operation is considered.

M2 was originally with only one operation and the processing of jobs 6 and 10 does not affect its original sequence, leading to $Reward_2 = 1$, as described in R2. Additionally, no changes were made on both machines M3 and M4, which got $Reward_3 = 1$ and $Reward_4 = 1$, respectively. The final measure for the sequence deviation of the resultant schedule present in Figure 4.4 is then $Sta_1 = 0.75$

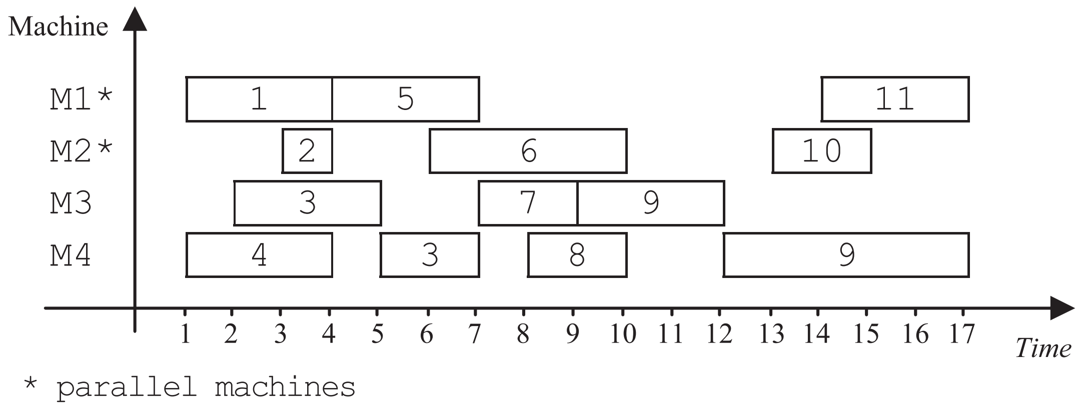
Sta_1 was defined at first as the main *Stability* measure for rescheduling because of the nature of the job shop problem presented by Sherwood Press, which has sequence dependent setup times. Afterwards, an additional measure was required, Sta_2, because time deviations could cause problems regarding raw material availability and/or personnel allocation for operating machines.

Sta_2 - Time Deviation

The second *Stability* component, Sta_2, considers the starting time deviation of operations in the initial and the new schedule. Let $start_j$ and $start'_j$ be the starting time of job j in the



(a)



(b)

Figure 4.8: Example of (a) an initial schedule with 10 jobs and its (b) resultant schedule after inserting a new job 11

initial and in the new schedule, respectively. Therefore, $TS_j = \max\left\{0, 1 - \frac{|start_j - start'_j|}{C_{max}}\right\}$, $TS_j \in [0,1]$ measures the starting time deviation for each job j. A maximum satisfaction is obtained when no time deviation is present between those starting times, and a minimum value when the absolute difference in starting times equals the length of the initial schedule, i.e. its makespan C_{max}. Note that TS_J defines a similar decreasing linear function as described for $SG_2 - SG_5$ and SG_{Make}. The only difference is that all jobs $j = 1, \ldots, N$ are individually evaluated and, subsequently, their TS_j values are combined into Sta_2 as:

$$Sta_2 = \frac{1}{N} \sum_{j=1}^{N} TS_j. \tag{4.4}$$

Once Sta_1 and Sta_2 have been calculated, an overall *Stability* measure is can be calculated as:

$$Stability = \frac{1}{2}(Sta_1 + Sta_2). \tag{4.5}$$

The resultant schedule present in Figure 4.4 has starting time deviations only for jobs 5 and 6, which are 4 and 2 time units, respectively. Therefore, $TS_5 = 0.76$ and $TS_6 = 0.88$ are obtained, as $TS_5 = \max\left\{0, 1 - \frac{|8-4|}{17}\right\}$ and $TS_6 = \max\left\{0, 1 - \frac{|4-6|}{17}\right\}$ with $C_{max} = 17$. No time deviations for the other jobs lead to a maximum satisfaction $TS_j = 1$. Note that job 10 has changed from machine M1 to M2, but its initial starting time is kept, which also results in $TS_{10} = 1$. Consequently, the final measure for the time deviation of the resultant schedule is $Sta_2 = 0.96$, which leads to an overall *Stability* of the schedule to be 0.79 as a result of the following calculation $(\frac{1}{2}(0.75 + 0.96))$.

4.3 Match-up Strategies for Rush Orders

This section introduces match-up strategies for the dynamic scheduling of rush orders. More details about inserting jobs with different levels of urgency are formally discussed in the following chapter.

Rush orders arrive everyday in the shop floor and they must be integrated into the current schedule as early as possible in order to achieve a good customer satisfaction. The pseudocode of the proposed match-up algorithm for rush orders is given in Figure 4.9. This match-up algorithm has three phases which are firstly outlined followed by a detailed description. In the first phase, steps 1–2, the rescheduling horizon, within which the operations of the new job will be accommodated, is defined. In the second phase, steps 3–6, a new scheduling problem containing operations within the calculated horizon and the new job are defined and solved. In the third phase, step 7, the newly generated schedule is integrated into the original one, checking and repairing possible overlaps between the unchanged part of the schedule and the newly generated partial schedule. The proposed algorithm considers for rescheduling one job at a time. If two or more jobs arrive simultaneously, priority is given to jobs with earlier due dates. If the due dates are the same then the order of jobs is randomly decided.

In step 1 of the algorithm, *initialStart* denotes the time of the arrival of the new job

```
Input: An initial schedule S, a new job j, initialStart
Output: A new schedule with job j integrated

1. Let startPoint be the latest completion time among operations whose processing time is
   crossed by initialStart
2. Calculate endpoint by collecting idle time on the machines required by the new job
3. Let O be the set of operations within startPoint and endPoint, plus the operations of job j
4. Update the release and due dates of jobs in O so that they lie within startPoint and endPoint
5. Let operations in O define a new scheduling problem S'
6. Solve S' using a genetic algorithm
7. Integrate S' into S, checking and removing overlaps
```

Figure 4.9: Pseudocode of the match-up algorithm for rush orders

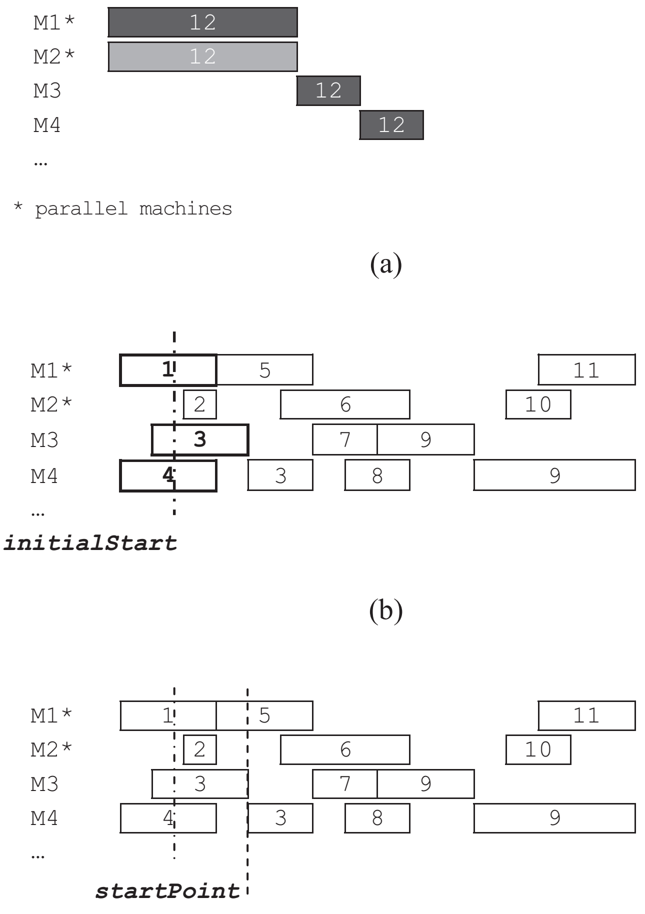


Figure 4.10: Example of a rush order arriving in the shop floor; (a) the new job processing requirements, (b) the calculation of *initialStart* and (c) the *startPoint* definition.

j increased by 48 hours, as required by Sherwood Press setting no changes for this time period in the shop floor.

For example, Figure 4.10(a) shows, using a Gantt chart, the processing requirements and precedence constraints defined for a new job 12. The first operation of this job has to be processed either on machine M1 or machine M2, which are parallel machines. After completing this execution, the next operation can be started on M3 and, subsequently, the last one is processed on M4. The operations of the initial schedule that have started their processing before *initialStart* must be completed before the rescheduling process begins. Therefore, all the operations that are being processed at *initialStart* are collected. These are the operations in Figure 4.10(b) that cross the *initialStart* line, i.e. jobs 1, 3 and 4 on machines M1, M3 and M4, respectively. The operation with the highest completion time among the collected operations determines the *startPoint*. In Figure 4.10(c), this is job 3 on M3. All the operations that can be completed before *startPoint* will resume their processing in this example that is job 2 on M2.

In order to calculate *endPoint* as in step 2, the algorithm collects idle times on the machines that are required by the new job. Four different strategies FW1– FW4 collecting idle times in a forward way are introduced, as illustrated with an example in Figure 4.11. Suppose that a job consisting of two operations arrives in the shop floor. The new job requires 4 and 2 processing time units on machines MA and MB, respectively, as in Figure 4.11(a). Four different strategies are introduced to collect idle times on machines:

FW1 Collect idle time on the required machines accumulating enough time, not necessarily as a single time window, until the collected idle time equals the new job processing requirements, as in Figure 4.11(b).

FW2 Collect idle time contained in a single time window, as in Figure 4.11(c).

FW3 As FW1, but considering the precedence constraints imposed on the operations of the new job. Figure 4.11 (d) shows idle time being collected on machines MA and MB. Note that the collection of time for the second operation on machine MB can start only after the required idle times for the first operation has been collected on machine MA.

FW4 As FW2, but considering the precedence constraints of the new job, as demonstrated in Figure 4.11(e).

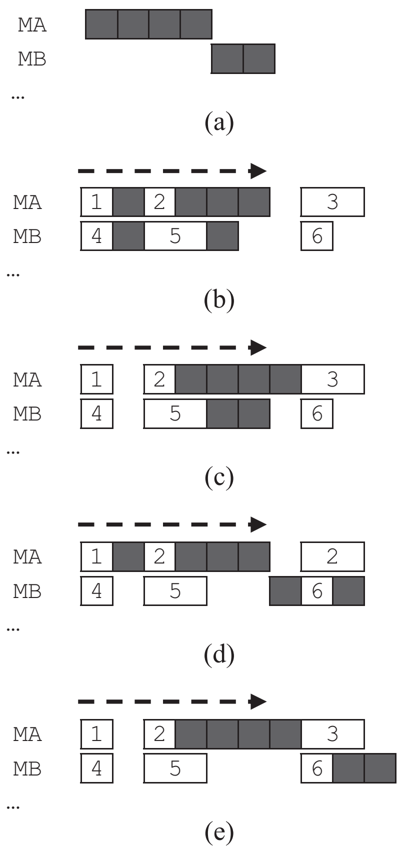
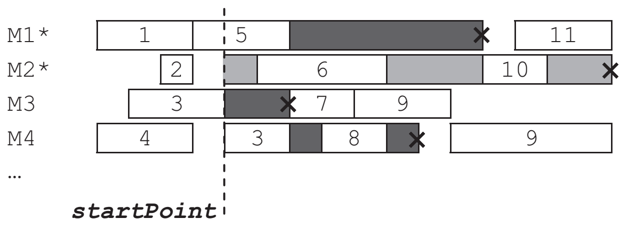
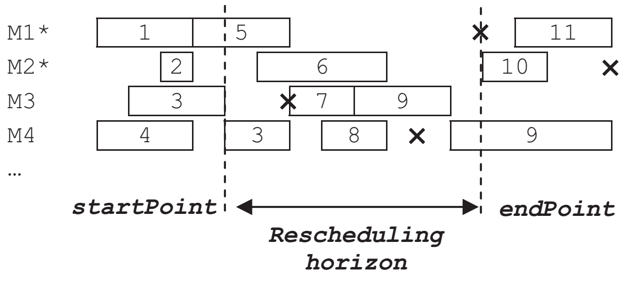


Figure 4.11: Example of the collection of idle times; (a) the new job requirements, (b) idle time collection using strategy FW1, (b) FW2, (c) FW3 and (e) FW4.

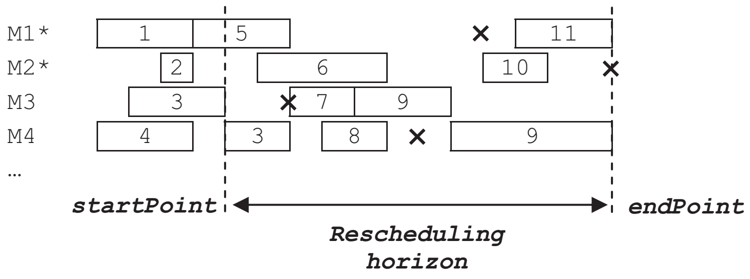
Let C_i be the completion time of a new job on machine i; then *endPoint* is defined as the maximum of C_i, $i = 1, \ldots, M$. Note that in the case of parallel machines, there are two or more possible completion time values for the same operation. Just one of these is considered for the calculation of *endPoint*. Here either the earliest or the latest of the completion times is used. In this way, 8 strategies are defined: S1-S4 that collect idle time using FW1-FW4, respectively, and consider the earliest completion time among the parallel machines, and S5-S8 which collect idle times using FW1-FW4, respectively, but consider the latest completion time. Consequently, S1– S4 is more likely to define smaller rescheduling horizons than S5– S8. Figure 4.12(a) demonstrates the idle time collection using strategy S1. Since the two accumulated times that extend the latest are on machines M1 and M2, which are parallel machines, there are two options for the definition of *endPoint*. The first option, using S1, considers the earliest completion time, as in Figure 4.12(b). The other option, using S5, uses the latest completion time, as in Figure 4.12(c).



(a)



(b)



(c)

Figure 4.12: Example of the calculation of the rescheduling horizon; (a) the collection of idle time on machines, (b) the calculation of *endPoint* using strategy S1 and (c) the calculation of *endPoint* using strategy S5.

Once the rescheduling horizon has been calculated, a new scheduling problem is defined. This problem requires the scheduling of the operations that lie within the rescheduling horizon and the operations of the new job. Let O be the set of all these operations (step 3) and let r_o and d_o be the release time and the due date of operation $o \in O$, respectively, in the original schedule. In order to keep the operations in O within the rescheduling horizon after rescheduling, the release times of the operations in O are set to $r_o^{new} = \max\{r_o, startPoint\}$, $\forall o \in O$, and the due dates are set to $d_o^{new} = \min\{d_o, endPoint\}$, $\forall o \in O$, (step 4). In step 5, a new scheduling problem that considers the operations in O with their updated release and due dates and with the objective of maximising the quality of the schedule is defined. A genetic algorithm (GA) is responsible to allocate jobs on machines as described in [86]. Note that different quality measures presented in section 4.2 can be used as objective functions for rescheduling affected operations O. More details and discussions about GA settings are presented in section 4.4 and subsection 4.4.3.

In the final phase (step 7), the initial partial schedule contained within the rescheduling horizon is replaced by the new schedule generated in step 6. It may be the case that the completion times of one or more operations in the new schedule are out of the rescheduling horizon. If this is the case, there is the possibility that such operations overlap with the operations from the initial schedule that start after the *endPoint*. These cases are identified, and operations from the initial schedule are shifted to the right as necessary in order to restore feasibility. Suppose that S1 was used to set a rescheduling horizon for inserting the new job j presented in Figure 4.10(a). The resultant schedule is shown in Figure 4.13(a) and there is an overlap between operations 12 and 10 on M2, since operations of the new job 12 were allocated outside of the rescheduling horizon. The schedule feasibility is restored when operation 10 on M2 is shifted to the right for 2 time units, as in Figure 4.13(b).

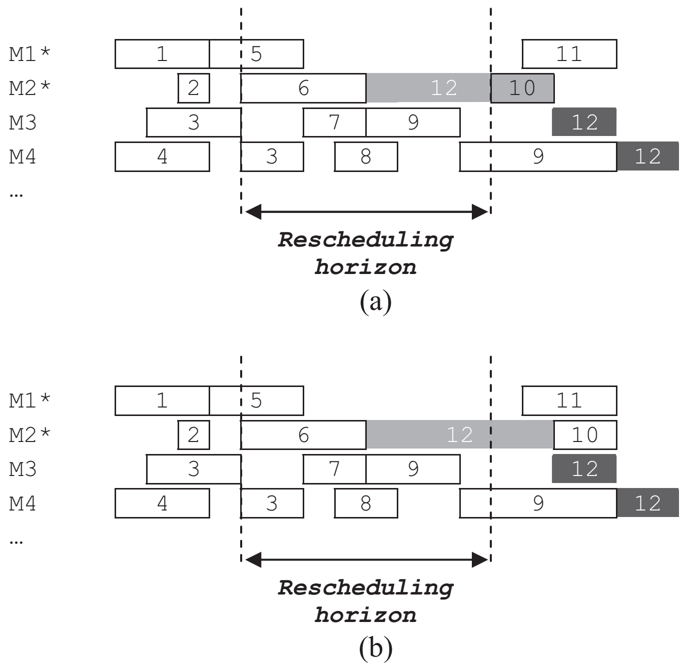


Figure 4.13: Example of overlapping between initial and the partial schedule generated by the rescheduling process; (a) the overlap between jobs 12 and 10 on M2 and (b) the feasibility restoration by right shifting the job 10 for 2 time units.

4.4 Experiments on Real World Data

Data sets obtained from Sherwood Press are used to test the *Performance* and *Stability* of the developed match-up algorithm. The data of the production orders of several months were used to produce schedules. In each instance, newly arriving jobs were randomly generated taking into account three parameters sat, $insTime$ and $jobSize$.

The saturation, sat, is used as an indicator of the amount of idle time in the shop floor and is calculated as the ratio between the makespan of the initial schedule and the sum of processing times of all operations:

$$sat = \frac{C_{max}}{\sum_{i=1}^{n} \sum_{j=1}^{M} p_{ij}}$$

where C_{max} is the length of the initial schedule (makespan) and p_{ij} is the processing time required by job i, $i = 1, \ldots, n$ on machine j, $j = 1, \ldots, M$. A large sat value indicates a highly saturated schedule, i.e. with a small amount of idle time. Three months with different saturation levels were selected and used to generate three instances. The saturation levels for each considered month are 1.85, 2.37 and 3.67. These values correspond to low, medium and high (i.e. $sat \in \{$low, medium,high$\}$) saturation levels, respectively.

The second parameter is the time of insertion of the new job, $insTime \in \{$beginning, middle, end$\}$, where “beginning”, “middle” and “end” are equal to 10%, 50% and 80% of the length of the schedule, respectively. The reason for considering the $insTime$ follows the observation that the workload of the shop varies at different points in the schedule. The workload in the middle of the schedule, for instance, is often higher than the workload at the beginning, which is higher than the workload at the end of the schedule.

The third parameter is the number of operations in the new job, $jobSize \in \{1,2,3,4,5\}$. The $jobSize$ value serves as a good indicator of the magnitude of the disturbance of the current schedule, which makes it an interesting parameter to investigate.

More details about characteristics of the experimental data is described in subsection 4.4.2.

Since there are three types of initial schedules of different saturation levels, and jobs arrive at three different times and they are of five different sizes, the total number of instances is $3 \times 3 \times 5 = 45$. Arriving jobs are not kept in the schedule as the experimentation progresses, on the contrary, once a job has been integrated into an initial schedule and the proper measures have been recorded, the job is removed, and the schedule is reset to its initial state ready to accommodate the next arriving job.

Initial schedules, before any disruption occurs, are generated using the genetic algorithm described in [86] with the objective of maximising the *Performance* measure. Ten different solutions are generated for each *sat* instance and their results are graphically presented in Figure 4.14. The *Performance* measure is calculated as an average between satisfaction grades SG_i, $i = 1, \dots 5$, previously described in section 4.2. A best solution is selected for each saturation level and subsequently they are used as initial solutions to insert new jobs in the scheduling problem. The aim is to investigate different scheduling scenarios.

The arriving jobs were integrated into the initial schedules using rescheduling strategies: $S1 - S8$, total rescheduling (T), right shift rescheduling (RS) and insertion in the end (E). In T, all operations after *feasiblePoint* are rescheduled with a genetic algorithm. In RS, the arriving job is integrated into the new schedule so that it is completed before its due date. The overlapping operations from the initial schedule, if any, are shifted to the right as required. In E, the new job is inserted right after the completion of the last operation of the initial schedule. The *Performance* and *Stability* measures were recorded for each strategy, each type of instance and for each arriving job.

It is important to highlight that schedules are generated using the genetic algorithm proposed by [86]. Their studies extensively investigate the static scheduling problem

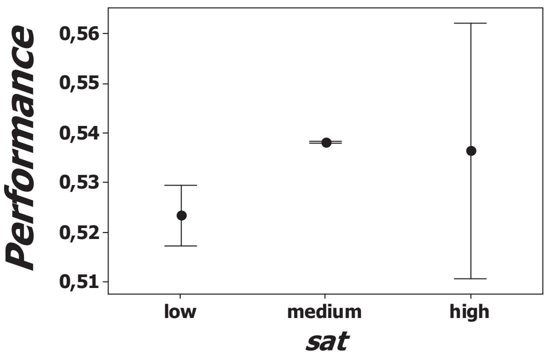


Figure 4.14: Overall results obtained by initial solutions with low, medium and high saturation values, the x-axis shows the saturation categories; the y-axis shows the mean (dot) and 95% confidence interval (vertical bars) of the achieved *Performance*

presented by Sherwood Press employing a GA to allocate jobs on machines [32,84,85,87]. More details are described in the following subsection. Note that this algorithm is also applied on rescheduling to accommodate newly arriving jobs, as previously mentioned in step 6 from Figure 4.9.

All algorithms were implemented in Visual C++. Testing was performed on a 2.16 GHz Centrino Duo PC with 1GB of RAM and running Windows XP. Results for different types of orders are presented and analysed in subsection 4.4.3.

4.4.1 Genetic Algorithm for Scheduling

Genetic algorithm is a bio-inspired method commonly used in optimisation problems [94]. It is based in natural evolution concepts, in which good solutions have a higher probability to remain through subsequent generations. Operators such as crossover and mutation are applied in order to explore the diversity of possible solutions. They have been employed in several scheduling / rescheduling problems mainly because they are able to create near optimal solutions for NP-hard problems [92, 111], as previously highlighted in Table 2.2 from Chapter 2.

The genetic algorithm mentioned in the previously subsection was specifically designed, tested and tuned for the scheduling problem presented by Sherwood Press [86]. Their main components are described as follows:

- **Chromosomes and population**: Each chromosome contains two sub-chromosomes. The first one represents the 18 available machines, and the second one has the 6 dispatching rules that are used for sequencing operations on the corresponding machines. Several chromosomes are generated in order to create a population of possible solutions;

- **Initialisation**: A random number $i = 1, \ldots, 18$ is assign to cells of the first sub-chromosome, in such a way that each cell has a distinctive i. Subsequently, one of the following 6 dispatching rules is selected to each cell of the second sub-chromosome, in which repetitions are allowed: (1) EDD - Early Due Date First, (2) SPT - Shortest Processing Time First, (3) LPT - Longest Processing Time First, (4) LRT Longest Remaining Processing Time First, (5) HP Highest Priority First,

(6) SFT - Same Family of Jobs Together. Note that the first four rules are well-established and widely used in the literature on job shop scheduling [91]. On the other hand, the last two are tailored to Sherwood Press with the aim of reducing the flow time of jobs of higher priority and scheduling group of jobs that belong to the same printing family, respectively. A graphical representation of two initialised chromosomes A and B is shown in Figure 4.15 (a);

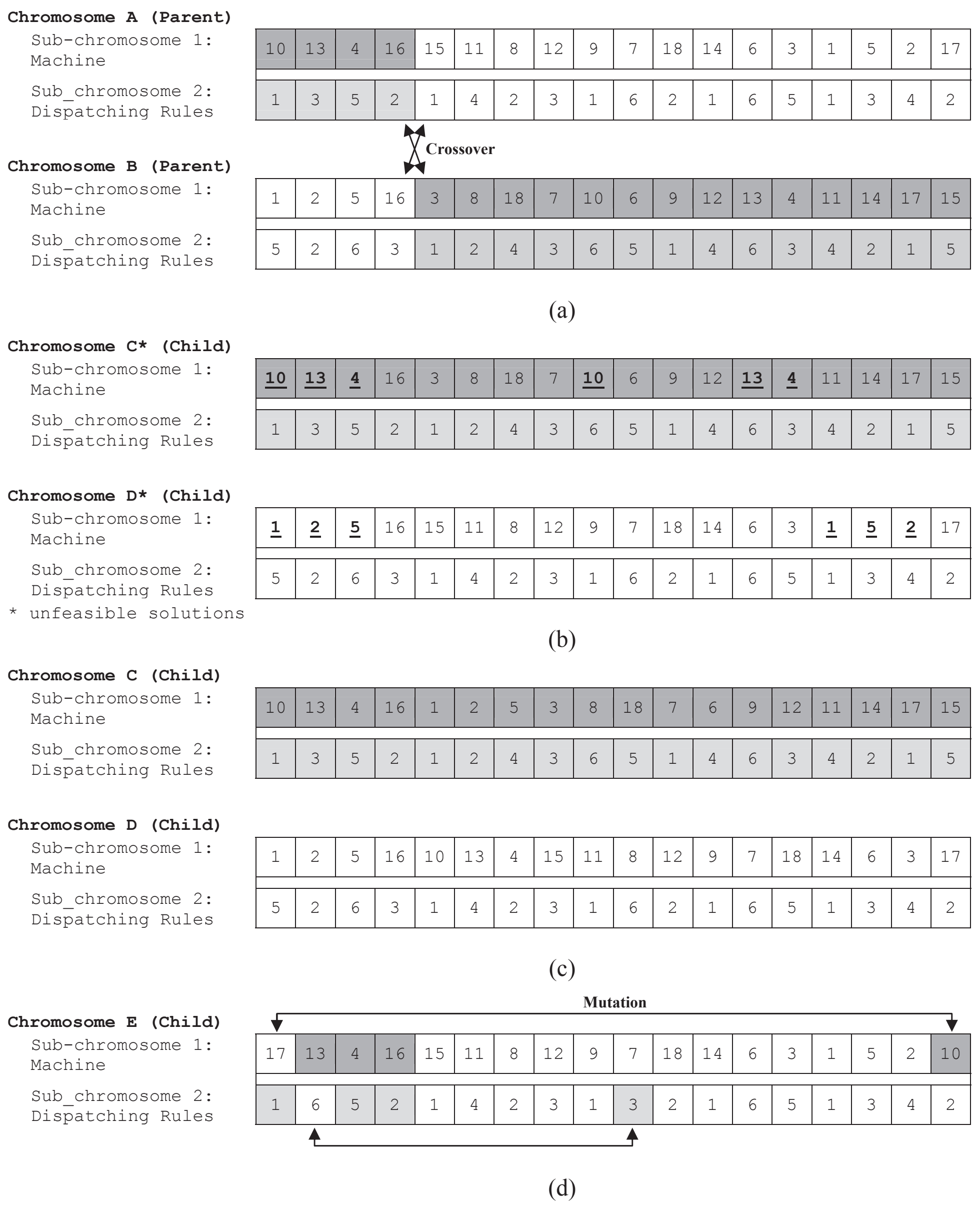


Figure 4.15: Components of a genetic algorithm for scheduling, (a) chromosome initialisation, (b) crossover generating unfeasible solutions, (c) repaired solutions, and (d) mutation.

- **Crossover**: This operator combines genes from two parents chromosomes to generate a new offspring. A single point W is randomly chosen in order to swap genes between parents. Figure 4.15 (b) shows an example of two offsprings being generated by swapping values on position 4 from both chromosomes present in Figure 4.15 (a). Unfortunately, infeasible solutions are generated in this example since repetition of *i* values are present in the machine sub-chromosomes of both offsprings. Consequently, a repair operation is required [9]. Child C has to be created by copying all values of the parent A up to position 4. Subsequently, the remaining cells are filled by scanning parent B from left to right and entering the machine numbers not already present. Child D is created in a similar way by reversing the roles of the parents. Figure 4.15 (c) shows the feasible resultant children following this repair mechanism. Note that no changes were required for the second sub-chromosomes and their orders remain the same in both Figures 4.15 (b) and (c);

- **Mutation**:A randomly chosen pair of genes exchange their positions in a sub-chromosome. Mutation is applied independently in both sub-chromosomes. Figure 4.15 (d) illustrates a child D being generated from the mutation of parent A, in which genes from positions 1-18 and 2-10 are exchanged from sub-chromosomes 1 and 2, respectively;

- **Generations**: Defines the number of iterations that the population of chromosomes will be subject to the processes of crossover, mutation and selection;

- **Fitness function**: Evaluates the quality of a given schedule based on the solution generated by its chromosome. The goal find a schedule with the highest possible fitness value among analysed generations. The expression 4.1 from section 4.2 is employed to assess this measure;

- **Selection**: A roulette-wheel-selection technique is applied for selecting chromosomes that will survive through consecutive generations. Chromosomes with higher fitness function values have a higher chance to be kept in the population of possible solutions;

- **Elitist Strategy**: The chromosome with the highest fitness is always kept to the next generation.

A series of experiments were done by [32, 86] to tune the GA parameters, in which the population size and number of generations were set to 50 and 500, respectively, whereas the crossover and mutation probabilities were set to 80% and 30%, respectively.

4.4.2 Characteristics of the Experimental Data

The following items provides complementary information about the experimental data. It is important to highlight that they are valid for all experiments investigated in this thesis.

- **Load-balancing**: All operations are evenly allocated between available parallel machines in order to process the required jobs as quickly as possible. This feature belongs to a previous study investigated by [84] and it is implemented in the scheduling algorithm;
- **Lot-sizing**: Some operations are eligible to be subdivided into smaller lots when they require large processing times. The aim is to manage the customer satisfaction delivering smaller lots in a shorter period of time. Again, this feature was originally introduced by [85, 86] and it is part of the scheduling process;
- **Number of jobs**: The number of jobs to be allocated varies between different months in Sherwood Press. For instance the databases used to set $sat \in$ {low, medium,high} has 39, 64 and 158 jobs, respectively. Note that a higher number of jobs does not necessarily mean the definition of a higher saturation level, since a schedule with small number of operations may require large processing times on machines;
- **Duration of operations**: As previously mentioned, some operations may have longer processing times than the others. For instance they are eligible to require 2 hours on a printing machine, as well as 5 days in a row.
- **Workload of the shop floor**: Number of required items from all jobs are summed up in order to define an overall workload of the shop floor, i.e. the three investigated months defining $sat \in$ {low, medium,high} have 3696356, 3955125 and 5120125

items to be produced, respectively. This number is used as a parameter to make decisions on lot-sizing.

Note that this items are described only as a reference to illustrate the data present in the investigated scheduling problem and to identify the studies developed by Fayad and Petrovic [84–86].

4.4.3 Rush Orders

Most of the jobs arriving in the shop floor of Sherwood Press are classified as rush orders, because they must be integrated in the current schedule as soon as possible. These newly arriving jobs are incorporated using the same genetic that was used to generate the initial schedule, as described in [86]. The original fitness algorithm function, expression 4.1 from section 4.2, was kept unchanged because it delivers good *Performance* results and the *Stability* of the schedule is maintained because only a part of the schedule must be modified.

This section presents the results obtained by the different rescheduling strategies as well as the results of statistical analysis of the effects of the problem parameters and the match-up strategies on *Performance* and *Stability*. The results obtained by all strategies are summarised in Table 4.1, which presents the average and standard deviation values for *Performance* and *Stability* attained by the investigated strategies for the instances grouped according to the different problem parameter values. Note that each cell in Table 4.1 represent the average result of 10 times executing the rescheduling algorithm. Additionally, the best strategies are highlighted in bold for each type of instance and numbers between brackets show the number executions done for each instance. In general, the match-up strategies (S1-S8) are superior to right shift (RS) and insertion in the end (E), and similar results to total rescheduling (T), with respect to *Performance*. As expected, RS and E deliver the most stable schedules. However, S1-S8 can be highlighted as superior to T with respect to *Stability*. Among the match-up strategies, S1-S4 seem to outperform strategies S5-S8 in most groups of instances. These results will be statistically verified next.

A comparison between *Performance* and *Stability* is presented in Figure 4.16, in which a scatter plot shows the trade-offs between these evaluation metrics. Few samples have

Table 4.1: Average and standard deviation values for *Performance* and *Stability* obtained by the rescheduling strategies for rush orders (larger vales are preferred)

		Performance - Rescheduling strategy										
		S1	S2	S3	S4	S5	S6	S7	S8	T	RS	E
sat	(15) low	0.534	0.534	0.534	0.534	0.528	0.528	0.531	0.531	**0.538**	0.528	0.510
	(15) medium	0.519	0.521	**0.526**	**0.526**	0.521	0.521	0.519	0.519	0.523	0.523	0.481
	(15) high	0.528	0.528	0.528	**0.529**	0.496	0.505	0.500	0.498	0.518	0.513	0.519
jobSize	(9) 1	**0.534**	**0.534**	**0.534**	**0.534**	0.523	0.524	0.523	0.523	0.530	0.532	0.526
	(9) 2	**0.532**	**0.532**	**0.532**	**0.532**	0.518	0.521	0.519	0.519	0.523	0.529	0.516
	(9) 3	0.529	0.529	0.530	0.530	0.516	0.519	0.518	0.517	**0.536**	0.524	0.504
	(9) 4	0.520	**0.526**	**0.526**	**0.526**	0.513	0.517	0.514	0.513	0.522	0.515	0.491
	(9) 5	0.519	0.517	**0.525**	**0.525**	0.506	0.510	0.509	0.508	0.521	0.507	0.479
insTime	(15) beginning	0.526	0.529	**0.534**	**0.534**	0.505	0.514	0.507	0.504	0.517	0.518	0.493
	(15) middle	0.527	0.527	0.528	0.527	0.514	0.515	0.517	0.517	**0.537**	0.521	0.493
	(15) end	0.526	0.526	0.526	**0.527**	0.526	0.526	0.526	0.526	0.526	0.526	0.525
(45) total average		0.527	0.528	0.529	**0.530**	0.515	0.518	0.517	0.516	0.526	0.521	0.503
(45) standard deviation		0,005	0.005	0.003	**0.003**	0.009	0.007	0.008	0.009	0.007	0.007	0.016
		Stability - Rescheduling strategy										
		S1	S2	S3	S4	S5	S6	S7	S8	T	RS	E
sat	(15) low	0.982	0.983	0.977	0.977	0.982	0.981	0.982	0.979	0.912	**1.000**	**1.000**
	(15) medium	0.965	0.965	0.966	0.966	0.966	0.967	0.966	0.965	0.943	**1.000**	**1.000**
	(15) high	0.991	0.991	0.990	0.993	0.961	0.954	0.958	0.958	0.878	**1.000**	**1.000**
jobSize	(9) 1	0.998	0.998	0.998	0.998	0.978	0.978	0.977	0.977	0.894	**1.000**	**1.000**
	(9) 2	0.998	0.998	0.998	0.998	0.982	0.978	0.983	0.981	0.939	**1.000**	**1.000**
	(9) 3	0.990	0.990	0.989	0.989	0.986	0.982	0.987	0.987	0.919	**1.000**	**1.000**
	(9) 4	0.970	0.971	0.973	0.972	0.967	0.961	0.962	0.962	0.911	**1.000**	**1.000**
	(9) 5	0.941	0.942	0.932	0.936	0.935	0.938	0.935	0.930	0.892	**1.000**	**1.000**
insTime	(15) beginning	0.973	0.975	0.972	0.973	0.947	0.940	0.944	0.942	0.838	**1.000**	**1.000**
	(15) middle	0.982	0.982	0.977	0.980	0.979	0.979	0.979	0.977	0.918	**1.000**	**1.000**
	(15) end	0.984	0.983	0.984	0.983	0.983	0.983	0.983	0.983	0.977	**1.000**	**1.000**
(45) total average		0.979	0.980	0.978	0.979	0.970	0.967	0.969	0.967	0.911	**1.000**	**1.000**
(45) standard deviation		0.016	0.016	0.018	0.017	0.016	0.016	0.017	0.017	0.035	**0.000**	**0.000**

Performance values between 0.3 and 0.45 and they are considered outliers. Figure 4.16 shows that variations on *Stability* are more expressive than the ones in *Performance*, which means that some rescheduling strategies can easily compromise the *Stability* of schedules. However, there is no indication that increasing *Stability* has positive or negative effects on *Performance* values, or vice-versa, which means that, even with the presence of the two conflicting criteria as *Performance* and *Stability*, it is possible to achieve highly stable and good quality schedules.

The statistical significance of the effects of problem parameters, match-up strategies, and the interactions among them on *Performance* and *Stability* was investigated by means

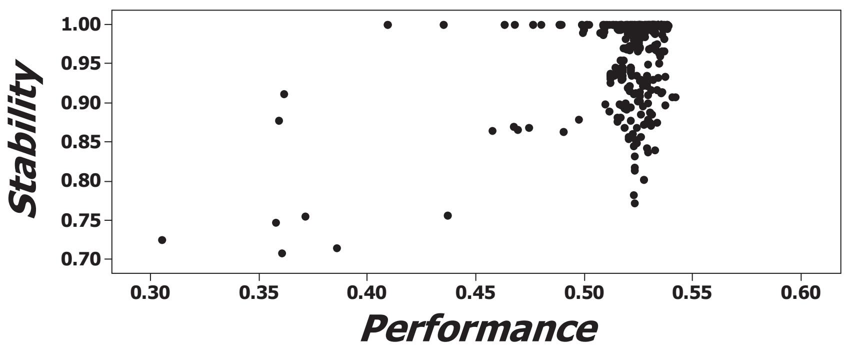


Figure 4.16: Trade-offs between *Performance* and *Stability* for rush orders

of the Analysis of Variance (ANOVA). The summary of the ANOVA is given in Table 4.2, where the individual effects of problem parameters and rescheduling strategy on *Performance* and *Stability* are labelled “main effects”, whereas the combined effects of the pairs of variables are labelled “interactions”. The $A*B$ notation refers to the interaction between parameters A and B. Values under the heading F value and P value are the value of the Fisher statistic of the corresponding row effect, and the probability of this value being due to mere chance, respectively. Effects with a P value ≤ 0.05 are considered to be significant. Results from ANOVA test shows that all “main effects” and “interactions” involving the parameter Strategy have influence on both *Performance* and *Stability* of the schedule. Further discussions on these results are presented next. R^2 measures the proportion of the variation of the observations around the mean. The relatively large R^2 values for both metrics, *Performance* and *Stability*, is an indicator of a high variability on the obtained results [68].

The fact that the effect on variability due to *Strategy* is significant implies that some rescheduling strategies are better than the others. A pairwise comparison test using Bon-

Table 4.2: Results of the ANOVA test for rush orders

		Performance		*Stability*	
		F value	P value	F value	P value
Main effects					
	Strategy	13.56	≤ 0.05	55.38	≤ 0.05
	sat	104.94	≤ 0.05	7.92	≤ 0.05
	jobSize	25.47	≤ 0.05	81.17	≤ 0.05
	insTime	18.24	≤ 0.05	108.40	≤ 0.05
Interactions					
	Strategy**sat*	6.49	≤ 0.05	8.06	≤ 0.05
	Strategy**jobSize*	0.66	≤ 0.05	2.15	≤ 0.05
	Strategy**insTime*	5.86	≤ 0.05	14.83	≤ 0.05
	R^2	0.71		0.83	

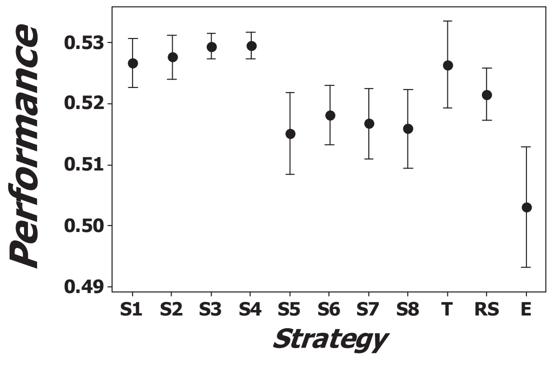
ferroni's correction [68] was carried out in order to identify which are those strategies that deliver higher *Performance* and *Stability* values. The results are given in Figure 4.17, with comparisons of *Performance* below the diagonal, and above of *Stability*. Each field in Figure 4.17 corresponds to the statistical test of the difference in means between the corresponding row-column strategies. The conclusion of each test is indicated with one of the following symbols: "$\leftarrow$" which indicates that the strategy in the row is superior; "$\uparrow$" which indicates that the column strategy is superior; and "=" which indicates that the strategies are non-distinguishable from each other. Additionally, the t statistic value and the p value are given on the top right and bottom right of each field, respectively. Large absolute t values and low p values confirm that the means of the corresponding row-column strategies are different, implying that one of them is superior. For example, the test in row S3, column RS, indicates that S3 obtains, on average, higher *Stability* values than RS. This holds with a t-test value of 5.70 and a $p \leq 0.05$.

Not surprisingly, RS and E deliver good values for *Stability* and T for *Performance*. However, this is achieved at the price of poor *Stability* for T and poor *Performance* for RS and E. Remarkably, the results obtained by the match-up strategies S1-S4 are statistically non-distinguishable from T for *Performance*, and strategies S1-S8 produce comparable results from RS and E for *Stability*. This result indicates that the newly introduced approaches posses the best attributes of the investigated rescheduling approaches, but do not exhibit their weaknesses. Figure 4.18(a) and 4.18(b) show these results graphically for *Performance* and *Stability*, respectively, using 95% confidence interval plots. In each plot, the dot indicates the average value on the whole set of instances obtained by the corresponding strategy in the $x-$axis. The vertical lines denote the 95% confidence interval of the mean value. Statistical differences are immediately detected when there is no overlap between the confidence intervals of two or more strategies. According to the interval plots in Figure 4.18(a) and Figure 4.18(b), strategies S1-S4 are the most competent with respect to both, *Performance* and *Stability*.

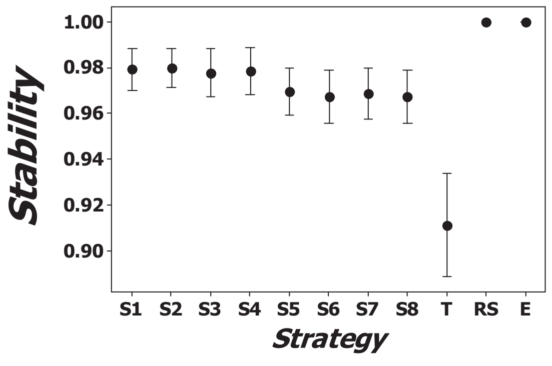
Strategies S5-S8 obtain, overall, low *Performance* values. In order to explain this behaviour, a correlation between the *Performance* and the $Delay$ that occur after rescheduling was measured. The $Delay$ represents the length of the overlap time period of the new schedule with the initial one after the rescheduling horizon. This is a consequence

Performance \ *Stability*	S1	S2	S3	S4	S5	S6	S7	S8	T	RS	E
S1		= 0.14 1.00	= -0.39 1.00	= -0.19 1.00	= -2.52 0.65	= -3.07 0.12	= -2.74 0.33	= -3.09 0.11	← -17.51 ≤0.05	↑ 5.30 ≤0.05	↑ 5.30 ≤0.05
S2	= 0.46 1.00		= -0.53 1.00	= -0.33 1.00	= -2.66 0.43	= -3.21 0.07	= -2.88 0.21	= -3.23 0.06	← -17.65 ≤0.05	↑ 5.16 ≤0.05	↑ 5.16 ≤0.05
S3	= 1.40 1.00	= 0.94 1.00		= 0.20 1.00	= -2.13 1.00	= -2.67 0.41	= -2.35 1.00	= -2.69 0.39	← -17.12 ≤0.05	↑ 5.70 ≤0.05	↑ 5.70 ≤0.05
S4	= 1.46 1.00	= 1.00 1.00	= 0.06 1.00		= -2.33 1.00	= -2.87 0.22	= -2.55 0.59	= -2.89 0.21	← -17.32 ≤0.05	↑ 5.50 ≤0.05	↑ 5.50 ≤0.05
S5	↑ -5.99 ≤0.05	↑ -6.45 ≤0.05	↑ -7.39 ≤0.05	↑ -7.45 ≤0.05		= -0.55 1.00	= -0.22 1.00	= -0.57 1.00	← -14.99 ≤0.05	↑ 7.82 ≤0.05	↑ 7.82 ≤0.05
S6	↑ -4.41 ≤0.05	↑ -4.87 ≤0.05	↑ -5.82 ≤0.05	↑ -5.87 ≤0.05	= 1.57 1.00		= 0.32 1.00	= -0.02 1.00	← -14.45 ≤0.05	↑ 8.37 ≤0.05	↑ 8.37 ≤0.05
S7	↑ -5.14 ≤0.05	↑ -5.61 ≤0.05	↑ -6.55 ≤0.05	↑ -6.60 ≤0.05	= 0.84 1.00	= -0.73 1.00		= -0.34 1.00	← -14.77 ≤0.05	↑ 8.05 ≤0.05	↑ 8.05 ≤0.05
S8	↑ -5.56 ≤0.05	↑ -6.02 ≤0.05	↑ -6.97 ≤0.05	↑ -7.02 ≤0.05	= 0.42 1.00	= -1.15 1.00	= -0.41 1.00		← -14.43 ≤0.05	↑ 8.39 ≤0.05	↑ 8.39 ≤0.05
T	= -0.15 1.00	= -0.61 1.00	= -1.55 1.00	= -1.61 1.00	← 5.84 ≤0.05	← 4.26 ≤0.05	← 4.99 ≤0.05	← 5.41 ≤0.05		↑ 22.82 ≤0.05	↑ 22.82 ≤0.05
RS	↑ -2.67 ≤0.05	↑ -3.13 ≤0.05	↑ -4.08 ≤0.05	↑ -4.13 ≤0.05	= 3.31 1.00	= 1.73 1.00	= 2.47 0.74	= 2.89 0.21	= -2.52 0.64		= 0.00 1.00
E	↑ -12.07 ≤0.05	↑ -12.53 ≤0.05	↑ -13.47 ≤0.05	↑ -13.53 ≤0.05	↑ -6.07 ≤0.05	↑ -7.65 ≤0.05	↑ -6.92 ≤0.05	↑ -6.50 ≤0.05	↑ -11.92 ≤0.05	↑ -9.39 ≤0.05	

Figure 4.17: Mean pairwise comparisons of *Performance* and *Stability* for rush orders



(a)



(b)

Figure 4.18: Overall results obtained by each rescheduling strategy; the x-axis shows the strategy; the y-axis shows the mean (dot) and 95% confidence interval (vertical bars) of *Performance* (a) and *Stability* for rush orders

of rescheduling jobs within the rescheduling horizon after which some jobs are tardy (i.e. they do not meet their due dates). The correlation between the *Performance* and the

Delay, $\rho = -0.42$, indicates that high delays caused by the rescheduling process are associated with low *Performance* values. This explains the low *Performance* values achieved by strategies S5-S8 which, as shown in Figure 4.19(a), are the ones that lead to the largest *Delay* values. On the other hand, S3 and S4, which seem to be the best performers according to Figure 4.18(a), cause the smallest delays. Note that strategy E obtained low *Performance* values even when it produces no *Delay* value. This situation occurs due to a higher probability of the new jobs being tardy when they are inserted in the end of the schedule. Additionally, match-up algorithms are relatively efficient since they took 10 seconds on average in the rescheduling process (see Figure 4.19(b)). This time is higher than the one required by RS and E, but lower than the one required by the total rescheduling. In any case, 10 seconds on average for the rescheduling of a one month production schedule is fast enough for a printing company industry and certainly for many other real world production shops.

Regarding the problem parameters, the ANOVA results in Table 4.2 shows that *sat*, *jobSize* and *insTime* have a significant influence on *Performance* and *Stability*. The nature of these effects is illustrated with the 95% confidence interval plots in Figure 4.20. The *x*-axis of plots (a)-(b), (c)-(d), and (e)-(f), measures the level of *sat*, *jobSize* and *insTime*, respectively. The *y*-axis shows the average values of *Performance*, (a), (c), (e), and *Stability*, (b), (d), (f) over all rescheduling strategies. In general, *Performance* and *Stability* values decrease when rescheduling is done on highly saturated schedules, rescheduling occurs at the beginning or in the middle of the schedule, or when the arriving job requires

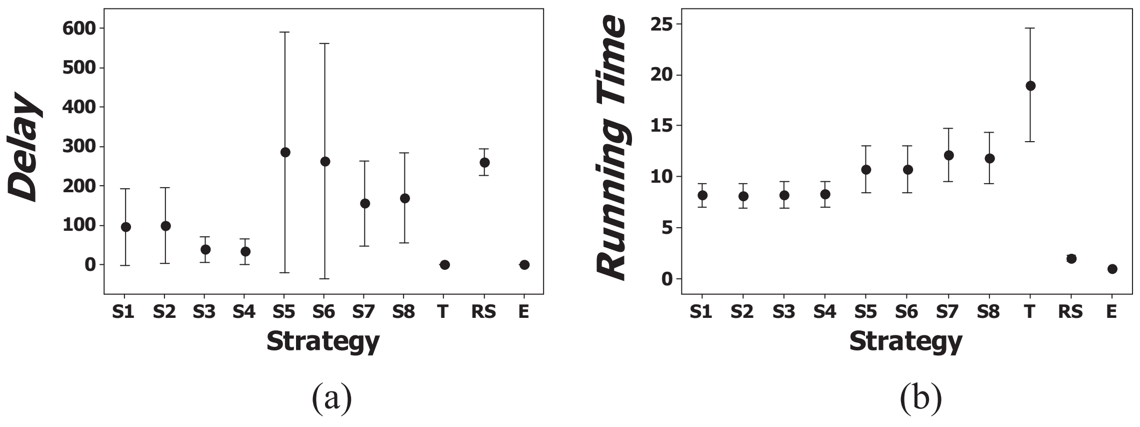


Figure 4.19: Overall results obtained by each rescheduling strategy on *Delay* (a) and *Running Time* (b) for rush orders

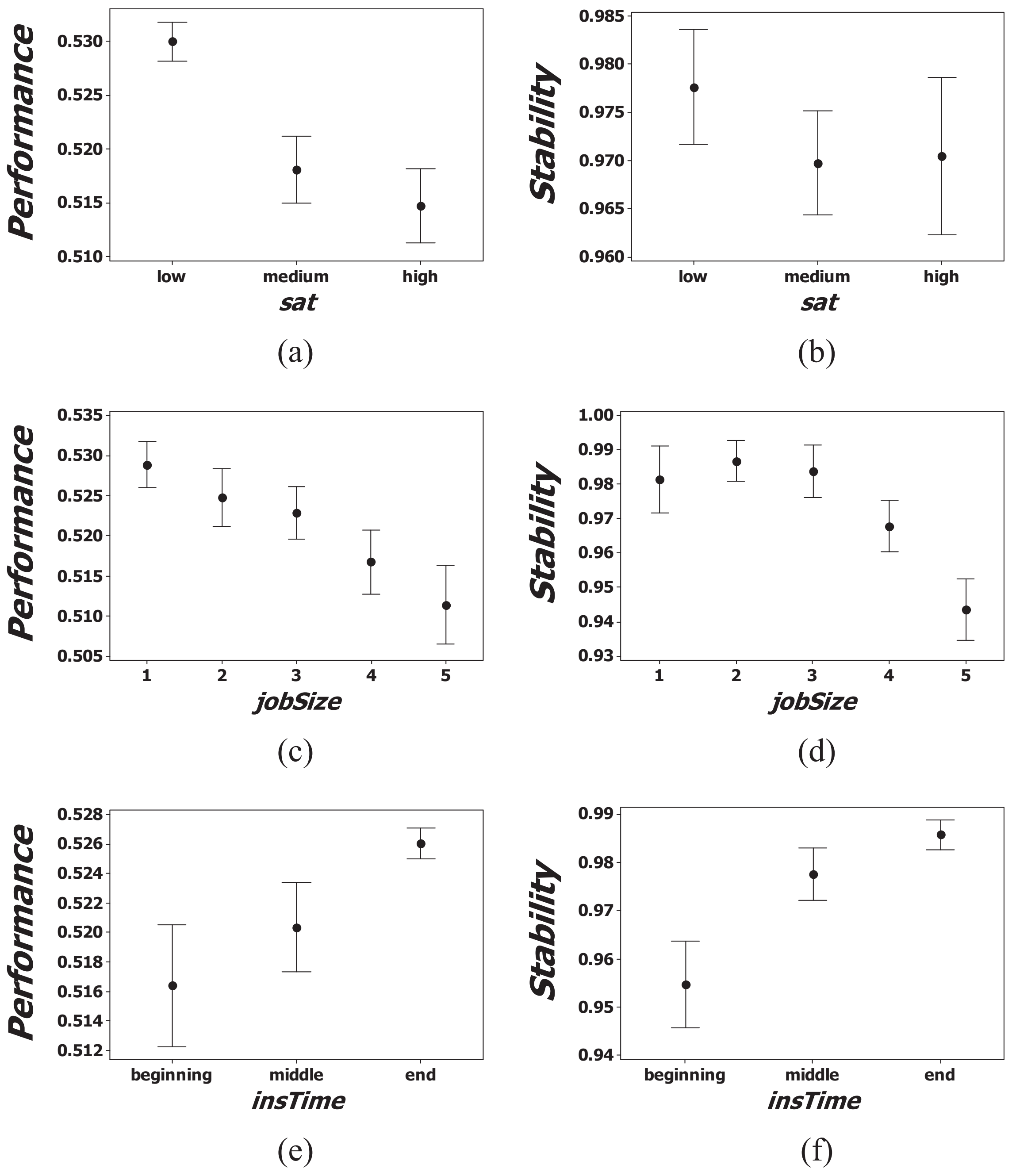


Figure 4.20: Main effects on *Performance* (a), (c), (e) and *Stability* (b), (d), (f) due to Strategy (a)-(b), *jobSize* (c)-(d) and *insTime* (e)-(f) for rush orders

many operations.

The ANOVA results, presented in Table 4.2, also identifies that all interactions of parameters involving *Strategy* are significant. These type of interactions indicate that some strategies are better at coping with certain problem conditions than others. That this is the case can be verified by Table 4.1. The three interactions involving *Strategy* were analysed and it was observed that strategies S1-S4 are either similar or superior to strategies S5-S8 under any scenario. This seems to contradict the existence of any interaction. However, the interactions exist because strategies S1-S4 are, under certain conditions, remarkably better than strategies S5-S8; under any other conditions they are only similar or slightly better. Regarding the *insTime*, for instance, S1-S4 are much preferred if the arriving job is to be inserted at the beginning of the schedule. Under this scenario, the *Delay* inserted by the rescheduling algorithm will have a stronger effect than if rescheduling occurs at the end of the schedule, and consequently strategies S1-S4 will be remarkably superior to S5-S8. A similar reasoning explains why strategies S1-S4 are also superior to S5-S8 when rescheduling occurs on highly saturated schedules (with high *sat* values). Additionally, it was observed that strategies S1-S4 are superior to S5-S8 when the arriving jobs have a small number of operations, 1 or 2. The number of operations to be inserted in the current schedule is directly proportional to the length of the rescheduling horizon. If the rescheduling horizon is relatively short, for example when inserting only one or two operations, the calculation of the *endPoint* performed differently by S1-S8 has a higher impact on the definition of the rescheduling horizon than if the rescheduling horizon is already large due to the insertion of a large number of operations. In this way, the small delay values obtained by S1-S4 lead to a better overall *Performance* and *Stability*.

Given the results in Table 4.1 and the statistical analysis, it is confirmed that match-up strategies are comparable to the right shift and insertion in the end strategies with respect to *Stability* and as good as the total rescheduling strategy with respect to *Performance*. The strategies S1-S4 seem to be slightly superior to the other match-up approaches and deliver the most consistent results under different problem scenarios, as demonstrated by Figure 4.18. The strategies S3 and S4 can also be highlighted for possible incorporation into the scheduling system of Sherwood Press, since they produce high values with respect to *Performance* and *Stability* and small *Delay* and *RunningTime* values, as shown in Figure 4.19.

4.5 Discussion

Match-up strategies are proposed to insert rush orders in a complex real world job shop scheduling/rescheduling problem, in which high performing and stable schedules are delivered in response to this typical disruption.

Statistical analysis reveal that, even with the presence of the two conflicting criteria as *Performance* and *Stability*, match-up strategies achieve high quality schedules under different problem instances, which highlight their strengths regarding possible scenarios tackled by Sherwood Press.

Match-up strategies S3 and S4 are candidates for possible incorporation into the scheduling / rescheduling system for Sherwood Press, since they produce good values for both *Performance* and *Stability* at a reasonable *RunningTime*.

Note that initial schedules and rescheduling of affected operations are done with the same five criteria fitness function described in [86], because good *Performance* results are delivered, while *Stability* is maintained by match-up strategies requiring only partial modifications on schedules. This feature is suitable to improvements and more detailed discussions are presented in the next chapter.

4.6 Summary

This chapter investigates a real world job shop scheduling/rescheduling problem from a printing company in Nottingham, UK. This problem is dynamic since new jobs with different levels of urgency arrive everyday in the shop floor and they have to be integrated into the existent schedule. Typical arriving jobs are rush orders, which means that they have to be processed as early as possible in the current schedule. This type of disruption is tackled first and the goal is to find appropriate rescheduling approaches to achieve high quality schedules.

A match-up algorithm, which accommodates new rush orders by using available idle times on machines, is proposed. Additionally, quality measures are introduced in order to identify good performing schedules. The motivation of the match-up algorithm is to modify only a part of the initial schedule in such a way that both *Stability* and *Performance* of the shop floor are kept, avoiding additional production costs. Several strategies to define rescheduling horizons were proposed and compared with other strategies, including total rescheduling, the right shift and insertion in the end of the schedule.

The obtained results were analysed and statistically validated. In summary, all match-up strategies do obtain reasonable values for both *Performance* and *Stability* regardless of the problem parameters. Strategies S1-S4 deliver better results because they set smaller rescheduling horizons than S5-S8. It was observed that match-up strategies are statistically non-distinguishable from total rescheduling with respect to *Performance* and comparable to the right shift and “insertion in the end” with respect to *Stability*. These encouraging results coupled with the fact that the proposed algorithms only take 10 seconds on average to reschedule a one month schedule, indicate that the proposed match-up strategies and particularly S1-S4 are adequate for the investigated and other similar production shops.

The following chapter continues of the investigation of match-up strategies, in which new improvements are proposed and the impact of other disruptions are investigated.

Chapter 5

Match-up Strategies for a Complex Real World Job Shop Problem - Improvements and Other Disruptions

5.1 Introduction

Encouraging results from the previous chapter show that match-up strategies deliver high performing schedules with a high stability when the arriving jobs are rush orders. In this chapter, a more general case, in which jobs may have different levels of urgency to be processed, is investigated. This generalisation requires further considerations in the algorithm design. Additionally, experimental results identify that match-up algorithms are suitable for improvements, since the associated genetic algorithm fitness function is able to control not only the schedule *Performance*, but also its *Stability*. This chapter emphasises the validity of hypothesis 1 described in Chapter 1.

The job shop scheduling problem presented by Sherwood Press - Nottingham, UK, is investigated in this chapter and the goal is to check the flexibility of match-up strategies under different types of disruption to achieve highly stable and good quality schedules.

The remaining of this chapter is organised as follows. Section 5.2 describes the match-up algorithm for jobs with different levels of urgency, referred to as normal orders. Section 5.3 analyses the proposed improvements on the genetic algorithm, which includes a comparison between the original version and improved one. Additionally, normal orders are statistically analysed and their results are compared with the ones presented by rush orders, in order to identify strengths of the proposed strategies. Sections 5.4 and 5.5 discuss and summarise the conclusions of this chapter.

5.2 Match-up Strategies for Normal Orders

This section introduces match-up strategies for the dynamic scheduling of normal orders. This type of orders set jobs to be processed before their due-date and they have the same priority of jobs already allocated in the current schedule. The pseudocode of the proposed match-up algorithm for normal orders is given in Figure 5.1. This algorithm resemble the previous one for rush orders, because they both define three phases: (1) the rescheduling horizon definition, in steps 1–2; (2) the subproblem definition, in steps 3–6; and (3) the integration of schedules in step 7. The main difference between them is the way how idle times are collected on machines in step 2. Note that reference points, such as $initialStart$, $startPoint$ and $endPoint$, are renamed in order to properly illustrate the rationale of the

```
Input: An initial schedule S, a new job $j$ with due date $d_j$, initialPoint
Output: A new schedule with job $j$ integrated

1. Let feasiblePoint be the latest completion time of operations whose processing time is crossed
   by initialPoint.
2. Calculate limitingPoint by collecting idle time starting from the due date of the new job $d_j$ in
   a backwards fashion towards the feasiblePoint.
     i  If $limitingPoint \geq feasiblePoint$
          - Let the rescheduling horizon be the time window between $limitingPoint$ and the due
            date $d_j$.
     ii Else
          - Recalculate limitingPoint by collecting idle times starting from feasiblePoint in a
            forwards direction towards the due date $d_j$.
          - Let the rescheduling horizon be the time window between $feasiblePoint$ and
            $limitingPoint$.
3. Let O be the set of operations within the rescheduling horizon, plus the operations of job $j$.
4. Update the release time and due dates of jobs in O so as to lie within the rescheduling horizon.
5. Let operations in O define a new scheduling problem $S'$.
6. Solve $S'$ using GA.
7. Integrate $S'$ into S checking and removing overlaps.
```

Figure 5.1: Pseudocode of the match-up algorithm for normal orders

proposed algorithm.

Figure 5.2(a) shows a new example for step 1, in which a new job 10 with due-date d_j has four operations subject to a precedence constraints. The first operation has to be processed either on M1 or M2, and then, after completing this execution, the next operation can be started on M3, and then on M4, and finally on M5. The company policy of 48 hours of no changes is applied and operations already started at this point must complete their processing before the rescheduling process can start, which set *initialPoint* and *feasiblePoint* respectively, as highlighted in Figure 5.2(b) and Figure 5.2(c). This example follows the same idea applied for rush orders, in which the completion time of Job 2 on M2 set *feasiblePoint*.

The *limitingPoint*, which defines one of the boundaries of the rescheduling horizon, is

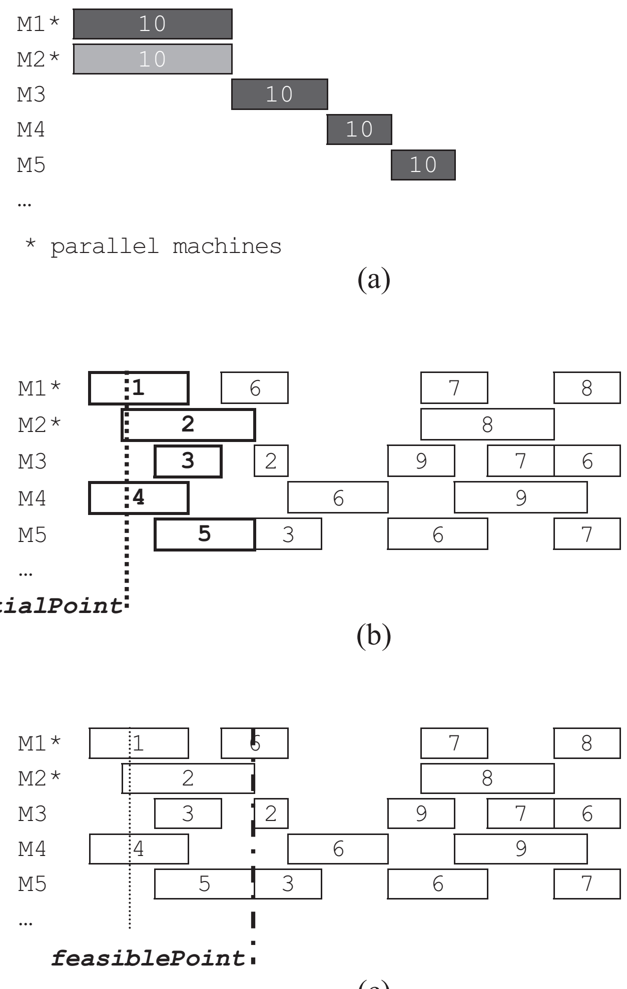


Figure 5.2: Example of a normal order arriving in the shop floor; (a) the new job processing requirements, (b) the calculation of *initialPoint* and (c) the *feasiblePoint* definition.

calculated either in a "backwards" or in a "forwards" direction depending on whether or not the amount of time between *feasiblePoint* and the due date of the new job, d_j, is large enough to accommodate the arriving job. If there is enough time to accommodate new job j, then the *limitingPoint* is calculated by collecting idle time in backwards direction starting from time d_j towards the $feasiblePoint$ (step 2.i). Time is accumulated until it can accommodate job j and defines the point refereed as $limitingPoint$. In this case, the rescheduling horizon is defined by the $limitingPoint$ and the due date d_j of job j. In the second case, where there is not enough time between the $feasiblePoint$ and d_j to contain new job j, the $limitingPoint$ is calculated by accumulating idle time on machines, but this time starting from the $feasiblePoint$ in a forwards direction towards the due date of job j (step 2.ii). As in the first case, the $limitingPoint$ labels the point when enough time to accommodate job j has been accumulated and the rescheduling horizon is defined by the $feasiblePoint$ and the $limitingPoint$. Note that, in this case, the due date of job j falls somewhere in between $feasiblePoint$ and $limitingPoint$. The match-up algorithm first tries to accumulate idle time in a backwards fashion, if this fails, i.e. if the accumulated time falls earlier than the $feasiblePoint$, then the idle time is accumulated in a forwards fashion.

Four additional strategies BW1– BW4 are introduced for the accumulation of idle times in a backwards fashion as illustrated with an example in Figure 5.3. Note that this example follows the same idea of the one presented for rush orders in the previous chapter, but idle times are collected from the due date of job j. BW1– BW2 collect partial and continuous time windows, respectively, as shown in Figure 5.3(b) and (c). On the other hand, BW3– BW2 collect idle as BW1– BW2 but they also consider precedence constraints imposed to the new job operations, as in Figure 5.3(d) and (e). Since idle times are collected on parallel machines, BW1– BW4 are extended to S1– S8, in which S1– S4 consider the latest time point and S5– S8 consider the earliest time point.

Figure 5.4(a) demonstrates idle times being collected for the new job j present in Figure 5.2(a). Since the accumulated times are on machines M1 and M2, which are parallel machines, there are two options for the definition of *limitingPoint*. The first option, using S1, is to consider the latest time point, as in Figure 5.4(b), in which a slightly smaller rescheduling horizon is defined. The other option, using $S5$, is to consider the earliest

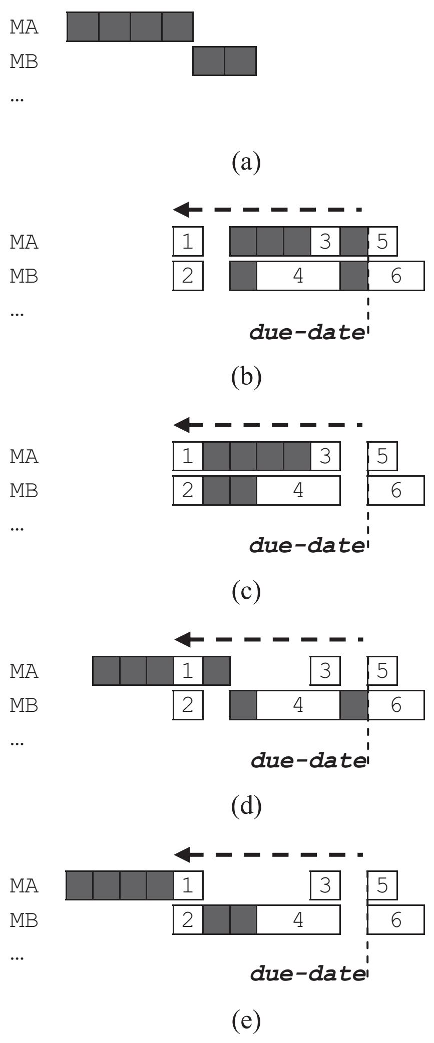
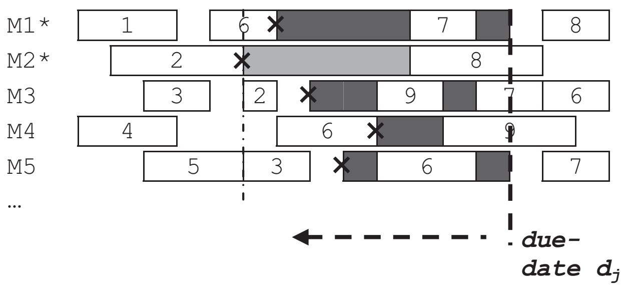


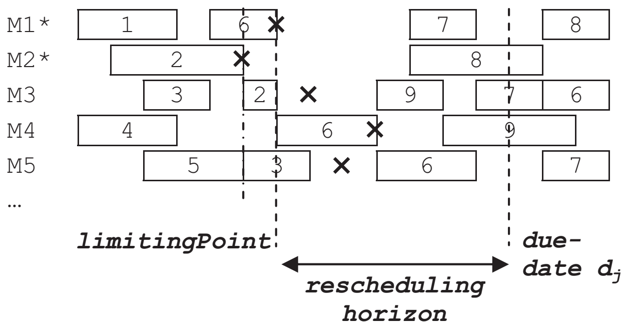
Figure 5.3: Example of the collection of idle times; (a) the new job requirements, (b) idle time collection using strategy BW1, (c) BW2, (d) BW3 and (e) BW4.

time point, as in Figure 5.4(c). The collection of idle times is also illustrated in Figure 5.6. In this case, the job to be inserted requires one more unit of processing time on M1, as in Figure 5.5(a). In this example the time accumulated in a backwards fashion using $S1$ in Figure 5.5(c) and $S5$ in Figure 5.5(d), is insufficient to accommodate the new job and the $limitingPoint$ is earlier than the $feasiblePoint$. In this case, time has to be accumulated in a forward fashion, as illustrated in Figure 5.6(a). This can be done using any of the strategies $S1 - S8$. For example, Figure 5.6(b) and Figure 5.6(c) show what the rescheduling horizon would be if $S1$ and $S5$ were used, respectively.

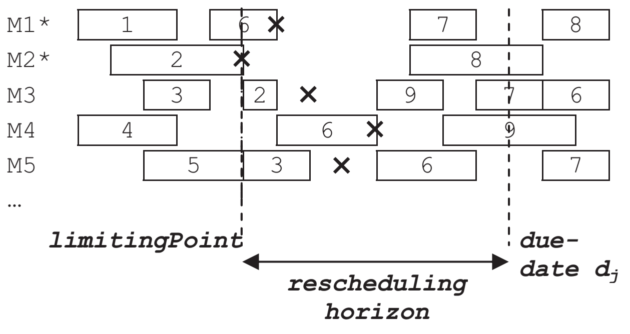
The remaining rescheduling actions, steps 3–7 from the algorithm in Figure 5.1, follow the same pattern presented for rush orders in the previous chapter. In step 3, the set



(a)



(b)



(c)

Figure 5.4: Example of the calculation of the rescheduling horizon; (a) the collection of idle time on machines, (b) the backwards calculation of *limitingPoint* using strategy S1 and (c) the backwards calculation of *limitingPoint* using strategy S5.

O aggregates affected operations within the rescheduling horizon with new job j. In step 4, both release time and due date of jobs in O are updated, in order to keep operations within the rescheduling horizon. In steps 5–6, a new scheduling subproblem S' is defined and solved using the previously developed genetic algorithm. In step 7, the feasibility of the schedule is verified, in which overlaps between the generated partial schedule and the initial one are solved by shifting operations to the right when necessary.

The algorithm presented in Figure 5.1 is able to manage normal and rush orders, since

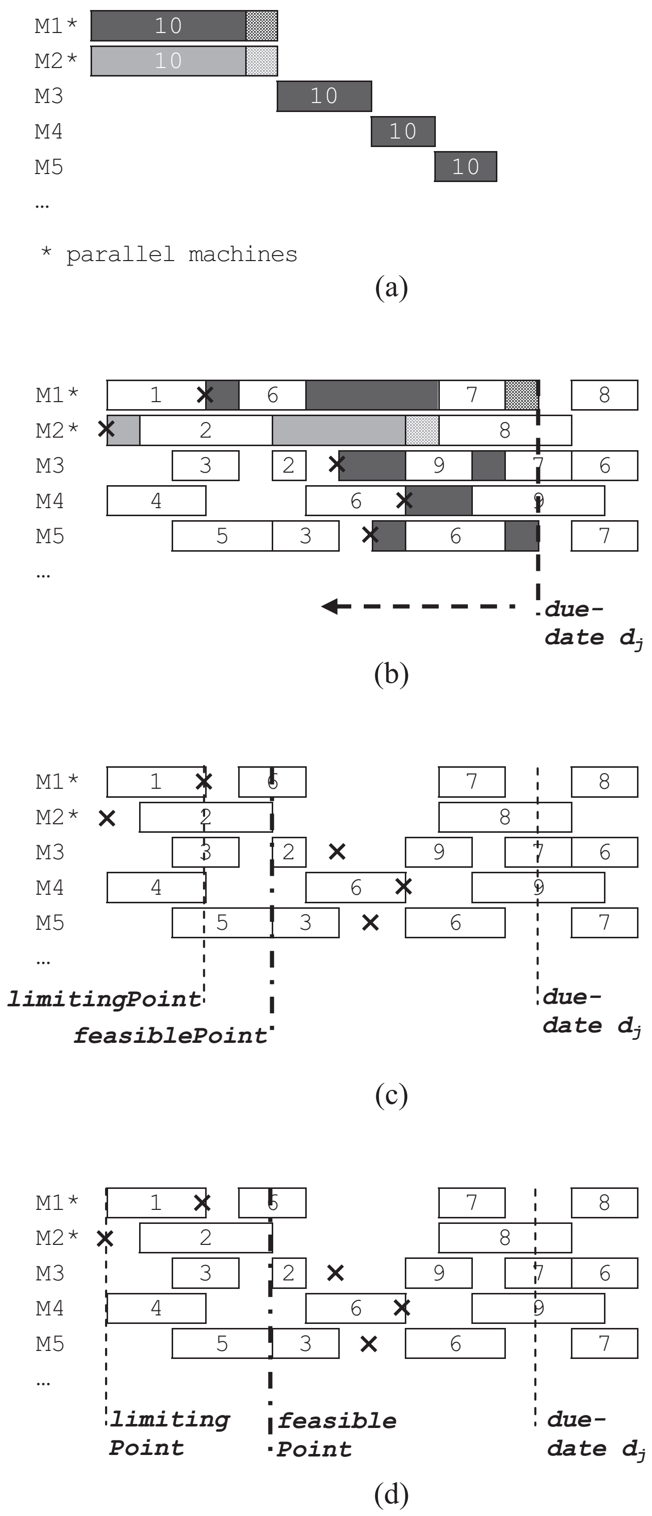
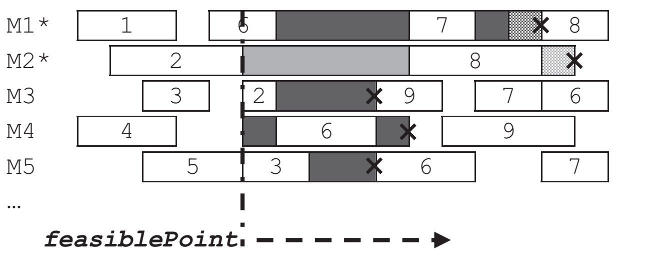
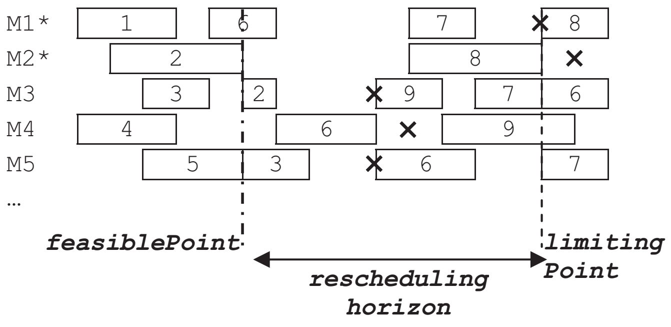


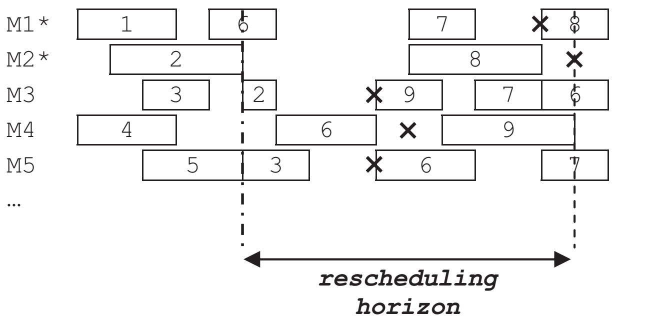
Figure 5.5: Example of the calculation of the rescheduling horizon; (a) new job $j = 10$ processing requirements, from those in Figure 5.2, extended by 1 time unit, (b) the collection of idle time on machines from the due date d_j, (c) the backwards calculation of *limitingPoint* using strategy $S1$ and (d) the backwards calculation of *limitingPoint* using strategy $S5$.



(a)



(b)



(c)

Figure 5.6: Example of the calculation of the rescheduling horizon; (a) the collection of idle time on machines from *feasiblePoint*, (b) the forwards calculation of *limitingPoint* using strategy *S*1 and (c) the forwards calculation of *limitingPoint* using strategy *S*5.

idle times are collected in both directions, backwards and forwards fashion. The only additional requirement is that rush orders must necessarily pass through step 2.ii, which sets idle times to be collected in a forward way. Consequently, a more general match-up algorithm, which is able to manage both types of arriving orders, is shown in Figure 5.7.

```
Input: An initial schedule $S$, a new job $j$ with due date $d_j$, $initialPoint$
Output: A new schedule with job $j$ integrated

1. Let feasiblePoint be the latest completion time of operations whose processing time is crossed
   by initialPoint.
2. Calculate limitingPoint by collecting idle time starting from the due date of the new job $d_j$ in
   a backwards fashion towards the feasiblePoint.
    i  If $limitingPoint \geq feasiblePoint$ and new job $j$ is a normal order
        - Let the rescheduling horizon be the time window between $limitingPoint$ and the due
          date $d_j$.
    ii Else
        - Recalculate limitingPoint by collecting idle times starting from feasiblePoint in a
          forwards direction towards the due date $d_j$.
        - Let the rescheduling horizon be the time window between $feasiblePoint$ and
          $limitingPoint$.
3. Let $O$ be the set of operations within the rescheduling horizon, plus the operations of job $j$.
4. Update the release time and due dates of jobs in $O$ so as to lie within the rescheduling horizon.
5. Let operations in $O$ define a new scheduling problem $S'$.
6. Solve $S'$ using GA.
7. Integrate $S'$ into $S$ checking and removing overlaps.
```

Figure 5.7: Pseudocode of the match-up algorithm for new orders

5.3 Experiments on Real World Data

Data sets obtained from Sherwood Press are used to test the *Performance* and *Stability* of the developed match-up algorithm shown in Figure 5.7. The data of the production orders of several months were used to produce schedules. In each instance, newly arriving jobs were randomly generated taking into account the same three parameters, sat, $insTime$ and $jobSize$, defined in the previous chapter. The combination of these parameters sets again a total of 45 analysed instances.

Initial schedules, previously selected for chapter 4, are used to here to insert the newly arriving jobs. Match-up strategies S1-S8, total rescheduling (T), right shift rescheduling (RS) and insertion in the end (E) are responsible again to integrate the new jobs into initial schedules. Both *Performance* and *Stability* are recorded for each strategy, each type of instance for each arriving job.

Features as population size, number of generations, crossover and mutation rates for the genetic algorithm were kept unchanged. Machine specifications, including hardware and software, also remain the same.

As mentioned in the previous chapter, improvements are proposed for the insertion of rush orders. These results are analysed and statistically validated before the investigation of normal orders. The following subsections formally discuss the obtained results.

5.3.1 Rush Orders - improved version

Results from experiments in chapter 4 show that there is a conflict between *Stability* and *Performance*, which means that schedules with a good *Stability* are typically poor regarding *Performance*, and vice-versa. A certain level of conflict is, of course, natural, since *Stability* and *Performance* are two conflicting criteria. The applied fitness function was optimising only the *Performance* during the rescheduling process and reasonable *Stability* results were found due to match-up strategies modifying only a part of the schedule. However, these results can be improved by optimising, instead of the *Performance*, the sum of the *makespan* and the *Stability* measures, previously introduced as SG_{Make} and $Stability$, expressions 4.2 and 4.5, from the previous chapter, respectively. Whereas it is natural to consider the $Stability$ during rescheduling, the use of the *makespan* objective rather than the $Performance$ is justified as follows.

The *Performance* of schedules is negatively correlated with the $Delay$ that they incur after rescheduling. This means that by considering the $Performance$ as fitness function for rescheduling, the new schedules within the rescheduling horizon are of a good local quality, but, once these are integrated into the initial schedule, the *Delay* incurred lessens the $Performance$ of the whole schedule. In order to minimise these $Delay$ values, it is necessary that new schedules within the rescheduling horizon are as short as possible; and this is achieved by minimising the *makespan*. Note that the proposed new settings also impose a double control of $Stability$, since expression 4.5 is now part of the fitness function and match-up strategies still modify only a part of the original schedule.

This subsection follows the same methodology used to analyse the original algorithm for rush orders. First, a complete statistical analysis regarding *Performance* and *Stability* of this new version is investigated. Subsequently, a comparison between the original algorithm and this improved version is presented in order to validate the expected improvements.

As expected, similar results on the statistical analysis are found, in which match-up algorithms are comparable to the right shift and insertion in the end strategies with respect to *Stability* and as good as the total rescheduling strategy with respect to *Performance*. More details about these results are discussed next.

The trade-offs between *Performance* and *Stability* are presented in Figure 5.8, in which samples between 0.3 and 0.45 for *Performance,* and between 0.7 and 0.9 for *Stability* are considered outliers. A more concentrated variation in *Stability* is now observed due to the new GA settings delivering highly stable schedules. Again, no positive or negative effects on *Performance* is observed when the *Stability* is increased, or vice-versa, which means that stable and good quality schedules are still delivered.

Table 5.1 summarises average and standard deviation results for *Performance* and *Stability* achieved by the investigated strategies for the different problem parameters. Each cell has an average result of 10 times executing the rescheduling algorithm. Additionally, the best strategies are highlighted in bold for each type of instance and numbers between brackets show the number executions done for each instance. Once again, match-up strategies combine good features of T, RS and E, since they deliver good results for both *Performance* and *Stability*. They are comparable to the best results presented by T and E, with respect to *Performance* and *Stability*, respectively. Note that E now represents the optimal *Stability* because it keeps both sequence and time of current operations. Among match strategies, S1-S4 outperform S5-S8 in most groups of instances. These results are statistically verified next.

The summary of the ANOVA is given in Table 5.2. All “main effects” and “interactions” involving the parameter Strategy have influences on both *Performance* and *Stability*

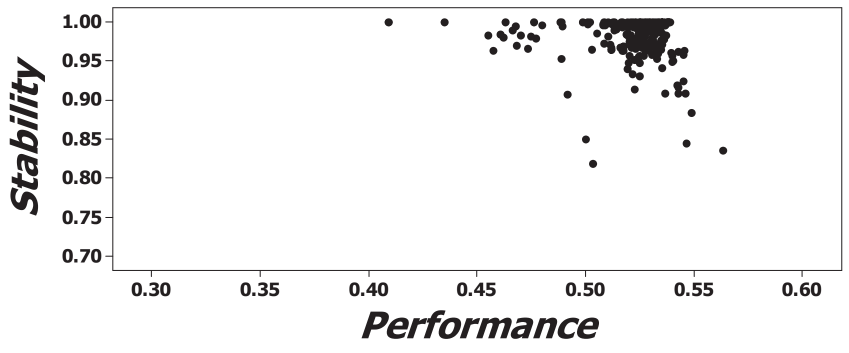


Figure 5.8: Trade-offs between *Performance* and *Stability* for rush orders - improved version

Table 5.1: Average and standard deviation values for *Performance* and *Stability* obtained by the rescheduling strategies for rush orders - improved version

		Performance - Rescheduling strategy										
		S1	S2	S3	S4	S5	S6	S7	S8	T	RS	E
sat	(15) low	0.533	0.533	0.533	0.532	0.532	0.532	0.531	0.529	**0.537**	0.528	0.510
	(15) medium	0.526	0.526	**0.527**	**0.527**	0.525	0.525	0.525	0.526	0.523	0.523	0.481
	(15) high	0.525	0.527	0.528	0.528	0.513	0.512	0.523	0.509	**0.529**	0.513	0.519
jobSize	(9) 1	0.535	0.535	0.535	0.535	0.532	0.533	0.532	0.532	**0.539**	0.532	0.526
	(9) 2	0.531	0.532	0.532	**0.533**	0.528	0.526	0.530	0.525	0.529	0.529	0.516
	(9) 3	**0.529**	**0.529**	**0.529**	**0.529**	0.523	0.522	0.527	0.521	0.523	0.524	0.504
	(9) 4	0.525	0.526	0.526	0.526	0.519	0.519	0.523	0.517	**0.531**	0.515	0.491
	(9) 5	0.521	0.523	0.523	0.522	0.515	0.516	0.519	0.513	**0.526**	0.507	0.479
insTime	(15) beginning	0.533	0.532	**0.535**	0.532	0.517	0.518	0.525	0.511	0.533	0.518	0.493
	(15) middle	0.525	0.528	0.526	0.528	0.527	0.526	0.528	0.528	**0.530**	0.521	0.493
	(15) end	**0.526**	**0.526**	**0.526**	**0.526**	**0.526**	**0.526**	**0.526**	**0.526**	**0.526**	**0.526**	0.525
(45) total average		0.528	0.529	0.529	0.529	0.523	0.523	0.526	0.521	**0.530**	0.521	0.503
(45) standard deviation		0.004	0.004	0.004	0.004	0.006	0.006	0.004	0.008	**0.005**	0.007	0.016
		Stability - Rescheduling strategy										
		S1	S2	S3	S4	S5	S6	S7	S8	T	RS	E
sat	(15) low	0.992	0.992	0.993	0.993	0.993	0.993	0.995	0.993	0.952	0.997	**1.000**
	(15) medium	0.983	0.983	0.983	0.983	0.983	0.983	0.981	0.982	0.972	0.998	**1.000**
	(15) high	0.995	0.995	0.995	0.995	0.987	0.986	0.982	0.981	0.920	0.998	**1.000**
jobSize	(9) 1	0.998	0.998	0.998	0.998	0.992	0.992	0.992	0.990	0.942	0.999	**1.000**
	(9) 2	0.996	0.997	0.996	0.996	0.993	0.992	0.990	0.991	0.956	0.998	**1.000**
	(9) 3	0.996	0.996	0.996	0.996	0.994	0.995	0.991	0.991	0.953	0.997	**1.000**
	(9) 4	0.985	0.985	0.986	0.986	0.985	0.984	0.981	0.981	0.952	0.997	**1.000**
	(9) 5	0.975	0.976	0.976	0.976	0.975	0.974	0.975	0.972	0.937	0.996	**1.000**
insTime	(15) beginning	0.990	0.991	0.991	0.990	0.983	0.982	0.982	0.981	0.909	0.995	**1.000**
	(15) middle	0.990	0.990	0.990	0.991	0.990	0.990	0.985	0.985	0.949	0.998	**1.000**
	(15) end	0.990	0.990	0.990	0.990	0.990	0.990	0.990	0.990	0.986	**1.000**	**1.000**
(45) total average		0.990	0.990	0.990	0.990	0.988	0.987	0.986	0.985	0.948	0.997	**1.000**
(45) standard deviation		0.006	0.006	0.006	0.006	0.006	0.006	0.006	0.006	0.020	0.001	**0.000**

Table 5.2: Results of the ANOVA test for rush orders - improved version

		Performance		*Stability*	
		F value	P value	F value	P value
Main effects					
	Strategy	27.08	≤ 0.05	85.48	≤ 0.05
	sat	47.61	≤ 0.05	16.07	≤ 0.05
	jobSize	52.84	≤ 0.05	48.62	≤ 0.05
	insTime	5.55	≤ 0.05	42.39	≤ 0.05
Interactions					
	Strategy**sat*	9.43	≤ 0.05	13.73	≤ 0.05
	Strategy**jobSize*	1.96	≤ 0.05	1.47	≤ 0.05
	Strategy**insTime*	8.64	≤ 0.05	19.33	≤ 0.05
	R^2	0.74		0.84	

metrics. A high variability on results is confirmed by the relatively large R^2 values. More details about ANOVA are discussed next. The Bonferroni's correction for pairwise comparisons is carried out to identify strategies which deliver superior results. These comparisons are given in Figure 5.9 with *Performance* results below the diagonal, and above for *Stability*.

As expected, RS and E deliver good *Stability* and poor *Performance*, while T has poor *Stability* and good *Performance*. Match-up strategies combine these good features, since strategies S1-S4 are statistically non-distinguishable from T for *Performance* and strategies S1-S8 produce comparable results to RS and E for *Stability*. Note that S1-S4 still deliver better *Performance* than S5-S8 because they set smaller rescheduling horizons, which affect a fewer number of operations on the rescheduling process, avoiding

Stability

Performance

	S1	S2	S3	S4	S5	S6	S7	S8	T	RS	E
S1	■	= 0.04 1.00	= 0.07 1.00	= 0.11 1.00	= -1.31 1.00	= -1.49 1.00	= -2.31 1.00	= -2.72 0.36	← -22.70 ≤0.05	↑ 3.90 ≤0.05	↑ 5.27 ≤0.05
S2	= 0.53 1.00	■	= 0.03 1.00	= 0.07 1.00	= -1.35 1.00	= -1.54 1.00	= -2.35 1.00	= -2.77 0.31	← -22.75 ≤0.05	↑ 3.86 ≤0.05	↑ 5.23 ≤0.05
S3	= 0.69 1.00	= 0.16 1.00	■	= 0.04 1.00	= -1.38 1.00	= -1.57 1.00	= -2.38 0.96	= -2.79 0.29	← -22.78 ≤0.05	↑ 3.83 ≤0.05	↑ 5.20 ≤0.05
S4	= 0.49 1.00	= -0.04 1.00	= -0.20 1.00	■	= -1.42 1.00	= -1.61 1.00	= -2.42 0.86	= -2.83 0.26	← -22.82 ≤0.05	↑ 3.79 ≤0.05	↑ 5.16 ≤0.05
S5	= -2.84 0.25	↑ -3.37 ≤0.05	↑ -3.53 ≤0.05	↑ -3.33 ≤0.05	■	= -0.19 1.00	= -1.00 1.00	= -1.42 1.00	← -21.40 ≤0.05	↑ 5.21 ≤0.05	↑ 6.58 ≤0.05
S6	= -2.95 0.17	↑ -3.48 ≤0.05	↑ -3.64 ≤0.05	↑ -3.44 ≤0.05	= -0.11 1.00	■	= -0.81 1.00	= -1.23 1.00	← -21.21 ≤0.05	↑ 5.39 ≤0.05	↑ 6.76 ≤0.05
S7	= -1.10 1.00	= -1.63 1.00	= -1.79 1.00	= -1.59 1.00	= 1.75 1.00	= 1.85 1.00	■	= -0.41 1.00	← -20.40 ≤0.05	↑ 6.21 ≤0.05	↑ 7.58 ≤0.05
S8	↑ -4.14 ≤0.05	↑ -4.67 ≤0.05	↑ -4.83 ≤0.05	↑ -4.63 ≤0.05	= -1.29 1.00	= -1.19 1.00	= -3.04 0.13	■	← -19.98 ≤0.05	↑ 6.62 ≤0.05	↑ 7.99 ≤0.05
T	= 1.08 1.00	= 0.55 1.00	= 0.40 1.00	= 0.59 1.00	← 3.93 ≤0.05	← 4.04 ≤0.05	= 2.18 1.00	← 5.22 ≤0.05	■	↑ 26.60 ≤0.05	↑ 27.97 ≤0.05
RS	↑ -4.05 ≤0.05	↑ -4.58 ≤0.05	↑ -4.73 ≤0.05	↑ -4.54 ≤0.05	= -1.20 1.00	= -1.09 1.00	= -2.95 0.18	= 0.09 1.00	↑ -5.13 ≤0.05	■	= 1.37 1.00
E	↑ -15.30 ≤0.05	↑ -15.83 ≤0.05	↑ -15.99 ≤0.05	↑ -15.79 ≤0.05	↑ -12.46 ≤0.05	↑ -12.35 ≤0.05	↑ -14.20 ≤0.05	↑ -11.16 ≤0.05	↑ -16.38 ≤0.05	↑ -11.25 ≤0.05	■

Figure 5.9: Mean pairwise comparisons of *Performance* and *Stability* for rush orders - improved version

the presence of large delays. Similar *Stability* results are obtained among all match-up strategies due to the applied new settings on the fitness function. These results are graphically shown in Figure 5.10(a) and 5.10(b) for *Performance* and *Stability*, respectively. Note that the *Stability* measure is now considering both sequence (Sta_1) and time (Sta_2) deviations, as described by the expressions 4.3 and 4.4 from chapter 4. Consequently, E delivers superior *Stability* results then RS, since E keeps the initial schedule intact, while RS requires some time deviations. Figure 5.11(a) shows the expected larger delays delivered by S5-S8 and RS. Additionally, Figure 5.11(b) confirms that match-up strategies are still efficient, since only 10 seconds are required on average to reschedule affected operations.

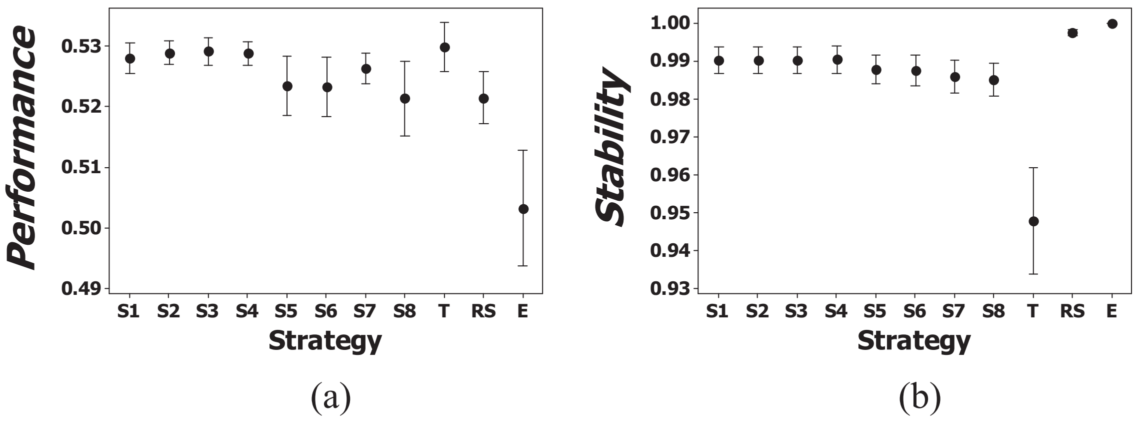


Figure 5.10: Overall results obtained by each rescheduling strategy; the x-axis shows the strategy; the y-axis shows the mean (dot) and 95% confidence interval (vertical bars) of *Performance* (a) and *Stability* for rush orders - improved version

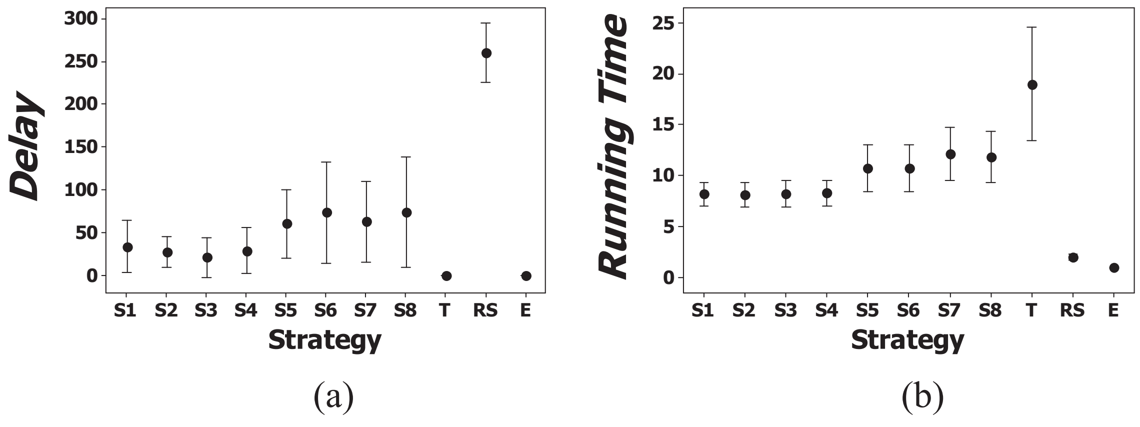


Figure 5.11: Overall results obtained by each rescheduling strategy on *Delay* (a) and *Running Time* (b) for rush orders - improved version

The parameters sat, *jobSize* and $insTime$ have a significant influence on both *Performance* and *Stability* over different applied rescheduling strategies, as shown by the ANOVA test results in Table 5.2. These effects are illustrated in Figure 5.12. As expected, *Performance* and *Stability* values decrease when rescheduling is done on highly saturated schedules, when the arriving job requires many operations, or when the rescheduling occurs at the beginning or in the middle of the schedule.

The ANOVA results in Table 5.2 together with the averages present in Figure 5.10 identify that the interaction between problem parameters are significant, which indicates that some strategies are better coping with certain conditions than others. Regarding *sat* and $insTime$, for instance, S1-S4 are much preferred when the rescheduling occurs either on highly saturated schedules or at the beginning of schedules. Under these scenarios, the $Delay$ inserted by the rescheduling algorithm will have a stronger effect than if rescheduling occurs at the end of the schedule or in less saturated schedules. Regarding *jobSize*, S1-S4 still superior than S5-S8 when jobs with a small number of operations are inserted, because they define smaller rescheduling horizons and, consequently, smaller delay values, leading to a better overall *Performance* and *Stability*.

Once again, match-up strategies leads to good quality schedules, which are highly stable as the right shift and insertion in the end strategies and as good as the total rescheduling strategy with respect to *Performance*. Strategies S1-S4 are slightly superior to the other match-up approaches and deliver the most consistent results under different problem scenarios, as demonstrated by Table 5.1 and Figure 5.10.

5.3.2 Comparison between Rush Orders using the original GA and the Improved Version

The improved version of the match-up algorithm so far have shown that it remains effective for inserting rush orders after statistical multi-comparison tests and analysis of variance were carried out on the previous subsection. Figure 5.13 presents comparative results between the original rescheduling algorithm introduced in chapter 4, which uses only the $Performance$ as the function to optimise during rescheduling (O), and the proposed improved version, that uses the sum of the makespan and $Stability$ (I).

The results obtained by the new version of the match-up algorithm (I) are remarkably superior to the earlier ones (O) with respect to both *Performance* and *Stability*. This improvements are due to smaller *Delay* values being generated with the minimisation of makespan and the effective control of *Stability* which is now integrated into the fitness function. Figure 5.14 shows the overall reduction of *Delay* values obtained by the im-

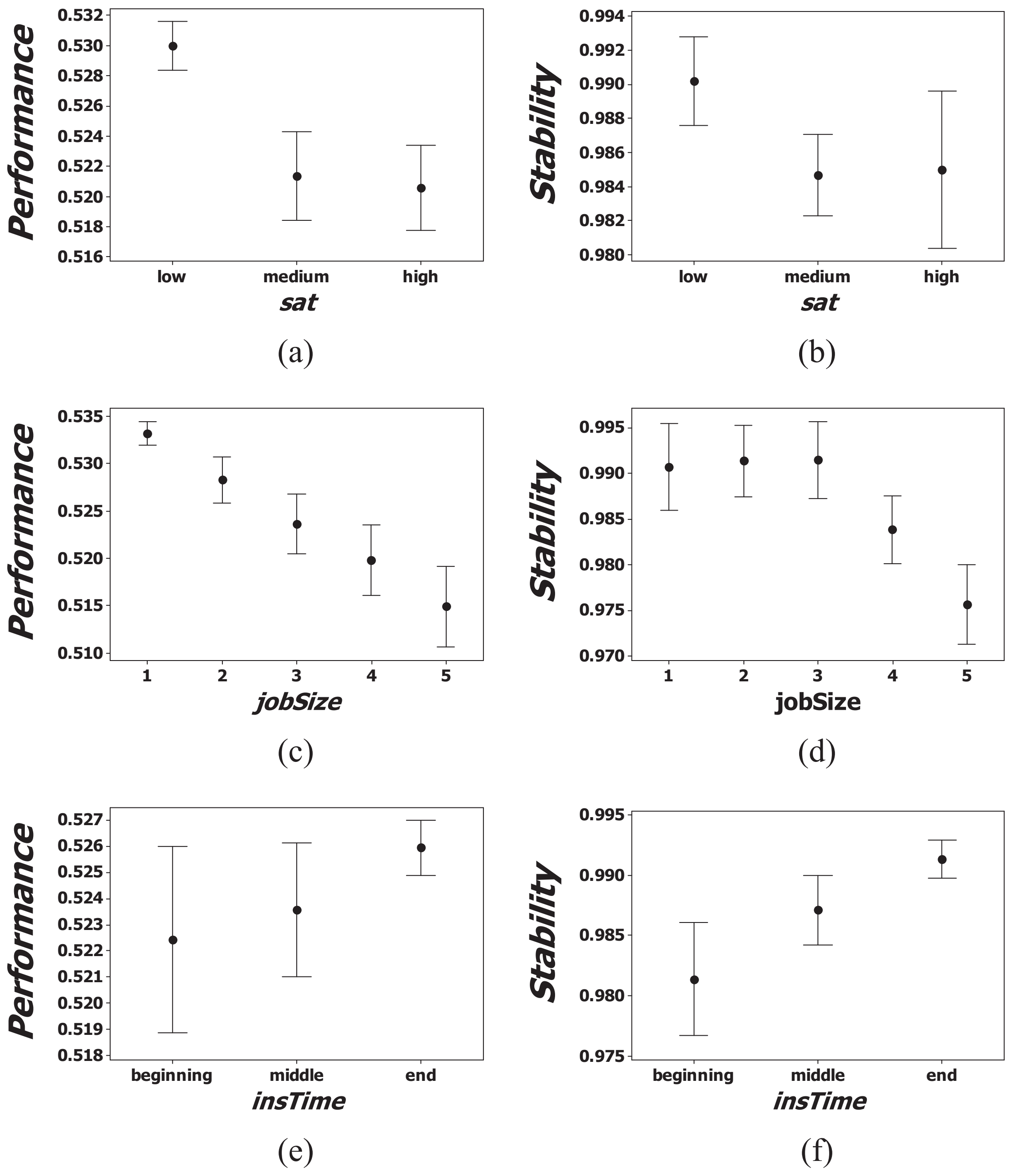
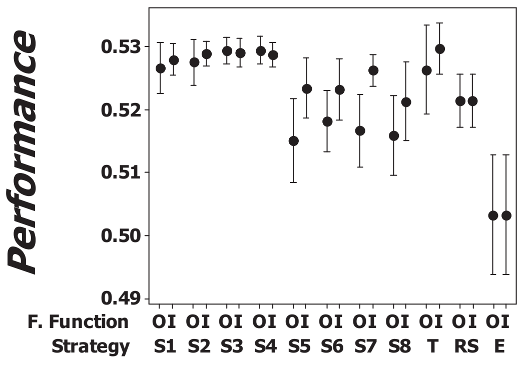
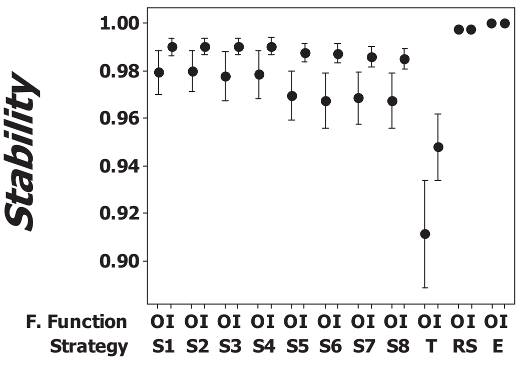


Figure 5.12: Main effects on *Performance* (a), (c), (e) and *Stability* (b), (d), (f) due to Strategy (a)-(b), *jobSize* (c)-(d) and *insTime* (e)-(f) for rush orders - improved version



(a)



(b)

Figure 5.13: *Performance* and *Stability* values obtained with the original fitness function (O) and the improved version (I)

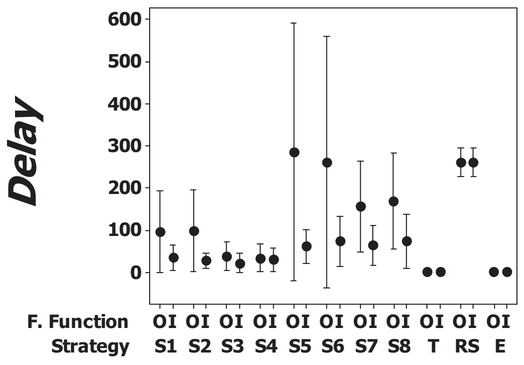


Figure 5.14: *Delay* values obtained with the original fitness function (O) and the improved version (I)

proved version (I) compared with the original one (O) for all rescheduling strategies. The negative correlation between *Delay* and *Performance* and the effective *Stability* control identify that the improved version of the match-up algorithm leads to schedules with better quality results. Note that no considerable improvements in *Performance* are observed for S3-S4, since their produced delays were already small.

5.3.3 Normal Orders

Normal orders are more flexible disturbances because their insertion are based on the job due-date, which gives a time window to make repair decisions.

This subsection follows the same methodology used to analyse the algorithm for rush orders, in which a complete statistical analysis is carried out to investigate both *Performance* and *Stability* of the repaired schedules. Subsequently, a comparison between rush and normal orders is also presented in order to identify the strengths of the match-up algorithm dealing with jobs with different levels of urgency. Note that the same settings of the genetic algorithm previously used in Section 5.3.1 is applied in this experiment because they deliver better overall results regarding all analysed metrics.

Similar results on the statistical analysis are found, in which match-up algorithms are comparable to the right shift and insertion in the end strategies with respect to *Stability* and as good as the total rescheduling strategy with respect to *Performance*. More details regarding these results are discussed next.

The trade-offs between *Performance* and *Stability* are presented in Figure 5.15, in which samples between 0.3 and 0.45 for *Performance*, and between 0.7 and 0.9 for *Stability* are considered outliers. Again, a more concentrated variation on *Stability* is observed due to the GA settings still delivering highly stable schedules. As expected, no positive or negative effects on *Performance* is observed when the *Stability* is increased, or vice-versa, which means that stable and good quality schedules are still delivered.

Each instance of the problem is executed 10 times and both average and standard deviation results obtained by the investigated strategies are shown in Table 5.3. Best strategies are highlighted in bold for each analysed instance. Once again, S1-S8 combine strengths of T, RS and E, delivering good quality stable schedules, which are similar to T regarding *Performance* and comparable to RS and E regarding *Stability*. Among match strategies, S1-S4 outperform S5-S8 in some groups of instances and these results are statistically verified next.

Table 5.4 shows the ANOVA results, in which all “main effects” and “interactions” involving the parameter Strategy have influences on *Performance* and *Stability*. A high

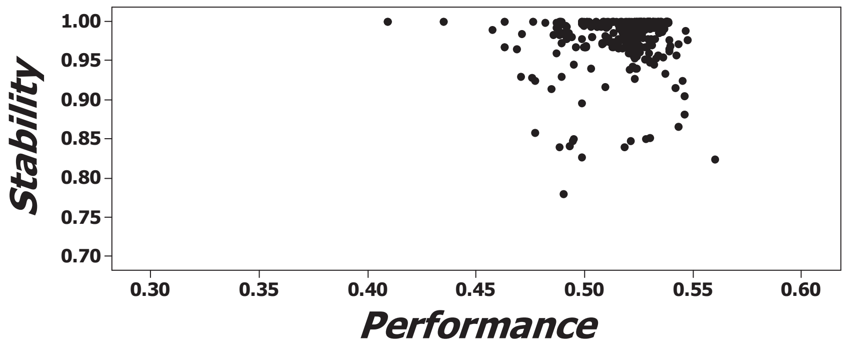


Figure 5.15: Trade-offs between *Performance* and *Stability* for normal orders

Table 5.3: Average and standard deviation values for *Performance* and *Stability* obtained by the rescheduling strategies for normal orders

		Performance - Rescheduling strategy										
		S1	S2	S3	S4	S5	S6	S7	S8	T	RS	E
sat	(15) low	0.528	0.528	0.526	0.526	0.532	0.533	0.529	0.529	**0.540**	0.524	0.510
	(15) medium	0.518	0.517	0.511	0.512	0.517	0.518	0.520	0.521	**0.523**	0.505	0.481
	(15) high	0.512	0.513	**0.526**	**0.526**	0.513	0.514	0.522	0.511	0.512	0.512	0.519
jobSize	(9) 1	0.524	0.524	**0.526**	**0.526**	0.521	0.524	0.525	0.525	**0.526**	0.525	**0.526**
	(9) 2	0.520	0.520	**0.531**	**0.531**	0.524	0.525	0.529	0.525	0.529	0.522	0.516
	(9) 3	0.522	0.523	0.521	0.522	0.524	0.524	0.527	0.521	**0.529**	0.517	0.504
	(9) 4	0.518	0.519	0.517	0.518	0.521	0.519	**0.523**	0.518	0.521	0.507	0.491
	(9) 5	0.512	0.512	0.510	0.510	0.514	0.515	0.515	0.512	**0.520**	0.496	0.479
insTime	(15) beginning	0.526	0.520	0.526	0.527	**0.529**	0.525	0.528	0.528	0.521	0.514	0.493
	(15) middle	0.512	0.519	0.523	0.523	0.512	0.518	0.523	0.511	**0.525**	0.513	0.493
	(15) end	0.520	0.520	0.514	0.513	0.521	0.521	0.521	0.521	**0.528**	0.513	0.525
(45) total average		0.519	0.520	0.521	0.521	0.521	0.521	0.524	0.520	**0.525**	0.513	0.503
(45) standard deviation		0.006	0.005	0.007	0.007	0.006	0.005	0.004	0.007	**0.007**	0.009	0.017
		Stability - Rescheduling strategy										
		S1	S2	S3	S4	S5	S6	S7	S8	T	RS	E
sat	(15) low	0.992	0.992	0.992	0.992	0.991	0.991	0.985	0.990	0.955	0.997	**1.000**
	(15) medium	0.985	0.985	0.984	0.985	0.984	0.984	0.983	0.983	0.967	0.997	**1.000**
	(15) high	0.986	0.993	0.994	0.994	0.977	0.989	0.979	0.981	0.846	0.998	**1.000**
jobSize	(9) 1	0.988	0.996	0.998	0.998	0.987	0.995	0.989	0.995	0.906	0.999	**1.000**
	(9) 2	0.994	0.997	0.998	0.998	0.994	0.996	0.994	0.989	0.932	0.999	**1.000**
	(9) 3	0.997	0.997	0.996	0.997	0.994	0.991	0.985	0.989	0.939	0.997	**1.000**
	(9) 4	0.986	0.986	0.986	0.986	0.978	0.985	0.981	0.977	0.925	0.997	**1.000**
	(9) 5	0.973	0.973	0.974	0.972	0.967	0.975	0.962	0.973	0.913	0.995	**1.000**
insTime	(15) beginning	0.991	0.991	0.990	0.990	0.990	0.990	0.986	0.989	0.925	0.995	**1.000**
	(15) middle	0.980	0.987	0.989	0.988	0.972	0.984	0.972	0.976	0.922	0.998	**1.000**
	(15) end	0.992	0.992	0.992	0.992	0.991	0.991	0.988	0.989	0.922	**1.000**	**1.000**
(45) total average		0.988	0.990	0.990	0.990	0.984	0.988	0.982	0.984	0.923	0.997	**1.000**
(45) standard deviation		0.007	0.007	0.007	0.008	0.009	0.006	0.009	0.007	0.031	0.002	**0.000**

Table 5.4: Results of the ANOVA test for normal orders

		Performance		*Stability*	
		F value	*P* value	*F* value	*P* value
Main effects					
	Strategy	14.33	≤ 0.05	169.44	≤ 0.05
	sat	90.99	≤ 0.05	60.71	≤ 0.05
	jobSize	41.93	≤ 0.05	55.91	≤ 0.05
	insTime	13.81	≤ 0.05	21.34	≤ 0.05
Interactions					
	Strategy**sat*	8.29	≤ 0.05	53.86	≤ 0.05
	Strategy**jobSize*	2.09	≤ 0.05	2.71	≤ 0.05
	Strategy**insTime*	7.58	≤ 0.05	1	≤ 0.05
	R^2	0.73		0.90	

variability on results is confirmed by the relatively large R^2 values. Additionally, the Bonferroni's correction for pairwise comparisons is carried out to identify strategies with superior results. These comparisons are given in Figure 5.16 with *Performance* results below the diagonal, and above for *Stability*.

Match-up strategies are still combining good features of T, RS and E, witout not exhibit their weaknesses, since all strategies are now statistically non-distinguishable from T for *Performance* and as good as RS and E for *Stability*. Note that, additionally, S2-S4 are statistically non-distinguishable from RS for *Stability*. Similar *Performance* and *Stability* results are obtained among all match-up strategies due to the effective control of *Delay* values by the GA fitness function. Note that the newly arriving jobs are inserted at different parts of the schedule based on the requirements of its due date. Consequently, there is a higher chance of using idle times present in the current schedule, which has a

Stability

Performance

	S1	S2	S3	S4	S5	S6	S7	S8	T	RS	E
S1	■	= 0.99 1.00	= 1.14 1.00	= 1.06 1.00	= -1.41 1.00	= 0.31 1.00	= -2.28 1.00	= -1.30 1.00	← -27.6 ≤0.05	↑ 4.17 ≤0.05	↑ 5.32 ≤0.05
S2	= 0.15 1.00	■	= 0.15 1.00	= 0.07 1.00	= -2.40 0.91	= -0.67 1.00	= -3.27 0.06	= -2.29 1.00	← -28.6 ≤0.05	= 3.19 0.08	↑ 4.33 ≤0.05
S3	= 0.88 1.00	= 0.73 1.00	■	= -0.08 1.00	= -2.55 0.60	= -0.82 1.00	← -3.42 ≤0.05	= -2.44 0.82	← -28.7 ≤0.05	= 3.04 0.13	↑ 4.18 ≤0.05
S4	= 1.05 1.00	= 0.90 1.00	= 0.17 1.00	■	= -2.47 0.74	= -0.74 1.00	← -3.34 ≤0.05	= -2.36 1.00	← -28.7 ≤0.05	= 3.12 0.10	↑ 4.26 ≤0.05
S5	= 0.83 1.00	= 0.68 1.00	= -0.05 1.00	= -0.22 1.00	■	= 1.73 1.00	= -0.87 1.00	= 0.11 1.00	← -26.2 ≤0.05	↑ 5.59 ≤0.05	↑ 6.73 ≤0.05
S6	= 1.09 1.00	= 0.94 1.00	= 0.21 1.00	= 0.04 1.00	= 0.26 1.00	■	= -2.59 0.53	= -1.62 1.00	← -27.9 ≤0.05	↑ 3.86 ≤0.05	↑ 5.00 ≤0.05
S7	= 2.48 0.71	= 2.34 1.00	= 1.60 1.00	= 1.44 1.00	= 1.66 1.00	= 1.40 1.00	■	= 0.98 1.00	← -25.3 ≤0.05	↑ 6.45 ≤0.05	↑ 7.60 ≤0.05
S8	= 0.46 1.00	= 0.31 1.00	= -0.42 1.00	= -0.59 1.00	= -0.37 1.00	= -0.63 1.00	= -2.02 1.00	■	← -26.3 ≤0.05	↑ 5.48 ≤0.05	↑ 6.62 ≤0.05
T	= 3.13 0.10	= 2.98 0.16	= 2.25 1.00	= 2.08 1.00	= 2.30 1.00	= 2.04 1.00	= 0.64 1.00	= 2.67 0.43	■	↑ 31.82 ≤0.05	↑ 32.97 ≤0.05
RS	↑ -3.33 ≤0.05	↑ -3.48 ≤0.05	↑ -4.22 ≤0.05	↑ -4.39 ≤0.05	↑ -4.17 ≤0.05	↑ -4.43 ≤0.05	↑ -5.82 ≤0.05	↑ -3.80 ≤0.05	↑ -6.46 ≤0.05	■	= 1.14 1.00
E	↑ -8.94 ≤0.05	↑ -9.09 ≤0.05	↑ -9.83 ≤0.05	↑ -10.0 ≤0.05	↑ -9.78 ≤0.05	↑ -10.0 ≤0.05	↑ -11.4 ≤0.05	↑ -9.41 ≤0.05	↑ -12.0 ≤0.05	↑ -5.61 ≤0.05	■

Figure 5.16: Mean pairwise comparisons of *Performance* and *Stability* for normal orders

positive effect on the reduction of delays. S1-S4 are again highlighted as the ones with the better overall results, as graphically shown in Figure 5.17(a) and 5.17(b) for *Performance* and *Stability*, respectively. Additionally, the *Delay* values delivered by all rescheduling strategies are presented in Figure 5.18. The running time parameter is not investigated again since they always deliver similar results, with all match-up strategies delivering reasonable execution times.

The ANOVA test results, present in Table 5.4, show that the parameters *sat*, *jobSize* and *insTime* have a significant influence on both *Performance* and *Stability* over different rescheduling strategies. These effects are graphically illustrated in Figure 5.19. As expected, *Performance* and *Stability* values decrease when rescheduling is done on highly saturated schedules, when the arriving job requires many operations, or when the rescheduling happens in the middle of the schedule. Note that rush orders start collecting

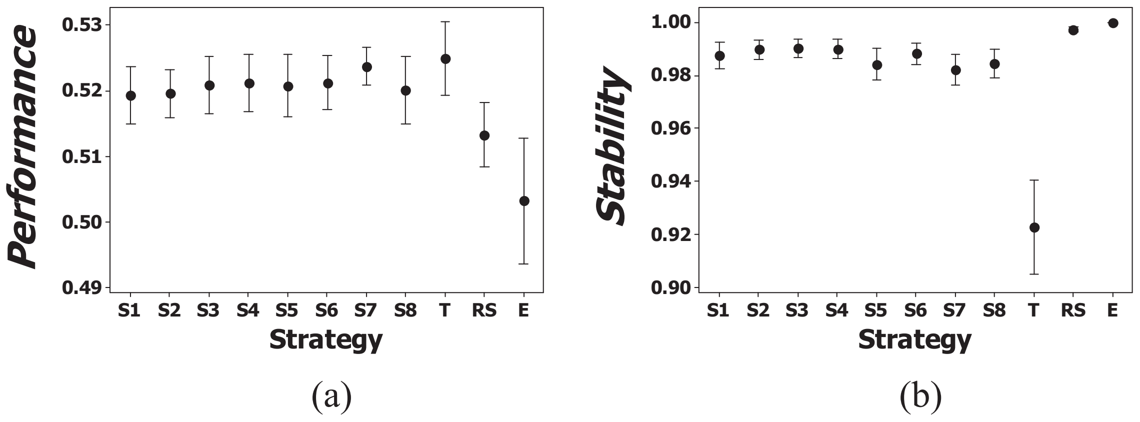


Figure 5.17: Overall results obtained by each rescheduling strategy; the x-axis shows the strategy; the y-axis shows the mean (dot) and 95% confidence interval (vertical bars) of *Performance* (a) and *Stability* (b) for normal orders

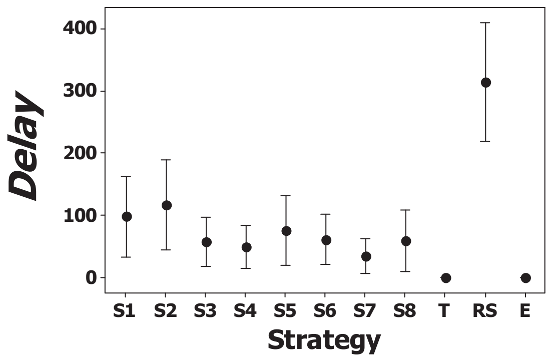


Figure 5.18: Overall results obtained by each rescheduling strategy on *Delay* for normal orders

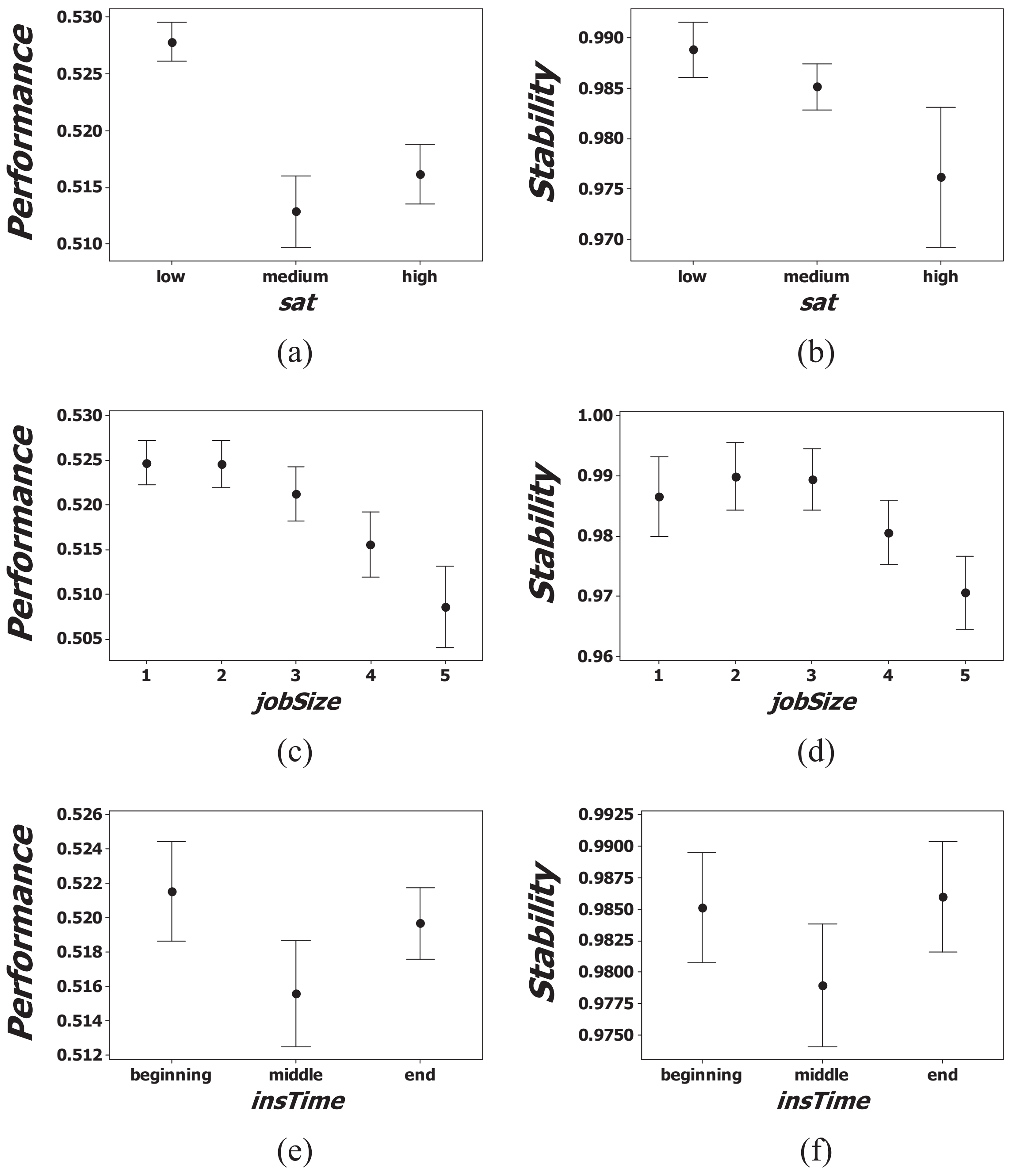


Figure 5.19: Main effects on *Performance* (a), (c), (e) and *Stability* (b), (d), (f) due to Strategy (a)-(b), *jobSize* (c)-(d) and *insTime* (e)-(f) for normal orders

idle times as soon as they arrive in the shop floor, which leads to higher saturation levels at the beginning and in the middle of the schedules. Contrarily, normal order start collecting idle times from due-dates, which mostly concentrate jobs in the middle of the schedule.

The ANOVA test also identify that the interaction between strategy and other problem parameters are significant, which indicates that some strategies are better coping with certain conditions than others. Regarding *sat* and *insTime*, for instance, S1-S4 are much preferred when the rescheduling occurs either on highly saturated schedules or in the middle of schedules. Under these scenarios, the *Delay* inserted by the rescheduling algorithm will have a stronger effect than if rescheduling occurs in less saturated schedules or at the end of the schedule. Regarding *jobSize*, S1-S4 still superior than S5-S8 when jobs with a small number of operations are inserted, because they define smaller rescheduling horizons and, consequently, smaller delay values, leading to a better overall *Performance* and *Stability*.

All match-up strategies are producing good quality schedules even when more flexible disturbances as normal orders occur in the shop floor. They still deliver highly stable solutions as the right shift and insertion in the end strategies and good *Performance* as the total rescheduling. In general, Strategies S1-S4 are slightly superior to the other match-up approaches and they deliver the most consistent results under different problem scenarios, as shown in Table 5.3 and Figure 5.17.

5.3.4 Comparison between Rush and Normal Orders

Jobs arriving in Sherwood press shop floor are either classified as rush or normal orders. It is expected to get superior results for normal ones due to its flexible nature, i.e. different due-dates, which gives a larger time window to make repair decisions when compared with rush ones. Rush orders (R) are analysed and their results for *Performance* and *Stability* are compared with the ones obtained by normal orders (N) for each rescheduling strategy, as shown in Figure 5.20 (a) and (b), respectively.

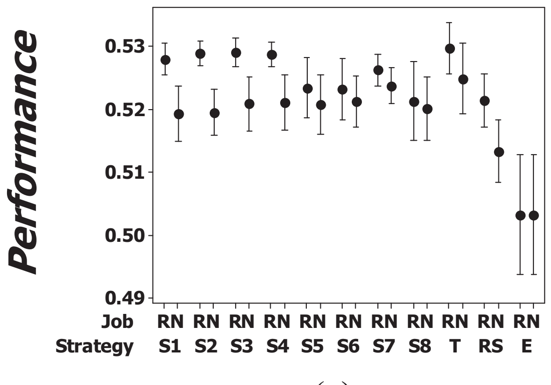
Surprisingly, rush orders achieve better *Performance* results because their due dates are not predefined as they need to be integrated as early as possible in the current schedule, which leads to good satisfaction grades regarding the tardiness of arriving jobs. Additionally, rush orders achieve slightly better overall results for *Stability* due to the fact that these disruptions are controlled as soon as they arrived in the shop floor, while regular orders keep changing different parts of the schedule based on different due dates values required by the newly arriving jobs.

5.4 Discussion

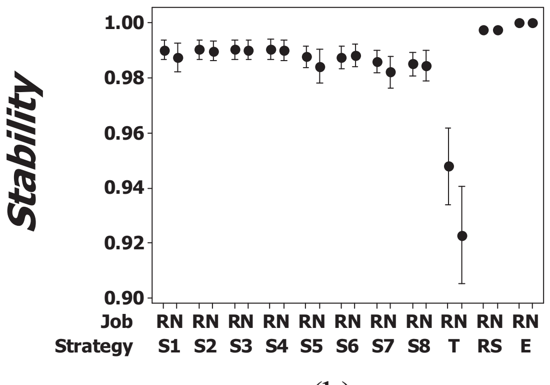
Improvements on the genetic algorithm fitness function led to a more effective control of both *Performance* and *Stability* of new schedules for the job shop scheduling/rescheduling problem presented by Sherwood Press. This effect is a result of an overall reduction of *Delay* values obtained by the improved version compared with the original one described in the previous chapter. Since the new settings bring remarkably superior results, they were also applied in experiments for normal orders. Note that there is a double control of *Stability*, since this feature is now part of the fitness function of the GA combined with match-up changing only specific parts of the schedule.

Statistical multi-comparison tests and analysis of variance reveal that match-up strategies are highly flexible to deal with complex disruptions, such as the ones which affects multiple resources in a shop floor as the arrival of rush and normal orders. This generalisation confirms that match-up strategies deliver highly stable and good performing schedules even when disruptions with different levels of urgency arises in the shop floor.

Match-up strategies S1-S4 remain highlighted as good candidates for possible incorporation into the scheduling/rescheduling system of Sherwood Press, since they produce good values for both *Performance* and *Stability* under different problem scenarios.



(a)



(b)

Figure 5.20: Rush (R) versus normal orders(N) for *Performance* and *Stability*

5.5 Summary

This chapter presents further investigation of the real world job shop scheduling/rescheduling problem presented by Sherwood Press, Nottingham, UK. A more general case of disruption is investigated, in which newly arriving jobs have different levels of urgency to be processed, referred here as normal orders. The main goal is to check the flexibility of match-up strategies under different types of disruption to achieve highly stable and good quality schedules.

Some design changes are proposed to adapt the original match-up algorithm to also accommodate normal orders. The obtained results were analysed and statistically validated. Surprisingly, rush orders deliver better overall results because their due dates are not predefined and the disruption is controlled as soon as they arrived in the shop floor.

Additionally, improvements on the genetic algorithm fitness function are proposed for rescheduling affected operations. Remarkably superior *Performance* and *Stability* results are found because the minimisation of the makespan reduces the overlaps between initial and new schedules, and the control of *Stability* soften changes on both sequence and processing time of operations.

In summary, match-up strategies are effective to manage different types of complex disruptions as the ones presented by Sherwood Press. They are able to combine the best attributes of total rescheduling, right shift and “insertion in the end”, in which good performing and highly stable schedules are delivered regardless of the problem parameters.

The next chapter investigates the combination of match-up strategies with a fuzzy robust scheduling system. The aim is to analyse if they are complimentary, regarding the use of idle times present on machines, to generate reliable schedules.

Chapter 6

Fuzzy approaches to robust job shop rescheduling

6.1 Introduction

This chapter considers a complex real world job shop rescheduling problem, in which jobs with different levels of urgency arrive every day in the shop floor and they need to be integrated in the existent schedule. A fuzzy scheduling system for inserting idle times on machines in order to produce initial robust schedules is developed; and a rescheduling system which uses match-up approaches accommodates the newly arriving jobs. The main goal is to investigate the quality of this combined system when the arriving jobs are either normal orders or rush ones. The obtained results and statistical analysis validate hypothesis 2 from Chapter 1, showing that a robust initial schedule combined with match-up rescheduling lead to higher quality and more reliable schedules even when jobs with different urgency levels arrive in a dynamic and uncertain shop floor.

The job shop scheduling problem presented by Sherwood Press - Nottingham, UK, is investigated in this chapter. The core idea is to find an appropriate scheduling/rescheduling approach to achieve a high quality schedule, regarding its *Performance* and *Stability*. The goal is to generate a robust schedule by inserting idle times on machines in order to reduce the negative effects of uncertainties that are present in the shop floor. On the other hand, match-up algorithms collect idle times on machines to define a *rescheduling horizon*, which is a part of the original schedule that is going to be modified to accommodate the newly arriving jobs. These strategies are complementary because both of them work with idle time control. The aim is to investigate possible effects of their combination.

Match-up algorithms have been only used for relatively simple scheduling problems. For instance, single machine problems are considered in [11] and [15], single stage with parallel machines problem in [12] and a match-up strategy coupled with a branch and bound algorithm is used for a flow shop problem in [3]. Robust scheduling, on the other hand, has been mostly investigated for machine breakdowns problems. For instance, fuzzy processing time and release time have been used in [35] and [21], branch and bound heuristics in [61] and [67], genetic algorithms in [47] and [59] and temporal protection based on historical data of the resources in [26]. Jobs with changing processing times are investigated in [28], which also applies fuzzy variables to set durations of operations. The research presented in this chapter and the resultant paper in [72] describe the only applications of match-up algorithms with robust scheduling to a complex real world job shop problem.

The remaining of this chapter is organised as follows. In sections 6.2 and 6.3, two different fuzzy rule-based systems are proposed to insert idle times on machines, in which databases with jobs requirements from Sherwood Press are used as reference for expected behaviour of the investigated shop floor. A comparison between these systems is also presented in order to decide which of them is more appropriate to apply to the analysed scheduling/rescheduling problem. Sections 6.4 and 6.5 discuss and summarise the conclusions of this chapter. Note that a discussion of match-up algorithms was already presented in the previous chapters, which shown their encouraging results regarding both *Performance* and *Stability* of resultant schedules.

6.2 Fuzzy Rule-based System for Robust Scheduling

The proposed fuzzy rule-based systems mimic the production manager's reasoning in making an initial robust schedule, in which information about jobs requirements are used as reference to insert idle times on machines. These extra idle times are inserted by extending the original processing times of operations. An overview of this process is described below.

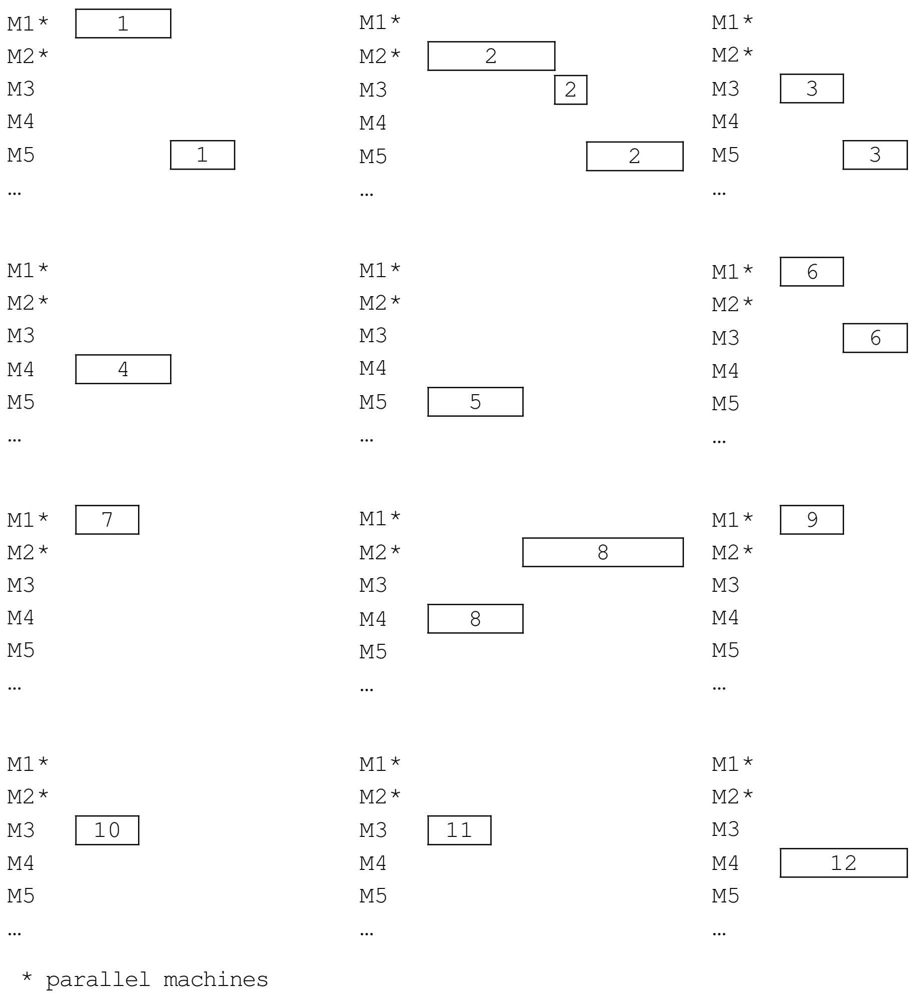
The proposed fuzzy systems decide for each operation o_{ij} whether to extend its processing time and how much to extend based on information on the workload of machines. Figure 6.1 (a) shows an example of 12 jobs that must be allocated on available machines. Each job has precedence constraints between operations. Note that jobs may have a different number of operations, i.e. jobs 1 and 2 have 2 and 3 operations, respectively. Machines M1 and M2 are parallel machines. After deciding on the extension for each operation, a schedule is generated using a genetic algorithm to optimise its *Performance*, shown in Figure 6.1 (b). The extension of original processing times generated by the fuzzy systems are highlighted in gray in Figure 6.1(b). In this resultant schedule, original processing times are restored and finishing times of operations are updated, leaving idle times on the machines and creating an initial robust solution, as shown in Figure 6.1 (c).

6.2.1 Fuzzy Rule-based System with Three Inputs

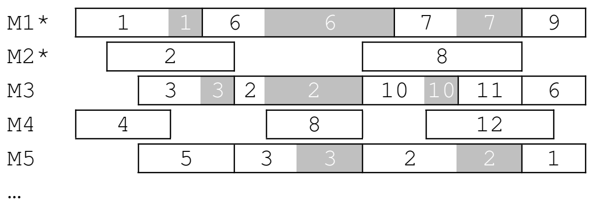
The first proposed fuzzy module has three antecedents and one consequent variable as shown in Figure 6.2. Databases from Sherwood Press provide information about the typical workload of machines in the shop floor. The main goal is to develop a system which can identify busy machines and busy periods in the time horizon.

The first antecedent, P_O, is the total processing time of operations for each machine. The second one, N_O, is the total number of operations that are required to be processed on a machine. They are both responsible to identify busy machines in the shop floor. Note that the information provided by N_O is also relevant because the investigated scheduling problem considers setup time between operations. Both P_O and N_O are represented by vectors, as shown in Figure 6.3 (a) and (b), respectively. M1 can be highlighted as a busy machine since it has a large value for both P_O and N_O.

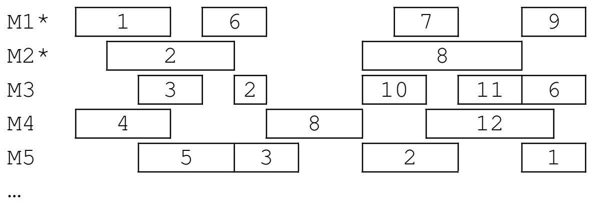
On the other hand, the third antecedent, PN_O, checks the possible number of operations that can be processed each minute of the scheduling horizon. Note that each job has a release and due-date time, defining a possible time window within which a job is processed. All jobs considered together can have overlapping time windows and PN_O measures the maximum number of overlapping operations that can be processed each minute of the scheduling horizon for each machine. For instance, lets consider only jobs 1 and 2 from Figure 6.1 (a). Hypothetically, they have the same release time and due-date,



(a)



(b)



(c)

Figure 6.1: Original processing times of jobs following their precedence constraints (a), schedule with extended processing times (b), and an initial robust schedule (c)

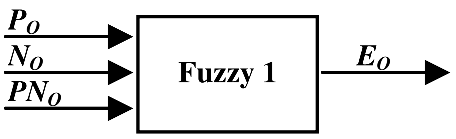


Figure 6.2: Fuzzy Rule-based System with three inputs and one output

P_O

M1	M2	...	M8	M9	...	M17	M18
18081	14828	...	2451	9297	...	8619	5523

(a)

N_O

M1	M2	...	M8	M9	...	M17	M18
111	104	...	26	95	...	19	40

(b)

PN_O

Time horizon	**M1**	M2	...	M8	M9	...	M17	M18
1	13	9	...	5	8	...	5	2
...	...	...	...	...	...	...	...	...
20000	**44**	37	...	6	34	...	10	16
...	...	...	...	...	...	...	...	...
30000	0	4	...	11	4	...	2	2
...	...	...	...	...	...	...	...	...
47700	0	0	...	0	1		0	0

(c)

Figure 6.3: Example of data present in vectors of total processing time of operations P_O (a) and total number of operations N_O (b), and in a matrix of total number of possible operations PN_O (c)

predefined as 1 and 10, respectively. A matrix 10 x 5 is then defined to set PN_O values for each instant of the time horizon for each of the 5 available machines. Note that two operations have to be processed on M5, i.e. one operation from each job. Consequently, column M5 set $PN_O = 2$ for the interval [1,10]. Similarly, M1-M3 set $PN_O = 1$. Note that M4 have $PN_O = 0$ because no operation has to be processed on this machine. The results are illustrated in Figure 6.4. A typical scheduling problem, however, set jobs with different intervals between release times and due-dates. Thus, each line may set different PN_O values based on these intervals. Figure 6.3 (c) shows an example of PN_O values calculated for 18 machines within a time horizon of 47700 time units. An operation with release time on 20000 requiring M1 can be highlighted as in a busy period because its PN_O has a relatively large value.

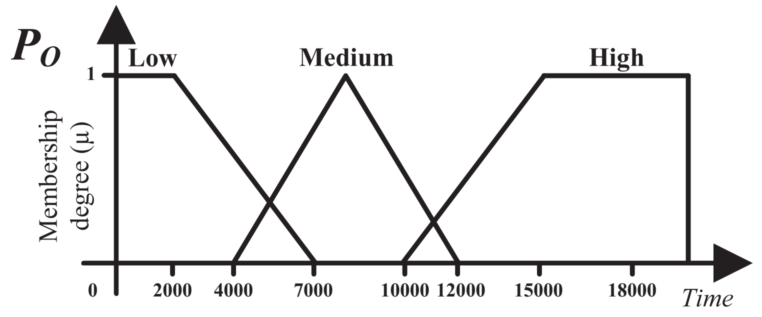
The consequent, E_O, is the extension value which is going to be applied to an original processing time of an operation. Extensions are generated within the interval [0,1], in with ”0” means no extension and ”1” increases the processing time a 100%. Note that extensions can be weighted following decisions made by the production manager.

The three input variables are described by three fuzzy sets, i.e. Low, Medium and High, whose membership functions are presented in Figure 6.5 (a), (b) and (c), respectively; the output is described by 5 fuzzy sets, i.e. No (no extension), Small, Regular, Large and Very Large, as in Figure 6.5 (d). Note that Very large has a more concentrated representation of the fuzzy set large in order to intensify its meaning, as described in Subsection 3.2.1 from Chapter 3.

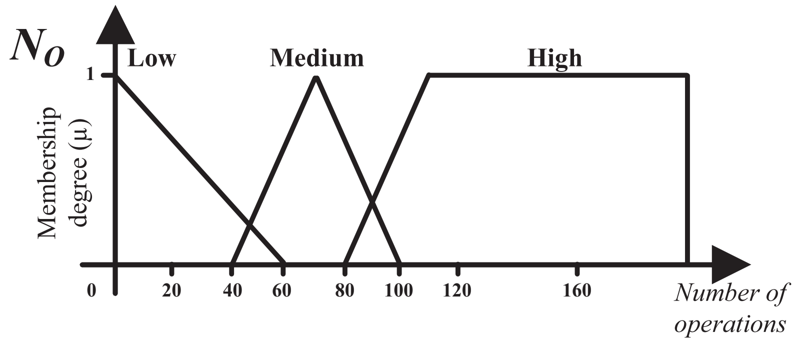
PN_O

Time horizon	M1	M2	M3	M4	**M5**
1	1	1	1	0	**2**
2	1	1	1	0	**2**
3	1	1	1	0	**2**
4	1	1	1	0	**2**
5	1	1	1	0	**2**
6	1	1	1	0	**2**
7	1	1	1	0	**2**
8	1	1	1	0	**2**
9	1	1	1	0	**2**
10	1	1	1	0	**2**

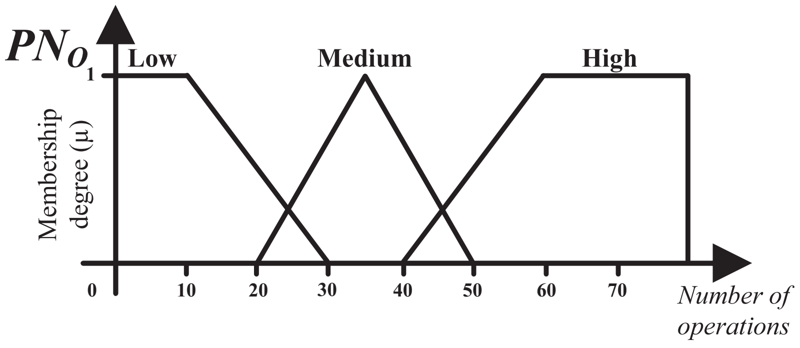
Figure 6.4: PN_O calculation only for jobs 1 and 2 from Figure 6.1 (a), assuming that they have the same release time and due-date, predefined as 1 and 10, respectively.



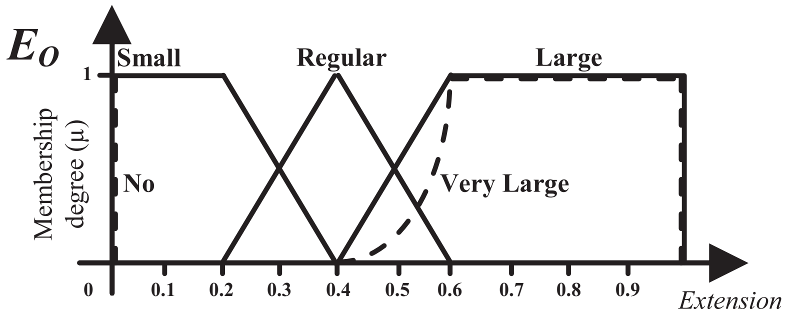
(a)



(b)



(c)



(d)

Figure 6.5: Membership functions for the variables total processing time of operations P_O (a), total number of operations N_O (b), total number of possible operations PN_O (c), and extension E_O (d)

The three input variables have the same shapes for their fuzzy sets Low, Medium and High. Information present in databases provided by Sherwood Press are used to configure them properly. The shapes for the output E_O follows the same pattern of the inputs, defining the meaning of the linguistic terms Small, Regular and Large. However two other fuzzy sets are required, in which no extensions are represented by the singleton set No and a hedge define Very Large fuzzy set. Note that those sets are introduced because it allows the fuzzy module to do a more refined decision on processing times extensions. Details about shapes and intervals for each fuzzy set are described in Table 6.1

Fuzzy rules [49] are defined which mimic the production manager's reasoning in making a robust schedule, namely operations to be processed on a "busy" machine in a "busy" period should be extended more than operations in a "less busy" part of the schedule. Fuzzy rules are shown in Table 6.2 in which a *Mamdani* style fuzzy inference is used [65]. Note that E_O generates "no" extension when all inputs variables P_O, N_O and PN_O have "Low" values. As soon as the input variables assume larger values, there is an incremental decision to generate larger extensions, as it can be observed when subsequent rows (or columns) from Table 6.2 are compared. The *min* operator is used in the evaluation of the premise of each rule. The defuzzification method *center of gravity* is applied to generate

Table 6.1: Fuzzy sets shapes and intervals for fuzzy rule-based system with three inputs

Variable	Fuzzy Set	Shape	Interval
P_O	Low	Trapezoidal	[0 0 2000 7000]
	Medium	Triangular	[4000 8000 12000]
	High	Trapezoidal	[10000 15000 20000 20000]
N_O	Low	Trapezoidal	[0 0 0 60]
	Medium	Triangular	[40 70 100]
	High	Trapezoidal	[80 110 200 200]
PN_O	Low	Trapezoidal	[0 0 10 30]
	Medium	Triangular	[20 35 50]
	High	Trapezoidal	[40 60 80 80]
E_O	No	Singleton	[0]
	Small	Trapezoidal	[0 0 2 4]
	Regular	Triangular	[2 4 6]
	Large	Trapezoidal	[4 6 10 10]
	Very Large	Gauss	[0.8 6.5 10 10]

Table 6.2: Fuzzy rules for fuzzy rule-based system with three inputs

		N_O = Low			N_O = Medium			N_O = High		
		P_O			P_O			P_O		
		Low	Medium	High	Low	Medium	High	Low	Medium	High
PN_O	Low	No	Small	Regular	Small	Regular	Large	Regular	Large	Very Large
	Medium	Small	Small	Regular	Small	Regular	Large	Regular	Large	Very Large
	High	Small	Regular	Large	Regular	Large	Very Large	Large	Very Large	Very Large

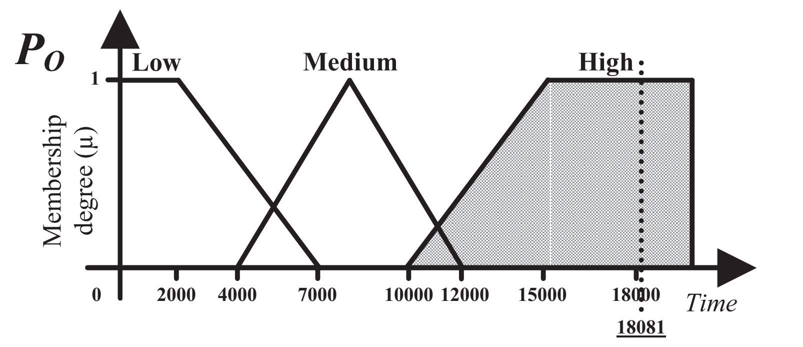
a crisp decision about the extension of the operation, given as percentage of its processing time.

From the example previously shown in Figure 6.3, a job requiring processing time on M1 with release date at 20000 has the following crisp numbers for the defined inputs, $P_O = 18081$, $N_O = 111$ and $PN_O = 44$, respectively. This numbers are converted into fuzzy numbers activating the fuzzy sets “High” for both inputs P_O and N_O, and the fuzzy sets “Medium” and “High” for the input variable PN_O, as shown in Figure 6.6. Subsequently, an output with “Very Large” extension is generated, because the inputs only activate rules which have this decision, as shown in highlighted cells in Table 6.2. The defuzzification method *center of gravity* transforms the activated area on the output E_O into a crisp number, generating an extension of 0.73, which means that the job processing time on M1 is going to be extended by 73%. Note that the production manager can add weights to these decisions in such a way that a maximum acceptable value for extensions is achieved.

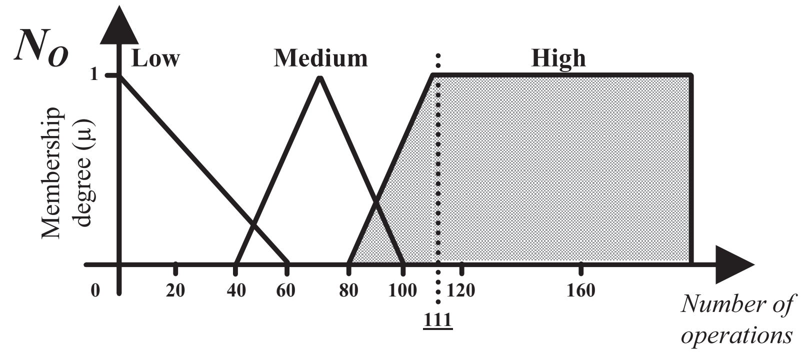
6.2.2 Fuzzy Rule-based System with Two Inputs

The second proposed fuzzy module has two antecedents and one consequent variable as shown in Figure 6.7. The main goal of this approach is to combine characteristics related to machine workload in only one variable, in such a way that a smaller number of fuzzy rules can be defined to identify busy machines and busy periods in the time horizon.

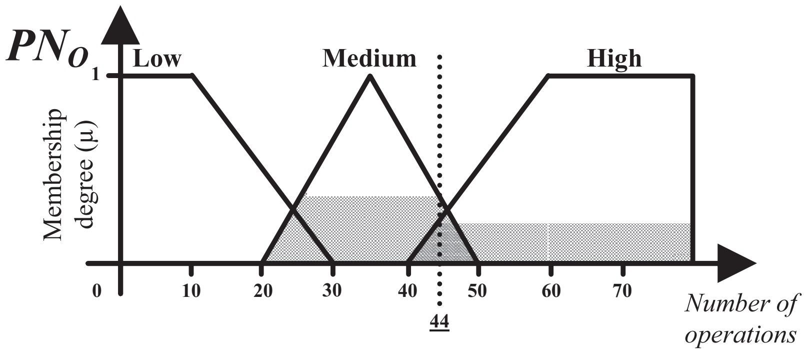
The first antecedent, *Mac*, combines two characteristics described in the previous model, i.e. P_O and N_O, total processing time of operations and total number of operations for each machine, respectively. Both characteristics are represented by vectors, as shown in Figure 6.8 (a) and (b). These vectors are both sorted in an ascending order and each machine receives a rank number within the interval [0,18] which identifies the workload of each machine, i.e. M1 got 18 as a rank number for both characteristics, which means that M1 is a busy machine. If different machines have the same number of operations or total processing times, then the ties are broken randomly. Subsequently, an average vector is defined to combine these two characteristics, as in Figure 6.8 (c). Note that a final rank number is set for each machine. If the average number is equal for different machines, a higher rank is given to the one with a larger number of operations because of the presence



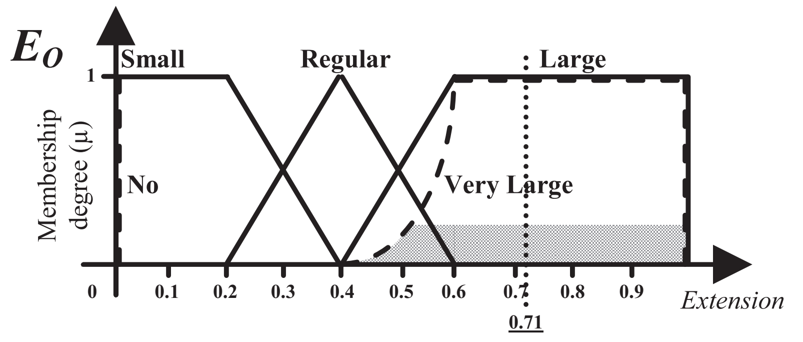
(a)



(b)



(c)



(d)

Figure 6.6: Fuzzy sets activated when a job requires processing time on M1 with release date on 20000 minutes

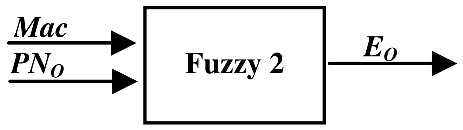


Figure 6.7: Fuzzy Rule-based System with two inputs and one output

of setup time in this job shop scheduling problem, i.e. M17 and M18 got both 9 as their rank and the tie is solved giving 8 to M17 and 9 to M18, due to the larger number of operations on M18.

The consequent PN_O is also used in this second approach to identify busy periods in the time horizon, which is represented by a matrix, as in Figure 6.8 (d). An operation with release time on 20000 requiring M1 is busier than another operation with release time on 30000 requiring M17, because its PN_O has a larger value. Additionally, the consequent, E_O, extends the processing times of operations following the same pattern presented in the subsection 6.2.1.

The input variables are both described by three fuzzy sets, i.e. ”NotBusy”, ”Normal” and ”Busy”, whose membership functions are presented in Figure 6.9 (a) and (b), respectively; and the output is described by 3 fuzzy sets, i.e. No (no extension), Small and Large, as in Figure 6.9 (c). The two input variables, *Mac* and PN_O, have similar shapes for their fuzzy sets. Several databases from Sherwood Press are used to set their size and shape properly. The fuzzy sets for the output E_O has the linguistic terms ”No”, ”Small” and ”Large”. Details about shapes and intervals for each fuzzy set are described in Table 6.3.

The fuzzy rules defined for this second approach follows the same idea of the first one, in which operations to be processed on a “busy” machine in a “busy” period should be extended more than operations in a “less busy” environment. The defined rules are shown in Table 6.4, in which a *Mamdani* style fuzzy inference is used. Note that larger

Table 6.3: Fuzzy sets shapes and intervals for fuzzy rule-based system with two inputs

Variable	Fuzzy Set	Shape	Interval
Mac	NotBusy	Trapezoidal	[1 1 2 9]
	Normal	Triangular	[5 10 15]
	Busy	Trapezoidal	[14 17 18 18]
PN_O	NotBusy	Trapezoidal	[0 0 10 30]
	Normal	Triangular	[20 35 50]
	Busy	Trapezoidal	[40 60 80 80]
E_O	No	Singleton	[0]
	Small	Trapezoidal	[0 0 3 6]
	Large	Trapezoidal	[4 7 10 10]

P_O - Characteristic 1

Machines	M1	M2	…	M8	M9	…	M17	M18
P_O	18081	14828	…	2451	9297	…	8619	5523
Sorted_P_O	18	16	…	4	12	…	11	7

(a)

N_O - Characteristic 2

Machines	M1	M2	…	M8	M9	…	M17	M18
N_O	111	104	…	26	95	…	19	40
Sorted_N_O	18	17	…	8	16	…	7	11

(b)

Mac **- Average between characteristics**

Machines	**M1**	M2	…	M8	M9	…	M17	M18
Average	18	16.5	…	6	14	…	9	9
Workload	**18**	16	…	6	14	…	8	9

(c)

PN_O

Time horizon	**M1**	M2	…	M8	M9	…	M17	M18
1	13	9	…	5	8	…	5	2
…	…	…	…	…	…	…	…	…
20000	**44**	37	…	6	34	…	10	16
…	…	…	…	…	…	…	…	…
30000	0	4	…	11	4	…	2	2
…	…	…	…	…	…	…	…	…
47700	0	0	…	0	1		0	0

(d)

Figure 6.8: Example of data present in vectors of total processing time of operations P_O - Characteristic 1 (a), total number of operations N_O (b) - Characteristic 2, Average of characteristics Mac (c) and a matrix of total number of possible operations PN_O (d)

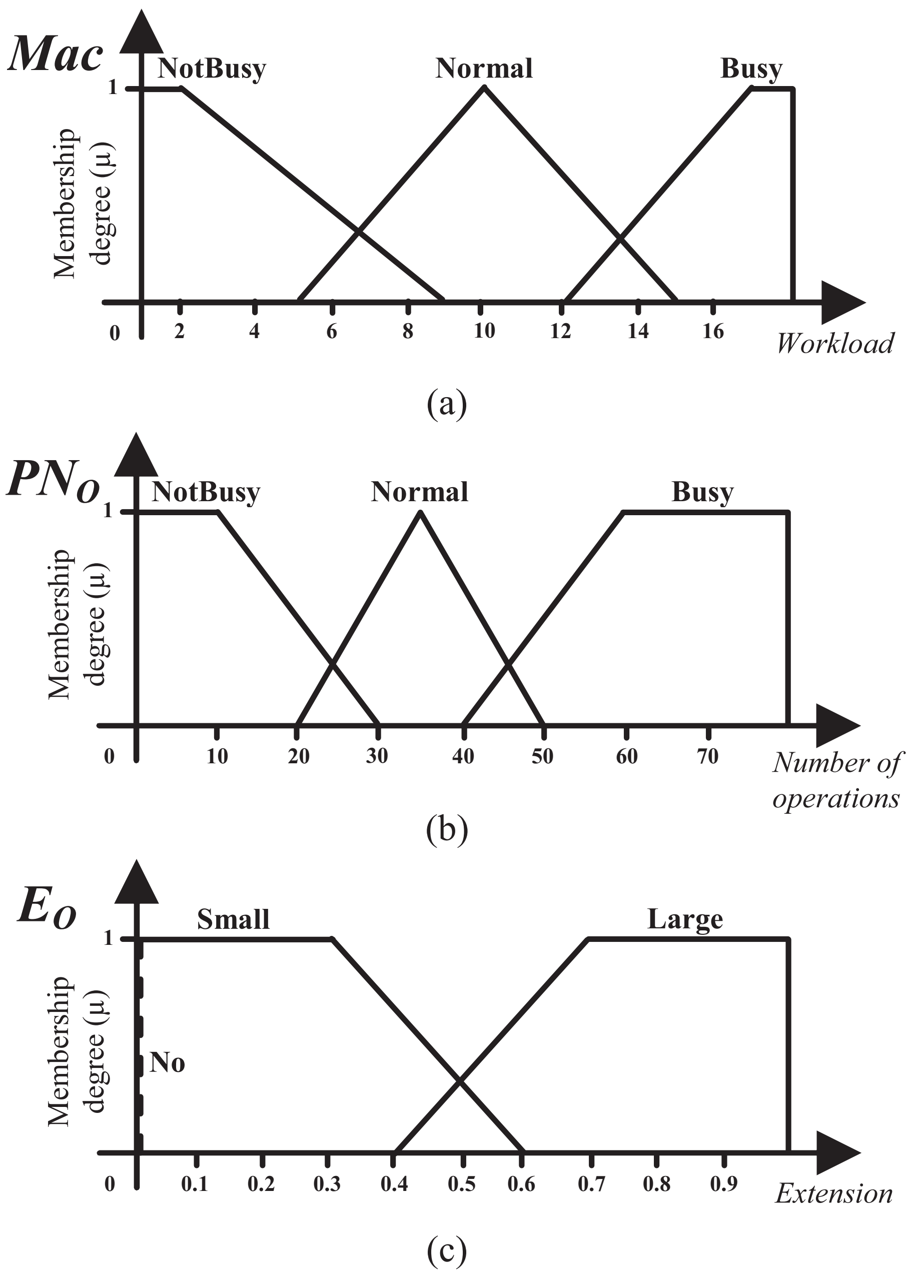


Figure 6.9: Membership functions for the variables total processing time of operations P_O (a), total number of operations N_O (b), total number of possible operations PN_O (c), and extension E_O (d)

Table 6.4: Fuzzy rules for fuzzy rule-based system with two inputs

		Mac		
		NotBusy	Normal	Busy
PN_O	NotBusy	No	Small	Small
	Normal	Small	Small	Large
	Busy	Small	Large	Large

extensions are generated when *Mac* and PN_O have larger values. The *min* operator is used in the evaluation of the premise of each rule and the defuzzification method *center of gravity* is applied to generate a crisp decision about the extension of each operation.

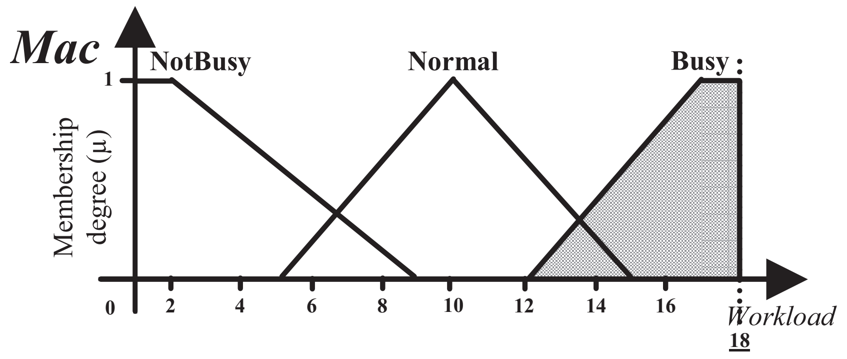
From the example previously shown in Figure 6.8, a job requiring processing time on M1 with release date on 20000 minutes has the following crisp numbers for the inputs, $Mac = 18$, and $PN_O = 44$, respectively. This numbers are converted into fuzzy numbers activating the different fuzzy sets, "Busy" for the input *Mac*, and both "Busy" and "Normal" for the input PN_O, as shown in Figure 6.10. Subsequently, an output with "Large" extension is generated,because both inputs only activate rules with this decision, as shown in highlighted cells in Table 6.4. The defuzzification method *center of gravity* transforms the activated area on the output E_O into a crisp number, generating a decision 0.73 extension for this example. Note that this fuzzy rule-based system defines a smaller number of rules when compared with the first approach. A comparison between results presented by both of them are discussed in the following section.

6.2.3 Fuzzy Rule-based Systems Analysis

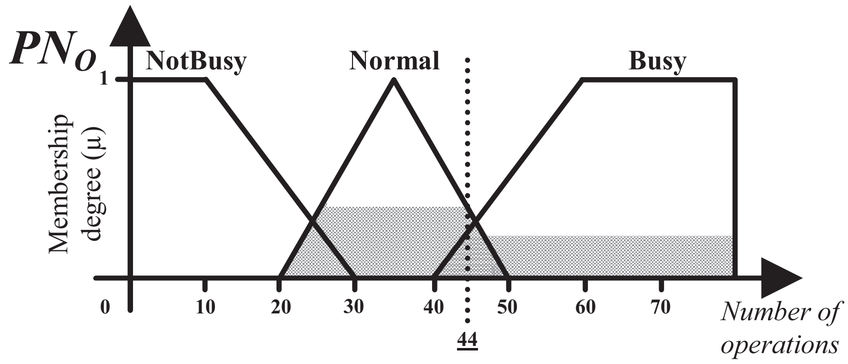
Data obtained from Sherwood Press are used to test the extensions generated by the previously described fuzzy rule-based systems. A set of 894 operation samples is evaluated, in which individual decisions are generated for each sample based on release times and machine requirements. Both fuzzy system aim to mimic the production manager decisions to create a initial robust schedule. The main differences between them are the number of inputs and their rules, i.e. Fuzzy1 has 3 inputs and 27 rules (as shown in Figure 6.2 and Table 6.2), and Fuzzy 2 has 2 inputs and 9 rules (as presented in Figure 6.7 and Table 6.4). Results obtained by both systems are presented in Figure 6.11.

Fuzzy1 can be highlighted as a more suitable approach because it generates smaller values for extensions than Fuzzy2, which avoids compromising the *Performance*. Its additional input variable allows a more refined decision, in which a larger number of rules leads to realistic extensions. Additionally, the meaning of P_O and N_0 are considered in a systematic way instead of combining these characteristics using a simple average vector *Mac*.

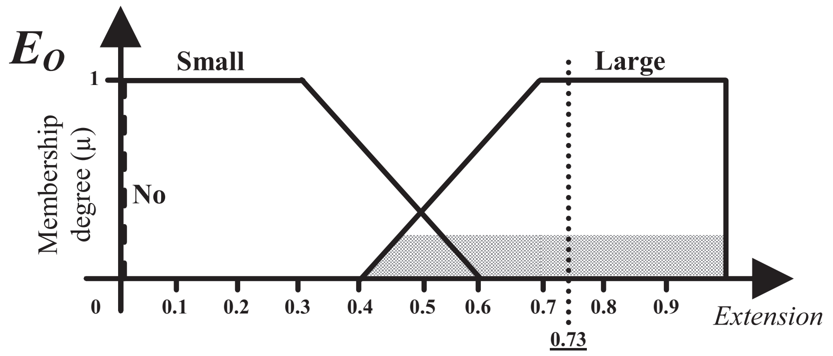
Some machines in the job shop problem have already a large amount of operations



(a)



(b)



(c)

Figure 6.10: Fuzzy sets activated when a job requires processing time on M1 with release date on 20000 minutes

to be processed. High extension levels, such as a 100%, could compromise too much the *Performance* of the schedule. In this way, a maximum processing time of 45 days is defined, P_{max}, and each machine i gets a weight w_i for their decisions on operations' extensions. Note that this maximum value is defined based on requirements of a typical month in Sherwood Press. Let M be the number of machines in the shop floor and P_{O_i} be the P_O value for each machine $i = 1, \dots M$. The goal is to check when P_{O_i} can be

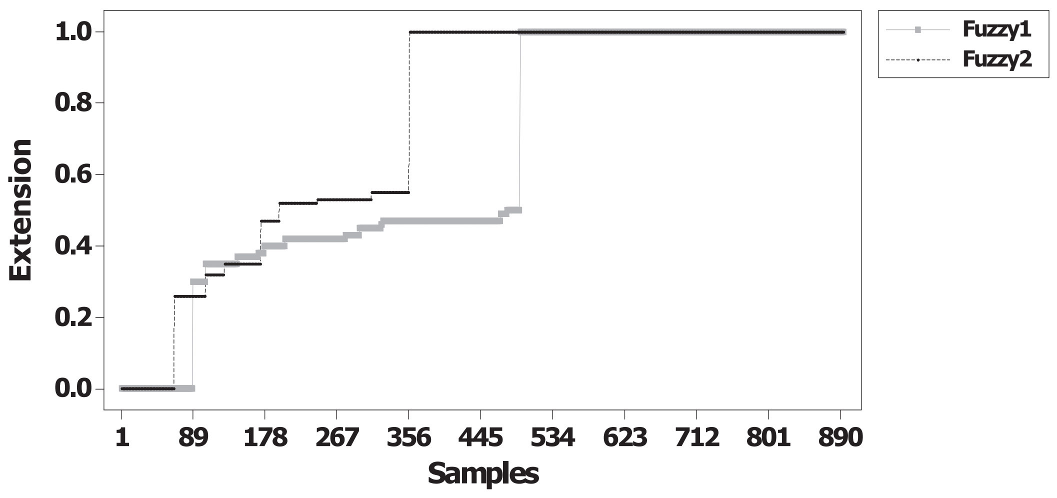


Figure 6.11: Comparison between Fuzzy1 and Fuzzy2

extended up to 100% without achieving P_{max}. The ratio $Ratio_i = \frac{P_{max}-P_{O_i}}{P_{O_i}}$ is calculated for each machine. If $Ratio > 1$ then $w_i = 1$, since all the operations can be extended up to 100%. If $Ratio < 1$ then $w_i = Ratio$, since a weight will guarantee that extensions will not surpass the P_{max} threshold. A comparison between the original Fuzzy1 and the weighted approach, Fuzzy1M, is shown in Figure 6.12.

As expected, Fuzzy1M delivers smaller extensions for operations because it considers already busy machines. This approach is selected to be used in the experiments because smaller extensions avoid unwanted deterioration of the schedule *Performance*.

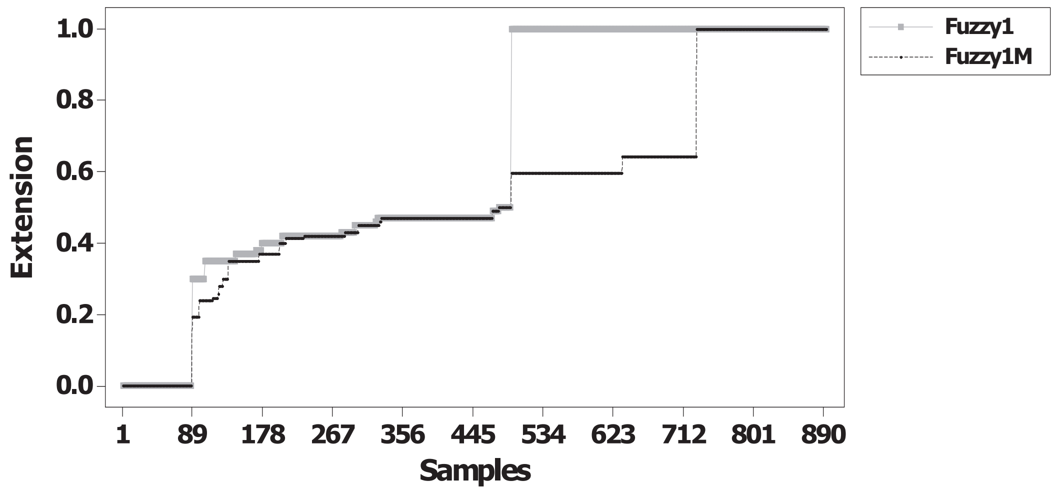


Figure 6.12: Comparison between Fuzzy1 and Fuzzy1M, which is a similar version with a maximum extension weight for each machine

6.3 Experiments on Real World Data

Data obtained from Sherwood Press are used to test the *Performance* and *Stability* of the proposed fuzzy rule-based system for robust scheduling combined with match-up algorithms for rescheduling. In each instance, new arriving jobs are randomly generated taking into account three parameters *jobSize*, *insTime* and *ext*. The first parameter is the number of operations in the new job, $jobSize \in \{1,2,3,4,5\}$. The *jobSize* value serves as a good indicator of the magnitude of the disturbance of the current schedule, which makes it an interesting parameter to investigate.

The second parameter is the time of insertion of the new job, *insTime* ∈ {beginning, middle, end} where "beginning", "middle" and "end" refer to an insertion point equal to 10%, 50% or 80% of the makespan of the initial schedule, respectively. The reason for considering the *insTime* follows the observation that the workload of the shop varies at different points in the schedule. The workload in the middle of the schedule, for instance, is often higher than the workload at the beginning, which is higher than the workload at the end of the schedule. Note that the parameter *sat*, previously investigated for match-up approaches, is not applied in this experiment because the parameter *insTime* already defines three different saturation levels, i.e. *insTime* ∈ {beginning, middle, end} reflects the same idea of *sat* ∈ {medium, high, low}.

The third parameter is the the extension level of operations, *ext*, which is used as an indicator of the amount of idle times inserted on machines. A range of different extensions are investigated based on possible protection levels applied on Sherwood Press shop floor. Two scenarios are investigated for this parameter. In the first scenario, the processing times of all operations are equally extended by 0%, 20%, 30%, 40% and 100%, denoted by E0, E2, E3, E4 and E10, respectively. In the second scenario, the developed fuzzy rule-based system is used to decide on the extension of the processing time of each operation. Note that an upper bound for extensions generated by the fuzzy system are also set to be 20%, 30%, 40% and 100%, while percentage of these bounds are determined by the consequent variable. These are denoted by F2, F3, F4 and F10, respectively. All extensions are subject to P_{max} threshold verification.

Figure 6.13 shows the average extension values applied to each analysed *ext* approach using 95% confidence interval plots. Fuzzy extensions set smaller values because each

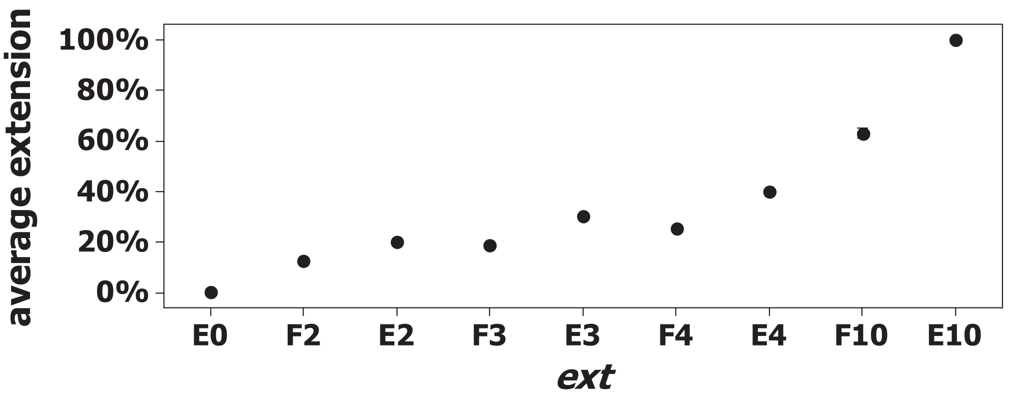


Figure 6.13: Average extension values for each *ext* approach

operation is individually analysed regarding the workload of the required machine and, consequently, F2, F3, F4 and F10 define smaller extensions than E2, E3, E4 and E10, respectively. Note that E0 represents the original schedule, which has no extended operations.

Since there are jobs with five different sizes, arriving at three different times, and nine possible extension levels, the total number of instances is $5 \times 3 \times 9 = 135$. For the purpose of experiments, arriving jobs are not kept in the schedule as the experimentation progresses, on the contrary, once a job has been integrated into an initial schedule and the proper measures have been recorded, the job is removed, and the schedule is reset to its initial state ready to accommodate the next arriving job.

Initial schedules are generated using the GA, described in [86] with the objective of maximising the *Performance* measure. Ten solutions are created for each *ext* instance and their results are shown in Figure 6.14. The *Performance* measure is an average of satisfaction grades SG_i, $i = 1, \dots 5$, as previously described in Chapter 4.

As expected, E0 delivers the best *Performance* results for an initial schedule because no extension is applied to operations. The pairs (Fx,Ex) with $x \in \{2,3,4,10\}$ show that the fuzzy approaches Fx achieve better *Performance* results than Ex, since each operation is analysed individually, leading to extensions when they are appropriate. Consequently, smaller extensions mostly lead to a better *Performance*. These results can be graphically observed in Figure 6.14. Smaller extensions produce schedules higher variability on their *Performance* results because the schedule does not become over saturated and, consequently, different solutions can be generated.

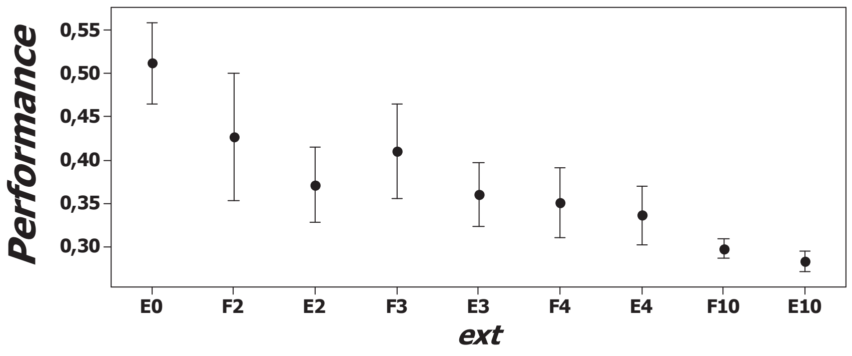


Figure 6.14: Overall results obtained by initial solutions after applying the extensions E0, F2, E2, F3, E3, F4, E4, F10 and E10

Schedules with the best initial solution for each extension are selected and they are subsequently used to insert the newly arriving jobs in the shop floor. Rescheduling strategies such as total rescheduling, right shift and insertion in the end are not applied in this experiment because they do not use the inserted idle times on machines. For instance, total rescheduling creates solutions from scratch and both right shift and insertion in the end insert new jobs without changing the sequence of operations.

Performance and *Stability* values are recorded for each tested approach on each instance of the problem. The obtained results for rush and normal orders are analysed and statistically validated in the following subsections. Additionally, a comparison between them is presented.

6.3.1 Rush Orders

New orders arrive everyday in the shop floor of Sherwood Press and most of them must be integrated in the current schedule as soon as possible. These orders are classified as rush orders. More details about this rescheduling process is described in chapter 4. This subsection presents the results obtained by the different extension levels, *ext*, as well as the results of statistical analysis of the effects of the problem parameters on *Performance* and *Stability*. Each instance of the problem is executed 10 times and both average and standard deviation results achieved by the investigated extensions *ext* are given in Table 6.5. Extensions with the best results for each instance are highlighted in bold. In general, fuzzy extensions are better than approaches using equally extended operations in most groups of instances. Additionally, F2 delivers the best *Performance*, and F10-E10 produce

Table 6.5: Average and standard deviation values for *Performance* and *Stability* obtained by the extension strategies for rush orders (larger vales are preferred)

		Performance - ext								
		E0	F2	E2	F3	E3	F4	E4	F10	E10
Strategy	(15) S1	0.527	**0.535**	0.444	0.505	0.433	0.465	0.417	0.312	0.298
	(15) S2	0.528	**0.535**	0.441	0.498	0.428	0.465	0.417	0.312	0.298
	(15) S3	0.528	**0.535**	0.441	0.498	0.433	0.465	0.415	0.312	0.299
	(15) S4	0.529	**0.531**	0.440	0.498	0.425	0.465	0.418	0.312	0.299
	(15) S5	0.515	**0.528**	0.439	0.498	0.428	0.460	0.417	0.312	0.299
	(15) S6	0.517	**0.518**	0.399	0.488	0.419	0.440	0.409	0.313	0.299
	(15) S7	0.51	**0.527**	0.436	0.493	0.428	0.460	0.415	0.312	0.299
	(15) S8	0.515	**0.519**	0.401	0.491	0.420	0.445	0.408	0.313	0.299
jobSize	(27) 1	0.53	**0.535**	0.440	0.508	0.432	0.466	0.418	0.313	0.299
	(27) 2	0.526	**0.533**	0.435	0.504	0.431	0.464	0.417	0.313	0.299
	(27) 3	0.521	**0.530**	0.430	0.495	0.426	0.458	0.416	0.312	0.298
	(27) 4	0.517	**0.524**	0.425	0.489	0.423	0.454	0.414	0.312	0.299
	(27) 5	0.512	**0.521**	0.421	0.485	0.420	0.450	0.410	0.312	0.298
insTime	(45) beginning	0.513	**0.528**	0.438	0.487	0.422	0.458	0.413	0.313	0.299
	(45) middle	0.523	**0.523**	0.407	0.496	0.425	0.454	0.411	0.312	0.298
	(45) end	0.528	**0.535**	0.445	0.506	0.433	0.463	0.420	0.313	0.299
(135) total average		0.521	**0.529**	0.430	0.496	0.427	0.458	0.415	0.312	0.299
(135) standard deviation		0.007	**0.006**	0.015	0.007	0.005	0.008	0.004	0.001	0.000

		Stability - ext								
		E0	F2	E2	F3	E3	F4	E4	F10	E10
Strategy	(15) S1	0.995	**0.996**	0.995	0.994	0.995	0.995	**0.996**	**0.996**	0.995
	(15) S2	0.995	0.995	0.994	0.994	0.994	0.995	0.994	**0.996**	0.995
	(15) S3	0.995	**0.996**	0.995	0.994	0.994	0.995	0.995	**0.996**	0.995
	(15) S4	0.995	0.994	0.994	0.994	0.994	0.995	0.993	**0.996**	0.995
	(15) S5	0.983	0.995	0.994	0.993	0.991	0.994	0.995	**0.996**	0.995
	(15) S6	0.984	0.981	0.979	0.981	0.982	0.985	0.986	0.994	**0.995**
	(15) S7	0.977	0.991	0.991	0.991	0.987	0.991	0.993	**0.996**	0.995
	(15) S8	0.979	0.980	0.979	0.979	0.978	0.983	0.985	0.994	**0.995**
jobSize	(27) 1	0.991	0.994	0.991	0.993	0.995	**0.999**	0.996	**0.999**	**0.999**
	(27) 2	0.988	0.995	0.994	0.995	0.995	0.997	0.996	**0.999**	**0.999**
	(27) 3	0.991	0.993	0.994	0.992	0.990	0.994	0.993	**0.999**	**0.999**
	(27) 4	0.987	0.990	0.992	0.988	0.987	0.988	0.989	**0.994**	**0.993**
	(27) 5	0.983	0.984	0.981	0.982	0.979	0.98	**0.985**	**0.985**	0.984
insTime	(45) beginning	0.979	0.990	0.992	0.990	0.992	0.992	0.992	**0.996**	0.995
	(45) middle	0.990	0.989	0.986	0.988	0.989	0.988	0.991	**0.996**	0.995
	(45) end	0.995	0.994	0.992	0.992	0.987	**0.996**	0.994	0.994	0.995
(135) total average		0.988	0.991	0.990	0.990	0.989	0.992	0.992	**0.995**	**0.995**
(135) standard deviation		0.006	0.005	0.006	0.005	0.006	0.005	0.004	**0.003**	**0.003**

schedules with better *Stability*. These results are statistically verified next.

A comparison between *Performance* and *Stability* is presented in Figure 6.15 in which a scatter plot shows the trade-offs between these evaluation metrics. Few samples have Stability values between 0.8 and 0.93 and they are considered outliers. Figure 6.15 shows

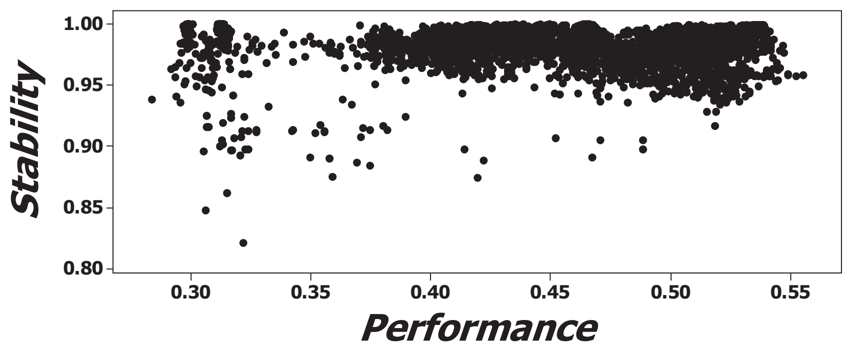


Figure 6.15: Trade-offs between *Performance* and *Stability* for rush orders

that there is a more concentrated variation on *Stability* when the *Performance* increases, which means that it may happen that increase of *Performance* also compromise *Stability*. However, there is no indication of increasing *Stability* having positive or negative effects on *Performance* values.

An ANOVA test checks the statistical significance of the effects of problem parameters, extension levels *ext* and the interactions among them on *Performance* and *Stability*. Results in Table 6.6 shows that all “main effects” and “interactions” influence both *Performance* and *Stability* of the schedule, since they achieve *P* values ≤ 0.05. Additionally, there is a higher variability on *Performance* results as its R^2 value is larger than the one achieved by *Stability*.

A pairwise comparison test using Bonferroni’s correction is given in Figure 6.16 in order to identify extensions that deliver higher *Performance* and *Stability*. Comparisons of *Performance* are below the diagonal, and above for *Stability*. These results combined with the averages shown in Table 6.5 give an overall behaviour of the analysed extension

Table 6.6: Results of the ANOVA test for rush orders

		Performance		*Stability*	
		F value	*P* value	*F* value	*P* value
Main effects					
	Strategy	288.95	≤ 0.05	501.05	≤ 0.05
	jobSize	356.64	≤ 0.05	1074.31	≤ 0.05
	insTime	677.39	≤ 0.05	147.9	≤ 0.05
	ext	55944.03	≤ 0.05	147.14	≤ 0.05
Interactions					
	Strategy**ext*	39.56	≤ 0.05	32.53	≤ 0.05
	*jobSize***ext*	16.4	≤ 0.05	17.6	≤ 0.05
	*insTime***ext*	138.45	≤ 0.05	112.49	≤ 0.05
	R^2	0.97		0.57	

Stability

Performance	E0	F2	E2	F3	E3	F4	E4	F10	E10
E0		↑ 8.96 ≤0.05	↑ 6.75 ≤0.05	↑ 6.00 ≤0.05	↑ 4.37 ≤0.05	↑ 11.24 ≤0.05	↑ 12.40 ≤0.05	↑ 21.61 ≤0.05	↑ 20.94 ≤0.05
F2	← 13.2 ≤0.05		= -1.91 1.00	= -2.56 0.37	← -3.97 ≤0.05	= 1.97 1.00	= 2.97 0.10	↑ 10.95 ≤0.05	↑ 10.37 ≤0.05
E2	↑ -157.4 ≤0.05	↑ -147.7 ≤0.05		= -0.64 1.00	= -2.05 1.00	↑ 3.88 ≤0.05	↑ 4.89 ≤0.05	↑ 12.86 ≤0.05	↑ 12.28 ≤0.05
F3	↑ -43.1 ≤0.05	↑ -48.8 ≤0.05	← 98.9 ≤0.05		= -1.41 1.00	↑ 4.53 ≤0.05	↑ 5.53 ≤0.05	↑ 13.51 ≤0.05	↑ 12.93 ≤0.05
E3	↑ -163.7 ≤0.05	↑ -153.2 ≤0.05	↑ -5.5 ≤0.05	↑ -104.4 ≤0.05		↑ 5.94 ≤0.05	↑ 6.94 ≤0.05	↑ 14.92 ≤0.05	↑ 14.34 ≤0.05
F4	↑ -108.8 ≤0.05	↑ -105.6 ≤0.05	← 42.1 ≤0.05	↑ -56.8 ≤0.05	← 47.6 ≤0.05		= 1.00 1.00	↑ 8.97 ≤0.05	↑ 8.39 ≤0.05
E4	↑ -184.3 ≤0.05	↑ -171.1 ≤0.05	↑ -23.3 ≤0.05	↑ -122.3 ≤0.05	↑ -17.8 ≤0.05	↑ -65.4 ≤0.05		↑ 7.97 ≤0.05	↑ 7.39 ≤0.05
F10	↑ -361.9 ≤0.05	↑ -324.8 ≤0.05	↑ -177.1 ≤0.05	↑ -276.0 ≤0.05	↑ -171.6 ≤0.05	↑ -219.2 ≤0.05	↑ -153.8 ≤0.05		= -0.58 1.00
E10	↑ -385.6 ≤0.05	↑ -345.4 ≤0.05	↑ -197.7 ≤0.05	↑ -296.6 ≤0.05	↑ -192.2 ≤0.05	↑ -239.8 ≤0.05	↑ -174.3 ≤0.05	↑ -20.56 ≤0.05	

Figure 6.16: Mean pairwise comparisons of *Performance* and *Stability* for rush orders

levels.

In general, a smaller extension leads to a higher *Performance*, which can be seem when the pairs (Fx,Ex), $x \in \{2,3,4,10\}$ are compared. Remarkably, F2 achieves superior *Performance* even when compared with the original schedule E0, because its strategic insertion of a small amount of idle times allows a better accommodation of the newly arriving jobs. Equally extended operations compromise too much the *Performance* even when small extensions are generated, i.e. E2 and E3 compared with larger fuzzy extensions F3 and F4, respectively.

On the other hand, higher *Stability* values are achieved when larger extensions are applied at the price of poor *Performance*. As expected, E10 and F10 deliver superior *Stability* and poor *Performance*. However, smaller extensions defined by Fx are able to achieve similar *Stability* to Ex, $x \in \{2,3,4,10\}$, which identify the strength of the proposed fuzzy system. Additionally, F2 can be highlighted again since the obtained results are statistically non-distinguishable to extensions up to 40%. This experiment achieved good *Stability* results for all parameters because match-up algorithms also helped keeping the schedule *Stability*. A summary of these results are graphically shown using 95% confidence interval plots in Figure 6.17 (a) and (b) for *Performance* and *Stability*, respectively. These results indicate that the newly introduced fuzzy ruled-based system posses the best attributes of the investigated extension approaches and overcome weaknesses, regarding their *Performance*.

Regarding the problem parameters, the Table 6.6 identifies that Strategy, *jobSize* and *insTime* have a significant influence on *Performance* and *Stability* when extending the processing times of operations. The nature of these effects is illustrated with the 95% confidence interval plots in Figure 6.18. The *x*-axis of plots (a)-(b), (c)-(d), and (e)-(f), measures the level of Strategy, *jobSize* and *insTime*, respectively. The *y*-axis shows the average values of *Performance*, (a)(c)(e), and *Stability*, (b)(d)(f), over different levels of extension *ext*. In general, *Performance* and *Stability* are superior by using S1-S4, when the arriving job requires a fewer number of operations and when the rescheduling is done in a less busy environment, i.e. at the end of the schedule.

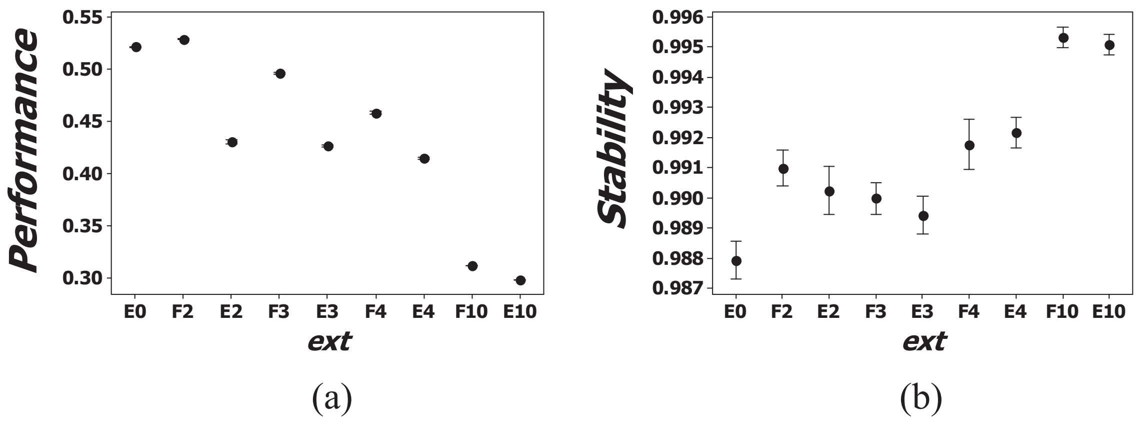


Figure 6.17: Overall results obtained by each extension *ext*; the x-axis shows the extension; the y-axis shows the mean (dot) and 95% confidence interval (vertical bars) *Performance* and *Stability* for rush orders

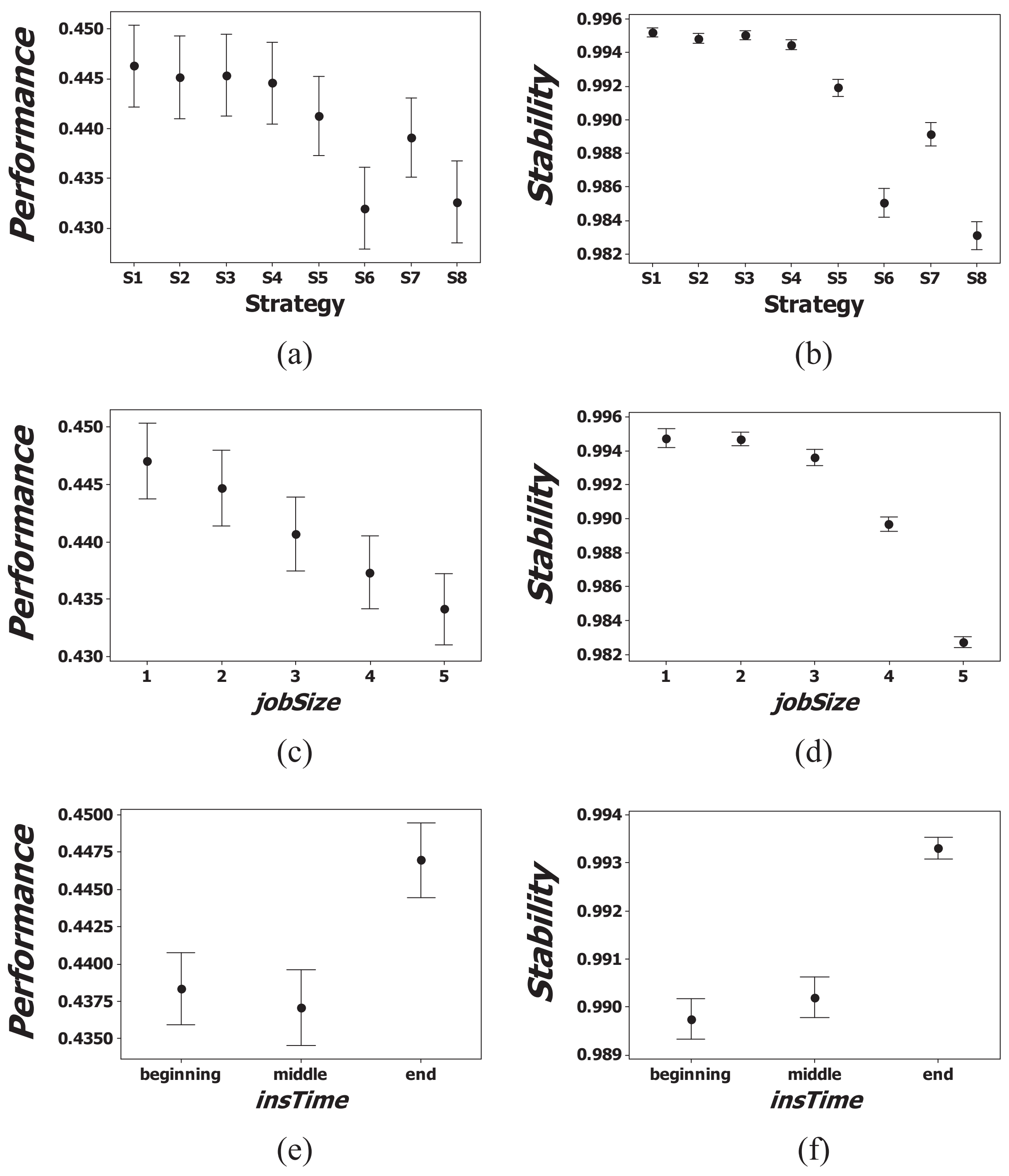


Figure 6.18: Main effects Strategy (a)-(b), *jobSize* (c)-(d) and *insTime* (e)-(f) on *Performance* (a), (c), (e) and *Stability* (b), (d), (f) for rush orders

Table 6.6 also shows that all interactions of parameters are significant. Particularly interesting are those interactions involving *ext* and any of the problem parameters. These type of interactions indicate that some extensions are better at coping with certain problem conditions than others. That this is the case can be verified by Table 6.5.

The three interactions involving *ext* were analysed. In general, large extensions *ext* lead to similar *Performance* and *Stability* results independently of the applied match-up strategy, *jobSize* and *insTime*. Schedules with a large amount of idle times, i.e. $ext \geq 40\%$ as the results in columns 6-9 from Table 6.5, always set similar rescheduling horizons, and consequently, similar *Performance* and *Stability* for all investigated instances. However, the interactions occur because small extensions for *ext* are better combined with strategies S1-S4, since they define smaller rescheduling horizons than S5-S8 and keep good *Performance* and *Stability*; *jobSize* with a smaller number of operations, i.e. 1 and 2 operations, because they make use of the inserted idle time without causing delays; and *insTime* in less busy parts of the schedule, i.e. at end, again because small rescheduling horizons are defined and the quality of the schedule is maintained.

Given the results in Table 6.5 and the statistical analysis, it is possible to conclude that the combination of fuzzy robust schedules with match-up algorithms for rescheduling brings more flexibility in a dynamic and uncertain environment, in which the strategic insertion of idle times on machines combined with minimal repair provided by match-up algorithms can reasonably well respond to disturbances that occur on a daily basis in Sherwood Press. The fuzzy extension F2 can be highlighted with the most consistent results under different problem scenarios, as demonstrated by Figure 6.18 and it can be considered for possible incorporation into the scheduling/rescheduling system of Sherwood Press, since it produces good values with respect to *Performance* and*Stability*, as shown in Figure 6.17.

6.3.2 Normal Orders

Normal orders define jobs with different levels of urgency. They represent a more flexible disturbance because their insertion are based on the job due-date, which gives a time window to make repair decisions. This subsection follows the same pattern of the statistical analysis presented for rush orders. More details about rescheduling normal orders is described in chapter 4. Table 6.7 summarises the obtained average and standard deviation results for *Performance* and *Stability* attained by the investigated *ext* levels for the different problem parameters after 10 execution times. Best results for each instance are highlighted in bold. As expected, fuzzy extensions are, in general, better than approaches using equally extended operations. Schedules with extension F2 have the best

Table 6.7: Average and standard deviation values for *Performance* and *Stability* obtained by the extension strategies for normal orders (larger vales are preferred)

		Performance - ext								
		E0	F2	E2	F3	E3	F4	E4	F10	E10
Strategy	(15) S1	0.513	**0.530**	0.433	0.497	0.428	0.459	0.414	0.312	0.299
	(15) S2	0.522	**0.525**	0.433	0.489	0.424	0.454	0.404	0.312	0.299
	(15) S3	0.525	**0.529**	0.437	0.499	0.426	0.459	0.415	0.312	0.299
	(15) S4	0.524	**0.529**	0.431	0.499	0.426	0.460	0.413	0.312	0.299
	(15) S5	0.509	**0.528**	0.434	0.496	0.428	0.460	0.416	0.312	0.299
	(15) S6	0.515	**0.520**	0.424	0.487	0.423	0.449	0.404	0.313	0.300
	(15) S7	0.506	**0.527**	0.433	0.493	0.425	0.459	0.412	0.312	0.299
	(15) S8	0.511	**0.516**	0.429	0.482	0.419	0.422	0.404	0.312	0.300
jobSize	(27) 1	0.525	**0.534**	0.445	0.504	0.432	0.462	0.418	0.313	0.300
	(27) 2	0.520	**0.531**	0.439	0.501	0.431	0.461	0.415	0.313	0.300
	(27) 3	0.516	**0.527**	0.433	0.492	0.427	0.457	0.411	0.312	0.299
	(27) 4	0.511	**0.521**	0.427	0.487	0.420	0.444	0.407	0.312	0.299
	(27) 5	0.506	**0.514**	0.413	0.480	0.414	0.439	0.400	0.311	0.298
insTime	(45) beginning	0.514	**0.522**	0.425	0.484	0.422	0.451	0.404	0.312	0.299
	(45) middle	0.506	**0.524**	0.431	0.487	0.423	0.449	0.405	0.312	0.298
	(45) end	0.526	**0.530**	0.439	0.507	0.430	0.458	0.421	0.312	0.300
(135) total average		0.516	**0.525**	0.432	0.493	0.425	0.453	0.410	0.312	0.299
(135) standard deviation		0.007	**0.006**	0.007	0.008	0.005	0.011	0.006	0.001	0.001
		Stability - ext								
		E0	F2	E2	F3	E3	F4	E4	F10	E10
Strategy	(15) S1	0.988	0.994	0.993	0.994	0.993	0.994	0.992	**0.996**	**0.996**
	(15) S2	0.993	0.991	0.990	0.992	0.989	0.992	0.989	**0.996**	**0.996**
	(15) S3	0.994	0.994	0.992	0.994	0.992	0.993	0.992	**0.996**	0.995
	(15) S4	0.994	0.994	0.992	0.993	0.990	0.993	0.992	**0.996**	0.995
	(15) S5	0.980	0.991	0.990	0.993	0.992	0.990	0.991	**0.996**	0.995
	(15) S6	0.984	0.984	0.986	0.986	0.985	0.983	0.983	**0.995**	**0.995**
	(15) S7	0.977	0.990	0.988	0.990	0.989	0.988	0.988	**0.995**	**0.995**
	(15) S8	0.980	0.980	0.982	0.981	0.974	0.962	0.976	**0.995**	**0.995**
jobSize	(27) 1	0.987	0.994	0.996	0.994	0.994	0.998	0.993	**1.000**	**1.000**
	(27) 2	0.990	0.994	0.994	0.994	0.993	0.994	0.993	0.999	**1.000**
	(27) 3	0.991	0.993	0.993	0.994	0.989	0.991	0.991	**0.999**	**0.999**
	(27) 4	0.984	0.988	0.989	0.988	0.987	0.978	0.986	**0.995**	0.993
	(27) 5	0.979	0.979	0.974	0.981	0.977	0.974	0.976	**0.985**	**0.985**
insTime	(45) beginning	0.990	0.989	0.991	0.989	0.989	0.989	0.991	**0.996**	0.995
	(45) middle	0.973	0.985	0.986	0.988	0.983	0.985	0.985	**0.995**	**0.995**
	(45) end	0.995	**0.995**	0.991	0.994	0.992	0.986	0.987	**0.995**	**0.995**
(135) total average		0.986	0.990	0.989	0.990	0.988	0.987	0.988	**0.995**	**0.995**
(135) standard deviation		0.007	0.005	0.005	0.004	0.006	0.009	0.005	**0.003**	**0.003**

Performance and a better *Stability* is delivered by schedules with extension F10 and E10. Further discussions are presented next.

Figure 6.19 shows the trade-offs between *Performance* and *Stability*, in which samples between 0.8 and 0.9 on y-axis are outliers. A more concentrated variation on *Stability* is

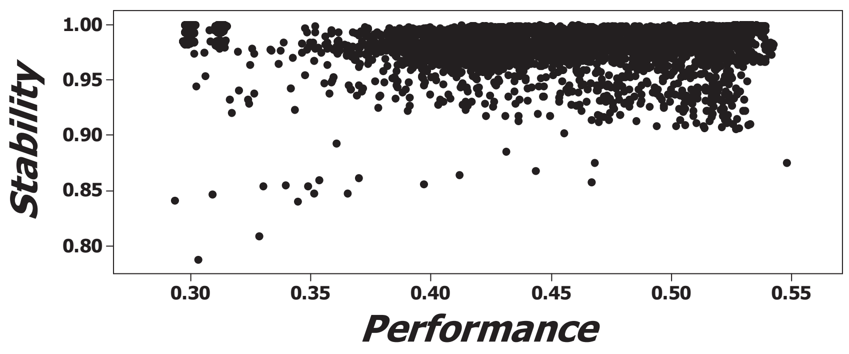


Figure 6.19: Trade-offs between *Performance* and *Stability* for normal orders

again observed when *Performance* increases, i.e. higher *Performance* results may compromise *Stability*. On the other hand, no positive or negative effects on *Performance* is observed when the *Stability* is increased.

Results of the ANOVA test for normal orders is shown in Table 6.8, in which all "main effects" and "interactions" have influences on both *Performance* and *Stability*. A higher variability of *Performance* is observed compared with *Stability* due to its large R^2 value of 0.97.

Figure 6.20 shows a pairwise comparison test using Bonferroni's correction, in which extensions with higher *Performance* and *Stability* values can be identified. As expected, higher *Performance* results are delivered by schedules with smaller extensions, as observed on pairs (Fx,Ex), $x \in \{2,3,4,10\}$. F2 deliver again superior *Performance* than E0, due to its strategic insertion of idle times combined with match-up rescheduling algorithms. Moreover, equally extended operations compromises too much the *Performance* even when small extensions are generated.

Table 6.8: Results of the ANOVA test for normal orders

		Performance		*Stability*	
		F value	P value	F value	P value
Main effects					
	Strategy	130.44	≤ 0.05	346.16	≤ 0.05
	jobSize	669.02	≤ 0.05	1278.59	≤ 0.05
	insTime	763.44	≤ 0.05	453.63	≤ 0.05
	ext	52713.23	≤ 0.05	184.74	≤ 0.05
Interactions					
	Strategy**ext*	26.84	≤ 0.05	29.98	≤ 0.05
	*jobSize***ext*	24.45	≤ 0.05	18.48	≤ 0.05
	*insTime***ext*	55.09	≤ 0.05	116.33	≤ 0.05
	R^2	0.97		0.58	

Stability

Performance

	E0	F2	E2	F3	E3	F4	E4	F10	E10
E0	■	↑ 8.98 ≤0.05	↑ 7.37 ≤0.05	↑ 10.37 ≤0.05	↑ 4.71 ≤0.05	= 1.42 1.00	↑ 3.93 ≤0.05	↑ 23.47 ≤0.05	↑ 22.87 ≤0.05
F2	← 18.2 ≤0.05	■	= -1.39 1.00	= 1.19 1.00	← -3.70 ≤0.05	← -6.55 ≤0.05	← -4.37 ≤0.05	↑ 12.54 ≤0.05	↑ 12.02 ≤0.05
E2	↑ -156.6 ≤0.05	↑ -151.4 ≤0.05	■	= 2.59 0.34	= -2.30 0.76	← -5.15 ≤0.05	= -2.97 0.10	↑ 13.94 ≤0.05	↑ 13.42 ≤0.05
F3	↑ -42.6 ≤0.05	↑ -52.6 ≤0.05	← 98.8 ≤0.05	■	← -4.90 ≤0.05	← -7.75 ≤0.05	← -5.57 ≤0.05	↑ 11.35 ≤0.05	↑ 10.82 ≤0.05
E3	↑ -169.3 ≤0.05	↑ -162.4 ≤0.05	↑ -11.0 ≤0.05	↑ -109.8 ≤0.05	■	= -2.84 0.15	= -0.66 1.00	↑ 16.25 ≤0.05	↑ 15.72 ≤0.05
F4	↑ -117.5 ≤0.05	↑ -117.5 ≤0.05	← 33.9 ≤0.05	↑ -64.9 ≤0.05	← 44.9 ≤0.05	■	= 2.18 1.00	↑ 19.10 ≤0.05	↑ 18.57 ≤0.05
E4	↑ -196.6 ≤0.05	↑ -186.1 ≤0.05	↑ -34.6 ≤0.05	↑ -133.4 ≤0.05	↑ -23.7 ≤0.05	↑ -68.5 ≤0.05	■	↑ 16.92 ≤0.05	↑ 16.40 ≤0.05
F10	↑ -379.7 ≤0.05	↑ -344.6 ≤0.05	↑ -193.2 ≤0.05	↑ -292.0 ≤0.05	↑ -182.2 ≤0.05	↑ -227.1 ≤0.05	↑ -158.5 ≤0.05	■	= -0.52 1.00
E10	↑ -404.0 ≤0.05	↑ -365.7 ≤0.05	↑ -214.3 ≤0.05	↑ -313.0 ≤0.05	↑ -203.3 ≤0.05	↑ -248.1 ≤0.05	↑ -179.6 ≤0.05	↑ -21.06 ≤0.05	■

Figure 6.20: Mean pairwise comparisons of *Performance* and *Stability* for normal orders

In general, the extensions Fx and Ex, $x \in \{2,3,4,10\}$, lead to superior *Stability* than E0. The pairs (Fx,Ex) deliver similar stability, which shows a strength of the proposed system regarding small extensions with reasonable *Stability*. However, no relatively large improvements on *Stability* are observed with extensions up to 40%, which indicates that the match-up algorithms already controls the schedule *Stability* in an effective way. Only larger extensions, i.e. E10 and F10, achieved higher *Stability* at the price of poor *Performance*. Additionally, F2 can be highlighted again since the obtained results are statistically superior or non-distinguishable to extensions up to 40%. Figure 6.21 (a) and (b) shows the overall results for *Performance* and *Stability*, respectively. These results

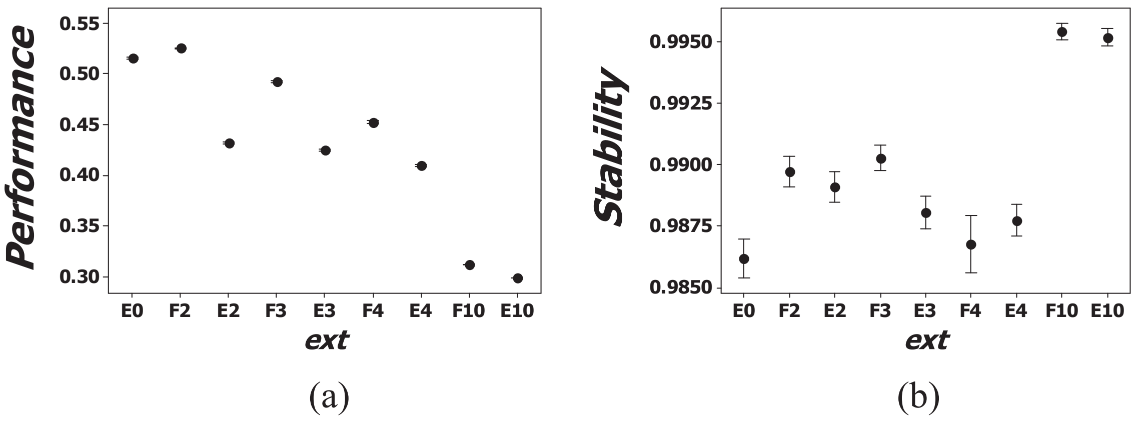


Figure 6.21: Overall results obtained by each extension *ext*; the x-axis shows the extension; the y-axis shows the mean (dot) and 95% confidence interval (vertical bars) *Performance* and *Stability* for normal orders

emphasises that the newly introduced fuzzy ruled-based system posses the best attributes of the investigated extension approaches.

The parameters Strategy, *jobSize* and *insTime* have a significant influence on both *Performance* and *Stability* over different levels of extension *ext*. These effects are illustrated in Figure 6.22. As expected, *Performance* and *Stability* are superior by using S1-S4, when the arriving job requires one of two operations and when the rescheduling is done at the end of the schedule.

The interactions between parameters are significant because small extensions *ext* are better combined with strategies S1-S4, which usually set small rescheduling horizons; *jobSize* with a 1 or 2 number of operations, which cause no delays; and *insTime* at the end of the schedule, because small rescheduling horizons are also defined. These combinations deliver again good quality schedules.

Fuzzy robust schedules has been again successfully combined with match-up algorithms to manage uncertainties present in a shop floor. The investigation of normal orders is a generalisation of different types of jobs that may arise in Sherwood Press. Schedules with fuzzy extension F2 are highlighted as the ones with the most consistent results under different problem scenarios, as shown in Figure 6.22, since it produces high quality stable schedules.

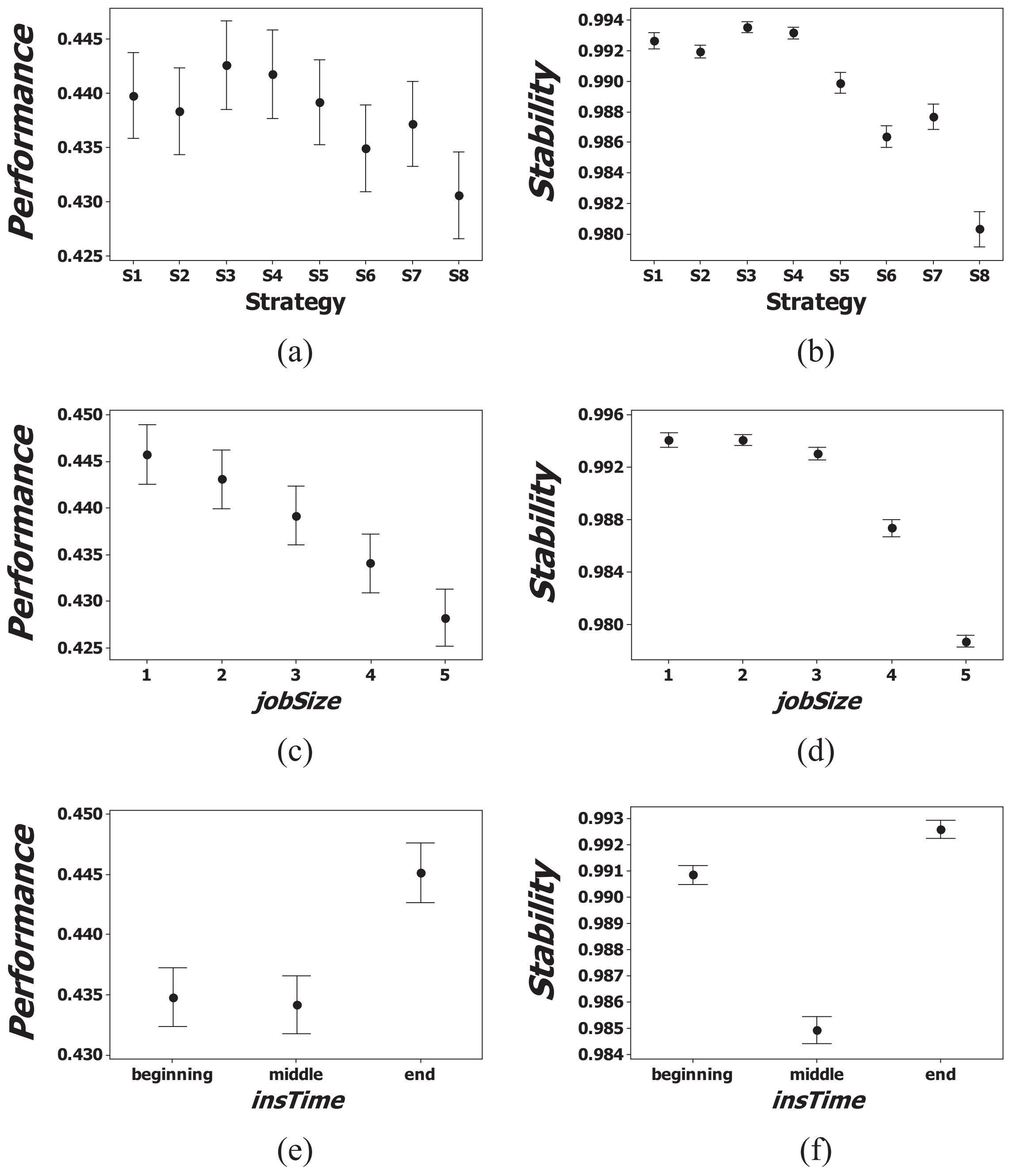


Figure 6.22: Main effects Strategy (a)-(b), *jobSize* (c)-(d) and *insTime* (e)-(f) on *Performance* (a), (c), (e) and *Stability* (b), (d), (f) for normal orders

6.3.3 Comparison between Rush and Normal Orders

Both rush (R) and normal (N) orders are typical disruptions arising on a daily basis in Sherwood Press. Their *Performance* and *Stability* results are compared in Figure 6.23 for each extension level *ext* and for each match-up strategy S1-S8.

Performance results obtained by different match-up strategies and extension levels

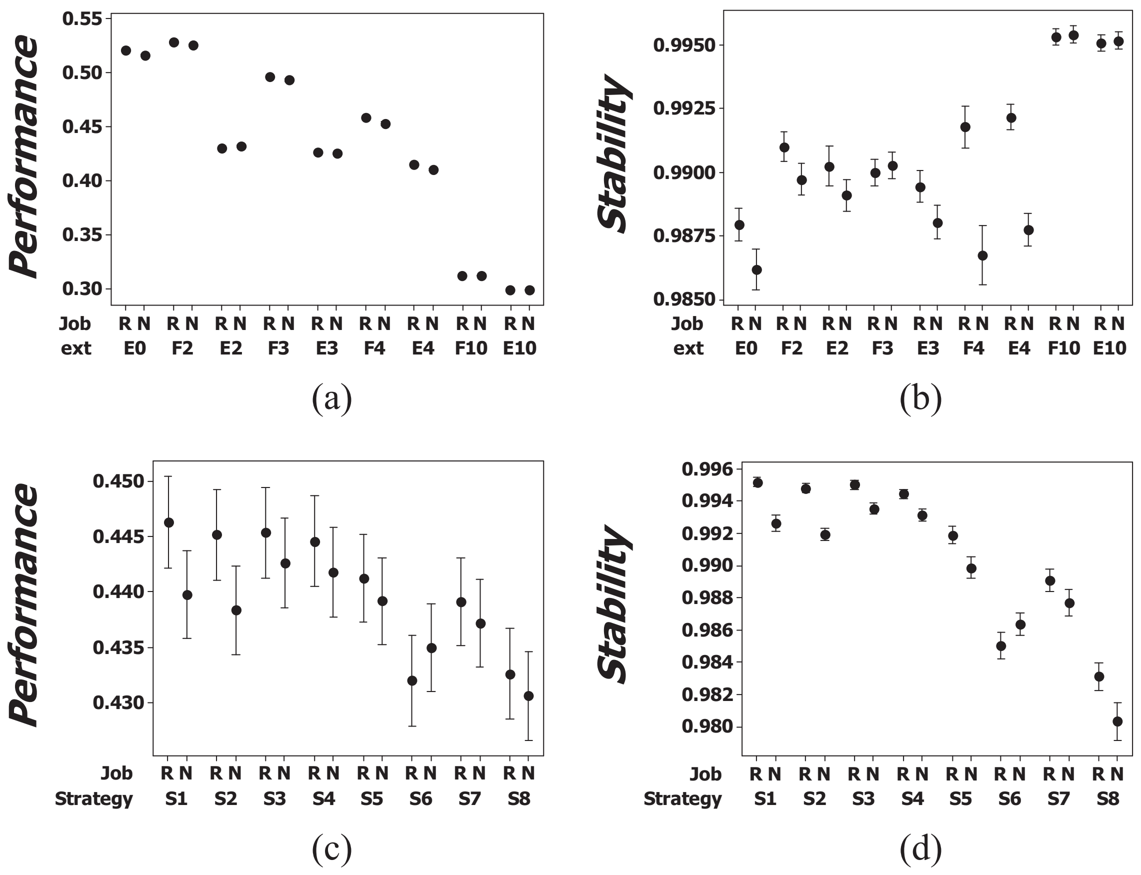


Figure 6.23: Rush (R) versus normal orders(N) for *Performance* (a) and (c), and *Stability* (b) and (d), respectively

ext for rush and normal orders are quite similar, which indicates a good flexibility of the proposed approaches for handling jobs with different levels of urgency.

Rush orders have no predefined due dates, and consequently, they deliver relatively better *Performance* results. Moreover, their *Stability* is generally better because they have to be aggregated as soon as possible, which avoids changing different parts of the schedule as it is always done by normal orders.

6.4 Discussion

Given the results for different types of arriving jobs and the statistical analysis carried out, it is possible to conclude that the combination of fuzzy robust schedules with match-up algorithms for rescheduling brings more flexibility in a dynamic and uncertain environment, in which the strategic insertion of idle times on machines combined with minimal repair provided by match-up algorithms can reasonably well respond to disturbances occurring on Sherwood Press.

Effects on quality measures indicates that it is possible by extending jobs' processing times to protect the schedule without compromising its *Performance* and *Stability*. Note that new orders usually compromise many resources in a shop floor and the improvements provided by fuzzy robust schedules may also assist on managing other relatively simple disturbances, such as operators doing late decisions, delays on raw material delivery and requirements of additional clean ups on machines.

The fuzzy extension F2 is highlighted as the one with the most consistent results under different problem scenarios and it is a candidate for possible incorporation into the scheduling/rescheduling system of Sherwood Press, since it produces high quality schedules with respect to both *Performance* and *Stability*.

6.5 Summary

A real world job shop scheduling/rescheduling problem is investigated in this chapter. The problem is dynamic since orders with different levels of urgency arrive every day in the shop floor and they need to be integrated in the existent schedule, without compromising its *Performance* and *Stability*. The proposed approach combines strengths of the robust scheduling, regarding control of future disturbances, and match-up rescheduling algorithms. These strategies are complementary because both of them work with idle time control.

Two fuzzy rule-based systems are proposed to insert idle times on machines, in which databases with jobs requirements from Sherwood Press are used as reference for expected behaviour in the shop floor. A comparison between these systems is presented in order to decide which among them is more appropriate to apply to this scheduling/rescheduling problem. Experiments with schedules with different amount of idle times are carried out in order to identify their possible effect on both *Performance* and *Stability*.

Two types of jobs arriving jobs are investigated, rush orders, which must be inserted as early as possible into the current schedule, and normal orders, which set jobs with different levels of urgency. The obtained results are analysed and statistically validated.

Additionally, a comparison between them is presented.

In summary, initial robust schedules combined with match-up rescheduling lead to higher quality and more reliable schedules even when jobs with different urgency levels arrive in a dynamic and uncertain shop floor.

The following chapter presents the conclusions of the studies investigated in this thesis, highlighting their relevance, limitations and future work.

Chapter 7

Conclusions

A real world job shop scheduling / rescheduling problem presented by a printing company in Nottingham, UK is investigated in this thesis. This problem is dynamic in its nature because unexpect events often occur on the shop floor. Typically, new orders arise on a daily basis and current allocations have to be changed in order to integrate them. These orders usually require processing time on different machines, and consequently, many available resources are often compromised. Match-up algorithms are applied as repair methods, because they are able to deliver stable and high quality schedules. These algorithms are subsequently combined with initial robust schedules with the aim of facilitating the accommodation of future disruptions and consequently producing more reliable and effective solutions.

Background and related work are presented in chapter 2, in which the investigated problem is situated within a rescheduling classification and match-up algorithms are highlighted as reasonable repair methods. Additionally, chapter 3 identifies the application of fuzzy logic concepts as a suitable approach to help modelling possible uncertainties present on a shop floor. The contributions of this thesis are described in chapters 4, 5 and 6. Chapter 4 discusses in detail the analysed scheduling / rescheduling problem, in which new match-up strategies are introduced to control a complex real world problem. These strategies accommodate disruptions by using available idle times on machines and consequently initial optimal solutions are kept unchanged as much as possible. Typical arriving jobs are rush orders and they have to be processed as soon as possible on the shop floor. This disruption is tackled first with the goal of checking the effectiveness of the proposed strategies on achieving stable and high quality schedules. As a matter of generalisation, orders with different levels of urgency are investigated in chapter 5 in which the flexibility of the proposed strategies are verified under different scenarios. Chapter 6 introduces a fuzzy scheduling approach for inserting idle times on machines, in which initial robust schedules are produced. The effects of combining this approach with match-up approaches for rescheduling are analysed due to the fact that they both work with idle time to manage disruptions.

7.1 Discussion

The application of match-up algorithms has been limited only to a small variety of problems, most of which are of a more theoretical than practical importance. This thesis and its resultant papers represent the only attempts to employ such algorithms in a complex real world shop floor which includes multiple criteria, setup times and disruptions affecting multiple resources. Additionally, strengths of fuzzy logic concepts are highlighted as a good approach on managing uncertainties present in real world problems. Despite of their success in solving many industry issues, research on fuzzy scheduling has been mainly focused on static scheduling environments. Consequently, this thesis uses their strengths applied to a dynamic complex job shop problem, in which fuzzy numbers are used to represent scheduling parameters and a fuzzy control system is combined with match-up algorithms aiming to produce robust and reliable high stable schedules.

New match-up strategies are initially introduced to manage a disruption that often occurs in the investigated problem, in which rush orders have to be integrated in a current schedule. Statistical multi-comparison tests and analysis of variance reveal that even with the presence of the two conflicting criteria *Performance* and *Stability* these algorithms produce high quality stable schedules on different problem instances, which highlight their strengths regarding possible scenarios tackled by the analysed printing company. It was observed that match-up algorithms posses the best attributes of other rescheduling strategies as “right shift”, “insertion in the end” and “total rescheduling”, but overcome their weaknesses in managing either *Performance* or *Stability*. Note that the genetic algorithm fitness function responsible to reschedule affected operations is identified as a limitation of the proposed approach. First, this function only optimises *Performance*, while match-up strategies are responsible to keep good *Stability* by requiring partial modifications of schedules. Second, there is no explicit strategy to keep repaired allocations within a same time window defined for rescheduling. Consequently, possible overlaps between current and repaired schedules inevitably compromise the overall quality of produced solutions.

Further investigation of match-up strategies are done in order to check they behaviour on repairing a more general case of disruption, in which newly arriving jobs have different levels of urgency, referred as normal orders. Statistical analysis confirms that these algorithms are highly flexible to deal with complex disruptions since they are able to deliver highly stable and good performing schedules even when disruptions with different levels of urgency arise in the shop floor. Additionally, improvements in the genetic algorithm fitness function for reallocating affected operations have a more effective control of both *Performance* and *Stability*. The new settings minimise the makespan, which reduces the overlaps between initial and new schedules, and maximise *Stability*, which reduces changes in both sequence and processing time of operations. Note that a double control of *Stability* is employed since match-up algorithms are now coupled with the new settings of the fitness function.

The use of idle times by match-up strategies indicates their potential to work cooperatively with initial robust schedules. A fuzzy scheduling system responsible for inserting idle times on machines is then proposed, in which robust schedules are produced. As a result, match-up algorithms are able to employ smaller changes in current schedules since they have a higher availability of idle times on machines. Other heuristics for robust scheduling are analysed; however they often compromise too much the *Performance* of schedules. Statistical analysis confirms that their combination is effective in managing both rush and normal orders, in which even more reliable high quality stable schedules are delivered.

In summary, match-up rescheduling algorithms and their combination with initial robust schedules set flexible approaches to manage complex disruptions that affect multiple resources in a dynamic and uncertain shop floor. These encouraging results highlight them as good candidates for possible incorporation into the scheduling / rescheduling system of investigated printing company and other similar production shops. The remarkable production of such good performing and highly stable schedules point up their relevance to both scheduling and rescheduling research communities.

It is important to highlight that scheduling / rescheduling solutions proposed by the research group from the University of Nottingham have been used by Sherwood Press.

7.2 Limitations

The study presented in thesis has the following limitations:

- *Match-up strategy selection*: strategies have to be selected manually by the system user;
- *Job insertion*: only one job can be inserted per time;
- *Overlap control*: the use of right-shift rescheduling to manage overlaps between schedules is considered sub-optimal, since they may compromise the schedule *Performance*;
- *Rescheduling horizon*: no strategies were implemented to prioritise the definition of rescheduling horizons where more idle times are available;
- *Scheduling / rescheduling solver*: only genetic algorithms were considered to allocate jobs on machines, while a comparison with different search methods would be beneficial.

Possible extensions considering these items are described in the following section.

7.3 Future work

The arrival of orders is considered as a generalisation of possible disruptions due to its ability to affect multiple resources available in a shop floor. However, further investigation into match-up algorithms and robust schedules can be done to analyse the specific effects generated by other types of disruptions, such as order changes, cancellation of jobs and requirements of rework when the product quality is not satisfactory. In addition, the resources changes can also be considered, such as multiple machine breakdowns, unavailability of raw materials, sickness of workers, among others depending on the specificity of the investigated scheduling / rescheduling problem.

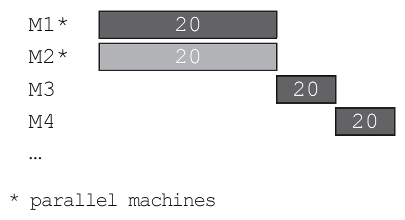
Rescheduling has been almost entirely focused on production scheduling. However, the proposed ideas can be extended to other problem domains, such as personnel scheduling and university timetabling to include disturbances like the absence of nurses, non-availability of lecturers and rooms, etc.

The following approaches illustrate possible strategies to improve the current scheduling / rescheduling system:

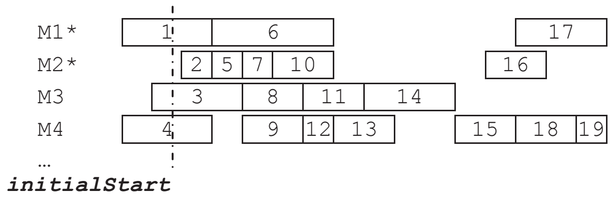
- *Dynamic selection of a best match-up strategy*: an optimisation model to dynamically select the most appropriated match-up strategy among S1-S8 aiming to deliver a best schedule at a certain moment;

- *Setting smaller rescheduling horizons*: the collection of idle times could start at a different rescheduling point. Figure 7.1 shows an example of a new job 20 requiring processing on machines M1 (or M2, which is a parallel machine), M3 and M4. The current time is highlighted by the variable *initialStart* and the original match-up algorithm set as a rescheduling *startPoint* the latest point when already started operations finish their processing, i.e. when the operation 3 on M3 is completed, as in Figure 7.1 (c). Note that there is no available idle time at this time. Consequently, a new approach would set as *startPoint* the first available point which has available idle times, as in Figure 7.1 (d). Subsequently, rescheduling horizons are calculated for both approaches. Figure 7.2 (a)-(b) and (c)-(d) shows the original and the new approach been employed, respectively, where match-up strategies S1 and S5 are applied for each approach. Smaller rescheduling horizons are then defined by the new approach, in which a smaller number of operations is affected during the rescheduling process. Their possible impacts in both *Performance* and *Stability* require further investigation.

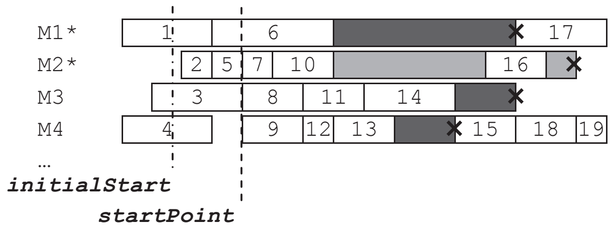
- *Insertion of multiple jobs per time*: multiple rescheduling horizons could be defined and a multi-agent system could be responsible to manage these sub-problems. Fig-



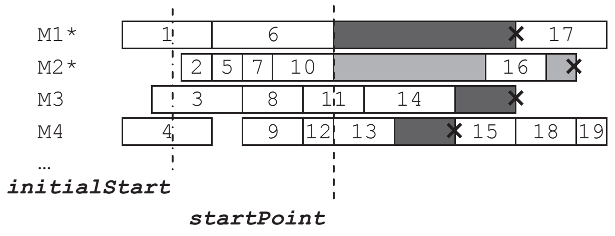
(a)



(b)



(c)



(d)

Figure 7.1: Alternative approach to set rescheduling horizons; (a) new job requirements, (b) current time represented by *initialStart*, (c) original approach setting *startPoint* and (d) new approach setting *startPoint*

ure 7.3 shows an example in which jobs 20 and 21 must be accommodated in a current schedule. Note that two independent rescheduling horizons are defined and their possible effects require further analyses;

- *Other optimisation methods for scheduling and rescheduling*: compare the application of match-up algorithms with other search methods such as simulated annealing, tabu search, branch and bound algorithms and Pareto efficient solutions. Note that

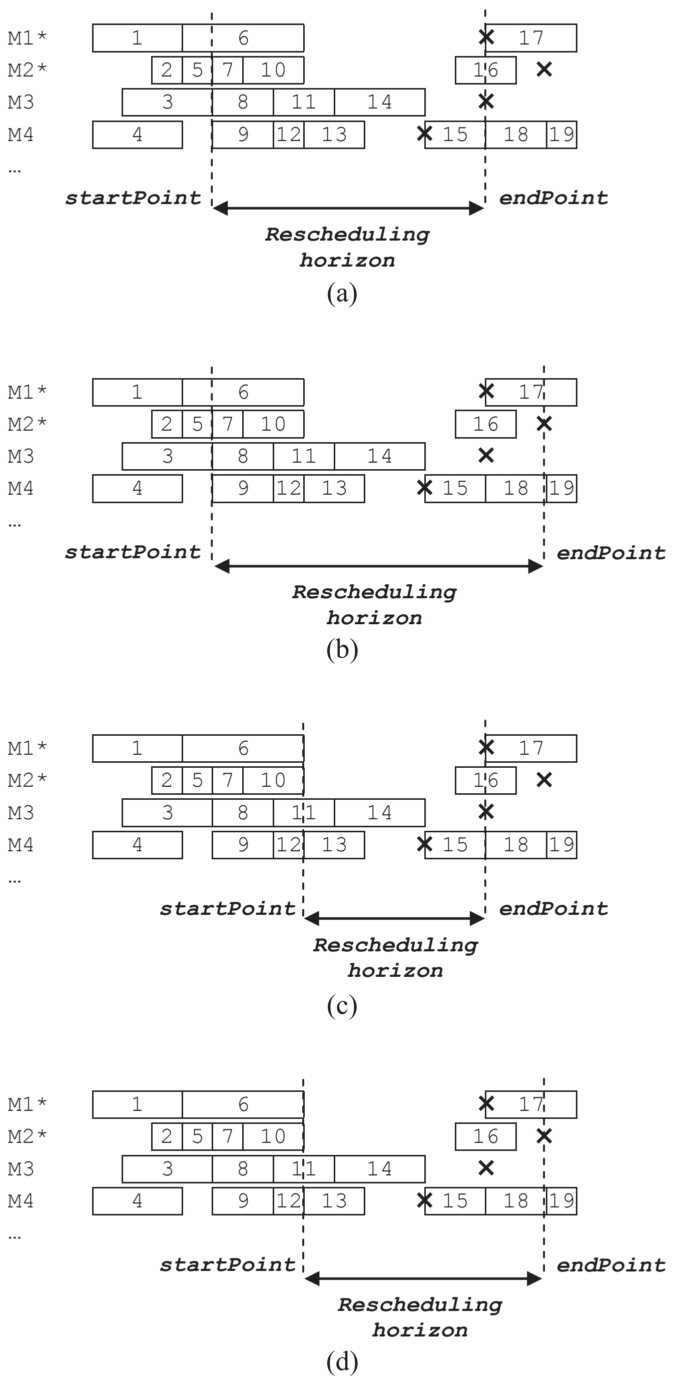
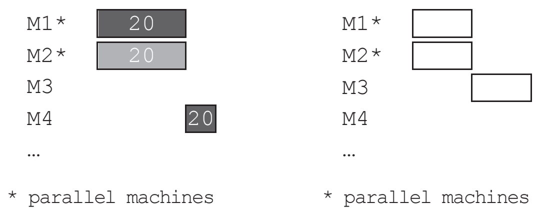
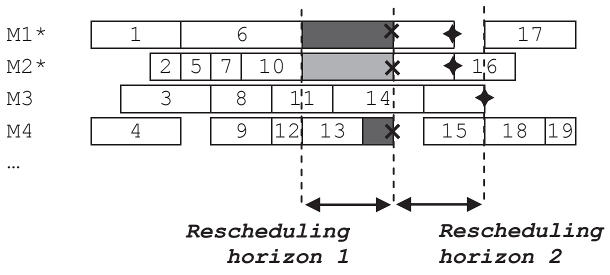


Figure 7.2: Setting smaller rescheduling horizons; (a)-(b) the calculation of *endPoint* using strategy S1 and S5 with the original approach, (c)-(d) the calculation of *endPoint* using strategy S1 and S5 with the new approach

genetic algorithms have been mainly used in this thesis because they previously provide encouraging results for the static scheduling problem presented by Sherwood Press - Nottingham, UK;



(a)



(b)

Figure 7.3: Setting multiple rescheduling horizons when inserting multiple jobs; (a) new jobs requirements, (b) the calculation of two rescheduling horizons

- *Other optimisation techniques to restore the schedule feasibility*: the previously mentioned search methods can also be applied to control possible overlaps between initial and repaired schedules on rescheduling. Further comparisons with these techniques would be significant to the research scope;

- *Other approaches to insert idle times on machines*: initial robust schedules can be also produced using other data analysis models such as clustering, neural networks, case-based reasoning and artificial intelligence agents. A comparison with these approaches would bring relevant discussions;

- *Preventive maintenance scheduling*: new “fake” jobs could be inserted as a preventive strategy. These jobs would generate extra idle times on machines, which could be use to allocate the maintenance of the available resources. A study to investigate their impact on disruptions such as machine breakdown and rework of jobs would be beneficial to the scheduling / rescheduling community.

Bibliography

[1] R. Abumaizar and J. Svestka. Rescheduling job shops under random disruptions. *International Journal of Production Research*, 35(7):2065–2082, 1997.

[2] M. Akturk, J. Ghoshb, and E. Gunesc. Scheduling with tool changes to minimize total completion time: Basic results and spt performance. *European Journal of Operational Research*, 157(3):784–790, 2004.

[3] M. Akturk and E. Gorgulu. Match-up scheduling under a machine breakdown. *European Journal of Operational Research*, 112(1):81–97, 1999.

[4] N. Al-Hinai and T. ElMekkawy. Robust and stable flexible job shop scheduling with random machinebreakdowns using a hybrid genetic algorithm. *International Journal of Production Economics*, 132(2):279–291, 2011.

[5] P. Alcan and H. Basligil. A genetic algorithm application using fuzzy processing-times in non-identical parallel machine scheduling problem. *Advances in Engineering Software*, 45(1):272–280, 2012.

[6] H. Aytug, S. Bhattacharyya, G. Koehler, and J. Snowdon. A review of machine learning in scheduling. *IEEE Transactions on Engineering Management*, 41(2):165–171, 1994.

[7] H. Aytug, G. Koehler, and J. Snowdon. Genetic learning of dynamic scheduling within a simulation environment. *Computers and Operations Research*, 21:909–925, 1994.

[8] H. Aytug, M. Lawley, K. McKay, S. Mohan, and R. Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161(1):86–110, 2005.

[9] T. Bagchi. *Multi-Objective Scheduling by Genetic Algorithms*. Kluwer Academic Publishers, 1999.

[10] S. Balin. Parallel machine scheduling with fuzzy processing times using a robust genetic algorithm and simulation. *Information Sciences*, 181(17):3551–3569, 2011.

[11] J. Bean and J. Birge. Match-up real-time scheduling. In *Proceedings of the Symposium on Real-Time Optimization in Automated Manufacturing Facilities, National Bureau of Standards, Special Publication 724*, pages 197–212, 1986.

[12] J. Bean, J. Birge, J. Mittenthal, and C. Noon. Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39(3):470–483, 1991.

[13] C. Bierwirth and D. Mattfeld. Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation*, 7(1):1–17, 1999.

[14] J. Birge and M. Dempster. Optimality conditions for match-up strategies in stochastic scheduling. Technical Report Technical Report 92 - 58, Department of Industrial and Operations Engineering, The University of Michigan, 1992.

[15] J. Birge and M. Dempster. Optimal match-up strategies in stochastic scheduling. *Discrete Applied Mathematics*, 57(2-3):105–120, 1995.

[16] E. Burke, M. Misir, G. Ochoa, and E. Ozcan. Learning heuristic selection in hyperheuristics for examination timetabling. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT'08)*, pages 1–4, Montreal,Canada, 18-22 August 2008.

[17] P. Caricato and A. Grieco. An online approach to dynamic rescheduling for production planning applications. *International Journal of Production Research*, 46(16):4597–4617, 2008.

[18] F. Chan, K. Choy, and Bibhushan. A genetic algorithm-based scheduler for multi-product parallel machine sheet metal job shop. *Expert Systems with Applications*, 38(7):8703–8715, 2011.

[19] C. Chase and P. Ramadge. On real-time scheduling policies for flexible manufacturing systems. *IEEE Transactions on Automatic Control*, 37(4):491–496, 1992.

[20] C. Chen and Y. Yih. Identifying attributes for knowledge-based development in dynamic scheduling environments. *International Journal of Production Research*, 34:1739–1755, 1996.

[21] W. Chiang and M. Fox. Protection against uncertainty in a deterministic schedule. In *Proceedings of The Fourth International Conference on Expert systems in Production and Operations Management*, pages 184–197, Hilton Head Island, SC, May 1990.

[22] B. Choi, K. Lee, J. Leung, and M. Pinedo. Flowshops with machine maintenance: Ordered and proportionate cases. *European Journal of Operational Research*, 207(1):97–104, 2010.

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

[24] P. Cowling and M. Johansson. Using real time information for effective dynamic scheduling. *European Journal of Operational Research*, 139(2):230–244, 2002.

[25] E. Cox. *The Fuzzy Systems Handbook*. Academic Press, Massachussets, 1994.

[26] A. Davenport, C. Gefflot, and J. Beck. Slack-based techniques for robust schedules. In *Proceedings of the Sixth European Conference on Planning*, pages 7–18, 2001.

[27] M. Donath and R. Graves. Flexible assembly systems: an approach for near real-time scheduling and routing of multiple products. *International Journal of Production Research*, 26(12):1903–1919, 1988.

[28] J. Dorn, R. Kerr, and G. Thalhammer. Reactive scheduling: improving the robustness of schedules and restricting the effects of shop floor disturbances by fuzzy reasoning. *International Journal of Human Computing Studies*, 42(6):687–704, 1995.

[29] D. Driankov, H. Hellendoorn, and M. Reinfrank. *An introduction to fuzzy control*. Springer, 1996.

[30] C. Duron, M. Loulyb, and J. Proth. The one machine scheduling problem: Insertion of a job under the real-time constraint. *European Journal of Operational Research*, 199(3):695–701, 2009.

[31] C. Duron, J. Proth, and Y. Wardi. Insertion of a random task in a schedule: a real-time approach. *European Journal of Operational Research*, 164(1):52–63, 2005.

[32] C. Fayad and S. Petrovic. A fuzzy genetic algorithm for real-world job shop scheduling. In *IEA/AIE '2005: Proceedings of the 18th international conference on Innovations in Applied Artificial Intelligence*, pages 524–533, London, UK, 2005. Springer-Verlag.

[33] F. Filip, G. Neagu, and D. Donciulescu. Job shop scheduling optimization in real-time production control. *Computers in Industry*, 4:395–403, 1983.

[34] J. Freeman. Fuzzy systems for control applications: The truck backer-upper. *The Mathematica Journal*, 4:64–69, 1994.

[35] H. Gao. *Building robust schedules using temporal protection an empirical study of constraint based scheduling under machine failure uncertainty*. PhD thesis, Toronto University, Canada, 1995.

[36] H. Gao, M. Fox, W. Chiang, and S. Hikita. Building robust schedules - an empirical study of single machine scheduling with uncertainty. Unpublished, 1995.

[37] J. Gibbs, G. Kendall, and E. Ozcan. Scheduling english football fixtures over the holiday period using hyper-heuristics. In *Proceedings of the 11th International Conference on Parallel Problem Solving From Nature, Lecture Notes in Computer Science*, volume 6238, pages 496–505, Krakow, Poland, 11-15 September 2011.

[38] F. Gomide, R. Gudwin, and R. Tanscheit. Conceitos fundamentais da teoria de conjuntos fuzzy e aplicacoes. In *Tutorials of 6th International Fuzzy Systems Association World Congress - IFSA 95*. Sao Paulo, SP, Brazil, 21-28 July 1995. (In Portuguese).

[39] B. Grabot and L. Geneste. Dispatching rules in scheduling: a fuzzy approach. *International Journal of Production Research*, 32(4):903–915, 1994.

[40] R. Guerra, S. Sandri, and M. Souza. Dynamics and design of autonomous attitude control of a satellite using fuzzy logic. In *Proceedings of COBEM97: Congresso Brasileiro de Engenharia Mecanica*, Bauru, SP, Brazil, 1997. ABCM.

[41] M. Hapke, A. Jaszkiewicz, and R. Slowinski. Fuzzy project scheduling system for software development. *Fuzzy Sets and Systems*, 67(1):101–117, 1994.

[42] W. Herroelen and R. Leus. Robust and reactive project scheduling: a review and classification of procedures. *International Journal of Production Research*, 42(8):1599–1620, 2004.

[43] T. Hong and T. Chuang. New triangular fuzzy johnson algorithm. *Computers and Industrial Engineering*, 36:179–200, 1999.

[44] W. Hopp and M. Spearman. *Factory Physics*. Irwin / McGraw-Hill, 1996.

[45] H. Ishibuchi and T. Murata. Flowshop scheduling with fuzzy duedate and fuzzy processing time. In R. Slowinski and M. Hapke, editors, *Scheduling Under Fuzziness*. Physica-Verlag, Heidelberg, 2000.

[46] H. Ishii and M. Tada. Single machine scheduling problem with fuzzy precedence relation. *European Journal of Operational Research*, 87:284–288, 1995.

[47] M. Jensen. Generating robust and flexible job shop schedules using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 7(3):275–288, 2003.

[48] G. Klir and T. Folger. *Fuzzy sets, uncertainty and information*. Prentice Hall, Englewood Cliffs, N.J., 1988.

[49] G. Klir and B. Yuan. *Fuzzy sets and fuzzy logic: theory and applications*. Prentice Hall, Upper Saddle River, NJ, 1995.

[50] B. Kosko. *Fuzzy Engineering*. Prentice Hall, Upper Saddle River, 1997.

[51] B. Kosko and S. Kong. *Comparison of Fuzzy and Neural Truck Backer-Upper Control Systems*. Prentice Hall, Englewood Cliffs, N.J., 1992.

[52] C. Koulamas, S. Gupta, and G. Kyparisis. A unified analysis for the single-machine scheduling problem with controllable and non-controllable variable job processing times. *European Journal of Operational Research*, 205(2):479–482, 2010.

[53] A. Koulouris and I. Kotelida. Simulation-based reactive scheduling in tomato processing plant with raw material uncertainty. *Computer Aided Chemical Engineering*, 29:1020–1024, 2011.

[54] J. Krucky. Fuzzy family setup assignment and machine balancing. *Hewlett-Packard Journal*, June:51–64, 1994.

[55] M. Kuroda and Z. Wang. Fuzzy job shop scheduling. *International Journal of Production Economics*, 44:45–51, 1996.

[56] P. Larsen. Industrial applications of fuzzy logic control. *International Journal of Man-Machine Studies*, 12:310, 1980.

[57] C. Lee. Fuzzy logic in control systems: Fuzzy logic controller - part i & part ii. *IEEE Transactions on Systems, Man and Cybernetics*, 20:404–435, 1990.

[58] J. Lee and Y. Kim. Minimizing the number of tardy jobs in a single-machine scheduling problem with periodic maintenance. *Computers & Operations Research*, 39(9):2196–2205, 2012.

[59] V. Leon, S. Wu, and R. Storer. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43, 1994.

[60] H. Li, Z. Li, L. Li, and B. Hu. A production rescheduling expert simulation system. *European Journal of Operational Research*, 124(2):283–293, 2000.

[61] C. Liao and W. Chen. Scheduling under machine breakdown in a continuous process industry. *Computers & Operations Research*, 31(3):415–428, 2004.

[62] C. Lu, S. Lin, and K. Ying. Robust scheduling on a single machine to minimize total flow time. *Computers & Operations Research*, 39(7):1682–1691, 2012.

[63] J. Lukasiewicz. On three-valued logic. In L. Borkowski, editor, *Selected works (Studies in logic and the foundations of mathematics)*, pages 87–88. North-Holland, Amsterdam, 1970. ISBN 0-7204-2252-3. Original Paper: J. Lukasiewicz, O logice trojwartosciowej, Ruch filozoficzny 5:170–171, 1920. (In Polish).

[64] T. Maekawa, K. Yasuda, R. Yokoyama, H. Ohtsuki, and Y. Mizukami. Fuzzy coordination of multi-objectives reflecting the operator's intention for dynamic generation rescheduling. *International Journal of Electrical Power and Energy Systems*, 14(5):314–320, 1992.

[65] E. Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. *Proceedings of IEEE*, 121(12):1585–1588, 1974.

[66] L. McKenzie. Turnpike theory. *Econometrica*, 44(5):841–865, 1976.

[67] S. Mehta and R. Uzsoy. Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing*, 12(1):15–38, 1999.

[68] R. Miller. *Simultaneous Statistical Inference*. Springer-Verlag, New York, 1991.

[69] P. Moratori, S. Petrovic, and D. Petrovic. A match-up algorithm for job shop rescheduling. In *Annual Operational Research Conference 49 (OR49)*, Edinburgh, UK, 4-6 September 2007.

[70] P. Moratori, S. Petrovic, and J. Vázquez-Rodríguez. Match-up strategies for job shop rescheduling. In N. Nguyen, L. Borzemski, A. Grzech, and M. Ali, editors, *New Frontiers in Applied Artificial Intelligence, Lecture Notes in Computer Science*, volume 5027, pages 119–128. Springer-Verlag, 2008.

[71] P. Moratori, S. Petrovic, and J. Vázquez-Rodríguez. Fuzzy approaches for robust job shop rescheduling. In *IEEE International Conference on Fuzzy Systems*, pages 1–7, Barcelona, Spain, July 18-23 2010.

[72] P. Moratori, S. Petrovic, and J. Vázquez-Rodríguez. Hibridisation of fuzzy robust scheduling with match-up approaches. In C. Antunes, D. Insua, and L. Dias, editors, *Proceedings of the 25th Mini-euro Conference Uncertainty and Robustness in Planning and Decision Making*, pages 1–7, Coimbra, Portugal, April 15-17 2010. ISBN 978-989-95055-3-7.

[73] P. Moratori, S. Petrovic, and J. Vázquez-Rodríguez. Integrating rush orders into existent schedules for a complex job shop. *Applied Intelligence*, 32(1):205–215, 2010.

[74] P. Moratori, S. Petrovic, and J. Vázquez-Rodríguez. Match-up approaches for a dynamic scheduling problem. *International Journal of Production Research*, 50(1):261–276, 2012.

[75] A. Muhlemann, A. Lockett, and C. Farn. Job shop scheduling heuristics and frequencies of scheduling. *International Journal of Production Research*, 20(2):227–241, 1982.

[76] T. Murata, M. Gen, and H. Ishibuchi. Multi-objective scheduling with fuzzy due-date. *Computers and Industrial Engineering*, 35:439–442, 1998.

[77] S. Noori-Darvish, I. Mahdavi, and N. Mahdavi-Amiri. A bi-objective possibilistic programming model for open shop scheduling problems with sequence-dependent setuptimes, fuzzy processing times, and fuzzy due dates. *Applied Soft Computing*, 12(4):1399–1416, 2012.

[78] G. Nurcahyo, S. Shamsuddin, R. Alias, and M. Sap. Selection of defuzzification method to obtain crisp value for representing uncertain data in a modified sweep algorithm. *JCS&T*, 3(2):22–28, 2003.

[79] D. Ouelhadj and S. Petrovic. Survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12(4):417–431, 2009.

[80] S. Panwalkar and W. Iskander. A survey of scheduling rules. *Operational Research*, 25:45–61, 1977.

[81] A. Patel. Simplest fuzzy pi controllers under various defuzzification methods. *International Journal of Computational Cognition*, 3:21–34, 2005.

[82] D. Petrovic and A. Duenas. A fuzzy logic based production scheduling/rescheduling in the presence of uncertain disruptions. *Fuzzy Sets and Systems*, 157:2273–2285, 2006.

[83] R. Petrovic and D. Petrovic. Multicriteria ranking of inventory replenishment policies in the presence of uncertainty in customer demand. *International Journal of Production Economics*, 71:439–446, 2001.

[84] S. Petrovic and C. Fayad. A genetic algorithm for job shop scheduling with load balancing. In S. Zhang and R. Jarvis, editors, *Proceedings of the 18th Australian Conference on Artificial Intelligence*, pages 339–348, Sydney, Australia, 5-9 December 2005.

[85] S. Petrovic, C. Fayad, and D. Petrovic. Job shop scheduling with lot-sizing and batching in an uncertain real-world environment. In G.Kendall, L.Lei, and M.Pinedo, editors, *MISTA 2005: Proceeding of the 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications*, pages 363–379, New York, USA, 18-21 July 2005.

[86] S. Petrovic, C. Fayad, D. Petrovic, E. Burke, and G. Kendall. Fuzzy job shop scheduling with lot-sizing. *Annals of Operations Research*, 159(1):275–292, 2008.

[87] S. Petrovic and M. Geiger. A fuzzy scheduling problem with dynamic job priorities and an extension to multiple criteria. In R. Meredith, G. Shanks, D. Arnold, and S. Carlsson, editors, *Decision Support in an Uncertain and Complex World: Proceedings of the 2004 IFIP International Conference on Decision Support Systems (DSS2004)*, pages 637–646, Prato, Tuscany, Italy, 1-3 July 2004. ISBN 0-7326-2269-7.

[88] S. Petrovic, D. Petrovic, and E. Burke. *Handbook of Production Planning*, chapter Fuzzy Logic Based Production Scheduling and Rescheduling in the Presence of Uncertainty. Springer, 2008. to appear.

[89] S. Petrovic, D. Petrovic, and E. Burke. Fuzzy logic based production scheduling and rescheduling in the presence of uncertainty. In K. Kempf, P. Keskinocak, and R. Uzsoy, editors, *Planning Production and Inventories in the Extended Enterprise*, volume 152, pages 531–562. Springer International Series in Operation Research and Management Science, 2011. ISBN 978-1-4419-8191-2.

[90] A. Pfeiffer, B. Kádár, and L. Monostori. Stability-oriented evaluation of hybrid rescheduling methods in a job-shop with machine breakdowns. *CIRP Journal of Manufacturing Systems*, 35(6):563–570, 2006.

[91] M. Pinedo. *Scheduling Theory, Algorithms and Systems*. Prentice Hall, 2002.

[92] G. Rabadi, M. Mollaghasemi, and G. Anagnostopoulos. A branch-and-bound algorithm for the early/tardy machine scheduling problem with a common due-date and sequence-dependent setup time. *Computers & Operations Research*, 31(10):1727–1751, 2004.

[93] R. Rangsaritratsamee, W. F. Jr, and M. Kurz. Dynamic rescheduling that simultaneously considers efficiency and stability. *Computers & Industrial Engineering*, 46(1):1–15, 2004.

[94] C. Reeves. Genetic algorithms and combinatorial optimisation. In V. Rayward-Smith, editor, *Applications of Modern Heuristic Techniques*. Alfred Waller Ltd, Henley-on-Thames, UK, 1995.

[95] C. Reeves. Landscapes, operators and heuristic search. *Annals of Operations Research*, 86:473–490, 1999.

[96] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009.

[97] N. Sadeh. Micro-opportunistic scheduling: The micro-boss factory scheduler. In *Intelligent Scheduling*, chapter 4, pages 99–136. Morgan Kaufmann, 1994.

[98] M. Sakawa and R. Kubota. Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms. *European Journal of Operational Research*, 120:393–407, 2000.

[99] S. Sandri and C. Correa. Logica nebulosa. In *Anais da V Escola de Redes Neurais*, pages 73–90, Sao Jose dos Campos, SP, Brazil, 1999. (In Portuguese).

[100] V. Sels, K. Craeymeersch, and M. Vanhoucke. A hybrid single and dual population search procedure for the jobshop scheduling problem. *European Journal of Operational Research*, 215(3):512–523, 2011.

[101] I. Shaw and M. Simoes. *Controle e Modelagem Fuzzy*. Edgard Blucher, 1999. (In Portuguese).

[102] W. Slany. Scheduling as a fuzzy multiple criteria optimization problem. *Fuzzy Sets and Systems*, 78(2):197–222, 1996.

[103] S. Smith. Reactive scheduling systems. In D. Brown and W. Scherer, editors, *Intelligent Scheduling Systems*. Kluwer Press, 1995.

[104] S. Smith, N. Muscettola, D. Matthys, P. Ow, and J. Potvin. Opis: an opportunistic factory scheduling system. In *IEA/AIE '90: Proceedings of the 3rd international conference on Industrial and engineering applications of artificial intelligence and expert systems*, pages 268–274, New York, NY, USA, 1990. ACM.

[105] P. Stanfield, R. King, and J. Joines. Scheduling arrivals to a production system in a fuzzy environment. *European Journal of Operational Research*, 93:75–87, 1996.

[106] J. Sun and D. Xue. A dynamic reactive scheduling mechanism for responding to changes of production orders and manufacturing resources. *Computers in Industry*, 46(2):189–207, 2001.

[107] E. Szelke and R. Kerr. Knowledge-based reactive scheduling. *Production Planning and Control*, 5(2):124–145, 1994.

[108] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man and Cybernetics*, 15:116–132, 1985.

[109] R. Tanscheit. *Study of a Rule-based Self-Organising Controller for Robotics Applications*. PhD thesis, University of London, 1988.

[110] S. Topaloglu and H. Selim. Nurse scheduling using fuzzy modeling approach. *Fuzzy Sets and Systems*, 161(11):1543–1563, 2010.

[111] I. Toroslu and Y. Arslanoglu. Genetic algorithm for the personnel assignment problem with multiple objectives. *Information Sciences*, 177(3):787–803, 2007.

[112] Y. Tsujimura, S. Park, S. Chang, and M. Gen. An effective method for solving flow shop scheduling problems with fuzzy processing times. *Computers and Industrial Engineering*, 25:239–242, 1993.

[113] T. Tsukamoto. An approach to fuzzy reasoning method. In M. Gupta, R. Regade, and R. Yager, editors, *Advances in Fuzzy Set Theory and Applications*. North-Holland, Amsterdam, 1979.

[114] G. Vieira, J. Herrmann, and E. Lin. Rescheduling manufacturing systems: A framework of strategies, policies and methods. *Journal of Scheduling*, 6(1):39–62, 2003.

[115] T. Wong, W. Leung, K. Mak, and R. Fung. Integrated process planning and scheduling / rescheduling - an agent-based approach. *International Journal of Production Research*, 44(18):3627–3655, 2006.

[116] S. Wu, R. Storer, and P. Chang. One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers and Operations Research*, 20(1):1–14, 1993.

[117] R. Yager. On ordered weighted averaging aggregation operators in multi-criteria decision making. In D. Dubois, H. Prade, and R. Yager, editors, *Readings in Fuzzy Sets for Intelligent Systems*, pages 80–87. Morgan Kaufmann, San Mateo, 1993.

[118] L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[119] L. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.

[120] X. Zhanga and S. Veldeb. Approximation algorithms for the parallel flowshop problem. *European Journal of Operational Research*, 216(3):544–552, 2012.

[121] H. Zimmermann. *Fuzzy Set Theory and its Applications*. Kluwer Academic Publishers, Dordrecht, 3rd edition, 1996.