

DISSERTAÇÃO DE MESTRADO Nº 1086

**SOLUTION OF A SCHEDULING PROBLEM USING AN ABSTRACTION OF
THE CLOSED LOOP BEHAVIOUR OF A DISCRETE EVENT SYSTEM**

Gustavo Caetano Rafael

DATA DA DEFESA: 12/11/2018

Universidade Federal de Minas Gerais

Escola de Engenharia

Programa de Pós-Graduação em Engenharia Elétrica

**SOLUTION OF A SCHEDULING PROBLEM USING AN
ABSTRACTION OF THE CLOSED LOOP BEHAVIOUR OF A
DISCRETE EVENT SYSTEM**

Gustavo Caetano Rafael

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do Título de Mestre em Engenharia Elétrica.

Orientadora: Profa. Patrícia Nascimento Pena

Belo Horizonte - MG

Novembro de 2018

R136s

Rafael, Gustavo Caetano.

Solution of a scheduling problem using an abstraction of the closed loop behaviour of a discrete event system [manuscrito] / Gustavo Caetano Rafael. - 2018.

55 f., enc.: il.

Orientadora: Patrícia Nascimento Pena.

Dissertação (mestrado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Bibliografia: f. 51-55.

1. Engenharia elétrica - Teses. 2. Algoritmos evolutivos - Teses. 3. Heurística - Teses. 4. Otimização combinatória - Teses. 5. Teoria dos autômatos - Teses. I. Pena, Patrícia Nascimento. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 621.3(043)

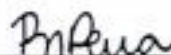
**"Solution of a Scheduling Problem Using an Abstraction of the
Closed Loop Behaviour of a Discrete Event System"**

Gustavo Caetano Rafael

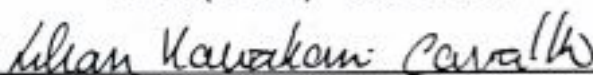
Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Mestre em Engenharia Elétrica.

Aprovada em 12 de novembro de 2018.

Por:



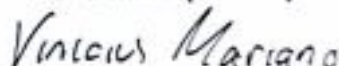
Prof. Dr. Patricia Nascimento Pena
DELT (UFMG) - Orientadora



Prof. Dra. Lilian Kawakami Carvalho
Departamento de Engenharia Elétrica (UFRJ)



Prof. Dr. Ricardo Hiroshi Caldeira Takahashi
DMAT (UFMG)



Prof. Dr. Vinicius Mariano Gonçalves
DEE (UFMG)

Acknowledgements

I would like to thank my father, Geraldo, for helping in my initial steps in my technical career (teaching me how to re-assembling my toys) and for the extra prays every night. My mother, Suzete, to whom I could not be more thankful, for the unconditional support (even when she was the only one believing on me), for the extra patience to teach me (in my earlier years) and for inspiring me and my sister to always learn more. To my little sister, Marianna, there are not enough words to say how important she was for this accomplishment. Thanks for listening me every night (and day), for motivating me to keep fighting every day of my life and for being my most dedicated career manager. To my aunts, Marly and Efigênia, for supporting me even when they did not understand what I was doing. They are an example of courage and faith and a source of inspiration on how a dedicated and responsible son should I be. To all my family (uncles, aunts, cousins, godson and goddaughter) thanks for supporting me in this achievement. In memory of my biggest supporter, my grandmother Geralda, who rests in peace in the glory of God.

Last, but certainly not least, I am very grateful to my advisor, Professor Patrícia Nascimento Pena, for the orientation, the patience and for pushing me to reach my best with her "Impress me!!!". I would like to give a special thanks to all my friends from the LACSED, for all the fun, for joining me in the "LAC-Tetrix's" moments and for the long discussions in the whiteboard. Thanks to the staff of UFMG's Engineering School that participated in my academic training. Moreover, I would like to acknowledge the funding agencies FAPEMIG, CAPES and CNPQ for funding my education.

To all my friends all around the world, in special Felipe Castro, Carlos Roberto jr. e Matheus Araújo, a big thanks for help during this whole journey (from undergraduate studies until the graduate school) you are the best investments that the Gustavo's Corporation did.

Agradecimentos

Eu gostaria de agradecer ao meu pai, Geraldo, por me ajudar nos meus primeiros passos em minha jornada técnica (me ensinado como remontar meus brinquedos) e pelas orações extras todas as noites. À minha mãe, Suzete, por seu apoio incondicional (mesmo nos momentos quando ela era a única pessoa que acreditava em mim), pela paciência extra ao me ensinar (nos meus primeiros anos escolares) e por inspirar a mim e minha irmã a sempre continuar aprendendo. À minha irmã, Marianna, não há palavras o bastante para descrever quão importante ela foi para essa conquista. Obrigado por sempre me ouvir todas as noites (e dias), por me motivar continuar lutando todos os dias da minha vida e ser minha mais dedicada empresária. As minhas tias, Marly e Efigênia, por me apoiarem mesmo quando elas não entendiam o que eu estava fazendo. Elas são um exemplo de coragem, fé e uma fonte de inspiração em como ser um bom filho. Aos meus familiares (tios, tias, primos, afilhado e afilhada) sou grato por terem me apoiado em mais essa conquista. Em memória da minha maior fã, minha avó Geralda, que descansa em paz na glória de Deus.

Por último, mas não menos importante, eu sou muito grato a minha orientadora, Professora Patrícia Nascimento Pena, por sua orientação, pela paciência e por me motivar à atingir meu melhor com seu "Me Impressionem!!!". Eu também gostaria de agradecer todos os meus amigos do LACSED, pelos momentos divertidos, por se juntarem a mim nos momentos de "LAC-Tetrix" (re-organização do laboratório) e nas longas discussões no quadro. Obrigado aos funcionários da Escola de Engenharia da UFMG, que participaram da minha vida acadêmica. Além disso, gostaria de agradecer as agências de financiamento à pesquisa FAPEMIG, CAPES e CNPQ, por investirem na minha educação.

À todos os meus amigos ao redor do mundo, em especial Felipe Castro, Carlos Roberto Jr. e Matheus Araújo, um grande obrigado por toda ajuda durante toda essa jornada (da graduação até a pós-graduação), vocês foram o melhor investimento da Gustavo Corporation.

If I have seen further than others, it is by standing upon the shoulders of giants. (Se eu pude enxergar mais longe, é por que me apoiei nos ombros de gigantes.)

Sir.Isaac Newton

Resumo

Esta dissertação aborda o problema de planejamento de produção no ambiente industrial sob a perspectiva de Sistemas a Eventos Discretos. Para tal, foi utilizada uma solução obtida pela aplicação da Teoria de Controle Supervisório como espaço de busca em um problema de otimização, juntamente com algoritmos evolucionários. Na literatura, essa abordagem foi descrita como CSO - Controle Supervisório e Otimização. Na CSO busca-se uma cadeia do comportamento em malha fechada, que minimiza o *makespan* para a produção de um lote de produtos. No presente trabalho, propõe-se uma heurística para geração de indivíduos garantidamente factíveis Rafael & Pena (2018), além do uso de uma abstração do comportamento em malha fechada. Por fim, a heurística proposta é aplicada em três estudos de caso. Para um deles (o Sistema Flexível de Manufatura) foi possível encontrar a solução ótima para todas as instâncias em que a mesma era conhecida.

Abstract

This dissertation deals with a production-planning problem in the industrial environment from the perspective of Discrete Event Systems. For such, a solution obtained by applying the Supervisory Control Theory (SCT) was used as the search space for the optimization problem, together with evolutionary algorithms. In the literature, this method was described as the SCO - Supervisory Control and Optimization approach. In the SCO, a string of the closed-loop behavior, that minimizes the *makespan* for the production of a batch of products is sought. In the present work, a heuristic that guarantees the generation of feasible individuals is proposed (Rafael & Pena, 2018), built from an abstraction of the closed-loop behavior. Lastly, the proposed heuristic is applied in three case studies. For one of them (the Flexible manufacturing system), in all instances that the optimal solution is known, it was found.

Contents

List of Figures	ix
List of Tables	xi
Acronyms	xii
List of Symbols	xiii
1 Introduction	1
1.1 Motivations	1
1.2 Scheduling	2
1.3 Scheduling and Supervisory Control Theory	3
1.4 Objectives	6
1.5 Thesis structure	6
1.6 Final Remarks	7
2 Preliminaries	8
2.1 Discrete Events Systems	8
2.2 Automata and Language Theory	9
2.2.1 Languages	9
2.2.2 Natural Projection	9
2.2.3 Automata	10
2.3 Supervisory Control Theory	10
2.4 Supervisor Abstraction	12
2.5 Makespan evaluation	14
2.6 Clonal Selection Algorithm	15
2.7 The case studies	17
2.7.1 The Cluster Tool	17
2.7.2 The Flexible Manufacturing System	20
2.7.3 The Search For the Optimal Sequences	22
2.8 Final Remarks	24

3	Methodology	26
3.1	Problem Statement	26
3.2	Definitions	27
3.3	Individual Generation algorithm	28
3.3.1	Integration: CSA and DS states	30
3.4	Setting the optimization parameters	31
3.5	Most Frequent Event Algorithm	37
3.6	Final Remarks	37
4	Experimental Results	39
4.1	Procedure	39
4.2	The Cluster Tool	40
4.2.1	One Robot Cluster Tool	41
4.2.2	Two Robots Cluster Tool	43
4.3	The Flexible Manufacturing System	44
4.4	Final Remarks	48
5	Conclusions	49
5.1	Future Research	50
	Bibliography	51

List of Figures

1.1	Scheduling application in a manufacturing system, (Malik & Pena, 2018).	3
2.1	Observer's property representation.	10
2.2	Automaton G	11
2.3	Supervisory Control Theory Structure.	11
2.4	Simplified Manufacturing System	13
2.5	G_k is the machine model ($k = 1, \dots, 4$) and E_j is the specification ($j = 1, 2, 3$).	13
2.6	Natural projection of the supervisor S to the set of controllable events, in the SMS.	13
2.7	Quantity specification (E_q) for a batch of size 2, Definition 2.	13
2.8	Automaton P_{E_q} , represents the parallel composition between the P and E_q .	14
2.9	The automaton that implements A	14
2.10	Standard Cluster Tool layout (Lee & Ni, 2012).	17
2.11	Cluster Tool layout and the number of robots in the mainframe.	18
2.12	One Robot Cluster Tool - all machines automaton model representation.	19
2.13	One Robot Cluster Tool - Buffers specification.	19
2.14	Flexible Manufacturing System.	20
2.15	Flexible Manufacturing System automata model.	21
2.16	Flexible Manufacturing System specifications.	22
3.1	Automaton G and its DS states filled in gray	28
3.2	Automaton G	28
3.3	P_{E_q} and the sequence s highlighted in gray.	30
3.4	P_{E_q} and the sequence s_{new} highlighted in gray.	30
3.5	Cloning process diagram.	32
3.6	The population size impacts in the average makespan and optimization time.	33
3.7	The population size impacts in the average makespan and optimization time.	34
3.8	The population size impacts in the average makespan and optimization time.	35
3.9	The stop criteria and impacts in the average makespan and optimization time.	35

3.10	The effects of different amounts of TMP individuals in the average makespan and in the optimization time.	36
3.11	The automaton G illustrates the presence of the most frequent events. . .	37

List of Tables

2.1	Scheduling problems and the number of solutions.	24
2.2	Scheduling problems and the total optimization time.	24
4.1	Time interval for the Cluster Tool with one robot and four process modules, (Nunes, 2018).	41
4.2	Optimization parameters used for the cluster tool with one robot problem. .	41
4.3	One robot cluster tool makespan optimization results	42
4.4	One robot cluster tool makespan optimization results for the average makespan in 30 runs.	42
4.5	Time interval for the Cluster Tool with two robot and four PM modules,(Nunes, 2018).	43
4.6	Optimization parameters used for the cluster tool with two robots problem.	43
4.7	Two robots cluster tool makespan optimization results	44
4.8	Two robots cluster tool makespan optimization results	44
4.9	Time interval between controllable and uncontrollable events for the FSM.	45
4.10	Optimization parameters used for the FMS problem.	45
4.11	FMS Makespan optimization using the proposed methods	45
4.12	Average and standard deviation of the FMS makespan in 30 runs of the CSA.	46
4.13	CSA with random initial population versus the MPT and the Formal Verification in the FMS Makespan optimization.	47
4.14	SCO (Costa et al., 2018) versus CSA with random initial population in the FMS Makespan optimization.	48

Acronyms

DES	Discrete Events Systems
SCT	Supervisory Control Theory
SCO	Supervisory Control Optimization
JSS	Job Shop Scheduling
DS	Divergent State
CSA	Clonal Selection Algorithm
BFS	Breadth First Search
MPT	Timed Maximum Parallelism
LMP	Logical Maximum Parallelism
MPS	Minimal Parts Set
SF	Small Factory
VNS	Variable Neighborhood Search
ITL	Industrial Transfer Line
CT	Cluster Tool
CTSP	Cluster Tool Scheduling Problem
PM	Process Module
FMS	Flexible Manufacturing System
NP	Nondeterministic Polynomial Time
LP	Linear Programming
MILP	Mixed Integer Linear Programming

List of Symbols

G	Automaton
S	Supervisor
\in	Belongs to
\forall	For all
\cap	Interception Operator
\cup	Union Operator
Σ	Alphabet or set of events
Σ_C	Set of Controllable Events
Σ_{UC}	Set of Uncontrollable Events
Σ^*	Set of all Words that can be created with the events of Σ including the Null word ε
ε	Null Word
Q	Set of States
Q_m	Set of Marked States
Q_D	Set of Divergent States
Γ	Active Events Function
$\mathcal{L}(G)$	Automaton G Generated Language
$\mathcal{L}_m(G)$	Automaton G Generated Language

1

Introduction

In this chapter, the motivations for this work, a literature review on scheduling problems and some techniques applied to solve them, in the context of Discrete Events System, are going to be presented. These problems appear in a wide range of industries and researchers apply some techniques that are going to be presented.

1.1 Motivations

In times of constant change in the economic scenario, organizations should not only reduce their losses, but also carefully invest their most important asset: time. In this scenario, the research in production planning and scheduling techniques becomes a key element for managing the ever increasing demand for productivity (Pinedo, 2016). Moreover, according to Wang et al. (2008), improving the production scheduling is a factor capable of giving competitive advantages to any industry. However, the problem of finding the sequence of events that minimize the production time (for a limited amount of products) is considered computationally hard, since its complexity is not polynomial (Garey & Johnson, 1979).

Many formalisms have been used to solve the problem of minimizing the production time (makespan) in scheduling problems within the context of Discrete Event Systems (DES) (Cassandras & Lafortune, 2009). We mention Petri Nets (López-Mellado et al., 2005), automata and languages (Alves et al., 2016a; Costa et al., 2018) and timed automata (Abdeddaïm et al., 2006; Su, 2012).

However, the present work is going to focus in automata and languages, in addition

of the Supervisory Control Theory (SCT) (Ramadge & Wonham, 1989), because they can offer a different perspective to solve planning problems. These techniques allow to model problems and its constraints using automata, besides it makes possible to create a minimally restrictive controller. Another significant aspect of using the SCT is the fact that, once the solution is found and some conditions are fulfilled (Vilela & Pena, 2016), it can be implemented in a real system without jeopardizing its operation.

1.2 Scheduling

A decision-making process that is regularly used in all types of manufacturing and services industries (Pinedo, 2016). Usually, a finite number of assets is assigned according to a set of tasks, this is made throughout a permutation on these elements (Pinedo, 2016). According to Baker & Trietsch (2013), the goal in a scheduling problem is to find the order in which a set of tasks must be performed so that a specific performance index is optimized.

In this scenario, finding a production schedule that minimizes the manufacturing time belongs to the non polynomial time class (NP)(Garey & Johnson, 1980). Methods in this class are computationally hard to solve due to its complexity. Often, in this type of problem, the set of all possible solutions is given by $(N!)^M$, where N represents the number of tasks and M the number of machines (Arisha et al., 2001).

One way to solve this kind of problems is to apply dynamic programming techniques to find exactly optimal solutions (Bellman & Dreyfus, 2015). However, the complexity starts to become prohibitive for medium size problems. Despite the inherited difficulty to solve scheduling problems over the years researchers developed several techniques to deal with this problem. For example the Gantt chart, Figure 1.1, illustrates the scheduling of a manufacturing system. In the y-axis each machine is represented, and in the x-axis their production time is represented by the rectangles. This kind of visualization tool allows us to understand what are the limiting factors in the production process.

The scheduling problem is often splitted into two different groups, the deterministic and the stochastic problems (Aytug et al., 2005). In the deterministic approach the operation time of each machine in a production system is known and constant during operation, while in a stochastic approach the time information takes into account a distribution and is represented by a random variable.

In the literature there are several techniques to solve scheduling problems. They usually are divided in two groups: traditional techniques (Analytical and Heuristic Methods) and advanced techniques (Artificial Intelligence, Genetic Algorithms and Hybrid Methods) (Arisha et al., 2001).

The analytical approach considers the problem in its total form, scheduling N jobs on M machines, meaning that the whole problem is considered. Approaches like Explicit Enu-

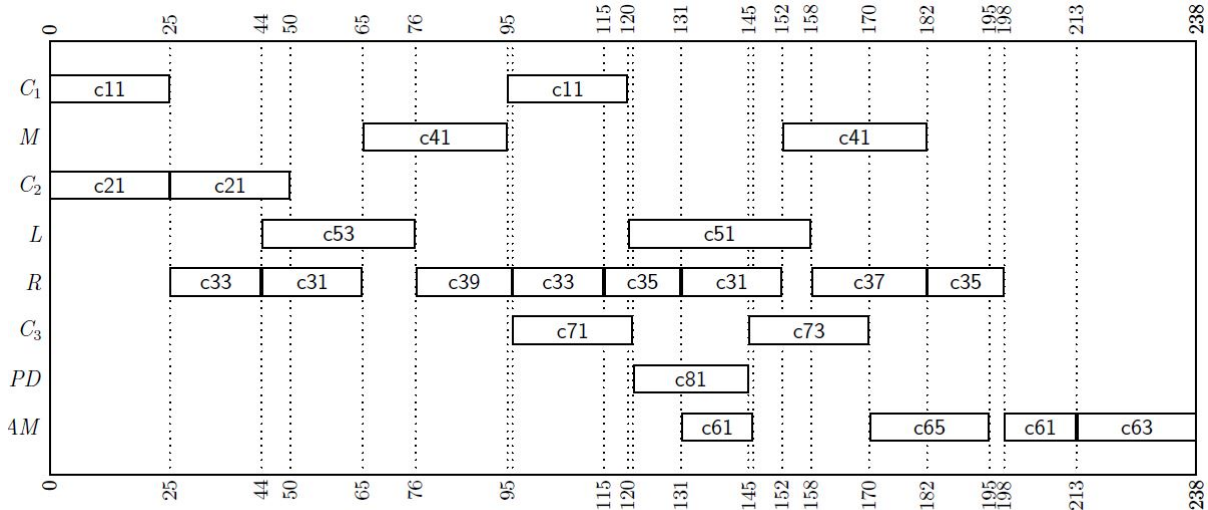


Figure 1.1: Scheduling application in a manufacturing system, (Malik & Pena, 2018).

meration (Baker & Trietsch, 2013; Brucker et al., 1994), Implicit Enumeration (Bellman & Dreyfus, 2015), Branch and Bound Algorithms (Brucker et al., 1994), Linear Programming (Charnes & Cooper, 1962; Panek et al., 2004), Integer Programming (Wagner, 1959; Kobetski & Fabian, 2006) are examples of the analytical method. A more in depth discussion is going to be made in Section 1.3.

The heuristic approach can be represented by a function h that returns an estimate $h(s)$ of the minimum cost $h^*(s)$ of getting from the state s to a goal state and that h is admissible if $0 \leq h(s) \leq h^*(s)$ for every state s (from which it follows that $h(s) = 0$ whenever s is a goal node) (Ghallab et al., 2016). In other words in every decision point the function h helps to decide which option will favour the optimization criteria. Some examples of heuristic methods are: Incremental Scheduling (Alves et al., 2016a), Neighborhood search techniques (Baker & Trietsch, 2013; Pena et al., 2016) and Lagrangian relaxation (Hoitomt et al., 1993).

Apart from the traditional methods, there are advanced techniques that apply artificial intelligence. Arisha et al. (2001) presents, as the most efficient methods to find the optimal solution, the Neural Networks (Masud et al., 2011), Fuzzy logic (Meziane et al., 2000) and genetic algorithms (Lee et al., 1997). In this work, an approach that is closer to the advanced methods is going to be used. In the methodology that will be presented, an evolutionary algorithm is going to be applied for the optimization and the solutions are provided by the Supervisory Control Theory.

1.3 Scheduling and Supervisory Control Theory

In the context of Supervisory Control Theory (SCT) (Ramadge & Wonham, 1989) many extensions have been proposed to solve scheduling problems (Kobetski & Fabian, 2006; Su

et al., 2012; Hill & Lafortune, 2016; Pena et al., 2016).

The works of Panek et al. (2004); Kobetski & Fabian (2006) combined the analytical techniques with SCT. The first proposed a method motivated by reachability algorithms. It presents an algorithm capable of generating lower bounds for the cost-to-go by embedded Linear Programming (LP) and the pruning of the search tree. The resulting algorithm reduces the computational effort when compared to the standard shortest path algorithm. According to Panek et al. (2004), the specification of a given job-shop¹ problem to be efficiently solved, using their technique is less time-consuming and cumbersome to design. Following this analytical approach, Kobetski & Fabian (2006) proposed a method that automatically combines Supervisory Control Theory and Mixed Integer Linear Programming (MILP). This technique aims to obtain the best of both methods. The SCT is used to model the problem and its constraints, then the resulting deterministic finite automaton (supervisor) is mathematically converted in such a way that the MILP can be applied directly. Other authors used a supervisor abstraction, obtained by SCT as the search space. This technique reduces the computational complexity of the scheduling problems. Hill & Lafortune (2016) apply a method, where each supervisor's transition is weighted and used to generate an abstraction, then this is employed to solve the planning problem. In a similar way, Su (2012) used the abstraction to simplify the time-weighted system design by masking out its local transitions so that subsequent time-optimal supervisor synthesis can be carried out in a more computationally efficient manner (Su, 2012). The key element on this approach is the encoding of the time information in max-plus matrices. It allows the aggregation of concurrent transition firings. The author shows that a component wise abstraction strategy is not only possible but also is less computationally demanding than creating an abstraction model for a system with a wide number of components.

A heuristic approach for scheduling problems is presented by Alves et al. (2016a). This work introduces a method called Maximum Parallelism, which has a heuristic that favors the total accumulated parallelism of equipment during production. This technique presents two distinct branches: one focused in a purely logical approach called Logical Maximum Parallelism (LMP) (Alves et al., 2016a) and the other strand combines the parallelism information and the task execution time to optimize the makespan, the Maximum Parallelism with Temporal constraints (MPT) (Alves et al., 2016b). Despite of the quality of the results and their low runtime, they are still local-optimal solutions for the makespan problem. The proposed heuristic has linear complexity in regard of the size of the problem, which makes it capable of producing solutions in a very time-efficient way. The MPT approach is used to compare with this work findings, in Chapter 4.

Malik & Pena (2018) apply the capabilities of model checking to solve the problem

¹ In a job shop with m machines each job has its own predetermined route to follow. A distinction is made between job shops in which each job visits each machine at most once and job shops in which a job may visit each machine more than once, (Pinedo, 2016).

of optimal task scheduling in a flexible manufacturing system (De Queiroz et al., 2005). In this approach, the system was modelled as a DES with a supervisor that implements the supervisory control theory. Then timing constraints were added to the model in the form of extended finite-state machines. In order to compute an optimal timed schedule, Supremica (Akesson et al., 2006), a software for synthesis, model checking simulation of discrete event systems was used.

The Supervisory Control Optimization (SCO) methodology was proposed to address scheduling problems in a manufacturing cell (Pena et al., 2016). In this approach, the Supervisory Control Theory encodes the problem constraints and an optimization technique is used to solve the problem of finding the optimal schedule. Here the controllable events are used to model the starting of operations in a specific machine and a production sequence (or set) is represented by a string of controllable events. In the SCO a specific word-encoding technique is employed, in which a small production set is concatenated several times to create the desired batch of products (Pena et al., 2016). The encoding is based on the fact that there are parts of the sequence (events) that have to remain in a specific order (a piece should always arrive in a conveyor before being painted, for example). Each invariant sub-sequence is uniquely encoded with a string of a repeated symbol from an alphabet. Since new individuals (new production sequences) are created by a permutation of the elements of the sequence such encoding allows to reduce the infeasibility. In the SCO, different optimization algorithms were tested: clonal selection algorithm (CSA) using a randomized method to generate solutions (Silva et al., 2011), CSA with a local search algorithm to improve the best individual of each generation (Oliveira et al., 2013), ant colony system algorithm with a 2-opt local search algorithm (Costa et al., 2012), among others. The optimization method with the best result was the Variable Neighborhood Search (VNS), used in (Pena et al., 2016).

Despite of the relative success, all previous works were limited to manufacture only small batches of product, because the generation of solutions for larger batches was heavily impacted by the occurrence of infeasible solutions causing a high computational cost. To solve this (Costa et al., 2018) problem, Costa et al. (2018) proposed the Supervisory Control Optimization Concatenation (SCO-CONCAT). SCO-CONCAT is based on the idea that to produce large batches it does not help optimizing the large batch itself. Typically, a manufacturing system has a limited number of products that can be manufactured at the same time. Previously optimized sequences (called Minimal Part Set strings, obtained using SCO, (Pena et al., 2016)) are concatenated and permuted to generate a pattern that is repeated a number of times until the size of the batch is obtained.

The SCO-CONCAT represents an important step for the evolution of the SCO and once the process of finding the MPS stops being its bottleneck, due to the generation of infeasible solutions, this methodology is going to be ready to face bigger challenges.

1.4 Objectives

The main objective of this dissertation is to propose a new heuristic method, based on the results of (Vilela & Pena, 2016) to generate solutions (or individuals) to be used with the clonal selection algorithm (De Castro & Von Zuben, 2002) for solving scheduling problems. This heuristic is built over an abstraction of the controllable closed-loop behavior (instead of the behavior itself) to generate the sequences that are going to be used in an optimization method. Once this new method is established, an Evolutionary Algorithm is going to be applied, in order to solve the scheduling problem. Moreover, another heuristic based on the frequency of the events is proposed. The major goal of this work is to find an efficient way to solve scheduling problems. As a result this work extends the Supervisory Control and Optimization (SCO) (Pena et al., 2016), using the abstraction of the closed-loop behavior (Vilela & Pena, 2016) to solve the problem of finding the optimal scheduling in a manufacturing system. As secondary objectives we mention:

- Develop new ways of generating feasible solutions;
- Find sequences that minimizes the makespan, for any number of products;
- Apply the proposed solutions to different scheduling problems in the Discrete Events Systems literature.

1.5 Thesis structure

This theses is organized as follows:

- **Chapter 1** Presents a literature review on scheduling problems and shows this problem from the perspective of Discrete Event Systems.
- **Chapter 2** Introduces preliminaries concepts of Discrete Events Systems and the main definitions for Supervisory Control Theory. Besides, the evolutionary algorithm used is explained and it ends discussing the impact of generating all possible solutions.
- **Chapter 3** Deepens in the methodology proposed introducing some new propositions and two algorithms that implements ideas presented. One is combined with an evolutionary algorithm and the second is a heuristic method to solve the problem.
- **Chapter 4** Tests the proposed ideas in this dissertation, by applying them in the three scheduling problems case studies.
- **Chapter 5** Concludes with the main contributions of this dissertation, a summary of the content discussed and presents some ideas for future developments.

1.6 Final Remarks

In this chapter the motivation and the objectives were introduced. Besides, the literature review on scheduling problems and the main concepts supporting the SCO were presented. This was important, because the present work extends the results of the SCO by applying a technique that assures the creation of feasible solutions only. These modifications in the SCO are expected to improve its time efficiency by reducing the optimization runtime. In the next chapter an in depth exploration of the Discrete Event Systems and of other concepts for this dissertation are going to be presented.

2

Preliminaries

In this chapter, some basic concepts of Discrete Event Systems, Supervisory Control Theory are presented. Also the theoretical results related to the abstraction of the closed-loop behavior, the makespan evaluation, the clonal selection algorithm and the case studies are presented. Lastly the impact of searching for all possible solutions is discussed.

2.1 Discrete Events Systems

Discrete Events Systems (DES) are dynamic systems whose interaction with external entities is made by stimulus called events. These events can be represented as: a push of a button, a sensor activation, the beginning or the ending of a task, a timed interruption call, or inner system's event. One characteristic of the event is its instantaneous nature that is responsible for its discrete aspect. Another aspect of the occurrence of events is the change of the system's configuration. This is known as state change, therefore, it is said that the occurrence of events leads to the transition of the system's states. As described by Cury (2001), Discrete Events Systems are dynamic systems that evolves according to abrupt occurrence of physical events that in general have irregular and unknown time intervals.

Due to the specificity of DES, it is not possible to apply conventional mathematical techniques like differential and difference equations that are very common when dealing with continuous, or discrete time variables, respectively.

2.2 Automata and Language Theory

2.2.1 Languages

Let Σ be a finite and nonempty set of events, referred to as an alphabet. The DES behavior is modeled by a finite group of strings over Σ . Σ^* is the set of all strings on Σ , including the empty string ε . A subset $L \subseteq \Sigma^*$ is called a language. The concatenation of strings $s, u \in \Sigma^*$ is written as su . A string $s \in \Sigma^*$ is called a prefix of $t \in \Sigma^*$, written $s \leq t$, if there exists $u \in \Sigma^*$ such that $su = t$. The prefix-closure \bar{L} of a language $L \subseteq \Sigma^*$ is the set of all prefixes of strings in L , i.e.:

$$\bar{L} = \{s \in \Sigma^* | s \leq t \text{ for some } t \in L\}.$$

2.2.2 Natural Projection

The natural projection $P_i : \Sigma^* \rightarrow \Sigma_i^*$ is an operation that maps the strings of Σ^* into strings of Σ_i , $\Sigma_i^* \subseteq \Sigma^*$, by erasing all the events that are not contained in Σ_i . The natural projection can be defined as:

$$\begin{aligned} P(\varepsilon) &:= \varepsilon \\ P(e) &:= \begin{cases} e & \text{if } e \in \Sigma_i^* \\ \varepsilon & \text{if } e \in \Sigma^* \setminus \Sigma_i^* \end{cases} \\ P(se) &:= P(s)P(e) \text{ for } s \in \Sigma^*, e \in \Sigma_i. \end{aligned}$$

The inverse projection $P_i^{-1} : \Sigma_i^* \rightarrow \Sigma^*$ is defined as $P_i^{-1}(L) = \{s \in \Sigma^* | P_i(s) \in L\}$ and is composed of all traces that, when projected, recover traces from L .

The natural projection can have a known property named Observer Property (OP) presented in Definition 1.

Definition 1. (Wong, 1998) *A language $L \subseteq \Sigma^*$, an alphabet $\Sigma_i \in \Sigma$ and $P : \Sigma^* \rightarrow \Sigma_i^*$ a natural projection that maps the strings in Σ^* into strings of Σ_i^* . If $(\forall a \in \bar{L})(\forall b \in \Sigma_i^*)$, such that $P(a)b \in P(L) \Rightarrow (\exists c \in \Sigma^*)P(ac) = P(a)b$ and $ac \in L$, then the natural projection $P(L)$ has the observer property.*

Figure 2.1 is going to be used to explain the Observer Property (Definition 1) as presented in Alves (2018). The inner ellipse on the left represents the language L , which is defined over the alphabet Σ , while the inner ellipse (to the right) represents the natural projection of this language to the alphabet Σ_i . Then, a string a that is a prefix of a string in L is chosen. After that, a is projected to the alphabet Σ_i and then the projection of a ($P_{\Sigma \rightarrow \Sigma_i}(a)$) is concatenated with a suffix b , composed only by elements of Σ_i . This results

in a string that is part of the projection of L , in another words, $P_{\Sigma \rightarrow \Sigma_i}(a)b \in P_{\Sigma \rightarrow \Sigma_i}(L)$. This operation is represented in Figure 2.1 by the blue arrows. If the projection has the Observer Property, then, for all a and b , such that $P_{\Sigma \rightarrow \Sigma_i}(a)b \in P_{\Sigma \rightarrow \Sigma_i}(L)$, there will be a c , such that $P_{\Sigma \rightarrow \Sigma_i}(ac) = P_{\Sigma \rightarrow \Sigma_i}(a)b$ and $ac \in L$. This is represented by the red arrows. As a consequence, $P_{\Sigma \rightarrow \Sigma_i}(c) = b$.

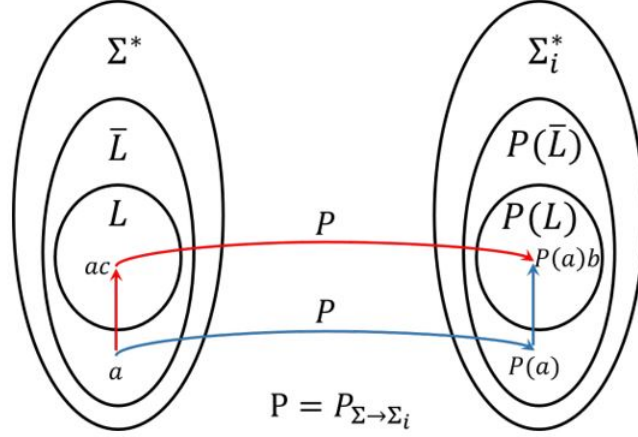


Figure 2.1: Observer's property representation.

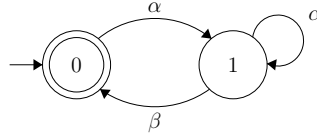
The language $P(L)$ is called abstraction, (Alves, 2018). If this abstraction has the observer property then it is called OP-abstraction. To verify this property the reader should look for the algorithm presented in Pena et al. (2014).

2.2.3 Automata

A finite automaton is a 5-tuple $G = (Q, \Sigma, \delta, q_0, Q_m)$ where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is the transition function, $q_0 \in Q$ is the initial state and $Q_m \subseteq Q$ is the set of marked states. The transition function can be extended to recognize words over Σ^* as $\delta(q, \sigma s) = q'$ if $\delta(q, \sigma) = x$ and $\delta(x, s) = q'$. The generated and marked language are, respectively, $\mathcal{L}(G) = \{s \in \Sigma^* | \delta(q_0, s) = q' \wedge q' \in Q\}$ and $\mathcal{L}_m(G) = \{s \in \Sigma^* | \delta(q_0, s) = q' \wedge q' \in Q_m\}$. The active event function $\Gamma : Q \rightarrow 2^\Sigma$, establishes that for any $q \in Q$, the set of events $\sigma \in \Sigma$ for which $\delta(q, \sigma)$ is defined. An automaton G is deterministic if $\delta(q, \sigma) = q_1$ and $\delta(q, \sigma) = q_2$ implies $q_1 = q_2$. In Figure 2.2, the set of states is $Q = \{0, 1\}$, the alphabet is $\Sigma = \{\alpha, \beta\}$, the marked state is $Q_m = \{0\}$ and the transition function is composed of: $\delta(0, \alpha) = 1$, $\delta(1, \alpha) = 1$ and $\delta(1, \beta) = 0$.

2.3 Supervisory Control Theory

Proposed by Ramadge & Wonham (1989), the Supervisory Control Theory (SCT) allows to synthesize an entity called Supervisor, which is capable of limiting the closed-loop

Figure 2.2: Automaton G

behavior of a system to a specific set of desired actions. This is important because it avoids that the system executes actions that lead to dangerous situations, or physical damage. In this context the supervisor implements the minimally restrictive behavior, meaning that it is permissive. So, it does not enforce a predetermined action to the system, only prevents undesirable situations. The basic structure representing a system under supervisory control can be seen in Figure 2.3. The behavior of the system is represented by automata and languages the supervisor imposes control over in order to prevent blockage and undesirable behaviors.

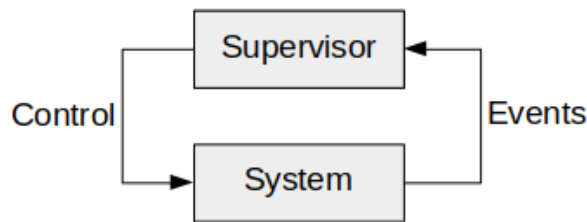


Figure 2.3: Supervisory Control Theory Structure.

The process of synthesizing a supervisor that embodies the entire system's behavior is presented in the SCT as the Monolithic Control. Depending on the system in question computing a supervisor to ensure the minimum makespan can be a NP-hard problem (Su & Woeginger, 2011).

The system to be controlled is called *plant*, the controller agent is called *supervisor* and the control problem is to find a supervisor that enforces the specifications in a minimally restrictive way.

The plant is modeled by an automaton $G = (Q, \Sigma, \delta, q_0, Q_m)$ and $\Sigma = \Sigma_c \cup \Sigma_{uc}$ where Σ_c is the set of controllable events that can be disabled by an external agent and Σ_{uc} is the set of uncontrollable events that cannot be disabled by an external agent. The plant represents the logical model of the DES and the supervisor's S role is to regulate the plant actions to meet a desired behavior K disabling only controllable events.

Let E be an automaton that represents the specification imposed over G . then $K = \mathcal{L}_m(G||E) \subseteq \mathcal{L}_m(G)$ is controllable with respect to G if $\overline{K}\Sigma_{uc} \cap \mathcal{L}(G) \subseteq \overline{K}$. There exists a non-blocking supervisor V for G such that $\mathcal{L}_m(V/G) = K$ if, and only if, K is controllable with respect to G . If K does not satisfy the condition, then the *supremal controllable and non-blocking sub-language* $\text{sup}\mathcal{C}(K, G)$ can be synthesized. It represents the least restrictive

non-blocking supervisor. For G and K , a monolithic supervisor automaton S , can be computed to represent $\text{sup}\mathcal{C}(K, G)$ such that $\mathcal{L}_m(S) = \text{sup}\mathcal{C}(K, G) \subseteq K$. The generated and marked language of a plant G under the action of a supervisor S are, respectively, $\mathcal{L}(S/G)$ and $\mathcal{L}_m(S/G) \subseteq \mathcal{L}(S/G)$. A supervisor S is called non-blocking when $\overline{\mathcal{L}_m(S/G)} = \mathcal{L}(S/G)$.

2.4 Supervisor Abstraction

In Vilela & Pena (2016), the conditions under which the natural projection of the closed-loop behavior to the controllable events, $P : \Sigma^* \rightarrow \Sigma_c^*$, keeps a good property of the original supervisor are presented. This is a theoretical result that allows to search for optimal solutions on a smaller universe, $P(S)$ instead of S . The above mentioned “good property” is, once a trace $s_{opt} \in P(S)$ is picked, that trace, when lifted to the original alphabet, can be executed to the end. This result is presented in Theorem 1.

Theorem 1. (Vilela & Pena, 2016) *Let G be a system, S be the supremal controllable sub-language contained in a desired language K and $P : \Sigma^* \rightarrow \Sigma_c^*$ the natural projection. For all sequences $s_{opt} \in \Sigma_c^*$ and $A = P^{-1}(s_{opt}) \cap S$, if P has the observer property then A is controllable with respect to $\mathcal{L}(G)$.*

Theorem 1 establishes that under certain conditions over the projection, the lifted language A (composed of all traces of S that project to s_{opt}) is controllable. The verification of controllability guarantees that all the interleavings that may arise in the lifted trace, if possible in the plant, are possible when implementing the plan in the system. In other words, to implement the controllable s_{opt} trace in the system, it is not necessary to disable uncontrollable events.

To find the set of all legal traces of controllable events that produce k products, we have to calculate $E_q || P(S) \subseteq P(S)$. We define E_q as in Definition 2.

Definition 2. *An automaton E_q is a quantity specification, for the production of a batch of k products, if it has the following characteristics:*

1. $\Sigma_{E_q} = \{\sigma\}$, where σ is the last controllable event of the event sequence that leads to the production of 1 product;
2. $L_m(E_q) = v \subseteq \Sigma_{E_q}^*$, where $|v| = k$ and k is the number of products in a batch.

Example 1. *The simplified manufacturing system (SMS) is an extension of the small factory problem (Wonham, 2015). It consists of four machines (M_1, M_2, M_3, M_4) and three unitary buffers (B_1, B_2, B_3), Figure 2.4. The controllable events $\Sigma_c = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ mark the starting of a process and the uncontrollable events $\Sigma_{uc} = \{\beta_1, \beta_2, \beta_3, \beta_4\}$ represent the process ending. This way, a machine (M_i) starts its operation when a controllable event α_i*

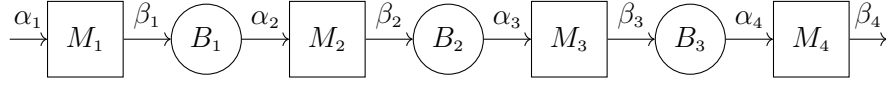


Figure 2.4: Simplified Manufacturing System

occurs and the process finalization is noticed by the correspondent uncontrollable event β_i . In this system a finished product must pass by all four machines.

The machines were modeled as two states automata G_k , which in the initial state I they are in Idle mode and the state W represents their operation mode. The control problem is to avoid underflow and overflow of the unitary buffers, in this case the buffer B_k is considered empty when it is in state 0 and full in state 1. The machines and Buffer specifications can be seen in Figure 2.5.

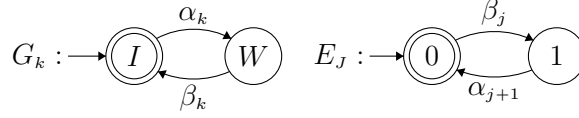


Figure 2.5: G_k is the machine model ($k = 1, \dots, 4$) and E_j is the specification ($j = 1, 2, 3$).

Once the plants (G_1, G_2, G_3, G_4) and the specifications (E_1, E_2, E_3) are defined they are used to compute a monolithic supervisor S and then a natural projection (P) on the controllable events is applied on S . Both were made by using UltraDES (Alves et al., 2017) and $P(S)$ is shown in the Figure 2.6.

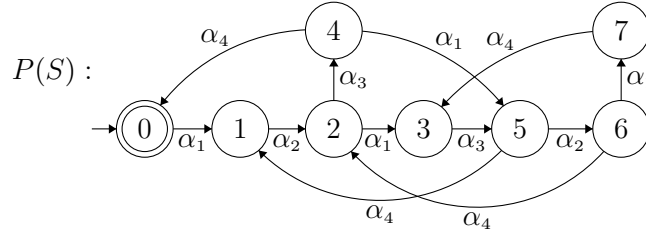


Figure 2.6: Natural projection of the supervisor S to the set of controllable events, in the SMS.

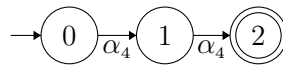


Figure 2.7: Quantity specification (E_q) for a batch of size 2, Definition 2.

To generate all the strings of controllable events that lead to the production of two products in the SMS, first the parallel composition between the natural projection ($P(S)$) and a quantity specification (E_q) is made ($E_q || P(S)$). The product specification (E_q) is presented in Definition 2. For a batch of size two, the quantity specification is shown in Figure 2.7.

The resulting automaton P_{E_q} (Figure 3.2) represents all possible sequences of controllable events, that once executed, lead to the manufacturing of two products in the SMS.

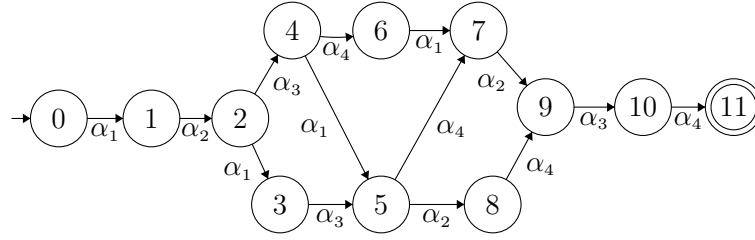


Figure 2.8: Automaton P_{E_q} , represents the parallel composition between the P and E_q .

Consider an optimal string of controllable events $s_{opt} = \alpha_1\alpha_2\alpha_1\alpha_3\alpha_2\alpha_4\alpha_3\alpha_4$, in G . According to Theorem 1, the lifted language A is controllable, thus adding uncontrollable events does not affect the string s_{opt} feasibility in regard to the plant. The automaton representing A can be found by applying: $A = P^{-1}(s_{opt}) \cap S$, Figure 2.9.

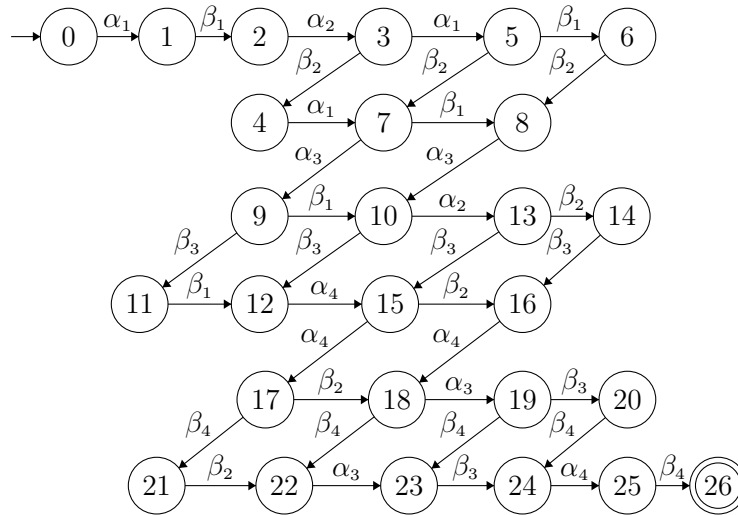


Figure 2.9: The automaton that implements A

The automaton in Figure 2.9 implements the lifted language A . It has all possible traces of S that project to s_{opt} . The result is valid for any s_{opt} and this sequence is going to be picked by the optimization algorithm.

2.5 Makespan evaluation

One key parameter to understand a manufacturing system's performance is the production time or makespan. This parameter represents the total time for a batch of products to be processed. In the context of Discrete Events Systems, more specifically in automata and languages, a batch of products can be represented as a sequence of events that are related to the machines in the plant.

To calculate the makespan, each machine processing time is known beforehand and it is represented by the time interval between the controllable event (starting the machine's operation) and the uncontrollable event (ending it). For calculation purpose we assume that the controllable event time is zero and the uncontrollable event happens a constant time value after the controllable.

In order to evaluate the makespan of this sequence, or string of events, in this work the temporal function (f_T) introduced by Alves et al. (2016b) is going to be applied. To understand how this function works, two definitions are going to be presented.

Definition 3. (Alves et al., 2016b) Consider $S = (Q, \Sigma, \delta, q_0, Q_m)$ a supervisor, the temporal function $f'_T : \Sigma^* \times \Sigma \rightarrow \mathbb{R}^*$, of S is defined as:

$$f'_T(s, \sigma) := \begin{cases} t & \text{if } \delta(s, \sigma) \text{ exists} \\ \infty & \text{otherwise} \end{cases}$$

For an event $\sigma \in \Sigma$ and a sequence $s \in \mathcal{L}(S/G)$, t is the time until σ happens, given that a sequence s has been executed. However, this is just not enough to evaluate the temporal information of a sequence of events. Thus, the temporal function must be extended:

Definition 4. (Alves et al., 2016b) Let $S = (Q, \Sigma, \delta, q_0, Q_m)$ be a supervisor, the expanded temporal function $f_T : \Sigma^* \rightarrow \mathbb{R}^*$, of S is defined as:

$$\begin{cases} f_T(\varepsilon) = 0 \\ f_T(s\sigma) = f_T(s) + f'_T(s, \sigma) \end{cases}$$

Definition 4 shows how to compute the makespan for a sequence.

2.6 Clonal Selection Algorithm

The Clonal Selection Algorithm (CSA) (De Castro & Von Zuben, 2002) was inspired in the principles of the immunological system of mammals, where the improvement of the solutions works as a metaphor for the immune system of the living organisms. In this scenario, each solution is treated as an individual and a population is composed by N individuals. First, several replicas of the current antibodies are made with the most useful antibodies receiving more replicas. Then, these replicas are randomly mutated. This process generates solutions that are similar to the original but they differ to some extent.

The pseudo-code for the CSA can be seen in Algorithm 1. The inputs for this algorithm are the number of individuals (N), the number of generations ($nGen$), the mutation rate (λ) and the number of products (nP). The outputs are the best solutions (individuals).

Algorithm 1: Clonal Selection Algorithm

```

Input  :  $N, nGen, \lambda, nP$ 
Output : Solution
1 Initial population of  $2N$  individuals ( $nP$ )
2  $Population \leftarrow Makespan.Evaluation(Population)$ 
3  $BestIndividuals \leftarrow Population.Sortedby(Makespan)$ 
4 for  $j \leftarrow 0$  to  $N$  do
5   |  $SelectedInd.add( BestIndividuals[j] )$ 
6 end
7 while Stop criterion not achieved do
8   for each  $Ind \leftarrow in Population$  do
9     |  $TestedIndividual.add(Ind)$ 
10    for  $k \leftarrow 0$  to  $Eval(Ind, noClones)$  do
11      |  $(Q_{DS_{new}}, S_{new}) \leftarrow Seq.Generator(Ind, \lambda)$ 
12      |  $newInd \leftarrow Makespan.Evaluation(s_{new}, Q_{DS_{new}})$ 
13      |  $TestedIndividual.Add(newInd)$ 
14    end
15     $NextGen.add(TestedIndividual.Min())$ 
16     $TestedIndividual.Clear()$ 
17  end
18   $Population \leftarrow NextGen$ 
19   $CheckMakespan(Population, nGen)$ 
20 end

```

Initially, the population has $3N$ random individuals (uniform distribution), this allows the exploration of the search space. Then, this population has its makespan evaluated (as described in Section 2.5) and the N best individuals (the ones with the smaller makespan) are selected, lines 1 to 5.

The number of clones ($nClones$) produced for each individual is calculated in terms of the function that guarantees that the best individuals generates $(\beta)nClones$ clones, while the worst ones generates $(1 - \beta)nClones$ clones. This parameter β may assume values between 0 and 1, and the parameter $nClones$ represents the maximum number of clones. The fitness function used is based on the ranking of the solutions which in this work is the makespan. Thus, the solutions with the smallest makespan are selected for the next generation. This version of the CSA uses as stop criteria: the maximum number of generations ($MaxGen$), or the number of generations without improvements ($nGen$).

Regarding the mutation, the clones are mutated and the parents are left untouched. Besides, its intensity parameter λ varies with a fitness function such that an individual with higher fitness (smaller makespan), suffers mutations that are less intense while individuals with lower fitness (meaning higher production time) have a more aggressive mutation.

2.7 The case studies

The proposed techniques were applied in two problems in the DES literature, the Cluster Tool (Uzsoy et al., 1994) and the Flexible Manufacturing System (De Queiroz et al., 2005). Each problem presented an unique optimization challenge involving the layout of the system and different types of products to be made. To assure that the abstraction proposed by Vilela & Pena (2016) could be used in the testing problems, they were checked for the presence of the observer property in the closed-loop projection. This was accomplished by using the method implemented in TCT (Feng & Wonham, 2006).

2.7.1 The Cluster Tool

The semiconductors industry has one of the highest specialized manufacturing facilities known (Lee & Ni, 2012), due to the critical nature of its processes. In this scenario, one of the key elements of the wafer manufacturing is the cluster tool (CT), Figure 2.10. This machine is responsible for the fabrication of the silicon wafer and it is capable to manage its most complex stages. As a result, there are studies related to the cluster tool scheduling problem (CTSP) like Uzsoy et al. (1994), Jula & Leachman (2010) and Shin et al. (2001). The cluster tool has four basic components:

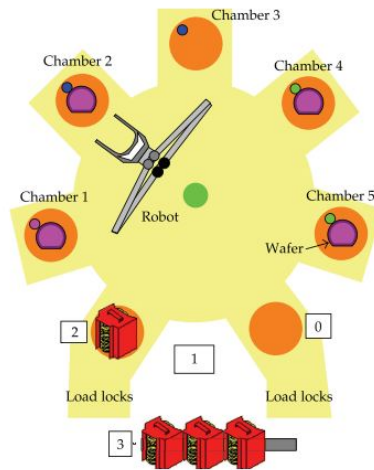


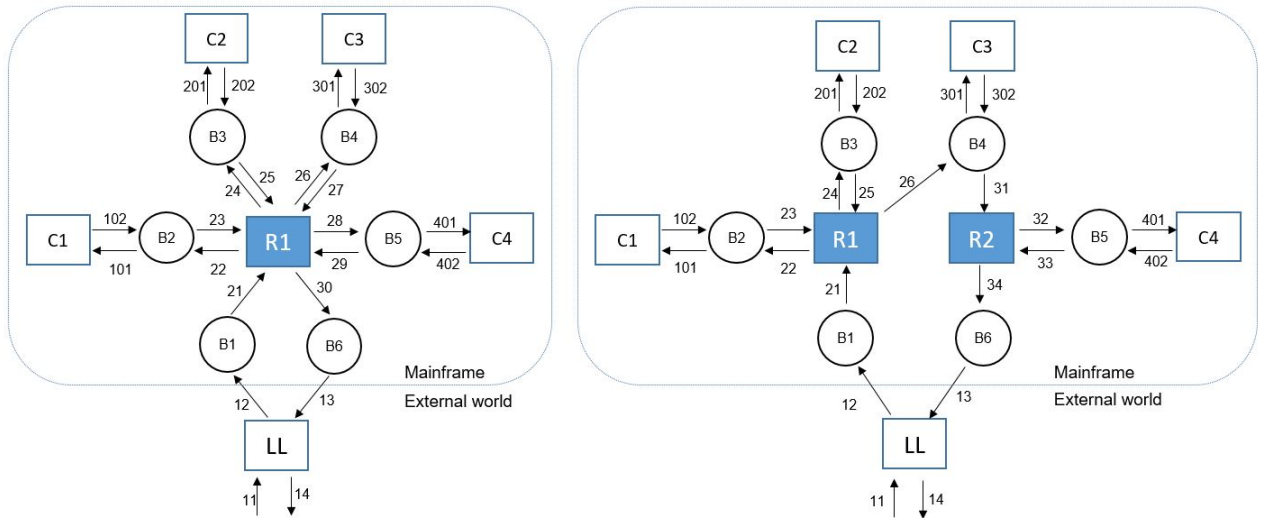
Figure 2.10: Standard Cluster Tool layout (Lee & Ni, 2012).

- Load locks (LL): the wafer's storage unit is responsible for protecting each wafer in the beginning of the process and at the end;
- Chambers (CH): the place where different stages of the wafer fabrication processes happens;
- Robot (RB): the agent behind the manipulation of the silicon wafer through the chambers;

- Mainframe: the place where all the components above are stored.

In this dissertation, two types of cluster architectures are evaluated. These layouts differ from each other by the number of robots in the mainframe (Figures 2.11), usually the cluster tool has one or two robots and in all architectures of this system, the load lock works as an interface with the external world. Another characteristic is the number of process modules for each layout that can be 4, 5 or 6.

The first layout to be introduced is the radial cluster tool with one robot inside the mainframe. This cluster tool architecture is shown in Figure 2.11.a with its main components and how they interact with each other. The arrows indicate the flow of the actions and the numbers attached to them represent the event. The controllable events are represented with odd numbers, while the uncontrollable events are represented with even numbers. As can be seen in Figure 2.11.a, there are three main components in the cluster tool, one load lock LL , one robot R_1 (in charge of the actions in the mainframe), six buffers (B_1, B_2, B_3, B_4, B_5 and B_6) and four process chambers (C_1, C_2, C_3 and C_4).



(a) Radial CT with one Robot

(b) Radial CT with two Robot

Figure 2.11: Cluster Tool layout and the number of robots in the mainframe.

Example 2. Consider the production of one wafer unit in the cluster tool, Figure 2.11.a. In a sequential manufacturing process each process chamber is visited only once, starting from C_1 up to C_4 . The production sequence, composed only by controllable events, would be:

- $Product = 11 - 21 - 101 - 23 - 201 - 25 - 301 - 27 - 401 - 29 - 13$.

To guarantee that the operation of this system does not reach a deadlock or a hazardous stage for the plant, the system first is modeled as an automaton G_P , which is the

parallel composition of all machines automata (Figure 2.12) and the supremal controllable sublanguage S is obtained. Then, the abstraction is calculated.

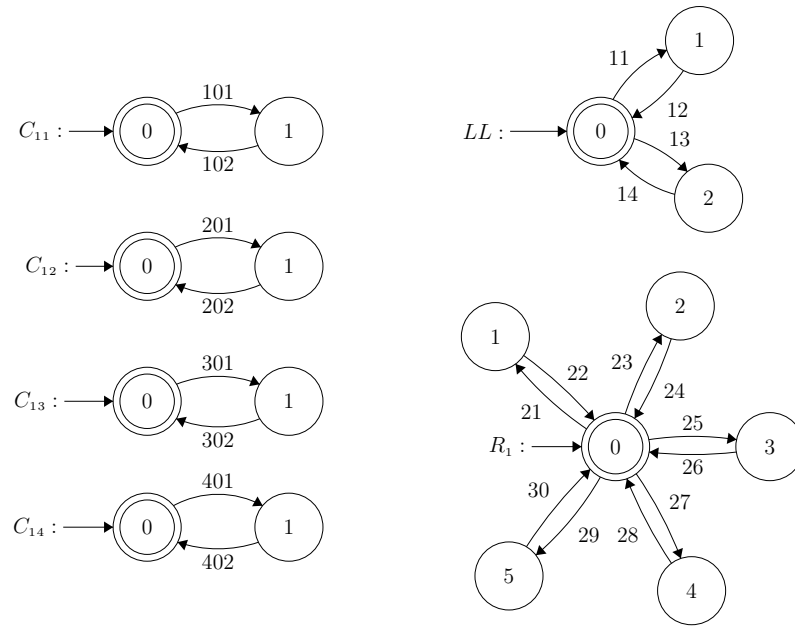


Figure 2.12: One Robot Cluster Tool - all machines automaton model representation.

The control problem is to guarantee no overflow and underflow in the buffers and also that the closed-loop behavior is nonblocking. The specifications (E_1, E_2, E_3, E_4, E_5 and E_6) were used to compute the supremal controllable sublanguage, Figure 2.13.

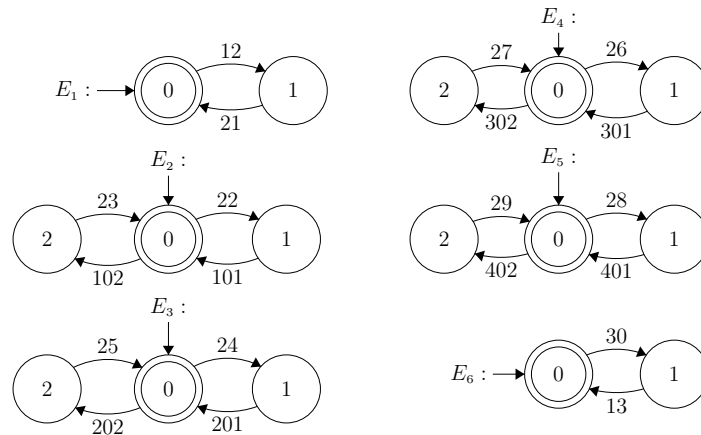


Figure 2.13: One Robot Cluster Tool - Buffers specification.

The second architecture studied was the cluster tool with two robots working in the mainframe, Figure 2.11.b.

Example 3. For the cluster tool with two robots in the mainframe the tasks can be splitted in two parts: those carried by $Robot_1$ and those carried by $Robot_2$.

- $Robot_1 = R1 = 21 - 23 - 25$;
- $Robot_2 = R2 = 31 - 33 - 13$.

As can be seen in Figure 2.11.b, there are four main components in the cluster tool: the load lock (LL) manages the interaction with the external world, two robots ($R1$ and $R2$) in charge of the actions in the mainframe, six buffers (B_1, B_2, B_3, B_4, B_5 and B_6) and four processes chambers (C_1, C_2, C_3 and C_4).

2.7.2 The Flexible Manufacturing System

The flexible manufacturing system (FMS) (De Queiroz et al., 2005) produces two types of products (A and B). There are eight machines in the FMS, three Conveyors (C_1, C_2 and C_3), a Mill, a Lathe, a Robot, a Painting Device (PD) and an Assembly Machine (AM). These devices are connected through unitary buffers (B_1 to B_8). In the FMS the controllable events are represented as odd numbers and the uncontrollable events as even numbers, as shown in Figure 2.14.

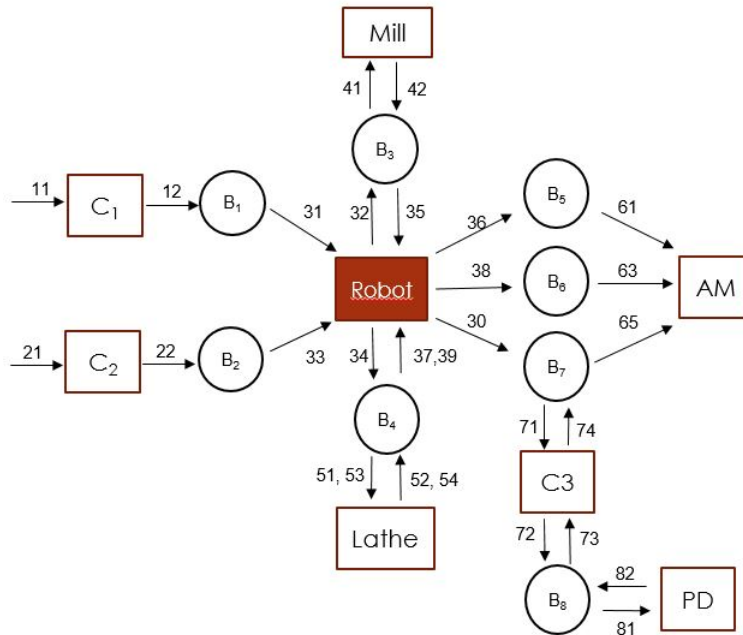


Figure 2.14: Flexible Manufacturing System.

The manufacturing of each product (A and B) can be expressed by a sequence of controllable events that needs to be executed in a specific order. The Product A is a block with a conical pin on top and product B is a block with a cylindrical painted pin. To manufacture one unit of product A, it is necessary to combine two parts: one $Base$ and one Pin_A . In a similar way, a product B is composed by the pair $Base$ and Pin_B . The sequence of controllable events representing these three elements are:

- $Base$: 11 – 31 – 41 – 35 – 61;
- Pin_A : 21 – 33 – 51 – 37 – 63;

- $Pin_B : 21 - 33 - 53 - 39 - 71 - 81 - 73 - 65.$

These sequences can be interleaved with each other in order to generate batches with different makespans. For each controllable event there is a correspondent uncontrollable event and the time between their occurrences represents the amount of time needed for a task to be completed. In the case of FMS there is an exception, the controllable event 61 that belongs to the Assembly Machine (AM). It does not have a correspondent uncontrollable event and after a minimal interval of 15 time units, it is followed by either event 63, or 65.

Example 4. *There are many ways of making one unit of product (A or B) in the FMS. Choose a simple manufacturing sequence represented by a sequential process, where first a Base is made and then Pin. These sequences of controllable events are as follows;*

- $Product\ A = Base + Pin_A = 11 - 31 - 41 - 35 - 61 - 21 - 33 - 51 - 37 - 63;$
- $Product\ B = Base + Pin_B = 11 - 31 - 41 - 35 - 61 - 21 - 33 - 53 - 39 - 71 - 81 - 73 - 65.$

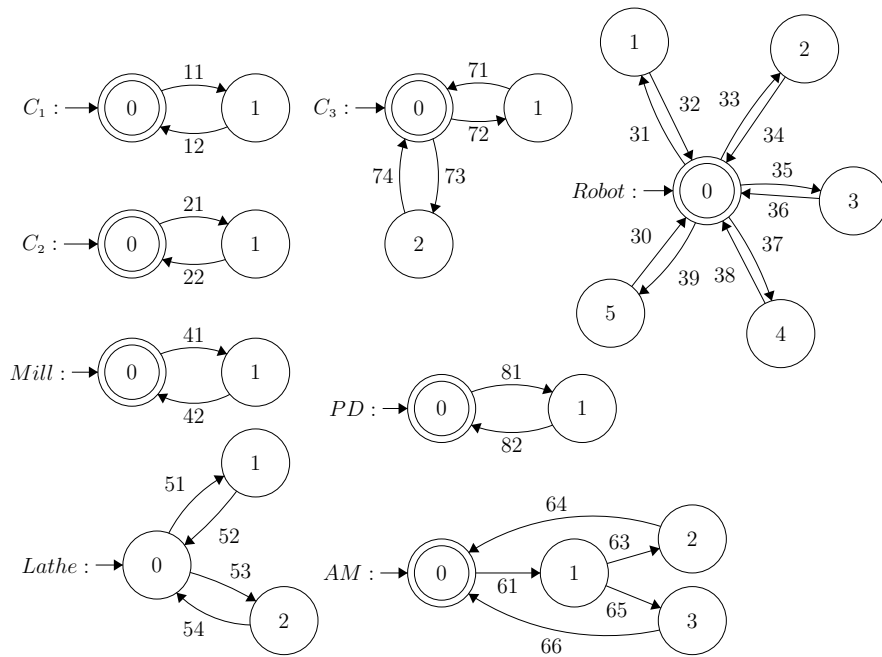


Figure 2.15: Flexible Manufacturing System automata model.

In Figure 2.15 all eight machines of the FMS are modeled as an automaton. The specifications used to compute the supremal controllable sublanguage are in Figure 2.16. The control problem is to guarantee no overflow and nor underflow in the buffers and also guarantee that the closed-loop behavior is nonblocking. To give an idea of the scale of this problem, the monolithic supervisor of the FMS used as an intermediate step to find the abstraction has 45,504 states and 200,124 transitions.

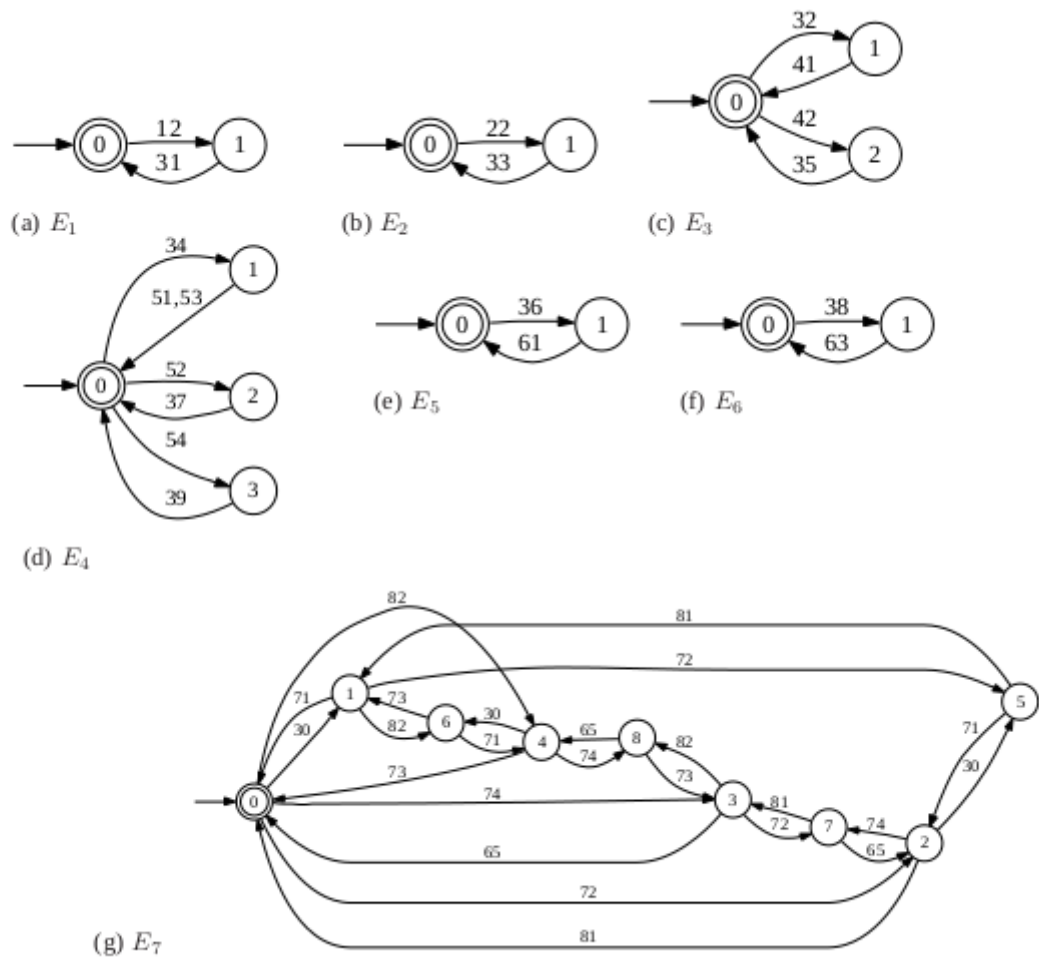


Figure 2.16: Flexible Manufacturing System specifications.

2.7.3 The Search For the Optimal Sequences

In this work an evolutionary algorithm (CSA) was employed in conjunction with a theoretic result that guarantees that under certain conditions an abstraction of the closed-loop behavior can be used to find the shortest makespan in a scheduling problem. Despite the promising aspect of this optimization technique, it is not possible to affirm that the solutions found are globally optimal because the technique is not an exact method.

Although scheduling problems belong to the NP class, which makes them hard to solve, it is known that the abstraction reduces the total search space. Therefore, the optimization becomes less computational costly. This way, based on the physical similarities between an abstraction automaton and an acyclic orientated graph, one can argue in favor of applying a shortest path algorithm like Dijkstra’s algorithm (Dijkstra, 1959), A^* (Hart et al., 1968), or Dynamic Programming (Bellman, 1958). However, finding the optimal path that leads to the smallest makespan is not a trivial problem. That is due to the fact that in the abstraction there are three characteristics that makes very time consuming and cumbersome to apply this methods, which are:

1. The transitions do not represent a cost value, but the kind of event (controllable, or uncontrollable);
2. If the time information is added, the weights become dynamic (varying in regard of the chosen path);
3. All the paths must have the same size, otherwise they cannot be applied in a real system.

Another way to search for the optimal solution within the abstraction is to evaluate all possible solutions for the shortest makespan. To accomplish this task a graph, or tree search algorithm, like the Breadth First Search (BFS) (Moore, 1959), should be applied on the abstracted automaton. The BFS begins its search in an arbitrary node of a graph and then explores all of the neighbor nodes at the same depth. Once it finishes the exploration, then it moves to the nodes at the next depth level. In this work, the implemented version of the BFS should begin in the initial state of the abstraction of the closed-loop behavior and keep the search until it reaches a marked state. Then, all possible solutions for a given batch of products can be checked and the shortest makespan found is the global optimum.

However, before starting to construct the production sequences from the abstraction of the closed-loop behavior, the number of possible solutions, for each one of the study cases (Cluster Tool and FSM) must be checked. Due to the explosion of states, it is important to know the size of the problem, in order to have enough computational resources to solve it. The effort necessary to do so can be measured and evaluated by applying the sequence counting algorithm from Alves (2018). In Table 2.1 the number of sequences found in the closed-loop behavior (monolithic supervisor) and their abstractions for four DES literature plants are presented:

- Simplified Manufacturing System (SMS) an extension of the *small factory problem* (Wonham, 2015),
- Industrial Transfer Line (ITL) (De Queiroz & Cury, 2000),
- Cluster Tool, with radial layout and one robot in the mainframe and four process modules (chambers) Section 2.7.1,
- Flexible Manufacturing System (FMS) Section 2.7.2.

For each case, the number of products manufactured were varied with the goal of understanding its impact on the number of possible sequences to be found. All the initial tests were executed on a notebook with an Intel Core I7-3537U 2.0 GHz processor with 8.0 GB of RAM. In addition, the UltraDES library from Alves et al. (2017) was used to compute the supervisor and the abstraction.

Table 2.1: Scheduling problems and the number of solutions.

Scheduling Problem	Num. of Machines	Num. of Products	Num. of Sequences in the Supervisor	Num. of Sequences in the Abstraction
SMS	5	4	444, 855, 492, 680	6, 392
		5	8.5×10^{15}	141, 696
		6	1.6×10^{20}	3, 140, 702
ITL	6	4	1.6×10^{19}	6, 331, 920
		5	1.7×10^{25}	686, 056, 800
		6	1.8×10^{31}	74, 333, 515, 200
Cluster Tool	6	4	1.3×10^{39}	1.0×10^{17}
		5	1.3×10^{52}	8.3×10^{22}
		6	1.2×10^{65}	6.5×10^{28}
FMS	8	(1,1)	9.7×10^{16}	466, 711, 684
		(2,2)	3.9×10^{42}	3.8×10^{21}
		(3,3)	$\gg 10^{65}$	4.1×10^{34}

The results shown in Table 2.1 make clear the magnitude of the presented problems. Even though the abstraction method reduces the search space, the number of solutions is still very high. The magnitude order for the FMS to produce two products ($A = 2$ and $B = 2$) is 10^{21} different solutions. Another aspect that should be addressed is the optimization time, because it can grow exponentially with the size of the problem. In Table 2.2, two scheduling problems are used to exemplify this issue. In the SMS, the manufacturing of 6 products has 3×10^6 feasible sequences (solutions). To generate and evaluate all of them took on average 12 minutes.

Table 2.2: Scheduling problems and the total optimization time.

Scheduling Problem	Number of Machines	Number of Products	Num. of Sequences in Abstraction	Optimization time (min.)	
SMS	5	6	3.1×10^6	12	× 148
FMS	8	(1,1)	4.6×10^8	1, 776*	

Based on SMS optimization time and the fact that the FMS is around 148 times bigger than SMS (in terms of number of sequences) the total optimization time for the FMS would take 1, 776 minutes (29.6 hours). And this is only for a small instance (manufacturing two products), even if the problem scaled linearly, it would be impracticable to solve it for larger batches due to time and computational constraints.

2.8 Final Remarks

This chapter presented the main concepts related to Discrete Events Systems. It covered the most important concepts used to understand the DES modeling of the scheduling problems and the supporting ideas. Besides, the optimization method and its main features were explained. At the end, based on the Section 2.7.3 an alternative to the optimization

method was introduced. In the next chapter it is going to be presented how the ideas introduced can be combined with the optimization methods.

3

Methodology

The goal of this work is to extend the results of the SCO and also to solve its main drawback, the generation of infeasible solutions. This issue deeply impacts the SCO efficiency, by increasing the optimization time. To solve this problem, a methodology that applies the theoretical results of Vilela & Pena (2016), that allows the use of the abstraction of the closed-loop behavior to generate strings for the optimization, is presented. These strings are used as individuals for the chosen optimization method (CSA), which has among the inputs, the set of individuals (population) and a mutation operator. Therefore, an algorithm that uses the abstraction for creating individuals and also controlling mutation was developed.

In this chapter two algorithms are going to be presented, the first is used to create the population and to generate mutations for the CSA algorithm. The second is a heuristic method that can be used as an alternative to the the CSA. However, before presenting some definitions, the optimization problem is going to be stated.

3.1 Problem Statement

The optimization problem is to minimize the total production time (makespan) required for a batch of products in a manufacturing system. Here, the problem is formally presented as stated in Pena et al. (2016):

- Let Σ be the set of events associated to a plant (commands and responses), which is divided into controllable events Σ_c and uncontrollable events Σ_{uc} ;

- Let P be the set of instances of events from Σ which are associated to the production of the complete batch, and let $P^c = P \cap \Sigma_c$ be the subset of controllable events of P which are associated to the production of the complete batch;
- Let P_k be an ordered set of the events of P , representing a production schedule candidate, $|P| = |P_k|$, and let P_k^c denote the ordered subset of controllable events in P_k ;
- Let $N = \{P_1^c, P_2^c, \dots, P_{|N|}^c\}$ be the set of all permutations of the elements of P_c , which result in feasible sequences, i.e., all ordered sets composed with the elements of P_c ;
- Let T_k denote the time elapsed while the plant processes the sequence P_k . If the sequence is unfeasible, $T_k = \infty$.

The scheduling optimization problem can be stated as:

$$P^* = \underset{k \in \{1, \dots, |N|\}}{\operatorname{argmin}} T_k.$$

3.2 Definitions

Before starting the optimization, two new properties regarding the transitions in and out of a given state are presented. They will be used in conjunction with two notable characteristics of an automaton state, the initial and marked states.

Definition 5. Let $G = (Q, \Sigma, \delta, q_0, Q_m)$ be a deterministic automaton. A state q is called a divergent state (DS) if $|\Gamma(q)| > 1$. The set $Q_D = \{q \in Q \mid \Gamma(q) > 1\}$ is the set of all DS states.

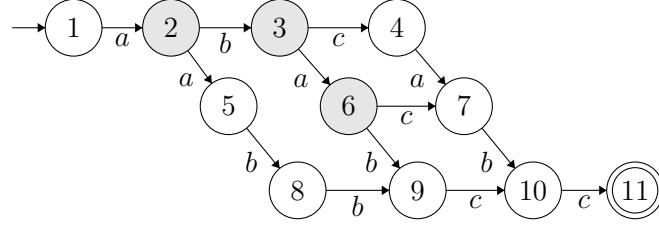
In words, if the number of *active events* is greater than one, this state is called a divergent state and they can all be grouped in Q_D .

Definition 6. Consider $s \in \mathcal{L}_m(G)$, then $Q_{DS}(s) = \{q \in Q_D \mid \delta(q_0, s') = q, s' \in \bar{s}\}$ is the set of divergent states in relation to s .

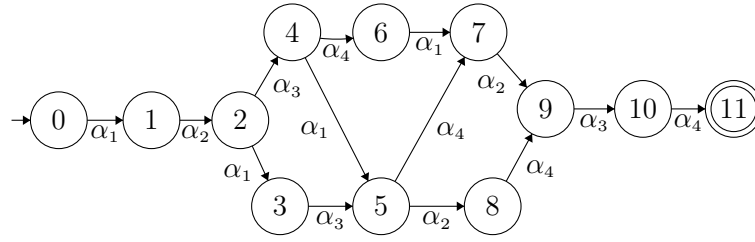
Definition 6 establishes the relationship between a string s and the DS states along its path.

Example 5. To clarify Definitions 5 and 6, let G be the automaton, which has only controllable events, Figure 3.1. The set of all DS is $Q_D = \{2, 3, 6\}$. Consider a sequence of events $s = abcabc$ in G . The set of all DS states in the path of s is $Q_{DS}(s) = \{2, 3\}$.

To find a sequence of events that minimizes the makespan we can use the abstraction (Theorem 1), Definitions 5 and 6. The aim is to create an initial sequence of controllable events in which making a small change in the events will not turn it infeasible, then evaluate the impact of such change in the overall makespan of the sequence.

Figure 3.1: Automaton G and its DS states filled in gray

Example 6. Consider a string of controllable events $s = \alpha_1\alpha_2\alpha_3\alpha_4\alpha_1\alpha_2\alpha_3\alpha_4$, in G , where the set of all DS is $Q_D = \{2, 4, 5\}$. In this scenario, $Q_{DS}(s) = \{2, 4\}$, Figure 3.2.

Figure 3.2: Automaton G .

In order to produce a new string (s_{new}), we change a subset of $Q_{DS}(s)$ by picking, from a divergent state, another path. Typically, we keep the initial part of the sequence (first states in the path). The idea is to find a slightly different sequence. So, we can pick state $4 \in Q_{DS}(s)$ and instead of continuing the sequence with event α_4 , we pick α_1 and complete the sequence until the marked state. The new sequence starts with $\alpha_1\alpha_2\alpha_3\alpha_1$ reaching a new divergent state. At this point, another choice has to be made, between continuing with α_4 or α_2 . Consider that α_4 is picked, then the complete sequence is $s_{new} = \alpha_1\alpha_2\alpha_3\alpha_1\alpha_4\alpha_2\alpha_3\alpha_4$ and the new set $Q_{DS}(s_{new}) = \{2, 4, 5\}$. This sequence is feasible when lifted to the complete alphabet because $P(S)$ has the observer property (Theorem 1). An algorithm is provided to make such changes in the individual, in order to find other individuals.

In the next section, the process of creating individuals for the clonal selection algorithm (discussed in Section 2.6) is going to be presented.

3.3 Individual Generation algorithm

This algorithm uses the closed-loop behavior of the system to encode a sequence of events (an individual) $s \in \mathcal{L}_m(G)$ and through a series of modifications (mutations) in the original sequence s a new individual for the optimization is generated.

Initially, an individual (Ind) is represented by a 3-tuple ($Ind = (s, Q_{DS}, Msp)$), where s is a sequence of controllable events, Q_{DS} is the set of divergent states in relation to s and Msp is the makespan.

In order to generate a new individual from *Ind* one parameter should be defined. This parameter is an analogy for the mutation and controls how much of the sequence is going to be modified. Its implementation in the algorithm is given by the percentage of preserved states or mutation rate (λ) of $Q_{DS}(s)$.

Besides, the active event function (Γ), the set of all states Q and the set of marked states (Q_m) are the inputs to the algorithm. The output solution is a 2-tuple ($Q_{DS_{new}}, s_{new}$) where $Q_{new} \subseteq Q$ is the set of divergent states and s_{new} is the new complete sequence of controllable events.

Algorithm 2: Individual Generator

Input : $P_{Eq}(Q, \Sigma, \delta, q_0, Q_m), \Gamma, Q_{DS}, s, \lambda$
Output : $Q_{DS_{new}}, s_{new}, Msp$

- 1 $Q_{DS_{new}} \leftarrow Select_{DS}(Q_{DS}, \lambda)$
- 2 $q \leftarrow Last(Q_{DS_{new}})$
- 3 $s_{new} \leftarrow NewSequence(s, q)$
- 4 **while** $q \notin Q_m$ **do**
- 5 **if** $|\Gamma(q)| > 1$ **then**
- 6 **if** $q \notin Q_{DS_{new}}$ **then** $Q_{DS_{new}} \cup \{q\}$
- 7 $\sigma \leftarrow Random(\Gamma(q))$
- 8 **else**
- 9 $\sigma \leftarrow \Gamma(q)$
- 10 **end**
- 11 $s_{new} \leftarrow s_{new}\sigma$
- 12 $q \leftarrow \delta(q, \sigma)$
- 13 **end**
- 14 $Msp \leftarrow Makespan.Evaluation(s_{new}, Q_{DS_{new}})$

In Algorithm 2, lines 1 to 3, the initial $round(\lambda \cdot |Q_{DS}(s)|)$ states are kept and included in $Q_{DS_{new}}$ (line 1) and the current state q becomes the last state in $Q_{DS_{new}}$ (line 2). Also, the prefix of the new sequence is obtained (line 3), by running the automaton from the initial state to state q passing through states of $Q_{DS_{new}}$. The sequence s_{new} is then interactively generated based on the possible continuations from the states reached (lines 4 to 13) until the marked state is reached. If another divergent state is reached in the path (line 5 to 8), the continuation from there is picked randomly and the state is added to $Q_{DS_{new}}$ (s_{new}). Otherwise, the available event (in a none divergent state) is concatenated with s_{new} . The end of the execution of this algorithm happens when it reaches the marked state. Two important aspects of the generating sequences from the closed-loop behavior are: the guarantee of none blocking states and no presence of loops which allows any path naturally flows to the marked state.

Example 7. We apply Algorithm 2 to Example 6 and the result is presented in Figure 3.3. The idea is to show how it can be used to generate a new sequence s_{new} from s . So,

let $s = \alpha_1\alpha_2\alpha_3\alpha_1\alpha_2\alpha_4\alpha_3\alpha_4$ and let $\lambda = 0.7$. The set $Q_{DS}(s) = \{2, 4, 5\}$ is the set of divergent states that are in the path of string s .

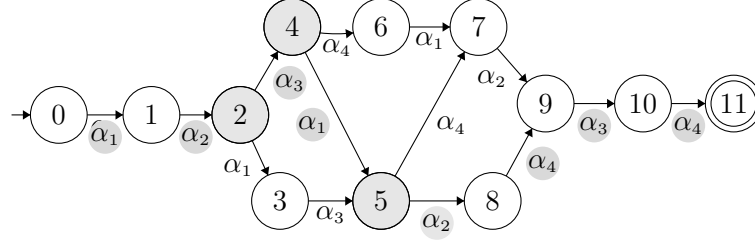


Figure 3.3: P_{E_q} and the sequence s highlighted in gray.

With $\lambda = 0.7$, 70% of the DS states in $Q_{DS}(s)$ are added to $Q_{DS}(s_{new}) = \{2, 4\}$. Then the string s_{new} is initialized as $s_{new} = \alpha_1\alpha_2\alpha_3$, in line 3. At this point, the execution enters the while loop. Since state $4 \in Q_{DS}$, we run lines 5 to 8, and pick a continuation randomly. Suppose that event α_4 is picked in line 7. Then, $s_{new} = \alpha_1\alpha_2\alpha_3\alpha_4$ and $q \leftarrow 6$. From state 6 on, no divergent states are reached, so the complete sequence is going to be $s_{new} = \alpha_1\alpha_2\alpha_3\alpha_4\alpha_1\alpha_2\alpha_3\alpha_4$, obtained by the execution of the “else” in lines 8 and 9 until state $11 \in Q_m$ is reached. In Figure 3.4, the resulting sequence s_{new} and its two DS states $Q_{DS}(s_{new}) = \{2, 4\}$ are highlighted in gray.

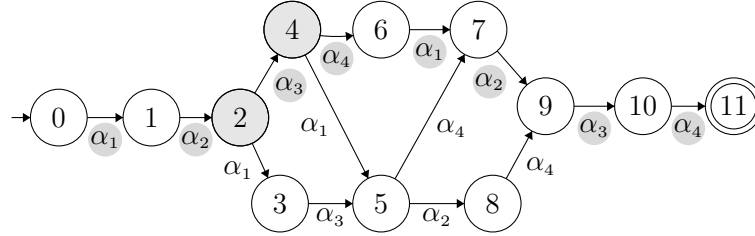


Figure 3.4: P_{E_q} and the sequence s_{new} highlighted in gray.

By applying the individual generator algorithm it is possible to create new feasible sequences of controllable events. This technique represents an efficient way to produce solutions to the optimization problem, because the new individual has embedded in its sequence of events the closed-loop behavior of the system which guarantees that it will run in the complete (non abstracted) system. Although the individual generator algorithm manages the creation of a new solution there are many other parameters responsible to improve the performance of the optimization. The behavior of these parameters is explained in Section 3.4.

3.3.1 Integration: CSA and DS states

Now that the basic concepts are presented, it is going to be explained how they are integrated with the CSA algorithm. A population with size N is generated by applying

Algorithm 2 N times which creates random individuals for this population. Each individual (Ind) is represent by 3-tuple $Ind = (seq, Q_{DS}(seq), Msp(seq))$:

- seq : Sequence of controllable events that produces a number of products;
- $Q_{DS}(seq)$: Set of divergent states related to the path of seq (Section 3.2);
- $Msp(seq)$: Total production time (makespan) of the sequence seq .

In this scenario, there are two parameters that directly impact the individuals of the new generation: the makespan and the set of divergent states. The first is used as fitness value and to sort the population in rising order so the best individual is on top with the smallest makespan and the sequence with the worst production time is on the bottom of the ledger. The second parameter holds a crucial information about the mutation process. As presented in Definition 6, the set of divergent states (Q_{DS}) is associated with the sequence of events (s). Therefore, controlling the number of DS states preserved in $Q_{DS}(s)$ allows us to determine the intensity of the mutation. In the CSA method, Algorithm 1, the parameter λ is responsible for the mutation rate. It controls how much an individual will be modified this is analogous to how many DS states are going to be preserved in Q_{DS} .

Example 8. *To better understand and clarify the mutation and the cloning processes a diagram is provided, Figure 3.5. On the Population box, the N individuals of a given population are presented sorted by makespan, from the best (on top) to the worst (on bottom). In front of each individual, there is a rectangle with its own set of DS states Q_{DS} . On top of all DS states sets, there is a ruler showing how much of its sequence is going to be preserved and inside the rectangle a light blue shade highlights the mutated DS states. The remaining area, the white part, represents the preserved DS states. In this example, the best individual has 80% of its DS states preserved and 20% of mutation, while the worst individual has only 10% preserved and 90% mutation. On the Offspring box, it is possible to observe that the best individual not only has more clones, but also a small part of its DS states are mutated. On the other hand, the worst individual (in the bottom) only has one clone and a small fraction of the parent's DS states is preserved.*

3.4 Setting the optimization parameters

The clonal search algorithm implemented has seven parameters, that are responsible for controlling the algorithm's ability of converging to a better solution and influence the optimization time. These parameters are:

- Population Size (N): sets the number of individuals in a population;
- Number of clones ($nClones$): represents the number of copies of a given individual;

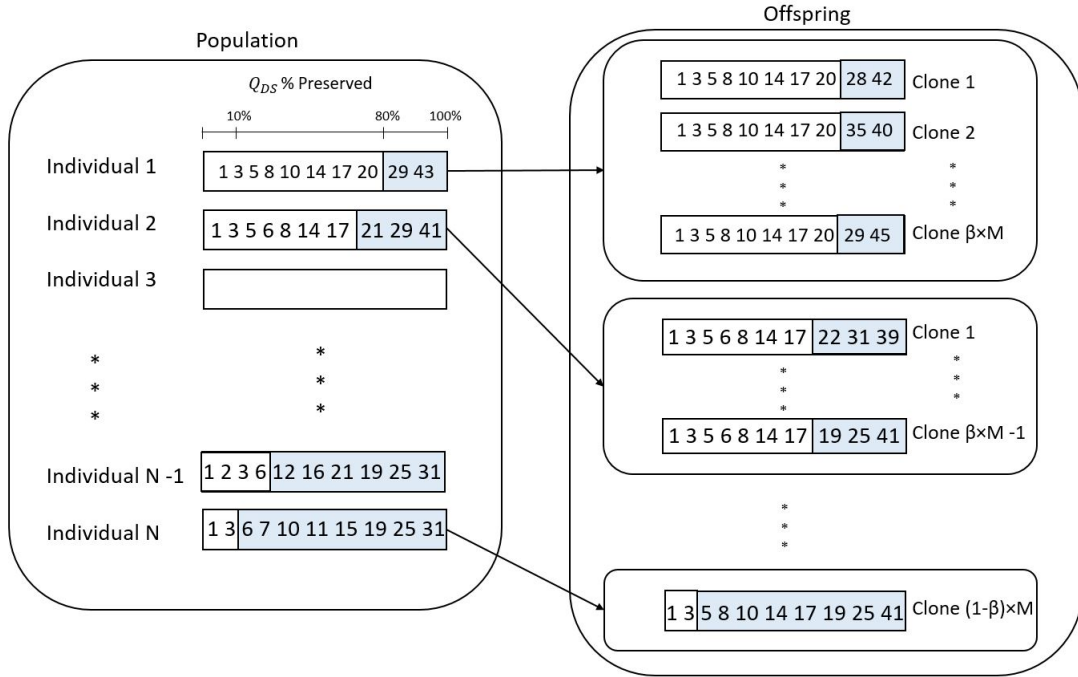


Figure 3.5: Cloning process diagram.

- Initial population (popRandom or popMixed): indicates the source of the individuals in the initial population, they can be randomly created or mixed ($X\%MPT + Y\%Random$), the quantities of X and Y are predetermined;
- Mutation rate (λ): determines the percentage of the individual that will be mutated;
- Stop criteria (nGen): sets the number of generations without improvements in the solution, in this case, without reaching a shorter makespan than the current one;
- Number of executions of the algorithm (nExec): indicates the number of executions of the optimization.

To assist the reader to understand how these parameters impact the overall performance of the clonal search algorithm, the FMS case was selected and the parameters (population size, number of Clones, mix initial population, mutation rate (λ) and number of generations) were tested. Despite of individual analysis of some parameters they are all correlated and to have a deep understanding of them a statistical analyses is necessary. However, the goal here is to give an overview of their behavior. In this analysis some basic values were predetermined and classified as *fixed*, if they do not change over the experiments and as *variable*, when they change. The parameters values and their classification are:

- Production of ten items ($A = 5, B = 5$) (Fixed),
- Number of Executions ($nExec = 50$) (Fixed),
- Number of individuals selected ($nIndSel = 1$) (Fixed),

- Population Size ($N = 100$) (Variable),
- Number of clones ($nClones = 5$) (Variable),
- Mutation rate ($\lambda = 5\%$) (Variable),
- Stop criteria ($nGen = 5$) (Variable),
- Mixed initial population (Variable).

The first parameter analyzed was the population size and how it impacts the average makespan and the total optimization time, which in these tests considered the total number of executions. Figure 3.6 shows the decrease of the average makespan as the size of the population increases as well as the the total optimization time. The reduction of the average makespan means that CSA is capable of finding more solutions with shorter makespan. On the other hand, the optimization time increases in what appears to be a linear rate as the population becomes larger. However, for populations larger than 50 individuals, the increase in time is more significant than the reduction of the average makespan. This way, a vertical rectangle is used to show the region, where the population size presents best trade off regarding the average makespan.

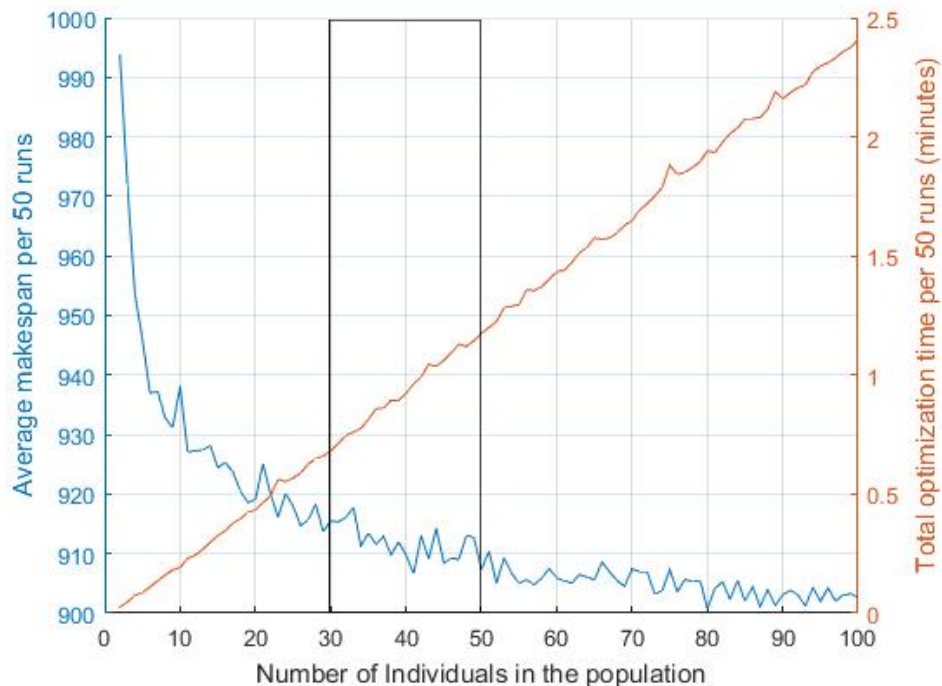


Figure 3.6: The population size impacts in the average makespan and optimization time.

Another parameter that impacts the CSA performance (total time and average makespan convergence) is the number of clones for each individual. As can be seen in Figure 3.7, values larger than 15 clones per individual do not have a significant reduction

on the average makespan, but it can drastically increase the total optimization time. This time the vertical rectangle area encapsulates the best values for the number of clones found during practical tests.

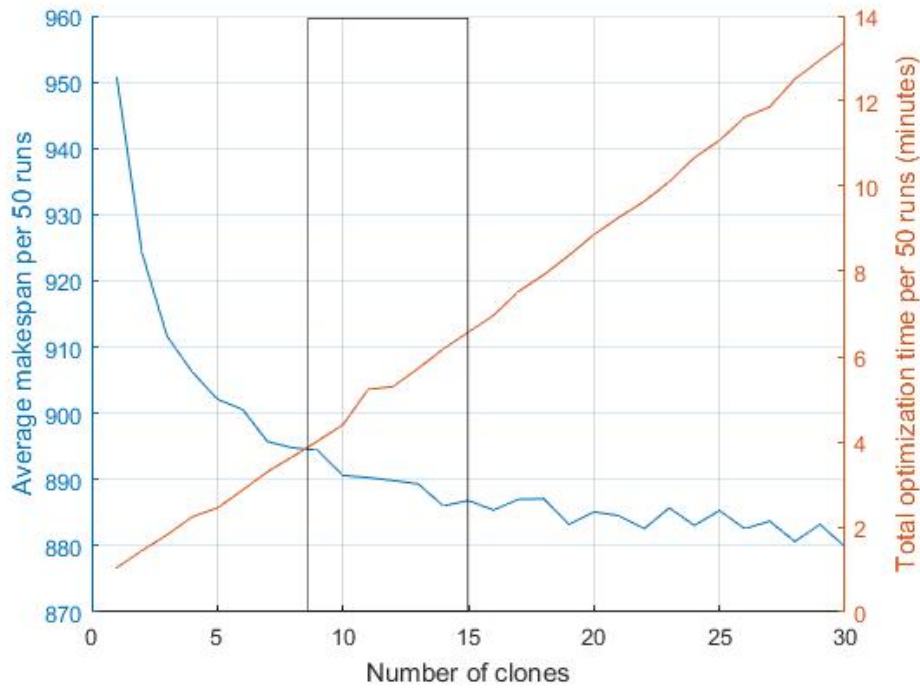


Figure 3.7: The population size impacts in the average makespan and optimization time.

In the clonal algorithm, the parameter related with making modifications in the individual is the mutation rate. Each generation the best individuals (the ones with the shortest makespan) have less intense mutation and the worst have a more aggressive mutation. Figure 3.8 shows the effects of varying this parameter. It can be noticed that the increase of the mutation rate leads to the reduction of the total optimization time. However, the mutation rate values above 0.5, or 50% increases of the average makespan, which reduces the capability of the algorithm to converge. During tests, the most significant results were found in the area inside the vertical rectangle in Figure 3.8.

The stop criteria used in this implementation of the CSA algorithm was the number of generations without improvements, which means, once a certain amount of generations is reached and the makespan has not got smaller, the algorithm stops the optimization, otherwise it resets the count and starts it all over again. It can be observed in Figure 3.9 that above 5 generations without improvements the total optimization time does not vary much, but the average makespan oscillates around the value of 900. Since the number of generations does not have a considerable impact in the overall time, the stop criteria choice relies in the values with the smaller average makespan.

The last parameter is the rate of individuals provided by the algorithm Timed Maximum Parallelism (MPT) in the initial population used. Mixing the individuals (random and

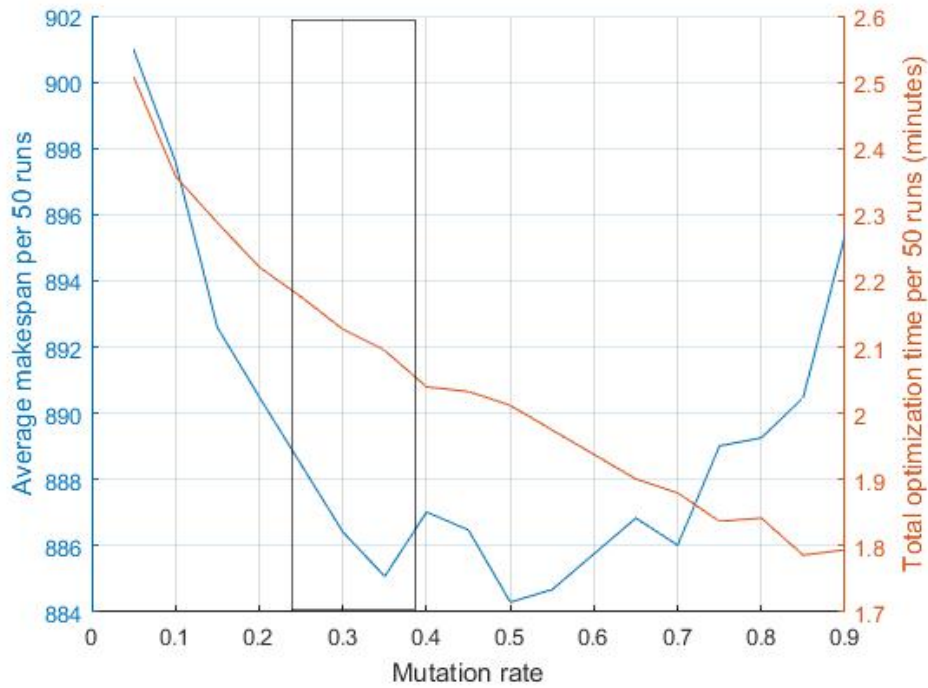


Figure 3.8: The population size impacts in the average makespan and optimization time.

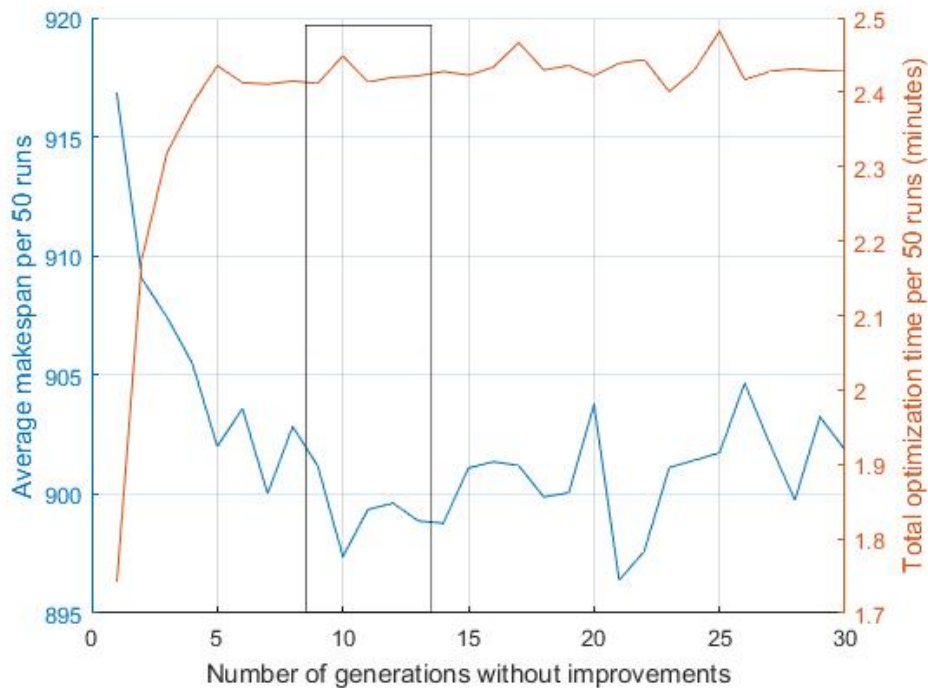


Figure 3.9: The stop criteria and impacts in the average makespan and optimization time.

from other methods) in the first population can help the optimization algorithm to wide its search, but it is necessary to determine the right ratio otherwise one type of solution becomes predominant and gets the algorithm stucked in a local solution. In order to understand how rate of individuals from the MPT influences the optimization three aspects

were observed: the average and the shortest makespan, and the total optimization time, Figure 3.10.

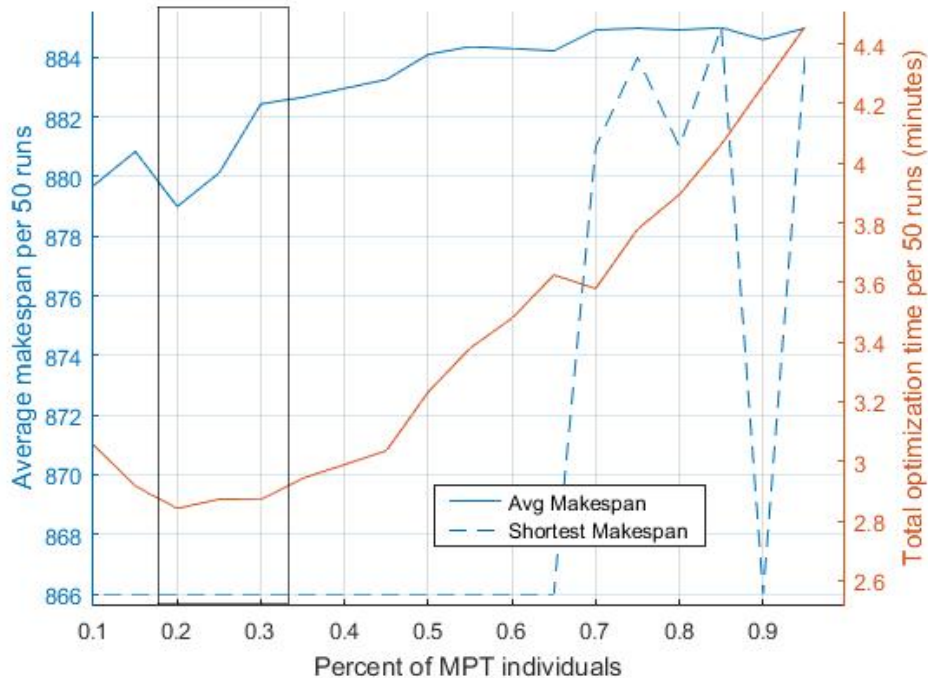


Figure 3.10: The effects of different amounts of TMP individuals in the average makespan and in the optimization time.

The analysis of Figure 3.10 makes clear that the lack of diversity (populations with more than 60% of individuals from the Timed Maximum Parallelism) resulted in solutions that were not able to assist the CSA algorithm to avoid the local optimum in most cases. Besides, as the diversity increases, in other words the percentage of individuals from MPT reduces (less than 50%), the average makespan becomes smaller than the value found by the MPT alone (885) and most instances tested were capable of finding the value of 866, which is the shortest found. Another impacted factor was the total optimization time. Its value decreases as the percentage of individuals from the MPT reduces. Besides, the dashed lines shows that most solutions below 0.65 of MPT individuals were capable to find the shortest value. Another significant aspect of having more than just one individual form for the MPT (despite of their similarity) is the fact that they suffer different mutation rates. This allows the algorithm to escape the local optimum because as the mutation rate increases the number of modifications on the individual increases too.

In this section a small over view on the CSA optimization parameters was provided by showing how their values impact the average makespan and the total optimization time. For each case study it was necessary to repeat these tests to understand how these parameters respond in a different problem and choose the appropriated values.

3.5 Most Frequent Event Algorithm

During some preliminary tests it was possible to observe one characteristic emerging from all the evaluated instances. In most of the cases, when a DS state was reached it became noticeable that the best solution always privileged events that had already happened, than events waiting for their first execution. This leads as to believe that the events on the beginning of the production are prioritized over the ones at the end of the process.

Example 9. Consider the automaton G , in Figure 3.11 state 2, it is possible to make events α_1 and α_3 , but only event α_1 was already made prior than reaching state 2. The same can be observed in the DS states 4 and 5.

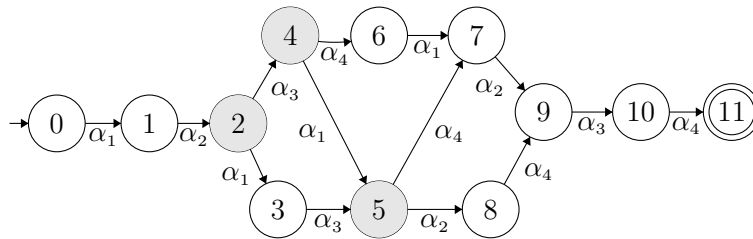


Figure 3.11: The automaton G illustrates the presence of the most frequent events.

Based on the characteristics observed the Most Frequent Event algorithm was developed, Algorithm 3. This method uses the automaton of the abstraction of the closed-loop behavior to generate a sequence of controllable events that minimizes the production time. In lines 1 and 2, the current state q becomes the initial state of G and Q_m is the set of marked states of G . This algorithm goes from the initial state q until the marked state $q_m \in Q_m$ passing through DS states along the way. The sequence s_{new} is iteratively generated (lines 11). Every DS state reached is stored in $Q_{DS_{new}}$ and the selected event σ is chosen by the function *GetMostFrequentEvent()*, line 6. This method returns the most frequent event based on the set of available events provided by the active events function $\Gamma(q)$ and the events in the most frequent in the *Events Table*. If no divergent state is reached in the path (line 7 to 9), the available event is concatenated with the sequence s_{new} and saved, line 11. The final steps are the updates of the Events Table by increasing the frequency of the selected event σ , and the current state, lines 11 and 12

3.6 Final Remarks

In this chapter some new properties regarding the states of an automaton are introduced. Then, it develops showing how this new properties are integrated in an algorithm to generate sequences/individuals for the optimization. In addition an exploratory example is given. It concludes by presenting a heuristic based on this new technique that works

Algorithm 3: Most Frequent Event

Input : G
Output : $Q_{DS_{new}}, s_{new}, Msp$

- 1 $Q_m \leftarrow \text{MarkedState}(G)$
- 2 $q \leftarrow \text{InitialState}(G)$
- 3 **while** $q \notin Q_m$ **do**
- 4 **if** $|\Gamma(q)| > 1$ **then**
- 5 $Q_{DS_{new}} \leftarrow q$
- 6 $\sigma \leftarrow \text{GetMostFrequentEvent}(\Gamma(q), \text{EventsTable}())$
- 7 **else**
- 8 $\sigma \leftarrow \Gamma(q)$
- 9 **end**
- 10 $s_{new} \leftarrow s_{new}\sigma$
- 11 $\text{Update}(\text{EventsTable}, \sigma)$
- 12 $q \leftarrow \delta(q, \sigma)$
- 13 **end**
- 14 $Msp \leftarrow \text{Makespan.Evaluation}(s_{new}, Q_{DS_{new}})$

independently of an optimization method to find solutions. In the next chapter, this two new methods (the individual generator algorithm and the most frequent event algorithm) will be applied on three study cases to evaluate their performances.

4

Experimental Results

In this chapter, the methodology presented in Chapter 3 was applied in two problems of the DES literature, the Cluster Tool (Uzsoy et al., 1994) and the Flexible Manufacturing System (De Queiroz et al., 2005). Each problem presented unique optimization challenges related to their layout and the amount of manufactured products. To accomplish this task, for each problem a set of production batches was established with different number of products to be made. Moreover, the timed maximum parallelism (Alves et al., 2016a) was used as a baseline method against the CSA algorithm and the MFE heuristic. In order to compare the results, the run time and makespan were used. However to assure that the abstraction presented in Section 2.4 could be used with these problems they were checked for the presence of the observer property. This was accomplished by applying the observer check method implemented in TCT (Feng & Wonham, 2006), all case studies presented passed in the evaluation. Finally, all tests were executed on a notebook with an Intel Core I7-3537U 2.0 GHz processor with 8.0 GB of RAM. In addition, the UltraDES library from Alves et al. (2017) was used to compute the supervisor and the abstraction.

4.1 Procedure

In order to apply the results presented in this work (Algorithms 2 and 3) to solve a scheduling problem, the following steps were performed:

- Obtain the model and specifications and compute the supremal controllable and nonblocking sublanguage S ;

- Apply the natural projection (P) to the controllable events over S ;
- Check the Observer Property in $P(S)$. If the projection is OP, then proceed;
- Make the parallel composition between the natural projection and the product specification, $P_{E_q} = P(S)||E_q$;
- Apply an optimization technique to find a sequence (s_{opt}) that minimizes makespan. In this work, two optimization techniques were proposed:
- In the CSA (Algorithm 1):
 - Input the abstraction P_{E_q} and the parameters (population size, mutation rate, number of clones and stop criteria). Those parameters are adjusted with a series of tests, as described in Section 3.4;
 - Generate the population (Algorithm 2), clone and mutate.
 - Evaluate each individual and the best one of each population survives to compose the next generation. This process is repeated until the stop criteria is reached.
 - Execute the CSA 30 times to calculate the average and the standard deviation of the makespan for a given production batch;
 - Output s_{opt} , the sequence with smaller makespan found.
- The MFE (Algorithm 3) can be executed directly.

In addition to basic CSA with random individuals in the initial population, two other sources of individuals were tested: MPT and MFE individuals. The reason for this approach is to understand the impact of these new sources of solutions in the CSA algorithm. In the next few sections, we present the results on each one of the systems described in Section 2.7.

4.2 The Cluster Tool

This manufacturing system can be constructed with many different layouts, for the purpose of this work, the radial architecture with one and two robots in the mainframe were considered.

Initially, five different batches of wafers had their makespan evaluated, with sizes of 6, 12, 25, 50 and 100. For each production batch, 30 runs of the optimization method CSA with three different initial populations (random, MPT and MFE individuals) were made. The algorithm Most Frequent Event (Section 3.5) was also applied.

4.2.1 One Robot Cluster Tool

The first problem tested was the radial cluster tool layout with one robot and the results can be seen in Table 4.3. Before starting to evaluate the makespan of the batches, it is necessary to know the production time associated with each machine presented in the set up. The CSO uses the time interval, between a pair of events (a controllable and its counter part uncontrollable) to evaluate the makespan. For the cluster tool with one robot, these time intervals are shown in Table 4.1.

Table 4.1: Time interval for the Cluster Tool with one robot and four process modules, (Nunes, 2018).

Machine	Controllable Event	Uncontrollable Event	Time Interval (t.u.)
LL	11	12	535
	13	14	706
R_1	21	22	474
	23	24	739
	25	26	781
	27	28	781
	29	30	690
C_1	101	102	50
C_2	201	202	50
C_3	301	302	50
C_4	401	402	50

Due to the possibility of varying the process chambers (C_1, C_2, C_3 and C_4) production time, they will be presented along with the optimization parameters in Table 4.2. In this table, the two first's columns show the cluster tool layout's configuration (number of chambers and process modules time interval), then the parameters for the optimization algorithm are presented, which tuning process was explained in Section 3.4. To simplify the tests, only one set of the optimization parameters was used to evaluate all five batches.

Table 4.2: Optimization parameters used for the cluster tool with one robot problem.

CT Characteristics		Optimization Parameters				
Number of Chambers	Process Module	Population Size	Mutation Rate %	Gen. Without Improvement	Number of Clones	Maximum of Generations
4	50	20	25	5	5	40

For this experiment, the two proposed optimization methods, the MFE algorithm and the CSA with three variation are going to be tested:

- the Most Frequent Event (MFE);
- the CSA + MFE: combines the CSA algorithm with an initial population of 20% of its individuals from the MFE heuristic and 80% randomly created;

- the CSA + MPT: combines the CSA algorithm with an initial population of 20% of its individuals from the MPT heuristic and 80% randomly created;
- the CSA with random initial population (CSA Rand Pop).

In Table 4.3, each optimization technique is represented by two columns, one for the Runtime (total simulation time in minutes - *RT*) and another for the makespan (*MKS*). Under both CSA variants, the runtime represents the total optimization time with the 30 executions of the optimization algorithm and in the makespan column only the shortest makespan was presented. All the proposed methods are compared against the *MPT* algorithm.

Table 4.3: One robot cluster tool makespan optimization results

Prod.	MPT		MFE		CSA + MPT		CSA Rand.	
	RT (min.)	MKS (t.u.)	RT (min.)	MKS (t.u.)	RT (min.)	MKS (t.u.)	RT (min.)	MKS (t.u.)
(6)	0.00143	22,131	0.00017	22,131	0.50	22,131	0.53	22,131
(12)	0.0022	42,921	0.00007	42,921	1.04	42,921	0.98	42,921
(25)	0.0052	87,966	0.00015	87,966	2.16	87,966	2.26	87,966
(50)	0.0110	174,591	0.00028	174,591	4.34	174,591	4.58	174,591
(100)	0.0223	347,841	0.0005	347,841	8.46	347,841	9.13	347,841

The results presented in Table 4.3 show that all proposed methods were capable of finding the same solution for the shortest makespan, which was the same found by the MPT. In addition, the MFE heuristic was the fastest method, regarding the runtime for all batches evaluated. Although, the CSA variants found the same makespan values as the MPT, in most instances tested the CSA with mixed initial population was faster than the CSA with random population.

In Table 4.4, the average makespan and the standard deviation (S.D.) found during the 30 runs of the CSA algorithm are presented. In this table, the results for all three variants of the CSA algorithm are the same as the smallest values in Table 4.3. The standard deviation value confirms that, in all runs the same makespan was found.

Table 4.4: One robot cluster tool makespan optimization results for the average makespan in 30 runs.

Prods	CSA + MFE Initial Population		CSA + MPT Initial Population		CSA Random Initial Population	
	Average Makespan (t.u.)	S.D.	Average Makespan (t.u.)	S.D.	Average Makespan (t.u.)	S.D.
(6)	22,131	0	(t.u.)	0	22,131	0
(12)	42,921	0	42,921	0	42,921	0
(25)	87,966	0	87,966	0	87,966	0
(50)	174,591	0	174,591	0	174,591	0
(100)	347,841	0	347,841	0	347,841	0

4.2.2 Two Robots Cluster Tool

The second architecture of the cluster tool evaluated had two robots working in the mainframe. Each robot is responsible for half of the wafer manufacturing process. As in Section 4.2.1, it is necessary to know the production time of each machine in this new set up, which is also done by using the time interval between events (controllable and its counter part uncontrollable), Table 4.5.

Table 4.5: Time interval for the Cluster Tool with two robot and four PM modules,(Nunes, 2018).

Machine	Controllable Event	Uncontrollable Event	Time Interval (t.u.)
LL	11	12	535
	13	14	706
R_1	21	22	474
	23	24	739
	25	26	781
R_2	31	32	672
	33	34	690
C1	101	102	50
C2	201	202	50
C3	301	302	50
C4	401	402	50

Once more, the optimization parameters used were found after applying a similar process to the one explained in Section 3.4 and they can be seen in Table 4.6. In this table, under CT Characteristics column there are the CT's number of process chambers ($C_1, C_2, C_3, \dots, C_N$) and process modules time interval and under Optimization Parameters the values employed in optimization algorithm. Despite of the fact that this problem has one extra robot in the mainframe the parameters were similar to the case with one robot.

Table 4.6: Optimization parameters used for the cluster tool with two robots problem.

CT Characteristics		Optimization Parameters				
Number of Chambers	Process Module	Population Size	Mutation Rate %	Gen. Without Improvement	Number of Clones	Maximum of Generations
4	50	20	25	5	5	40

Again, the two optimization methods proposed the Most Frequent Event (MFE) and the CSA with three different initial populations (MFE, MPT and Random) are going to be tested. Here 20% of the initial population comes from the MPT, or MFE heuristic and 80% are randomly created for the CSA.

The results presented in Table 4.7 show that all CSA variants tested were capable of reaching the same shortest makespan value in all batches evaluated, as accomplished by

MPT. Despite of its shortest runtime for all batches, the pure MFE heuristic was not capable to find the same, or a smaller, value for the shortest makespan than the MPT.

Table 4.7: Two robots cluster tool makespan optimization results

Prod	MPT		MFE		CSA + MPT		CSA Rand	
	RT (min.)	MKS (t.u.)	RT (min.)	MKS (t.u.)	RT (min.)	MKS (t.u.)	RT (min.)	MKS (t.u.)
(6)	0.0002	14,767	0.0002	18,367	0.49	14,767	0.66	14,767
(12)	0.0003	26,731	0.0001	34,771	0.96	26,731	1.19	26,731
(25)	0.0007	52,653	0.0001	70,313	1.97	52,653	2.53	52,653
(50)	0.0012	102,503	0.0003	138,663	3.86	102,503	4.81	102,503
(100)	0.0026	202,203	0.0006	275,363	7.54	202,203	9.38	202,203

Once again, the results in Table 4.8 show that in all 30 runs the algorithm (in its three variants of the CSA) converged for the smallest solution found (similar to the MPT method). This is confirmed with the standard deviation equals to zero.

Table 4.8: Two robots cluster tool makespan optimization results

Prods	CSA + MFE Initial Population		CSA + MPT Initial Population		CSA Random Initial Population	
	Average Makespan (t.u.)	S.D.	Average Makespan (t.u.)	S.D.	Average Makespan (t.u.)	S.D.
(6)	14,767	0	14,767	0	14,767	0
(12)	26,731	0	26,731	0	26,731	0
(25)	52,653	0	52,653	0	52,653	0
(50)	102,503	0	102,503	0	102,503	0
(100)	202,203	0	202,203	0	202,203	0

4.3 The Flexible Manufacturing System

The FMS can manufacture two types of products (A and B). Each batch, the same amount of products A and B were made with batches ranging from 1 to 10 products.

The first step before starting to evaluate the makespan of the batches is to know the production time associated with each machine presented in the set up. In Table 4.9, the time intervals for the machines in FMS expressed in time units (t.u) are shown. In this table the controllable event 61 does not have an uncontrollable counterpart, so after it happens 15 t.u later, it allows the events 63 or 65 to happen. The optimization parameters used are presented in Table 4.10.

The results using the MFE heuristic and the CSA algorithm variants are presented in Table 4.11, where the number of products ($Prods$) A and B are identified as (nA , nB). For

Table 4.9: Time interval between controllable and uncontrollable events for the FSM.

Machine	Control. Events	Uncontrol. Events	Time Interval (t.u.)
C1	11	12	26
C2	21	22	26
Robot	31	32	22
	33	34	20
	35	36	17
	37	38	25
	39	30	21
Mill	41	42	31
Lathe	51	52	39
	53	54	33
AM	61	-	15
	63	64	27
	65	66	27
C3	71	72	26
	73	74	26
PD	81	82	25

Table 4.10: Optimization parameters used for the FMS problem.

Optimization Parameters				
Population Size	Mutation Rate %	Stop Criteria	Number of Clones	Maximum of Generations
25	15	11	15	30

this experiment, the CSA with mixed initial population (from MFE and MPT) had 15% of the total initial population and 85% randomly created, this values were defined during preliminary tests.

Table 4.11: FMS Makespan optimization using the proposed methods

Prods.	MFE		CSA + MFE Initial Population		CSA + MPT Initial Population		CSA Random Initial Population	
	RT (min.)	MKS (t.u)	RT (min.)	MKS (t.u)	RT (min.)	MKS (t.u)	RT (min.)	MKS (t.u)
(1, 1)	0.0002	307	0.81	238	0.45	238	0.50	238
(2, 2)	0.00002	590	1.44	395	0.83	395	0.85	395
(3, 3)	0.00002	807	1.49	552	1.11	552	1.25	552
(4, 4)	0.00002	1,027	1.96	709	1.22	709	1.52	709
(5, 5)	0.00005	1,247	2.21	866	1.37	866	1.53	866
(6, 6)	0.0001	1,467	2.87	1,023	1.53	1,023	2.03	1,023
(7, 7)	0.00011	1,670	3.56	1,180	1.44	1,180	2.40	1,180
(8, 8)	0.00011	1,907	4.26	1,337	1.64	1,337	3.30	1,337
(9, 9)	0.00012	2,127	5.03	1,494	1.75	1,513	3.81	1,494
(10, 10)	0.00025	2,347	4.44	1,651	2.01	1,670	4.98	1,651

As can be seen in Table 4.11, the three CSA variants performed better than the MFE

approach. The CSA using a random initial population performed slightly better than the CSA + MFE in regard to the optimization time. A surprising result came from the CSA + MPT, which was not able to find the smallest makespan for a couple of batches ((9, 9) and (10, 10)). It was expected that the populations with the mixed individuals would perform better once pre-optimized individuals were used. It turned out that the algorithm was stuck in local optimal sequences while the random population allowed a wider exploration of the search universe.

In order to understand the differences among the three variants of the initial population of the CSA, the average and the standard deviation of the makespan were analyzed and the results placed in Table 4.12. This table shows that for the batches of sizes (1, 1) and (2, 2) the three variants of the CSA were capable of finding the smallest makespan on every run with zero standard deviation. But only the CSA + MPT and the CSA Random were able to have the same result for the batch (3, 3). When compared with the CSA + MFE, the CSA Random had the smallest standard deviation in most of the tested instances. And despite the fact that the CSA + MPT has the smallest standard deviation, it was capable of finding the smallest makespan for the batches of (9, 9) and (10, 10). For this reason the CSA Random is going to be used as base case to compare with other methods.

Table 4.12: Average and standard deviation of the FMS makespan in 30 runs of the CSA.

Prods.	CSA + MFE Initial Population		CSA + MPT Initial Population		CSA Random Initial Population	
	Average Makespan. (t.u.)	S.D.	Average Makespan. (t.u.)	S.D.	Average Makespan. (t.u.)	S.D.
(1, 1)	238.0	0	238.0	0	238.0	0
(2, 2)	395.0	0	395.0	0	395.0	0
(3, 3)	553.1	4.1	552.0	0	552.0	0
(4, 4)	713.1	7.3	713.2	7.3	714.0	8,3
(5, 5)	878.5	9.9	878.1	8.4	874.9	9.5
(6, 6)	1,044.8	15.0	1,043.2	11.8	1,037.3	6.7
(7, 7)	1,208.9	18.1	1,196.4	5.7	1,204.2	12.1
(8, 8)	1,365.13	20.9	1,355.7	0.79	1,358.7	12.9
(9, 9)	1,525.3	15.2	1,513	0	1,522.9	15.1
(10, 10)	1,693.9	26.27	1669.4	3.5	1687.1	16.8

In Table 4.13, the optimization results applying the CSA with random initial population, the MPT method and the formal verification approach (Malik & Pena, 2018) are presented. The formal verification evaluates each and every sequence that produces a batch of each size, so it finds the optimal sequence. That justifies its always higher runtime. Using Malik & Pena (2018), it is possible to evaluate the quality of the solution.

In this context, one surprising result came from the CSA using random initial population. The CSA algorithm was capable of finding the sequence with the optimal makespan value

Table 4.13: CSA with random initial population versus the MPT and the Formal Verification in the FMS Makespan optimization.

Prods.	MPT		Formal Verification		CSA Random Initial Population	
	RT (min.)	MKS (t.u.)	RT (min.)	MKS (t.u.)	RT (min.)	MKS (t.u.)
(1, 1)	0.002	272	0.6	238	0.50	238
(2, 2)	0.004	414	2.7	395	0.85	395
(3, 3)	0.009	571	5.0	552	1.25	552
(4, 4)	0.018	728	7.5	709	1.52	709
(5, 5)	0.027	885	10.2	866	1.53	866
(6, 6)	0.04	1,042	14.0	1,023	2.03	1,023
(7, 7)	0.056	1,199	17.5	1,180	2.40	1,180
(8, 8)	0.074	1,356	21.5	1,337	3.29	1,337
(9, 9)	0.095	1,513	24.5	1,494	3.81	1,494
(10, 10)	0.12	1,670	30.2	1,651	4.98	1,651
(11, 11)	0.48	1,827	36.4	1,808	5.87	1,808
(12, 12)	0.54	1,984	42.0	1,965	6.75	1,983
(13, 13)	0.62	2,141	45.1	2,122	7.42	2,155
(14, 14)	0.71	2,298	53.4	2,279	8.66	2,297
(15, 15)	0.84	2,455	61.7	2,436	8.55	2,469
(16, 16)	0.96	2,612	-	-	9.17	2,644
(17, 17)	1.07	2,769	-	-	10.06	2,798
(18, 18)	1.21	2,926	-	-	10.24	2,979
(19, 19)	1.41	3,083	-	-	11.23	3,144
(20, 20)	1.48	3,240	-	-	14.39	3,308

for instances ranging from (1, 1) until (11, 11). Besides, it was accomplished six times faster, than the verification method in the production batch (11, 11), Table 4.13. The fact that the algorithm is capable of evaluating batches larger than (15, 15), it is another advantage over the Formal Verification that is limited to (15, 15).

However, despite the success on evaluating larger batches, from the instance (12, 12) on, the decay in performance became noticeable and was evidenced by the increase on the makespan, when compared with the MPT. A possible solution for this issue is to re-calibrate the algorithm's parameters, because as they are presented in Table 4.10, it seems that they are better adjusted for batches (1, 1) to (11, 11).

To understand the level of progress brought by the new method of generating individuals (proposed in this work) for the SCO, the number of evaluations of the objective function and the total optimization time (in minutes) were compared to Costa et al. (2018). The results can be seen in Table 4.14, where four initial batches (from 1, 1 to 4, 4) are tested.

In all instances tested the CSA with Random initial population made less evaluations of the objective function (O.F.) and in the worst cases the batch with size (4, 4) the number of evaluation was around 3,500 times smaller. For this instance the SCO normally spent

Table 4.14: SCO (Costa et al., 2018) versus CSA with random initial population in the FMS Makespan optimization.

Products	SCO		CSA Random Population	
	Average Number of Evaluation of O.F.	Total Time (Min.)	Average Number of Evaluation of O.F.	Total Time (Min.)
(1,1)	34,680	5.09	2,171	0.56
(2,2)	541,970	95.62	2,216	0.94
(3,3)	2,151,400	636.18	2,345	1.49
(4,4)	6,746,900	3,256.16	1,946	1.67

3,256 minutes, which are equivalent to 54 hours while the CSA Random would need 1.67 minutes.

4.4 Final Remarks

This chapter presented the experimental results for the proposed techniques on three optimization problems. It was possible to see that the two new methods (the CSA + individual generation heuristic and the MFE heuristic) were capable of producing solutions that in some cases were faster than the baseline (MPT) and in others it was able to find the same solution as the exactly method but faster than it. Besides when compared with the previous version of the SCO significant gains in speed were reported.

5

Conclusions

In this dissertation two methods for solving the problem of finding the optimal scheduling were presented. The first was an algorithm that creates solutions/individuals for an evolutionary algorithm (the clonal selection algorithm). One particular aspect of these methods was the use of the abstraction of the closed-loop behavior in the creation of individuals. The second method was the Most Frequent Event algorithm which is a heuristic algorithm that favors the events that more often are available during the operation of the system. In spite of not guaranteeing optimal results both methods in general were capable of finding the best solutions. The MFE was the fastest regarding the run time in all instances tested, while the CSA + Random individual generation algorithm presented the shortest makespan values.

This work applies the theoretical result of Vilela & Pena (2016) to generate solutions for the optimization of scheduling problems. One of the major advantages of using the abstraction is the reduction of the problem's search universe. The proposed method used the supervisor control theory to create a solution that encodes the closed-loop behavior of the system and combine it with the clonal search algorithm. In addition with this technique no unfeasible sequence is produced, in another words, all solutions found can be executed in the system making the approach very efficient.

Our findings in the studies cases show that, in problems where the optimal solution is known, the proposed method was able to find it. That was shown in the FMS study case, where the same solution obtained by the formal verification method could be found for production batches ranging from (1,1) to (11,11) when using the CSA with random

individuals. This result was presented in Rafael & Pena (2018). Besides, the proposed method was capable of solving the FMS problem for batches larger than the Formal Verification. Despite this fortunate outcome the process of setting the optimization parameters proved to be one of the biggest challenges of this work because of considerable number of parameters to set and their correlation.

5.1 Future Research

Some possible future developments:

- *Evaluate the SCO-CONCAT performance, when the traditional SCO is replaced for the results found in this work, in the process of generating the MPS;*
- *Use other heuristic methods integrated with the proposed methodology like the VNS, the Ant colony algorithm among others;*
- *Extend the methodology proposed to multiobjective optimization problems;*
- *Develop an automatic method for setting the parameters of the CSA;*
- *Adjust the most frequently event heuristic to improve its solutions for imbalanced number of events systems.*

Bibliography

- Abdeddaïm, Y., Asarin, E., Maler, O., et al. (2006). Scheduling with timed automata. *Theoretical Computer Science*, 354(2), 272–300. Cited in page 1.
- Akesson, K., Fabian, M., Flordal, H., & Malik, R. (2006). Supremica-an integrated environment for verification, synthesis and simulation of discrete event systems. In *8th International Workshop on Discrete Event Systems (WODES)* (pp. 384–385).: IEEE. Cited in page 5.
- Alves, L. V., Bravo, H. J., Pena, P. N., & Takahashi, R. H. (2016a). Planning on discrete events systems: A logical approach. In *International Conference on Automation Science and Engineering (CASE)* (pp. 1055–1060).: IEEE. Cited 4 times in the pages 1, 3, 4, and 39.
- Alves, L. V., Martins, L. R., & Pena, P. N. (2017). Ultrades - a library for modeling, analysis and control of discrete event systems. *Proceedings of the 20th World Congress of the International Federation of Automatic Control*, 50(1), 5831–5836. Cited 3 times in the pages 13, 23, and 39.
- Alves, L. V., Pena, P. N., & Takahashi, R. H. (2016b). Planejamento da produção baseado no critério do máximo paralelismo com restrições temporais. *Anais do XXI Congresso Brasileiro de Automática, CBA*. Cited 2 times in the pages 4 and 15.
- Alves, M. C. (2018). Abstrações de supervisores localmente modulares para aplicação na solução de problemas de planejamento, dissertação (mestrado), 2018. Programa de Pós - Graduação em Engenharia Elétrica - PPGEE. Cited 3 times in the pages 9, 10, and 23.
- Arisha, A., Young, P., & El Baradie, M. (2001). Job shop scheduling problem: an overview. *International Conference for Flexible Automation and Intelligent Manufacturing (FAIM 01)*, (pp. 682–689). Cited 2 times in the pages 2 and 3.
- Aytug, H., Lawley, M. A., McKay, K., Mohan, S., & Uzsoy, R. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161(1), 86–110. Cited in page 2.

- Baker, K. R. & Trietsch, D. (2013). *Principles of sequencing and scheduling*. John Wiley & Sons. Cited 2 times in the pages 2 and 3.
- Bellman, R. (1958). On a routing problem. *Quarterly of applied mathematics*, 16(1), 87–90. Cited in page 22.
- Bellman, R. E. & Dreyfus, S. E. (2015). *Applied dynamic programming*. Princeton University Press. Cited 2 times in the pages 2 and 3.
- Brucker, P., Jurisch, B., & Sievers, B. (1994). A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49(1-3), 107–127. Cited in page 3.
- Cassandras, C. G. & Lafortune, S. (2009). *Introduction to discrete event systems*. Springer Science & Business Media. Cited in page 1.
- Charnes, A. & Cooper, W. W. (1962). Programming with linear fractional functionals. *Naval Research logistics quarterly*, 9(3-4), 181–186. Cited in page 3.
- Costa, T. A., de Oliveira, A. C., Pena, P. P., & Takahashi, R. H. (2012). An ant system algorithm for task scheduling in a flexible manufacturing cell with supervisory control. In *XIX Congresso Brasileiro de Automática, CBA*, volume 12 (pp. 2515–2522). Cited in page 5.
- Costa, T. A., Pena, P. N., & Takahashi, R. H. (2018). Sco-concat: a solution to a planning problem in flexible manufacturing systems using supervisory control theory and optimization techniques. *Journal of Control, Automation and Electrical Systems*, (pp. 1–12). Cited 5 times in the pages xi, 1, 5, 47, and 48.
- Cury, J. E. R. (2001). Teoria de controle supervisório de sistemas a eventos discretos. *V Simpósio Brasileiro de Automação Inteligente (Minicurso)*. Cited in page 8.
- De Castro, L. N. & Von Zuben, F. J. (2002). Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6(3), 239–251. Cited 2 times in the pages 6 and 15.
- De Queiroz, M. H. & Cury, J. E. (2000). Modular supervisory control of large scale discrete event systems. In *Discrete Event Systems* (pp. 103–110). Springer. Cited in page 23.
- De Queiroz, M. H., Cury, J. E., & Wonham, W. M. (2005). Multitasking supervisory control of discrete event systems. *Discrete Event Dynamic Systems*, 15(4), 375–395. Cited 4 times in the pages 5, 17, 20, and 39.
- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1), 269–271. Cited in page 22.

- Feng, L. & Wonham, W. M. (2006). Tct: A computation tool for supervisory control synthesis. In *8th International Workshop on Discrete Event Systems(WODES)* (pp. 388–389).: IEEE. Cited 2 times in the pages 17 and 39.
- Garey, M. R. & Johnson, D. S. (1979). Computers and intractability: a guide to np-completeness. Cited in page 1.
- Garey, M. R. & Johnson, D. S. (1980). Computers and intractability: A guide to the theory of np -completeness. *Bulletin (New Series) of the American Mathematical Society*, 3(2), 898–904. Cited in page 2.
- Ghallab, M., Nau, D., & Traverso, P. (2016). *Automated planning and acting*. Cambridge University Press. Cited in page 3.
- Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100–107. Cited in page 22.
- Hill, R. C. & Lafortune, S. (2016). Planning under abstraction within a supervisory control context. In *55th Conference on Decision and Control (CDC)* (pp. 4770–4777).: IEEE. Cited in page 4.
- Hoitomt, D. J., Luh, P. B., & Pattipati, K. R. (1993). A practical approach to job-shop scheduling problems. *IEEE Transactions on Robotics and Automation*, 9(1), 1–13. Cited in page 3.
- Jula, P. & Leachman, R. C. (2010). Coordinated multistage scheduling of parallel batch-processing machines under multiresource constraints. *Operations Research*, 58(4-part-1), 933–947. Cited in page 17.
- Kobetski, A. & Fabian, M. (2006). Scheduling of discrete event systems using mixed integer linear programming. In *2006 8th International Workshop on Discrete Event Systems* (pp. 76–81).: IEEE. Cited 2 times in the pages 3 and 4.
- Lee, C.-Y., Piramuthu, S., & Tsai, Y.-K. (1997). Job shop scheduling with a genetic algorithm and machine learning. *International Journal of Production Research*, 35(4), 1171–1191. Cited in page 3.
- Lee, S. & Ni, J. (2012). Genetic algorithm for job scheduling with maintenance consideration in semiconductor manufacturing process. *Mathematical Problems in Engineering*, 2012. Cited 2 times in the pages ix and 17.
- López-Mellado, E., Villanueva-Paredes, N., & Almeyda-Canepa, H. (2005). Modelling of batch production systems using petri nets with dynamic tokens. *Mathematics and Computers in Simulation*, 67(6), 541–558. Cited in page 1.

- Malik, R. & Pena, P. N. (2018). Optimal task scheduling in a flexible manufacturing system using model checking. In *2018 14th International Workshop on Discrete Event Systems (WODES)*: IEEE. Cited 4 times in the pages ix, 3, 4, and 46.
- Masud, A., Al Bashir, M., & Islam, M. Z. (2011). Approach to job-shop scheduling problem using rule extraction neural network model. *Global Journal of Computer Science and Technology*. Cited in page 3.
- Meziane, F., Vadera, S., Kobbacy, K., & Proudlove, N. (2000). Intelligent systems in manufacturing: current developments and future prospects. *Integrated Manufacturing Systems*, 11(4), 218–238. Cited in page 3.
- Moore, E. F. (1959). The shortest path through a maze. In *Proceedings of the International Symposium on Switching Theory, 1959* (pp. 285–292). Cited in page 23.
- Nunes, M. J. (2018). Estudo de desempenho do cluster tool - abordagem baseada na teoria de controle supervisorio, dissertação (mestrado), 2018. Programa de Pós - Graduação em Engenharia Elétrica - PPGE. Cited 3 times in the pages xi, 41, and 43.
- Oliveira, A. C., Costa, T. A., Pena, P. N., & Takahashi, R. H. (2013). Clonal selection algorithms for task scheduling in a flexible manufacturing cell with supervisory control. In *2013 IEEE Congress on Evolutionary Computation (CEC)* (pp. 982–988).: IEEE. Cited in page 5.
- Panek, S., Stursberg, O., & Engell, S. (2004). Job-shop scheduling by combining reachability analysis with linear programming. In *Proceedings of the 7th International Workshop on Discrete Event Systems* (pp. 199–204). Cited 2 times in the pages 3 and 4.
- Pena, P. N., Bravo, H. J., da Cunha, A. E., Malik, R., Lafortune, S., & Cury, J. E. (2014). Verification of the observer property in discrete event systems. *IEEE Transactions on Automatic Control*, 59(8), 2176–2181. Cited in page 10.
- Pena, P. N., Costa, T. A., Silva, R. S., & Takahashi, R. H. (2016). Control of flexible manufacturing systems under model uncertainty using supervisory control theory and evolutionary computation schedule synthesis. *Information Sciences*, 329, 491–502. Cited 5 times in the pages 3, 4, 5, 6, and 26.
- Pinedo, M. L. (2016). *Scheduling: theory, algorithms, and systems*. Springer. Cited 3 times in the pages 1, 2, and 4.
- Rafael, G. C. & Pena, P. N. (2018). Using an abstraction of the supervisor to solve a planning problem in manufacturing systems. *Anais do XXII Congresso Brasileiro de Automática, CBA*. Cited 3 times in the pages v, vi, and 50.

- Ramadge, P. J. & Wonham, W. M. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77(1), 81–98. Cited 3 times in the pages 2, 3, and 10.
- Shin, Y.-H., Lee, T.-E., Kim, J.-H., & Lee, H.-Y. (2001). Modeling and implementing a real-time scheduler for dual-armed cluster tools. *Computers in Industry*, 45(1), 13–27. Cited in page 17.
- Silva, R. S., Oliveira, A. C., Pena, P. N., & Takahashi, R. H. (2011). Algoritmo clonal para job shop scheduling com controle supervisorio. *X Simpósio Brasileiro de Automação Inteligente*, (pp. 1376–1381). Cited in page 5.
- Su, R. (2012). Abstraction-based synthesis of timed supervisors for time-weighted systems. *Proceedings of the International Federation of Automatic Control, IFAC, Volume*, 45(29), 128–134. Cited 2 times in the pages 1 and 4.
- Su, R., Van Schuppen, J. H., & Rooda, J. E. (2012). The synthesis of time optimal supervisors by using heaps-of-pieces. *IEEE Transactions on Automatic Control*, 57(1), 105–118. Cited in page 3.
- Su, R. & Woeginger, G. (2011). String execution time for finite languages: Max is easy, min is hard. *Automatica*, 47(10), 2326–2329. Cited in page 11.
- Uzsoy, R., Lee, C.-Y., & Martin-Vega, L. A. (1994). A review of production planning and scheduling models in the semiconductor industry part ii: Shop-floor control. *IIE Transactions*, 26(5), 44–55. Cited 2 times in the pages 17 and 39.
- Vilela, J. N. & Pena, P. N. (2016). Supervisor abstraction to deal with planning problems in manufacturing systems. In *13th International Workshop on Discrete Event Systems (WODES)* (pp. 117–122). Cited 6 times in the pages 2, 6, 12, 17, 26, and 49.
- Wagner, H. M. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2), 131–140. Cited in page 3.
- Wang, W., Yuan, C., & Liu, X. (2008). A fuzzy approach to multi-product mixed production job shop scheduling algorithm. In *Fifth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, volume 1 (pp. 95–99).: IEEE. Cited in page 1.
- Wong, K. (1998). On the complexity of projections of discrete-event systems. In *Proceedings of the International Workshop on Discrete Event Systems (WODES)* (pp. 201–206). Cited in page 9.
- Wonham, W. M. (2015). Supervisory control of discrete-event systems. *Encyclopedia of Systems and Control*, (pp. 1396–1404). Cited 2 times in the pages 12 and 23.