

ULRR

Simulation-optimization with machine learning applied to production planning and scheduling - case: semiconductor manufacturing

Item Type	Thesis
Authors	Ghasemi, Amir
Download date	2026-05-17 00:02:17
Item License	https://creativecommons.org/licenses/by-nc-sa/1.0/
Link to Item	https://hdl.handle.net/10344/10373

**Simulation-Optimization with
Machine Learning Applied to
Production Planning and
Scheduling - Case: Semiconductor
Manufacturing**

Amir Ghasemi

Supervised by Prof Cathal Heavey



UNIVERSITY of LIMERICK
OLLSCOIL LUIMNIGH

A thesis presented for the degree of Doctor of Philosophy

SCHOOL OF ENGINEERING

UNIVERSITY OF LIMERICK

Date

Dedication

To all my teachers who have taught me to learn...

To my father, mother, and sister for their love and endless support ...

To my wife, Sarah, who has been a constant source of support and encouragement during the challenges of life...

Declaration

I hereby declare that this thesis is entirely my own work and has not been submitted for any other awards at this or any other academic establishment. Where use has been made of the work of other people, it has been fully acknowledged and referenced.

Amir Ghasemi

12/1/2021

Acknowledgements

First, I would like to express my deepest thanks to my supervisor Cathal Heavey. I had a privilege to work with him during the last three years and was provided an invaluable experience under his supervision. I am deeply grateful for his immeasurable support, guidance, and encouragement in both academic and nonacademic affairs throughout my Ph.D. life as well as his substantial reviews and corrections on this dissertation.

My sincere thanks go to John Fowler and Yohanes Kristianto Nugroho for attending to my Ph.D. viva and acting as external and internal examiners for the defense of my dissertation.

I wish to thank to Georg Laipple and Oliver Schoenherr for allowing me to have on-site visits to an actual ASIC company and accesses to the wafer fabrication area.

I wish also to express my special thanks to Radhia Azzouz for her supervision and support to write my articles.

I would like to acknowledge to all my friends and all my colleagues in Enterprise Research Centre (ERC) and all the academic staff of the Department of Design and Manufacturing Technology at University of Limerick for their kind support, help and friendship.

Finally, I would like to express my gratitude to my family who were always in my corner when I needed them.

Contents

1	Introduction	1
1.1	Motivation	1
1.1.1	Simulation	6
1.1.2	Optimization	8
1.1.3	Machine Learning (ML)	8
1.1.4	Hybrid Methods	9
1.2	Problem Statement	10
1.3	State of the Art	11
1.3.1	Capacity Allocation in the Photolithography Area	11
1.3.2	Learning Based Evolutionary SO Method Applied to SJSSPs .	13
1.4	Thesis Objectives	15
1.5	Research Questions	16
1.6	Thesis Outline	17
1.6.1	Chapter 2 - Research Methodology	18
1.6.2	Chapter 3 - Optimizing Capacity Allocation in Semiconductor Manufacturing Photolithography Area - Case Study: Robert Bosch	18
1.6.3	Chapter 4 - Estimating The Scheduling Performance in Stochas- tic Job Shop Production Environments Using Genetic Programming- Based Metamodels - Case Study: Robert Bosch	18
1.6.4	Chapter 5 - Evolutionary Learning Based Simulation Opti- mization for Stochastic Job Shop Scheduling Problems	19

1.6.5	Chapter 6 - Conclusions and Recommendations for Future Work	19
2	Research Methodology	20
2.1	Introduction	20
2.2	Research Methods in PP&S	20
2.2.1	Survey Research	20
2.2.2	Case Research	21
2.2.3	Action Research	23
2.2.4	Quantitative Modeling and Simulation	25
2.3	The Research Methodology used in this Thesis	27
2.4	Conclusions	29
3	Optimizing Job Assignment in Semiconductor Manufacturing Photolithography Area – Case Study: Robert Bosch	31
3.1	Introduction	31
3.2	Literature Review	35
3.3	Problem Statement	39
3.3.1	CAPPA Complexity Proof	42
3.4	Improved Reference Group Genetic Algorithm (IRGGA)	44
3.4.1	Solution Structure and Greedy Heuristic-based Initialization	45
3.4.2	Crossover and Mutation operators using imitation and distinguish procedures	47
3.5	Experiments and Results	51
3.5.1	Parameters Design	51
3.5.2	Validation of GA Algorithms	52
3.5.3	Comparison of GA algorithms	53
3.6	Conclusions	63
4	Estimating Production Planning Performance in Jobshop Production Environments Under Uncertainty by Genetic Programming-	

Case Study: Robert Bosch	67
4.1 Introduction	67
4.2 Literature Review	72
4.3 Mathematical Model for Optimization of SJSSP	75
4.4 SJSSP DESM Metamodeling Using Genetic Programming (GP)	80
4.4.1 Queue Positions Based Learning Vector (QPBLV)	83
4.4.2 Machines Positions Based Learning Vector (MPBLV)	84
4.4.3 Machines Positions Interaction Based Learning Vector (MPI- BLV)	88
4.5 Data Preparation For Experimentation	90
4.5.1 Input Data for the SJJSP	90
4.5.2 GP calibration	93
4.5.3 Training and Evaluation Data Sets Design	94
4.6 Experiments	96
4.6.1 Evaluation of Training Vectors	98
4.6.2 Evaluation of Quality of Data Sets and Simulation Replications	103
4.7 Conclusions	105
5 Evolutionary Learning Based Simulation Optimization for Stochas- tic Job Shop Scheduling Problems	108
5.1 Introduction	108
5.2 Literature Review	111
5.2.1 Solutions of Stochastic Job Shop Scheduling Problems	113
5.3 Mathematical Model for Optimization of SJSSP	118
5.4 Evolutionary Learning Based Simulation Optimization (ELBSO) Method	124
5.4.1 ELBSO Solution Structure	125
5.4.2 GP Preparation: Metamodeling the SJSSP DESM using GP in ELBSO	126
5.4.3 Phase 1 of ELBSO algorithm	133
5.4.4 Phase 2 of ELBSO algorithm	135

5.5	GP Calibration	136
5.6	Experiment Results and Sensitivity Analysis	140
5.6.1	Comparison with ESOO Horng, Lin, and Yang (2012) and ESOOCBA Yang, Lv, et al. (2014)	140
5.6.2	Comparing ELBSO with GA, PSO, and FA proposed by Shen and Zhu (2016)	142
5.6.3	Comparative Analysis of ELBSO with ESOO Solution Algo- rithms and Dispatching Rules	144
5.6.4	Runtime Comparison of ELBSO and ESSO	149
5.7	Discussion	152
5.8	Conclusions	153
6	Conclusion	155
6.1	Introduction	155
6.2	Summary of Main Conclusions	155
6.3	Recommendations for Further Research	157
A	Glossary of Acronyms	183
B	Data sets to evaluate IRGGA	184
C	IRGGA MATLAB Codes	191
D	ELBSO MATLAB Codes	202

List of Figures

1.1	Bosch Supply Chain Network.	3
2.1	Main stages of confirmatory survey research (Forza 2002).	22
2.2	Overall framework for conducting case research (Voss, Tsikriktsis, and Frohlich 2002).	24
2.3	Cyclical four-step process of AR (Coughlan and Coughlan 2002).	25
2.4	Quantitative research cycle (Will M. Bertrand and Fransoo 2002).	27
2.5	Research methodology in this thesis.	28
2.6	Bosch data set sample.	29
2.7	Fitted distributions to processing times.	30
3.1	CAPPA example.	40
3.2	General concept of IRGGA	50
3.3	Taguchi test results.	52
3.4	Experiments design diagram.	55
3.5	Bosch data set processing time distribution.	55
3.6	Maximum load boxplots variation with M_{Rate} , P_{Rate} and R_{Rate} changes for problem 1.	63
3.7	Analysis of CAPPA sensitivity to P_{Rate} and M_{Rate}	64
3.8	Analysis of CAPPA sensitivity to P_{Rate} and R_{Rate}	64
4.1	The difference between typical evolutionary SOs and evolutionary SOs integrated with metamodels.	69
4.2	Proposed Job Shop Scheduling Problem Example.	76

4.3	Proposed GP metamodeling procedure general framework.	80
4.4	Data sets box plots.	93
4.5	Taguchi test results.	94
4.6	EIP to obtain SJSSP data sets.	96
4.7	The general SJSSP metamodeling accuracy analysis structure.	97
4.8	Data sets box plots of DS1, DS2, DS3, and DS4.	99
4.9	GP training by MPIBLV line chart.	102
4.10	Data sets box plots for MPIBLV.	104
4.11	<i>SDF</i> line charts based on number of simulation replications and the different quality of input data.	106
5.1	Proposed job shop scheduling problem example.	121
5.2	The general framework of ELBSO method.	125
5.3	Solution structure in ELBSO.	126
5.4	Tree-based encoding of a function, e.g., $(\sin(x_0) \times \log(x_1)) + x_2^2$, in GP for symbolic regression illustrating functional primitives and terminal set (Can and Heavey 2011).	128
5.5	Proposed GP metamodeling procedure general framework.	129
5.6	Crossover in ELBSO.	135
5.7	Mutation in ELBSO.	135
5.8	Taguchi test results.	138
5.9	GP line chart result.	139
5.10	10000 seconds termination criteria.	152
B.1	CAPPA machines input, where in each row the capability of each machine in running a certain process (cap1, cap2, etc.) is showed by 1, and 0 otherwise.	184

B.2	CAPPA inputs sample 1 showing the processing time (Time) of each layer of orders planned to be produced in a certain week (Week) and needs certain reticle (Reticle) and process capability (Capability), while it is critical (Critical=1) or not (Critical=0).	185
B.3	CAPPA inputs sample 2.	186
B.4	CAPPA input sample 3.	187
B.5	CAPPA inputs sample 4.	188
B.6	CAPPA inputs sample 5.	189
B.7	CAPPA inputs sample 6.	190
C.1	IRGGA (assigning parameters 1).	191
C.2	IRGGA main code (assigning parameters 2).	192
C.3	IRGGA main code (assigning parameters 3).	192
C.4	IRGGA main code (initial population 1).	193
C.5	IRGGA main code (initial population 2).	194
C.6	IRGGA main code (initial population 3).	195
C.7	IRGGA main code (initial population 4).	196
C.8	IRGGA main code (main loop 1).	197
C.9	IRGGA main code (main loop 2).	198
C.10	IRGGA main code (main loop 3).	199
C.11	IRGGA main code (main loop 4).	200
C.12	IRGGA Fitness Calculation Code.	201
C.13	IRGGA Recombination Code.	201
D.1	ELBSO main code (parameters assignment 1).	202
D.2	ELBSO Main Code (initial population 1).	203
D.3	ELBSO Main Code (initial population 2).	204
D.4	ELBSO Main Code (neighborhood search 1).	204
D.5	ELBSO Main Code (neighborhood search 2).	205
D.6	ELBSO Main Code (simulation replications).	205

D.7 Simulation Function in ELBSO. 206

D.8 Simulation Function in ELBSO 2. 207

D.9 Training Function in ELBSO. 208

List of Tables

1	List of publications.	xv
3.1	Summary of publications on CAPPa with MC=Machine process capability constraints; MD=Machine dedication constraints; R=Reticule constraints; EX=Exact optimization; SH=Simple Heuristic; LS=Local-search; CS=Constraint Satisfaction; S=Simulation; P=(Maximize) profit; C=(Minimize) cost; LL= (Minimize) load levelling; TH=(Maximize) throughput; CT=(Minimize) cycle time.	39
3.2	Notations table.	41
3.3	Chromosome structure	45
3.4	Taguchi test parameters.	52
3.5	IRGGA and CPLEX Comparison on small size CAPPa instances (CPU.T represents CPU Time).	53
3.6	Test problem instances under Number of Machines ED (Table 3.7).	54
3.7	Empirical distribution (ED) for number of machines.	54
3.8	Comparative experiments results	59
3.9	<i>t</i> -test results	61
4.1	Notations table.	77
4.2	Operations flow matrix.	92
4.3	Jobs due dates.	92
4.4	Taguchi test parameters.	94
4.5	Training Data Sets Statistical Analysis.	98
4.6	GP training vectors comparison.	102

4.7	Training data sets statistical analysis for MPIBLV.	104
4.8	Algorithms comparison.	105
5.1	The relevant published journal papers of job shop problem under uncertainty; Notations: MS=(Minimize) Make-Span, TE=(Minimize) Tardiness and/or Earliness, TC=(Minimize) Total Cost, C-FT=(Minimize) Completion-Flow Time, TR=(Maximize) Total Revenue, RB=Robust, ST-P=Stochastic-Probabilistic, FU=Fuzzy, UP=Uncertain Parameter, DR=Dispatching Rules, HE=Heuristic, MH=Meta Heuristic, SI=Simulation, MM=Metamodel.	118
5.2	Notations table.	120
5.3	Taguchi test parameters.	138
5.4	The test problem environment presented by Horng, Lin, and Yang (2012).	141
5.5	Job due dates in the test problem presented by Horng, Lin, and Yang (2012).	141
5.6	Algorithms comparison.	142
5.7	Test problem environment presented by Shen and Zhu (2016).	143
5.8	Job due dates, CE , and CT values in test problem presented by Shen and Zhu (2016).	143
5.9	Jobs due dates, CE , and CT values in test problem 1 presented by Shen and Zhu (2016)	143
5.10	Operations flow and their processing times in test problems set 2.	145
5.11	Jobs due dates in test problems set 2.	145
5.12	Test problems set 2 instances.	145
5.13	Test problem set 2 results	147
5.14	t -test results	148
5.15	Algorithms TI analysis.	151
A.1	List of Acronyms.	183

List of Publications to Date

In Table 1, the extracted publications from this thesis are listed. We note also the state of the paper in case it is not currently published and how information from it feeds the thesis.

Table 1: List of publications.

Publication	Notes
Ghasemi, Amir, Cathal Heavey, and Georg Laipple (2018). “A Review of Simulation-optimization Methods with Applications to Semiconductor Operational Problems”. Winter Simulation Conference.	Part of Chapters 4 and 5.
Ghasemi, Amir, Cathal Heavey, and Kamil Erkan Kabak (2018). “Implementing a New Genetic Algorithm to Solve the Capacity Allocation Problem in the Photolithography Area”. Winter Simulation Conference.	Part of Chapter 3.
Ghasemi, Amir, Radhia Azzouz, Kamil Erkan Kabak, and Cathal Heavey (2020). “Optimizing capacity allocation in semiconductor manufacturing photolithography area – Case study: RobertBosch”. Journal of Manufacturing Systems 54, page 123–137.	Presented in Chapter 3.
Ghasemi, Amir, Amir Ashuri, and Cathal Heavey (2021). “Evolutionary Learning Based Simulation Optimization for Stochastic Job Shop Scheduling Problems”. Applied Soft Computing	Accepted and Under Revision. Presented in Chapter 5.
Ghasemi, Amir, and Cathal Heavey (2021). “Estimating The Scheduling Performance in Stochastic Job Shop Production Environments Using Genetic Programming-Based Metamodels - Case Study: Robert Bosch”. Computers and Operations Research	Under Revision. Presented in Chapter 4.

Simulation-Optimization with Machine Learning Applied to Production Planning and Scheduling - Case: Semiconductor Manufacturing

Amir Ghasemi

Abstract

Industry 4.0, which may eventually represent a fourth industrial revolution, is a complex technological system that has been widely discussed and researched, having a great influence in industrial, since it introduces relevant advancements that are related with smart and future factories. Production Planning and Scheduling (PP&S) paradigms within industries are one of the main sectors influenced by Industry 4.0 advancements. Undoubtedly, capacity allocation and production scheduling are important aspects in PP&S to be studied by researchers. From a globalized perspective, semiconductor manufacturing is one of the main contributors to support the Industry 4.0 era. Thus, in the first phase of this research, a new Mixed Integer Linear Programming (MILP) model for a Capacity Allocation Problem in a Photolithography Area (CAPPA) is proposed. To solve CAPPA, an improved Genetic Algorithm (GA) named Improved Reference Group GA (IRGGA) is used to solve CAPPA efficiently by improving the generation of the initial population. In the second phase, a novel metamodeling approach is proposed to metamodel a Discrete Event Simulation Model (DESM) of a Stochastic Job Shop Scheduling Problem (SJSSP). Finally, the designed metamodeling approach is integrated within an evolutionary Simulation Optimization (SO) method called an Evolutionary Learning Based Simulation Optimization (ELBSO) algorithm. Using a comprehensive experimental analysis this algorithm is examined and proof of its superiority is provided.

Chapter 1

Introduction

1.1 Motivation

Continual attempts of researchers and industry engineers to invent new technologies, to promote the efficiency level of existing systems, and to plan for sustainable development have fostered four industrial revolutions in the last two hundred years. The First Industrial Revolution took hold in England in the middle of the 18th century and was potentiated by the invention of the steam engine. During the second half of the 19th century, the Second Industrial Revolution was developed in Europe and USA. This revolution was characterized by mass production and the replacement of steam by chemical and electrical energy. To meet the growing demand, several technologies in industry and mechanization have been developed, such as the assembly line with automatic operations, allowing increases in productivity. The invention of the Integrated Circuit (microchip) was the technological advancement that has triggered the Third Industrial Revolution. The use of electronics and Information Technology (IT) to achieve further automation in production is the key feature of this revolution that emerged in the last years of the 20th century in many industrialized countries around the world. In the last years, with the growing advancements in manufacturing processes and technology, many new global concepts have emerged (Schmidt et al. 2015).

The term “Industry 4.0” has become an increasingly important topic in the last

few years. This concept appeared firstly in an article published in November 2011 by the German government that resulted from an initiative regarding high-tech strategy for 2020 (Zhou, Taigang Liu, and Lifeng Zhou 2015). The global industrial landscape has changed deeply in the last years and is a result of successive technological developments and innovations. Industry 4.0 can be tentatively compared with three industrial revolutions; that represent the main disruptive changes in manufacturing that have resulted from several technological advances. Industry 4.0, which may eventually represent a fourth industrial revolution, is a complex technological system that has been widely discussed and researched, having a great influence in the industrial sector, since it introduces relevant advancements that are related with smart and future factories. This emerging Industry 4.0 concept is an umbrella term for a new industrial paradigm that embraces a set of future industrial developments regarding Cyber-Physical Systems (CPS), Internet of Things (IoT), Internet of Services (IoS), Robotics, Big Data, Cloud Manufacturing and Augmented Reality. Together, these generate and leverage on the concept of ‘smart factories’ to comprise the next industrial revolution in manufacturing, characterised by increased flexibility, productivity, efficiency, and sustainability, ultimately ensuring competitiveness in the global market (Weyer et al. 2015). Bringing together many of these advances, the fourth industrial revolution threatens to radically change the traditional Production Planning and Scheduling (PP&S) paradigm within industrial sectors (Dolgui et al. 2019). Thus, it is necessary to advance existing Decision Support Tools (DSTs) and design new DSTs used by PP&S sectors within industries to comply with such multidirectional changes.

As mentioned, the invention of the Integrated Circuit triggered the third industrial revolution. Undoubtedly, semiconductor manufacturing is again one of the flagship industries initiating Industry 4.0. The semiconductor manufacturing industry is the fastest evolving and most highly competitive industry in the world. According to Moore’s law, the number of transistors on integrated circuits doubles approximately every 2 years, and consequently new technologies appear (Moore

1998).

The research presented in this thesis has been funded by the Productive4.0 project (*Productive4.0 - A European co-funded innovation and lighthouse project on Digital Industry 2021*). The European publicly funded project, Productive4.0, is carried out under Horizon2020, within the Electronic Components and Systems for European Leadership (ECSEL), which was established in 2016. The Productive4.0 project is coordinated by Infineon, Germany. It involves 109 partners from science and industry with several research objectives to be achieved. One of those objectives is the development of a hierarchical master planning system for supply chains with a simulation based hierarchical supply chain optimization system to optimize the production scenarios of the master planning system. Since the description of the physical system structure is based on a real system, which is then generalized, most of the data analysis work was carried out using the example of the BOSCH semiconductor plant in Reutlingen, Germany.

The structure of the Bosch physical supply chain can in general be characterized as a hierarchical system (see Figure 1.1). Within this system, there are three levels: the top level, the macro level, and the micro level, which are described below.

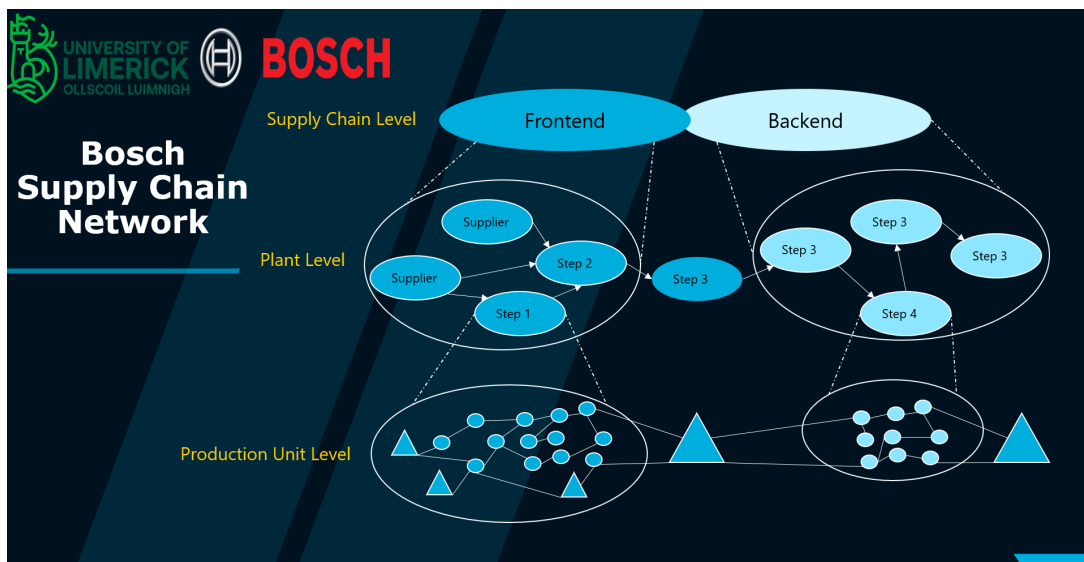


Figure 1.1: Bosch Supply Chain Network.

At the top level, also called Supply Chain Level (top row in Figure 1.1), an aggregated perspective is taken, whereby the node characteristics result from the

aggregation of the characteristics of the associated child nodes on the so-called Plant Level (middle row in Figure 1.1). Thereby, it will be an important task to aggregate data and characteristics from the lower levels of the supply chain to a higher level. In most cases, the nodes on this level represent one physical production location, but one location could also be split into different supply chain level nodes if necessary for a more suitable description of the system. On the top level, the semiconductor manufacturing supply chain can be divided into two parts separated by a goods warehouse: Front-end and Back-end. The Front-end contains all the semiconductor production steps on a wafer, while the Back-end contains all cutting, measurement, and assembly steps to build a microelectronic component. The warehouse in between exists to separate the customer demands from the wafer production, which is necessary due to the different time horizons of customer orders and production cycle times.

The macro level is the Plant Level. Within one plant, there usually is more than one production unit. Most of the time, the production steps within one plant are clustered either functionally or in segments along the product stream. The different production units are characterized by independent shop floors and shifts. Often, different production units are locally separated into different buildings. Thereby, the different production units are often planned and controlled separately, and the interaction and information exchange processes are not continuous. One of the positive side effects of this project will be an increase in the interaction and information exchange between different entities on this level. The micro level is the Production Unit Level. Within one production unit, a large number of different equipment types (machine groups) work together in a work flow. One equipment group may contain an arbitrary number of identical equipment.

The semiconductor industry is characterized by an increasing complexity of manufacturing processes in terms of elementary operations as well as of the number of constraints, finer geometries to realize on chips, increasing degrees of automation combined with equipment and infrastructure costs, high renewal rates of technolo-

gies that lead to a rapid obsolescence of products, an ever-increasing pressure on the cost of wafers due to worldwide competition, and high customer requirements in terms of quality (Mönch, Uzsoy, and Fowler 2018). Within production units, photolithography, requiring high capital investments, is one of the most crucial processes in semiconductor manufacturing. It is mostly regarded as a bottleneck process due to the layered nature of wafer fabrication, especially for the case of Application Specific Integrated Circuit (ASIC) fabrication environments with high mix product portfolio and low volume. Essentially, the photolithography process includes three main steps. These steps are coat, expose, and develop. First, the wafer is coated with a thin film of a photosensitive polymer, called the photoresist strip. Then, in the “expose” step, the wafer is exposed with ultraviolet light (UV) in order to print the circuit pattern onto the wafer. This is done using a reticle, which is a chrome patterned glass that defines the circuit pattern. This pattern tends to be unique for each layer. In an ASIC fab, a diverse range of recipes exist due the range of products produced. Finally, polymerized sections of photoresist are removed from the exposed wafer. Since the circuits are made up of layers, with every wafer passing through the photolithography area up to 40 times, this typically makes photolithography a bottleneck resource. With the performance of a system determined by the bottleneck resource, optimal capacity allocation and job scheduling of a photolithography work area ensures improvement in the performance of the whole fab (Ghasemi, Azzouz, et al. 2020). It is worth mentioning that, similar to many production environments, the photolithography workstation can be viewed as a job shop production environment. Indeed, job shop scheduling problems are one of the most challenging problems in most production industries (Zhang, Ding, et al. 2019). Thus, the prime goal of this research is to develop DSTs supporting PP&S paradigms within photolithography workstations, which results in the efficiency improvement of the micro supply chain level.

In fact, PP&S is a decision-making process in manufacturing and service sectors that deals with the allocation of capacities (e.g., human, machine, money) to tasks

in a certain sequence and over given periods of time. This process is complex, and it aims to optimise tactical and operational activities by leveraging on available production data, which may also include previous planning results (i.e., scheduling and rescheduling) (Pinedo 2016). Since usually, from the time frame perspective, the capacity allocation is a mid-term problem, and the scheduling problem is a short-term one, they are mainly categorized in tactical and operational decision making problems, respectively (Mönch, Uzsoy, and Fowler 2018). In this thesis, we considered both capacity allocation and scheduling problems within a photolithography workstation in semiconductor manufacturing systems.

Due to the extent of scheduling problems, there are several classifications of scheduling problems in the literature. Often, scheduling problems are distinguishable based on three involving factors: machine environment, job characteristics, and objective function(s) to be minimized. Machine environment covers single machine, parallel machines, job shop, flow shop, open shop, etc. Job characteristics consist of the presence of preemption (resume or repeat), precedence constraints between jobs, presence of release dates, presence of deadlines, batching problems, sequence-dependent setup times, etc. Moreover, two types of objective function are the most common: bottleneck objective functions (e.g., Makespan) and sum objective functions (e.g., Tardiness), (Pinedo 2016). To solve capacity allocation and scheduling problems within industries, a variety of approaches mainly based on Simulation, Optimization, and Machine Learning (ML) concepts have been proposed by both researchers and industry engineers. In the following, these concepts are defined one by one.

1.1.1 Simulation

Computer simulation provides a modeling and evaluation tool for complex systems that are analytically intractable. Since its inception in the 1950s, simulation has been successfully employed to improve the design of complex systems. Traditionally, simulations have been used to evaluate system designs to determine if they meet

various operational objectives. In doing so, simulations save time and resources by establishing a proof of concept prior to building a physical system for prototyping. A natural extension is then to use the simulation model to evaluate several alternative system designs, to select the design that has the best performance according to the simulation model. This use of simulations reflects both their strengths and their limitations.

On the one hand, unlike analytical models that often require overly simplistic (and in practice, hard to accept) assumptions for tractability, simulation models capably leverage specificity to improve fidelity to the system being modeled. Therefore, simulation model outputs can provide reliable estimates of systems' performance for decision makers to consider before a physical system is built. On the other hand, the accuracy of the simulation results comes with two prerequisites. First, simulation models often need to capture important system processes and operations with incredible detail. As such, simulation models are often highly complex and computationally expensive to run. The computational cost is further increased when multiple replications of simulations are required to control the noise in stochastic simulations. Second, there must be a sufficient amount of high-quality data accessible to the simulation modelers so they can estimate probability distribution models for various sources of randomness and uncertainty in the system (Xu et al. 2016).

There are three main modeling approaches in the context of simulation model design: Discrete Event Simulation (DES) modeling, System Dynamics modeling, Agent Based modeling, and Multi-method modeling. DES Models (DESMs) are event-driven dynamical systems (i.e., the state transitions are initiated by events, rather than a clock). In the last couple of decades, there has been an increase in the research on DESMs applied to PP&S problems. System dynamics is a highly abstract method of modeling. It ignores the fine details of a system, such as the individual properties of people, products, or events, and produces a general representation of a complex system. These abstract simulation models may be used for long-term strategic modeling and simulation. Agent based modeling focuses on

the individual active components of a system. With Agent based modeling, active entities, known as agents, must be identified and their behavior defined. They can be vehicles, equipment, products, or even companies.

1.1.2 Optimization

Optimization is the process of choosing the best tradeoffs between different factors in a system to achieve desirable outcomes. The notion of different factors means that there are different solutions, and the notion of achieving desirable outcomes declares that there is an objective of seeking improvements on how to find the best solution. Optimization methods have been proposed for a wide range of production-related problems since the 1950s, addressing problems of long-term aggregate production planning, mid-term capacity allocation to different products, lot sizing, and product cycling, and detailed short-term production scheduling (Missbauer and Uzsoy 2011). There is a variety of optimization techniques applied to PP&S problems, which include global methods (e.g., Mixed Integer Programming (MIP) and Branch and Bound (B&B)), local methods (e.g., Heuristics and Metaheuristics), and hybrid methods (e.g., Hybrid Metaheuristic-Exact optimizers). Global methods are mainly based on mathematical optimization and usually are not applicable to large and real size complex problems as they seek for the global optimum, which might take a huge amount of computation time for such problems. In contrast, local methods solve problems in a reasonable time while the solution is not necessarily the global optimum. Moreover, hybrid methods combine features from both optimization classes mentioned.

1.1.3 Machine Learning (ML)

In general, learning can be defined as the process of improving one's ability to do a task, processes of inferencing and memorising, where inference is deemed to be any kind of reasoning or knowledge transformation. That is, inference results in the production of new knowledge. Functionally speaking, learning will often consist of

filtering data to obtain interesting information and then refining this in some way so that useful facets remain. Thus, data is distilled into knowledge, whence it is stored for future use (Dutton and Conroy 1997). This could be defined as the philosophy of ML. In other words, the goal of ML is to program computers to use example data or experience to solve a given problem. In fact, together with the rising popularity of ML research and the growth of the available data amounts, the applications of the developed methods have found their way into various industrial fields. Within the context of PP&S, ML has been applied in various areas. Neural Networks (NNs), Genetic Programming (GP), and Gaussian process regression are the main ML techniques applied to PP&S problems (Takeda-Berger et al. 2020). The main usages of these techniques consist of switching dispatching rules in dynamic PP&S environments (Mouelhi-Chibani and Pierreval 2010), estimating objective functions (Azadeh, Moghaddam, et al. 2010), metamodeling complex simulation models (Can and Heavey 2011), and training agents for various usages (Akyol and Bayhan 2007).

1.1.4 Hybrid Methods

In the era of Industry 4.0, considering emerging complex PP&S problems within industries, hybrid methods have had the best performance to be used as DSTs (Shahzad and Mebariki 2012). In other words, hybrid methods combining Simulation, Optimization, and ML concepts within their architectures have been of interest among both researchers and industry practitioners. Accordingly, hybrid Simulation Optimization (SO) methods have been one of the most promising techniques to solve complex real PP&S problems (Ghasemi, Heavey, and Laipple 2018). Simulation models have been used as one of the DSTs to analyze real industrial systems by considering stochastic parameters and/or variables of different systems. On the other hand, optimization techniques are key tools to improve decisions within almost all systems. Integrating simulation models with optimization methods could establish promising decision support tools benefiting from the advantages of both tools. That is the main idea to support proposing SOs for different complex and stochas-

tic industrial problems. However, simulation models of such problems are highly time-intensive, causing SO implementations to be infeasible for real and large-sized industrial problems. Thus, accurate and fast simulation metamodel implementations are necessary to be integrated within SOs (Can and Heavey 2012).

1.2 Problem Statement

As discussed above, there is a significant need for efficient methods to support the decision making process for PP&S sectors within semiconductor industries in the era of Industry 4.0. Moreover, the photolithography area is a well-known bottleneck workstation in semiconductor manufacturing, and the photolithography workstation can be viewed as a job shop production environment. On the other hand, simulation, optimization, and ML techniques are the main tools to design efficient DSTs for PP&S paradigms within manufacturing systems. Thus, the prime goal of this thesis is to design efficient simulation, optimization, and ML-based DSTs to tackle PP&S problems in manufacturing systems, specifically, photolithography workstations in semiconductor manufacturing. Besides, SO methods are one of the most promising techniques to design DSTs for complex and real PP&S problems, while integrating them with simulation metamodels could strengthen them by reducing their computation time intensity. Thus, this thesis presents:

- A Mixed Integer Linear Programming (MILP) model for Capacity Allocation Problem in Photolithography Area (CAPPA) in semiconductor manufacturing;
- A novel Genetic Algorithm (GA) to solve CAPPA;
- Application of the proposed capacity allocation approach to the data sets captured from a Bosch semiconductor front-end fab;
- A mathematical model for Stochastic Job Shop Scheduling Problems (SJSSPs) considering production time uncertainties;

- A Discrete Event Simulation Model (DESM) of SJSSP.
- A new implementation of GP in metamodeling SJSSP DESMs to be used in SO techniques;
- An evaluation of three training vectors for metamodeling GP from DESMs of SJSSPs;
- A comprehensive sensitivity analysis of the accuracy of GP to metamodel SJSSPs within SOs based on Bosch data sets;
- A new Evolutionary Learning Based Simulation Optimization (ELBSO) technique to solve SJSSPs.

1.3 State of the Art

The state of the art of this research is categorised into two main domains: capacity allocation in the photolithography area and learning based evolutionary SO methods applied to SJSSPs.

1.3.1 Capacity Allocation in the Photolithography Area

A large amount of research has been carried out on the operations of semiconductor manufacturing (Mönch, Uzsoy, and Fowler 2018; Mönch, Fowler, and Mason 2013; Uzsoy, Lee, and Martin-Vega 1992). The photolithography process is considered a key process in wafer fabrication due to the complex technology used, the critical dimensions involved, and re-entrant flows, and thus this is considered the bottleneck in most fabrication lines (Ghasemi, Heavey, and Kabak 2018). The notion of a bottleneck is important in many planning methods. A bottleneck is a group of similar machines that limits the production rate. As a result of this, the importance of an efficient machine capacity plan for this tool is required (Mönch, Fowler, and Mason 2013). Capacity planning problems appear in many forms and have attracted

thousands of research papers. To structure this large body of research, comprehensive literature reviews by Martínez-Costa et al. (2014) and Wu, Erkoç, and Karabuk (2005) have been published. As stated by Martínez-Costa et al. (2014), deterministic models have got most of the attention in solving capacity allocation models.

There are three main special constraints effecting capacity allocation in the photolithography area: machine process windows, machine dedication, and reticle sharing constraints (Ghasemi, Heavey, and Kabak 2018). Machine process window constraints define that each operation requires a certain machine(s) to be produced. Machine dedication refers to critical operations which must be assigned to a certain machine to insure quality. Reticle sharing constraint considers the total number of sharing of masks to produce wafers. It is crucial to consider all these constraints together to obtain an accurate capacity allocation plan. In the literature, all these constraints are rarely considered in an integrated manner (Ghasemi, Azzouz, et al. 2020). On the other hand, CAPP is a NP-Hard problem. Thus, heuristics and metaheuristics are the most widely applied methods to solve CAPP in the literature (Chen, Chen, and Liang 2016).

This thesis extends the existing literature on CAPP by:

- Proposing a MILP mathematical model for CAPP considering all critical constraints;
- Proofing the complexity of CAPP mathematically;
- Presenting a novel GA named an Improved Reference Group Genetic Algorithm (IRGGA);
- Implementing the proposed method on Bosch photolithography data sets;
- Providing a comprehensive experimental analysis to prove the effectiveness of proposed methods.

The detailed literature review on CAPP and the existing methods to solve it are provided in Section 3.2 on page 35.

1.3.2 Learning Based Evolutionary SO Method Applied to SJSSPs

Over recent years, a large body of research has been published on Job Shop Scheduling Problems (JSSP), which are one of the basic models used in manufacturing. JSSP is one of the famous mathematical optimization problems that has been proved to be NP-hard (Horng, Lin, and Yang 2012). Although stochasticity has been known as a crucial part of most industrial operations in the scheduling literature, there is little attention to solving SJSSPs, while Deterministic Job Shop Scheduling Problems (DJSSPs) have been widely researched (Winands, Adan, and Houtum 2011).

Recent advances in SO research and the explosive growth in computing power have made it possible to optimize complex manufacturing system problems. In fact, DESMs are important tools used as a predictor of performance, allowing examination of the likely behavior of a proposed manufacturing system under experimental conditions (Trigueiro de Sousa Junior et al. 2019). While DESMs do not directly provide explanations for the observed system behavior, it is essentially a trial and error methodology, and although attention to experimental design techniques enhances its value, it does not provide a method of optimization. On the other hand, optimization techniques are key tools to improve decisions within almost all systems.

Integrating simulation models with optimization methods could establish promising DSTs benefiting from the advantages of both tools. Thus, SO techniques have been known as one of the most promising techniques to tackle large and stochastic real production problems such as SJSSPs (Ghasemi, Heavey, and Laipple 2018). Researchers applied SOs to SJSSPs to allow an optimizer(s) to seek better solutions when integrated with DESMs (Figueira and Almada-Lobo 2014).

However, the objective calculated using simulation replications has a high computation cost. Therefore, most researchers have applied three main techniques to replace the high number of simulation replications. One approach used by several researchers, such as Horng, Lin, and Yang (2012) and Yang, Lv, et al. (2014), and Akker, Blokland, and Hoogeveen (2013), is to use a small number of simulation repli-

cations executed during the search phase. For instance, although it is reported by Horng, Lin, and Yang (2012) that 10^5 simulation replications are enough to ensure the accuracy of the objective values, they performed 368 simulation replications to calculate the objective values of the proposed SJSSP (in the exploration phase, i.e., phase 1 of Ordinal Optimization (OO)). A second approach is where researchers such as Shen and Zhu (2016) and Jamili (2019), convert SJSSP mathematical models to deterministic ones where a level of robustness and/or confidence is achieved when optimized. Typically, evolutionary methods are proposed to solve the deterministic models. A third approach is where metamodels are used. This was reported in Horng, Yang, and Lin (2012) where an Radial Basis Functions (RBF) metamodel was used in phase 1 of OO for the optimization of a hotel booking limit problem. The third approach is the most advantageous one as other methods ignore a series of stochastic scenarios while they do not provide information on the ignored scenarios. However, due to the highly complex nature of SJSSPs, there is no metamodeling method applied to SJSSPs within the literature.

To metamodel DESMs, researchers have presented a variety of methods and concepts, such as RBF (Hussain, Barton, and Joshi 2002), Kriging (KG) metamodeling (Kleijnen 2009), while Artificial Neural Network (ANN) has been known as the predominant approach to metamodel DESMs (Dunke and Nickel 2020). Can and Heavey (2012) compared both GP and ANN in metamodeling DESMs. For the industrial case studies considered, the results showed that GP outperforms ANN in metamodeling DESMs. Surprisingly, there is no research in the literature implementing GP-based metamodels to SJSSP DESMs.

This thesis extends the existing literature on methods applied to SJSSPs by:

- Proposing a new GP-based SJSSP DESM metamodeling procedure consisting of three novel training vectors;
- Training the proposed metamodel with various data sets from a Bosch photolithography workstation;

- Designing and implementing a new SO method called ELBSO that integrates an evolutionary SO algorithm with a GP-based metamodels;
- Implementing the proposed approach to the Bosch case study data;
- Providing a comprehensive experimental analysis to proof the effectiveness of proposed methods.

The detailed literature review on SO and metamodeling techniques applied to SJSSPs are provided in Sections 4.2 on 72 and 5.2 on 111.

1.4 Thesis Objectives

From the above brief literature review, the existing research gaps and expected contributions in this thesis are defined. In this regard, the objectives of this thesis are:

- To propose simulation, optimization, and ML-based tools supporting PP&S paradigms within semiconductor photolithography toolsets.
- To perform an optimization-based offline capacity allocation tool for the photolithography workstation in semiconductor manufacturing;
- To design an accurate metamodeling technique to replace highly time-intensive simulation replications within evolutionary SO methods in solving SJSSPs;
- To establish an evolutionary SO framework for solving SJSSPs in a short period of time, that depending on the system, could be used in real time;
- To implement the proposed DSTs on real case data obtained from a Bosch front-end fab;
- To propose a comprehensive verification and validation methodology for the presented novel approaches.

1.5 Research Questions

Designing DSTs for PP&S for the case of semiconductor manufacturing (as discussed above) and combining this with the thesis objectives raises several research questions. These research questions help to structure the focus areas of the research.

Q1: How to design an optimization-based framework to solve CAPPAs accurately while considering all critical constraints?

Capacity allocation is one of the most critical planning problems within bottleneck workstations of almost all industries. As photolithography is a well-known semiconductor front-end fab's bottleneck, optimizing CAPPAs could improve the utilization rate of semiconductor manufacturing systems. There are a couple of research articles addressing CAPPAs within the literature. However, they rarely have considered all critical constraints together. For instance, both machine process windows and machine dedication constraints were addressed by (Chung, Huang, and Lee 2006) and (Chung, Huang, and Lee 2008). In the former study, Chung, Huang, and Lee (2006) presented a mixed-integer programming (MIP) model in order to level or balance the load over machines based on the average utilization rates. A demonstrative example and a real-world application were given in this study. In the other study, Chung, Huang, and Lee (2008) referred to the capacity allocation problem as CAPPAs and they proposed heuristics to solve it. Clearly, considering all three main critical constraints in an integrated manner is missing from the literature of CAPPAs (Ghasemi, Azzouz, et al. 2020). Thus, in this thesis answering the following questions are of interest: How to design an optimization model solving CAPPAs while considering all three critical constraints together?; How to tackle the considered CAPPAs accurately?; and finally, How to verify the proposed optimization approach?

Q2: What are the influencing factors in obtaining near optimal solutions in CAPPAs problems?

From the practical point of view, it is important to understand influences of each critical constraint on the capacity allocation plan. Thus, in this thesis, using a

sensitivity analysis framework, the impact of each critical factor for obtaining near optimal solutions for CAPPA problems is examined.

Q3: How to design and train a metamodel to replace a DESM in a SJSSP?

Although SO techniques provide promising solutions for large and complex stochastic problems, simulation model execution is potentially expensive in terms of computation time. Moreover, Can and Heavey (2012) showed the effectiveness of GP to metamodel DESM production models.. Thereby, in this thesis, we attempted to replace simulation replications with a GP-based metamodel in the case of SJSSP. This provides the opportunity to implement evolutionary SO methods for complex optimization problems without the requirement of extensive computer resources for simulation replications. Moreover, this replacement enables SO methods to allocate more computation budgets on strengthening the optimization core rather than on simulation replications. However, designing, training, and implementation of such a metamodel is not trivial. Thus, in this thesis, the procedure of designing a metamodel which is accurate enough to replace simulation replications within evolutionary SO methods to solve SJSSPs is investigated.

Q4: How to design and implement evolutionary SO integrated with metamodels to solve SJSSPs?

After constructing the metamodel, in line with the prime goal of this thesis (developing an efficient evolutionary SO method to solve SJSSPs), the framework to design a new evolutionary SO integrated with metamodels is investigated.

1.6 Thesis Outline

Therefore, this chapter highlights motivations, thesis objectives, and research questions. In this section, each chapter is briefly summarized to give a synopsis of the thesis.

1.6.1 Chapter 2 - Research Methodology

This chapter aims to detail methodologies used in this thesis.

1.6.2 Chapter 3 - Optimizing Capacity Allocation in Semiconductor Manufacturing Photolithography Area - Case Study: Robert Bosch

In this chapter, firstly, a literature review highlighting research trends and gaps in modeling and solving CAPP is provided. Next, an MILP model for CAPP is proposed. Then, after proving the complexity of CAPP mathematically, a new GA named IRGGA is proposed to solve CAPP. Finally, by implementing IRGGA to Bosch photolithography data sets, its performance is compared with a set of heuristics and metaheuristics.

1.6.3 Chapter 4 - Estimating The Scheduling Performance in Stochastic Job Shop Production Environments Using Genetic Programming-Based Metamodels - Case Study: Robert Bosch

In this chapter, firstly, a literature review highlighting research trends and gaps in metamodeling DESMs in the case of SJSSP is provided. Next, the training of metamodels to replace simulation replications is evaluated, with a particular focus on SJSSP and GP. Then, three different training vectors inspired by special features of SJSSP for the generation of GP metamodels are evaluated. Finally, an evaluation is carried out using realistic industrial data from a Bosch fab photolithography toolset, with the data used within an evolutionary optimization algorithm, similar to how the metamodel would be used within SO.

1.6.4 Chapter 5 - Evolutionary Learning Based Simulation Optimization for Stochastic Job Shop Scheduling Problems

In this chapter, firstly, a literature review highlighting research trends and gaps in implementing SOs to SJSSPs is provided. Then, a new SO method is proposed, denoted ELBSO, that consists of a learning procedure using GP and an evolutionary optimization structure embedded in OO to solve the SJSSP. To solve the proposed problem, a SO mathematical definition for SJSSP is developed. Next, a novel GP training strategy to learn from SJSSP is developed to metamodel the SJSSP DESM. Then, a two-phase ELBSO algorithm is presented which is an evolutionary SO based method with reduced computations due to the use of GP in estimating the objective values of solutions. Finally, using sets of standard SJSSP experiments, the developed ELBSO is compared with several published results in the literature.

1.6.5 Chapter 6 - Conclusions and Recommendations for Future Work

At the end of the thesis, a summary of generalized conclusions are presented as well as discussions and considerations for further research.

Chapter 2

Research Methodology

2.1 Introduction

As mentioned in Chapter 1, simulation, optimization, and metamodeling concepts are utilised to solve Capacity Allocation Problem in a Photolithography Area (CAPPA) and Stochastic Job Shop Scheduling Problem (SJSSP) in this thesis. The purpose of this chapter is to describe the methodologies that can be used in Production Planning and Scheduling (PP&S) and the selected methodologies used in this study are presented.

2.2 Research Methods in PP&S

In the PP&S area, four research methodologies are available to use separately or conjointly. These are survey research, case research, action research, and quantitative modelling and simulation. Now, a brief introduction of these methodologies is given by defining briefly these research methodologies and briefly touching on how they can be implemented.

2.2.1 Survey Research

Typically, a survey can be defined as a grouping of information which represents individuals by means of mailed questionnaires, telephone calls, interviews, etc, or

social units of individuals (Forza 2002). According to this purpose, survey research is defined into three different categories. These categories are (1) exploratory survey research, (2) confirmatory (or theory testing or explanatory) survey research, and (3) descriptive survey research. The purpose of exploratory survey research is to obtain an initial understanding of a subject and its support for an initial structuring for a more detailed survey. Generally, it has no model and helps to determine the feasible limits of a theory. The purpose of confirmatory survey research is to test the sufficiency of theoretically well-defined concepts on account of hypothesised connections within the limit of models (Forza 2002). The purpose of descriptive survey research is to understand the connections to a particular phenomenon and to explain it for theory building and refinement. Among these survey research types, confirmatory survey research assumes a theoretical model which involves operational definitions, hypotheses, and definitions of conditions and relationships. Forza (2002) highlights that the lack of theoretical background in surveys is one of the biggest drawbacks in Operations Management. For this reason, surveys which define explicit theories are suggested to allow testing of these theories in the early stage of survey research. Accordingly, the stages of a confirmatory survey (or theory testing) are carefully structured. Figure 2.1 illustrates the main stages and sub-stages during the process of a confirmatory survey research.

Besides a theoretical domain, the main stages of a theory testing survey encompasses design and pilot testing. This involves sub-processes such as defining constraints, target sample, data collection method, test procedures and assessing measurement quality, collecting data for theory testing, data analysis process which tests a hypothesis and report generation stage with necessary conclusions and results as given in Figure 2.1.

2.2.2 Case Research

Case research is one of the most commonly used research methods, especially in building new theories and concepts by researchers and practitioners in the scientific

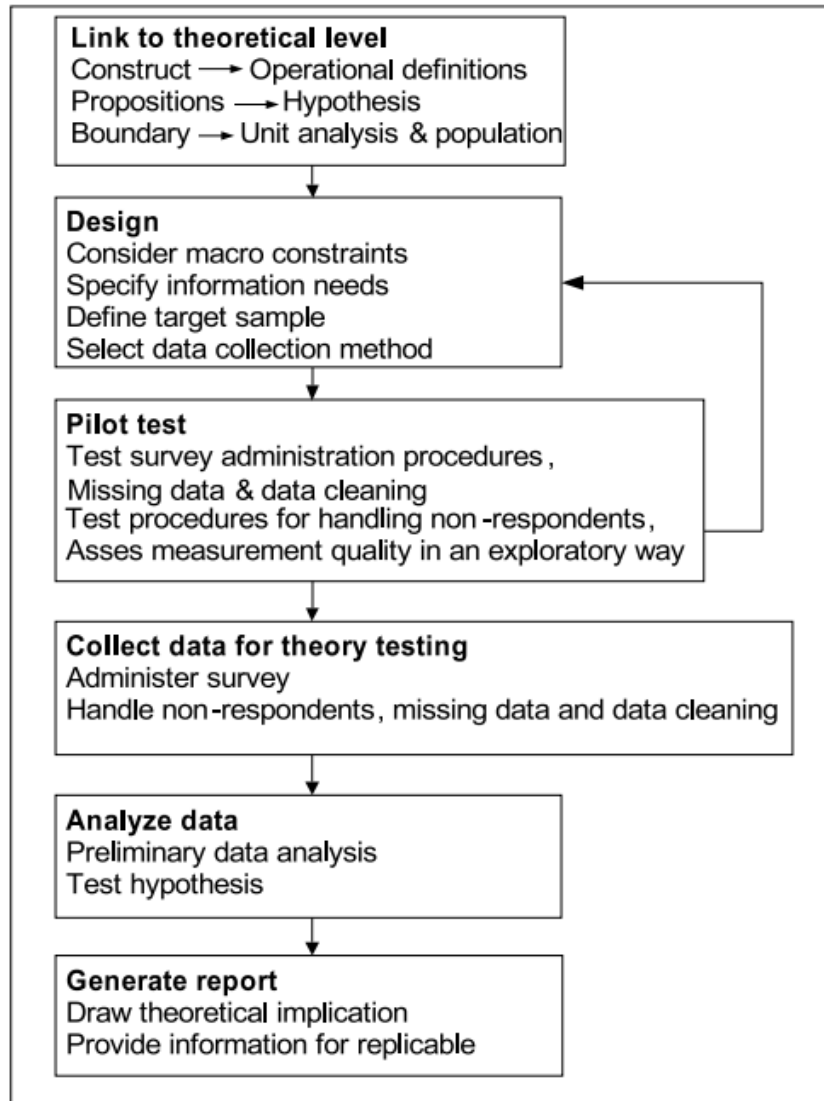


Figure 2.1: Main stages of confirmatory survey research (Forza 2002).

community. However, difficulties such as the requirement for qualified interviewers, high attention to generalizations from limited case boundaries, and the high level of time involved make case research not an easy methodology to implement (Voss, Tsiriktsis, and Frohlich 2002). On the other hand, the results of a case research study may have a great effect on improving new theories and concepts. Leonard-Barton (1990) defines a case study as “case study is a history of a past or current phenomenon, drawn from multiple sources of evidence. It can include data from direct observation and systematic interviewing as well as from public and private archives. In fact, any fact relevant to the stream of events describing the phenomenon is a potential datum in a case study since the context is impor-

tant” (Leonard-Barton 1990). In addition, case studies can be applied for different research objectives like exploration, theory building, theory testing, and theory extension/refinement. Voss, Tsikriktsis, and Frohlich (2002) describes the process of developing case research and conducting field research at length. The overall framework for conducting case research is summarized in Figure 2.2. In general, the process of case research starts with a research framework developed with research questions (see Figure 2.2). Especially, “why”, “what” and “how” type of research queries are widely acknowledged in case study research for the set of existing events on which the researcher has little or no control. Developing a conceptual framework is also considered useful in this early phase to discuss the constructs and variables in the study. This is followed by the phase for choosing cases which includes selection of the number of cases, case selection and sampling, specifying the parameters and factors. At this stage, selecting a single case provides a more in-depth analysis unlike multiple cases (Voss, Tsikriktsis, and Frohlich 2002). The next phase is developing research instruments and protocols. This phase consists of structured interviews, events, formal or informal observations, or a review of related literature. After field research is conducted by contacting people, collecting field data, conducting interviews, recording and clarifying objective data, determining cause and effect relationships (Voss, Tsikriktsis, and Frohlich 2002). Then, field research is followed by documentation and analysis of data. The data analysis phase may have searched cross-case patterns, hypothesis development and testing, and enfolding literature (Figure 2.2).

2.2.3 Action Research

As its name applies, the objective of Action Research (AR) is both to take action and to generate knowledge or information on applied action so that the results of AR are both action and research, not only built and generated knowledge as in the positivist science (Coughlan and Coughlan 2002). Coughlan and Coughlan (2002) discusses four distinct characteristics of AR. The first characteristic is research in

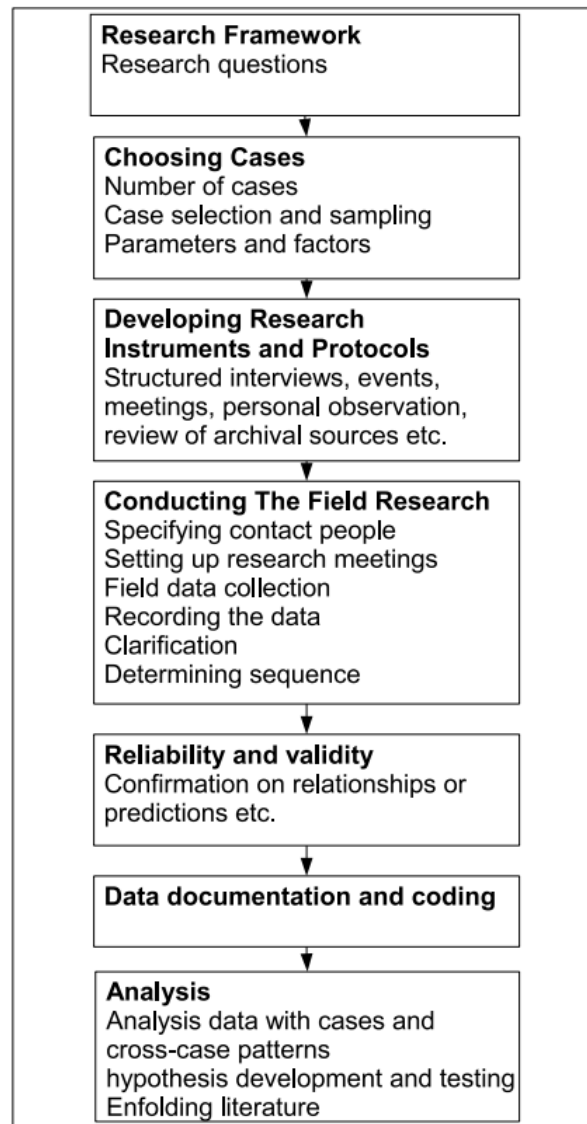


Figure 2.2: Overall framework for conducting case research (Voss, Tsirikrisis, and Frohlich 2002).

action, not research about action. Researchers take actions rather than only making observations under a particular context and purpose. The process of taking actions is cyclical, which involves an iterative cycle of planning, implementing, evaluating, and diagnosing actions. Figure 2.3 shows a cyclical four-step process of AR.

The second characteristic of AR is participatory action. As opposed to traditional research which takes members of the context or system as the objects of the study, the members of the system take part actively in the cyclical AR process illustrated in Figure 2.3 (Coughlan and Coughlan 2002). The third characteristic of AR is the concurrency of action. In AR process, the scientific knowledge is built up

continuously through effective actions in a cyclic fashion. The last characteristic of AR is described as a sequence of events. These events are described as an approach for problem solving as illustrated in a cyclic AR process which collects data, takes action solutions to practical problems with active participation of the members in the system, and brings solutions in an iterative cycle (Coughlan and Coughlan 2002). Implementation of AR involves three main phases. These phases are: (1) pre-step to conceptualize context and purpose, (2) six main steps which are connected with data and actions: data gathering, data feedback, data analysis, action planning, implementation of action and evaluation of outcomes, and (3) a meta-step for monitoring the overall AR process (Coughlan and Coughlan 2002). In the field of PP&S, Westbrook (1995) analysed potential applications of AR in PP&S.

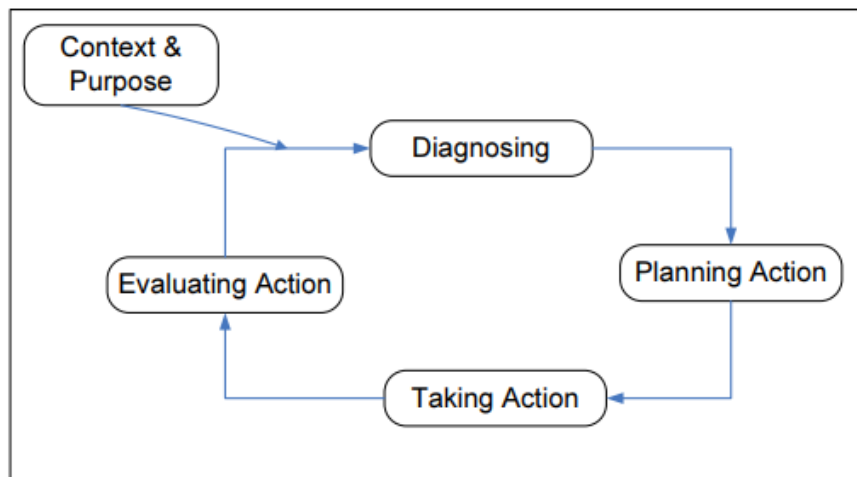


Figure 2.3: Cyclical four-step process of AR (Coughlan and Coughlan 2002).

2.2.4 Quantitative Modeling and Simulation

Quantitative modelling and simulation is another well-known method used in PP&S. Model based quantitative research or quantitative modelling is defined by Will M. Bertrand and Fransoo (2002) as being based on a set of variables that vary over a specific domain, while quantitative and casual relationships have been defined between these variables. In that case, performance variables can be physical variables (e.g., inventory position, utilisation rate) or economic variables (e.g.,

profit, costs or revenues). This type of research is referred to as operational research (in UK) or operations research (in USA). Quantitative models are grouped under two classes: axiomatic and empirical research. The first class is axiomatic research which is related to an idealised model by using associated theorems. The second class is empirical research, which aims to recreate a reality by building a model (Will M. Bertrand and Fransoo 2002). Based on the objective of research, each class can be divided under two categories: descriptive and normative (prescriptive) research. Both axiomatic and empirical research can be prescriptive and descriptive. Prescriptive or normative research is about developing policies, strategies or actions, finding the optimal of a defined problem, improving existing results. Descriptive research is about analysing a model to understand and explain its characteristics. Usually, axiomatic research is prescriptive while empirical research is descriptive. The quantitative research cycle presented by Will M. Bertrand and Fransoo (2002) is represented in Figure 2.4, which includes four phases: conceptualisation, modelling, model solving, and implementation. In conceptualisation, variables are selected and the scope of the problem and the model are introduced. In modelling, the relationships between the variables are defined. In the model solving part, mathematical or statistical techniques are used. In implementation, the results of the model are implemented, which may create a new cycle. Moreover, based on this research type, both the start and finish nodes in the route can be changed.

In fact, the selection of above-mentioned methodologies depends on the purpose of the research, requirements, and available resources to conduct a research. Thus, each research method has its own merits. To maximise the benefits of different methodologies, they can be used in combination with each other. For instance, in the literature of PP&S, quantitative and case research approaches are the most applied methodologies (Fazel Zarandi et al. 2020; Chong, Appa Iyer Sivakumar, and Gay 2003).

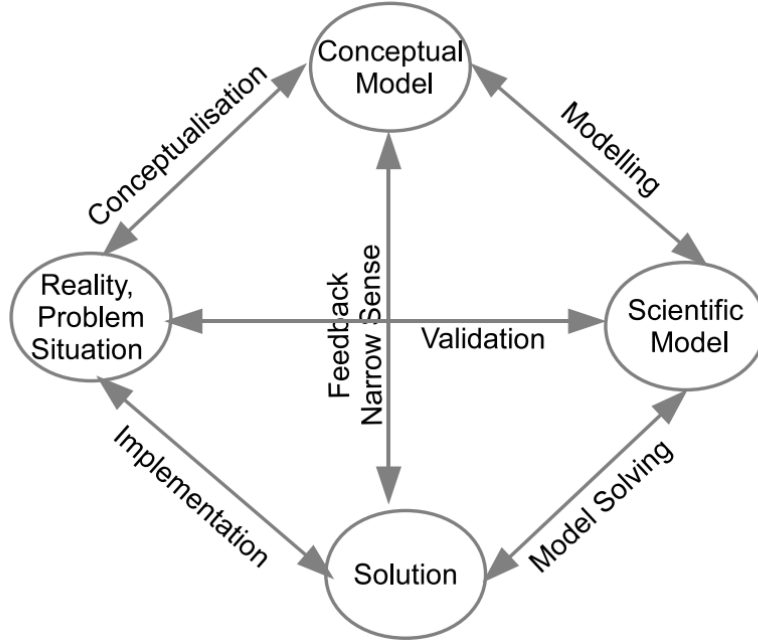


Figure 2.4: Quantitative research cycle (Will M. Bertrand and Fransoo 2002).

2.3 The Research Methodology used in this Thesis

As mentioned above, this research has been conducted as a part of Productive4.0 project (*Productive4.0 - A European co-funded innovation and lighthouse project on Digital Industry* 2021). Thus, to improve the efficiency level within semiconductor manufacturing supply chain networks, the prime goal of this research has been to propose simulation, optimization, and Machine Learning based (ML) Decision Support Tools (DSTs) supporting PP&S paradigms within photolithography toolsets (see Section 1.1). On the other hand, case research and quantitative research methods have been the most desirable research methodologies in the literature of PP&S. Thus, in this thesis, the data provided by Bosch is used as the case study, and a quantitative empirical research methodology is considered to answer research questions. Figure 2.5 shows the the research methodology conducted in this thesis. As mentioned above, in this thesis, CAPP and SJSSPs are addressed, which are the two main problems within the context of photolithography PP&S. Both CAPP and SJSSP are well conceptualized, modelled, and many examples are solved within the

literature. Thus, there are available solutions for both mentioned classes of PP&S problems. In this thesis, using mathematical modeling and simulation techniques, both Capacity Allocation Problem in Photolithography Area (CAPPA) and SJSSP problems are modeled. To verify the designed models, comparisons with existing results and models are conducted. In the next stage, before solving the designed models, the analysed data sets from the case research are used to quantify the model parameters.

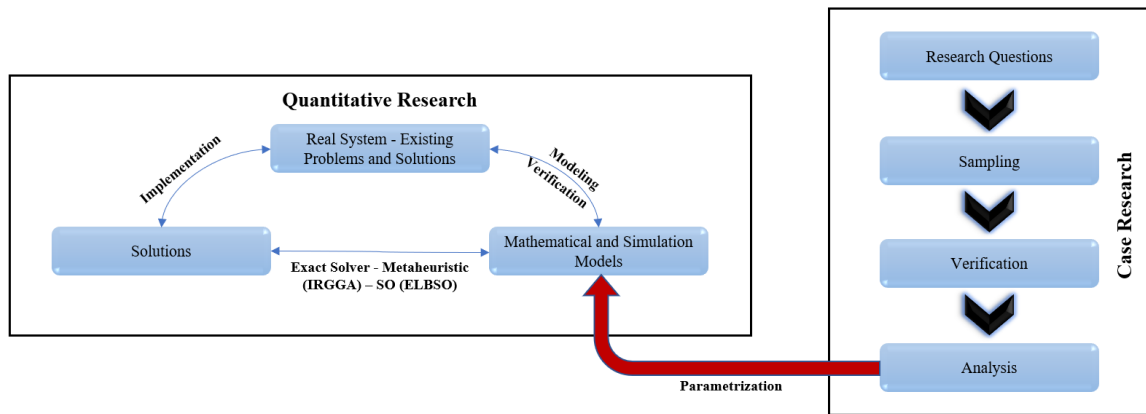


Figure 2.5: Research methodology in this thesis.

As mentioned above, the obtained data sets are from a Bosch front-end fab. Accordingly, data was collected for 3 months from a front-end Bosch fab with a data set consisting of 929,178 rows of data. To clarify, these data sets include the size of lots (TRACKINMAINQTY), photolithography entrance times in each route (TRACKINTIME), and photolithography exit times in each route (TRACKOUTTIME) (see Figure 2.6). Using track in and track out times of each lot, the processing time for each lot could be easily calculated. It is worth mentioning here that all data sets are sampled from the Manufacturing Execution System (MES) at the Bosch fab and are verified by Bosch before analysis. since some lots does not require the photolithography in each route, their processing time in the photolithography area equals to zero. Therefore, all jobs with processing times equal to zero are deleted.

Figure 2.7 shows the fitted distributions on processing time data sets using Matlab Fitting App (Mathlab Fitting App 2020). Although the Normal distribution can describe features of the processing times with a good estimation, it contains

TRACKINTIME	TRACKOUTTIME	TRACKINMAINQTY
Datetime	Datetime	Number
TRACKINTIME	TRACKOUTTIME	TRACKINMAINQTY
12/06/2015 19:50:43	12/06/2015 19:50:43	1
12/06/2015 19:50:44	12/06/2015 20:12:52	1
10/06/2015 17:19:23	10/06/2015 17:19:23	1
10/06/2015 17:19:25	10/06/2015 17:55:32	1
01/07/2015 21:56:56	01/07/2015 21:56:56	1
01/07/2015 21:56:57	01/07/2015 22:31:13	1
20/10/2017 14:17:50	20/10/2017 14:17:50	1
20/10/2017 14:27:33	20/10/2017 14:27:33	1
10/04/2015 01:35:26	10/04/2015 01:35:27	7
10/04/2015 01:35:28	10/04/2015 02:02:25	7
18/04/2015 21:38:17	18/04/2015 21:38:17	7
18/04/2015 21:38:18	18/04/2015 22:11:29	7
30/07/2015 14:49:32	30/07/2015 14:49:32	1
30/07/2015 14:49:33	30/07/2015 15:13:27	1
19/05/2015 19:15:50	19/05/2015 19:15:51	25
19/05/2015 19:15:52	19/05/2015 19:58:14	25
06/05/2015 13:52:56	06/05/2015 13:52:56	25
06/05/2015 13:52:58	06/05/2015 14:24:18	25
25/02/2016 20:17:54	25/02/2016 20:17:55	13
25/02/2016 20:17:55	25/02/2016 20:51:35	13
01/07/2016 03:07:33	01/07/2016 03:42:22	3
22/04/2015 14:27:19	22/04/2015 14:27:20	13
22/04/2015 14:27:21	22/04/2015 15:00:28	13
24/06/2016 15:25:37	24/06/2016 15:25:37	3
24/06/2016 15:25:38	24/06/2016 15:55:26	3
25/06/2016 22:50:09	25/06/2016 22:50:09	3
25/06/2016 22:50:10	25/06/2016 23:26:43	3

Figure 2.6: Bosch data set sample.

negative numbers. Therefore, a Gamma distribution is used which guarantee both positive numbers for processing times and a good quality of estimation ¹. This is used to parameterize the designed quantitative models in Chapters 3 and 4. Then, the developed solution methods are used throughout the thesis.

2.4 Conclusions

In this chapter, the methodology of this thesis is defined. After describing different research methodologies in PP&S, in line with main goals of this research, quantitative models and case research are selected to be conducted in this thesis. The case research includes data sets from a Bosch front-end fab photolithography worksta-

¹The data set analysis has already been published in Ghasemi, Azzouz, et al. (2020).

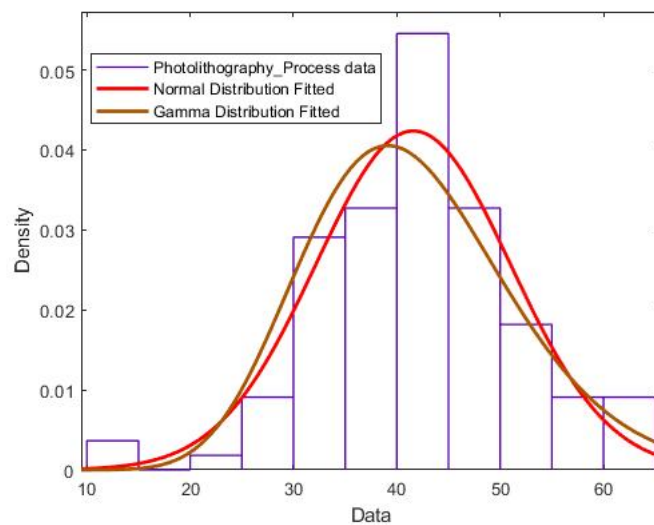


Figure 2.7: Fitted distributions to processing times.

tion, which is used to parametrize quantitative models. In Chapters 3 and 4, the case research is used to design CAPP and to structure metamodels, respectively. Then the designed metamodel is utilised in Chapter 5 to design an evolutionary learning-based simulation optimization method for SJSSPs.

Chapter 3

Optimizing Job Assignment in Semiconductor Manufacturing Photolithography Area – Case Study: Robert Bosch

3.1 Introduction

Photolithography, requiring high capital investments, is one of the most crucial processes in semiconductor manufacturing. It is mostly regarded as a bottleneck process due to the layered nature of wafer fabrication, especially for the case of Application Specific Integrated Circuit (ASIC) fabrication environments with high mix product portfolios and low volumes. Essentially, the photolithography process includes three main steps. These steps are coat, expose, and develop. First, the wafer is coated with a thin film of a photosensitive polymer, called the photoresist strip. Then, in the “expose” step, the wafer is exposed with ultraviolet light (UV) in order to print the circuit pattern onto the wafer. This is done using a reticle, which is a chrome patterned glass that defines the circuit pattern. This pattern tends to be unique for each layer. In an ASIC fab, a diverse range of recipes exist

due the range of products produced. Finally, polymerized sections of photoresist are removed from the exposed wafer. Since the circuits are made up of layers, with every wafer passing through the photolithography area up to 40 times, this typically makes photolithography a bottleneck resource. With the performance of a system determined by the bottleneck resource, good capacity allocation of a photolithography work area ensures improvement in the performance of the whole fab. In this chapter, we consider the capacity allocation problem for the photolithography toolset, which is commonly referred to as the CAPP problem (Chung, Huang, and Lee 2006). In this problem three different constraints exist which distinguish CAPP from typical allocation and scheduling problems. First, certain machines within the photolithography tool will be qualified for different recipes (machine process capability constraints). Secondly, for some critical layers, certain machines within the toolset will be required to be used to ensure the quality of the integrated circuit (machine dedication constraints). Thirdly, the number of times a reticle shares between different layers should be less than its maximum number of sharing amount (maximum reticles sharing constraints). In summary, CAPP refers to allocating order layers on different machines by considering the above-mentioned constraints.

Due to several manufacturing requirements, semiconductor production is extremely cost intensive. Semiconductor microchips with around 1000 different production steps and re-entrant loops in a job shop production system may face lead times of one month and more. Photolithography work area is one of the most crucial steps of wafer fabrication. Since the circuits are made up of layers, every wafer passes through the photolithography area up to 40 times. Thus, the photolithography work area is often regarded as a bottleneck process. According to the fundamental concept of the theory of constraints, the performance of a system is determined by the bottleneck resource in that system. Thus, photolithography has a direct impact on the product cycle time. Moreover, there is a great demand in the market for semiconductors, which makes every production second count in terms of profitability. (To illustrate, the revenue is around \$1000 per wafer). Understanding

factors influencing the cycle time would have direct effects on improving product lead times and consequently the revenue of the company.

Although machine process capability, machine dedication, and maximum reticles sharing constraints are critical in most real photolithography fabs, the majority of researchers have considered them separately to reduce problem complexity (e.g. (Chung, Huang, and Lee 2006) and (Çatay, Erengüç, and Vakharia 2003)). In this chapter, we are considering all of them together, which makes the problem in practice more interesting but also more difficult to solve. Chen, Chen, and Liang (2016) is the only work that modelled all these constraints integratedly. However, for the sake of simplicity, they considered reticles availability as a predefined parameter and not a part of the solution to be optimized. From a practical point of view, the photolithography workstation costs upwards of \$70 million per machine (ELINFOR 2019), in addition, reticles in 2010 cost between \$1K – \$200K with, at the usage rate of 1800 wafers per reticle, a mask contribution to the cost of photolithography ranges from \$0.55 – \$111.11 per wafer exposure (Levinson 2010). Therefore, optimizing the capacity allocation will not lead to significant benefits if the reticles are not optimally shared, which limits the applicability of the approach developed by (Chen, Chen, and Liang 2016).

In this research, data was collected for 3 months from a front-end Bosch fab with a data set consisting of 929,178 rows of data. Firstly, we retrieve important and relevant information about data using classification methods. This helped to classify data into different classes, with patterns for each data class identified. After analyzing this complex data set, which consisted of operations, tools and lots features, experiments and sensitivity analysis was carried out. Using Rstudio software, both track in and track out times of the photolithography department for each lot in the data set were extracted. Then distributions were fitted to different CAPPAs instances. In the following (Section 3.5.3) the experiment design steps are described in detail.

The GA presented here has been customized to solve the CAPPAs problem. The

closest work to our research was Chen, Chen, and Liang (2016) who did not consider any form of adaptation that could accelerate the algorithm convergence by directing the search to the feasible area. We mainly redefined the initialization, mutation, and crossover operators. We used a greedy heuristic to seed the initial random population, which enables the algorithm to start with solutions with a higher quality. In addition, we used imitation and avoidance operators reported by Beheshtinia, Ghasemi, and Farokhnia (2017) which enables the algorithm to search only in the feasible area. Beheshtinia, Ghasemi, and Farokhnia (2017) just provided a general concept of Reference Group-based GA without defining core elements implementation which limited its use. In this chapter, we defined different algorithm components making possible the expansion of the use of this concept.

In a final step in this research, a sensitivity analysis is provided that provides information to practitioners in balancing the three different constraints found in managing the photolithography tool set, machine process capability, machine dedication and reticle constraints. In summary, this chapter provides

- A mixed integer linear formulation presented by (Ghasemi, Heavey, and Kabak 2018) is extended by adding maximum reticles sharing constraints integrated with machine process capability and machine dedication constraints.
- It is then demonstrated that CAPP is an NP-Hard problem. We also proved that it cannot be optimally solved with the exact solver IBM ILOG CPLEX software except for small size problems.
- Therefore, a new GA, named IRGGA, is developed to obtain approximate near optimal solutions. IRGGA is essentially based on the GA proposed by Beheshtinia, Ghasemi, and Farokhnia (2017), named Reference Group GA (RGGA), which we have updated by developing a heuristic to create the initial population instead of the random initialization used in RGGA. This improvement customizes the RGGA, by improving its performance in solving the CAPP. This is due to directing the search towards the feasible space.

- To evaluate our algorithm, the experimental study is conducted using different sets of real data gathered from a Bosch semiconductor facility. Comparative experiments are done against two GAs reported in (Chen, Chen, and Liang 2016) and (Beheshtinia, Ghasemi, and Farokhnia 2017). Moreover, different sensitivity analysis of the three different constraints, machine process capability, machine dedication and maximum reticles sharing in CAPPA are provided.

The remainder of this chapter is organized as follows. Section 3.2 provides a literature review in solving this problem. Section 3.3 presents a mathematical formulation for the problem while Section 3.4 describes the proposed GA. In Section 3.5, the results of the conducted experiments are reported and discussed. Finally, Section 3.6 concludes the chapter and looks at possible follow-on studies.

3.2 Literature Review

A large amount of research has been carried out on the operations of semiconductor manufacturing (Mönch, Uzsoy, and Fowler 2018; Mönch, Fowler, and Mason 2013; Uzsoy, Lee, and Martin-Vega 1992). The photolithography process is considered a key process in wafer fabrication due to the complex technology used, critical dimensions involved, and re-entrant flows and is considered the bottleneck in most fabrication lines (Lee and Lee 2003). As a result of this, the importance of an efficient machines capacity plan for this tool is required (Mönch, Fowler, and Mason 2013). Capacity planning problems appear in many forms and have attracted thousands of research papers. To structure this large body of research, comprehensive literature reviews by Martínez-Costa et al. (2014) and Wu, Erkoc, and Karabuk (2005) have been published. As stated by Martínez-Costa et al. (2014), deterministic models have got much attention in solving capacity allocation models. Game theoretic approaches have been widely applied to capacity planning problems in strategic and tactical levels of manufacturing organizations (Renna and Argoneto 2010). Renna and Argoneto (2010) developed a distributed approach for a network of independent

enterprises, able to facilitate the capacity process by using a multi-agent architecture and a cooperative protocol. In another application of game theory concepts in solving capacity allocation problems, Seok and Nof (2014) proposed an adaptive Collaborative Demand and Capacity Sharing (CDCS) protocol based on dynamic contract mechanism. Liu, Zhang, et al. (2017) proposed a model of cloud manufacturing resource service sharing based on the Gale-Shapley algorithm and analysed it in the context of fluctuating resource service supply and demand. Although game theory methods has a broad range of use in production and supply chain planning, it has been scarcely applied to the CAPP as a result of the high complexity of semiconductor manufacturing operational problems.

With regard to machine capacity allocation models in photolithography, one of the earliest studies on both capacity allocation problem and machine capability is the study by Leachman and Carmon (1992), in which they defined a production plan by presenting a linear programming (LP) model in order to maximize the total profit. A similar production plan with an LP model was also presented by Hung and Cheng (2002). The former study scrutinized machine process capability constraints by introducing an alternative machine set' that is defined to represent a capability for a particular operation, and the capacity limitations of these machine sets are indicated by proposed models (i.e., step-separated, workload allocation and direct mix formulations) with the assumption of identical or proportional processing times (Leachman and Carmon 1992). For LP formulations, the number of decision variables increases due to revisits of products to process areas because of the number of alternative machine types to the power of the number of re-entrant visits (Leachman and Carmon 1992).

Regarding the latter study, Hung and Cheng (2002) presented the capacity partition generation procedure (CPGP) in which the uniformity assumption is relaxed in the direct mix formulation provided by Leachman and Carmon (1992) with the capacity set generation procedure (CSGP). In another earlier study, Toktay and Uzsoy (1998) transformed the capacity allocation problem into a maximum flow

network problem for maximizing throughput. Their mathematical formulation includes not only machine capabilities but also tooling and set-up constraints together with integer side constraints. They compared results of the problem by two proposed heuristics, i.e., greedy and extended heuristics. On a later study, Akçalı and Uzsoy (2000) decomposed the shift-scheduling problem into two sub-problems which are capacity allocation and lot sequencing, in order to analyze them sequentially. To solve the problem, Capacity Allocation Routine (CAR) was applied by the greedy heuristic defined by Toktay and Uzsoy (1998), and embedded in a simulation model. Also, two different sets of capabilities (i.e., operation-stepper matrices) were defined as fully-flexible and nested sets. That is, the fully-flexible set was defined for processing capability of all operations, and the nested set was defined for the processing capability of a partial set of operations. Their simulation experiments included analyses of stepper capabilities, reticle and setup constraints.

In another study, Çatay, Erengüç, and Vakharia (2003) presented a multiperiod mixed-integer programming model (MILP) with duplicated tools to minimize total costs, specifically, total of machine tool operating, new tool acquisition and inventory holding costs. They used a Lagrangian-based relaxation algorithm to solve the problem with the assumption of known demands and capacity limits. Moreover, they classified tool groups as primary, defined as the most efficient tools to process particular operations, and also alternative tools when additional capacity is required. By taking into account reticle availability constraints and investments on tool capabilities, Kabak et al. (2013) provided a summary of the proposed actions under varying levels of demand to maintain tool wait times and utilization in a detailed simulation model of a photolithography area.

Regarding machine dedication constraints, they were evaluated in some studies that considered the capacity allocation problem. To illustrate, Akçalı, Nemoto, and Uzsoy (2001) pointed out that a flexible machine dedication policy, that is, a lot does not have any restrictions on a stepper at any layer, significantly effects the average and variation of the photolithography cycle times.

Both machine process windows and machine dedication constraints were addressed by Chung, Huang, and Lee (2006) and Chung, Huang, and Lee (2008). In the former study, Chung, Huang, and Lee (2006) presented a mixed-integer programming (MIP) model in order to level or balance the load over machines based on the average utilization rates. A demonstrative example and a real-world application were given in this study. In another study, Chung, Huang, and Lee (2008) referred to the capacity allocation problem as CAPP and they mention that the problem is NP-hard by stating that it is a subset of the Load Balanced Demand Point Assignment Problem (LBDPAP) which is NP-hard (Low and Fang 2005). As a result, the computational time to solve the CAPP problem for large instances takes much time (Chung, Huang, and Lee 2008). To overcome the computational complexity, later studies on CAPP introduced heuristic applications. In another work, Pham et al. (2008) proposed an integer programming framework to minimize the production cost by considering a dedicated photolithography machine constraint.

Table 3.1 presents an overview of publications on CAPP. CAPP has been known as one of the most complex problems in terms of scale in semiconductor manufacturing. Optimizing capacity allocation plays a key role in operational photolithography production line smoothness. It results in reducing cycle times and WIP in the well-known bottleneck of semiconductor fab. However, there is a research gap in modelling and solving CAPP large-scale problems that is caused by complex features of CAPP (machine process capability, machine dedication and maximum reticles sharing constraints). One can notice from Table 3.1 that the majority of these works did not consider all CAPP constraints simultaneously except the work of (Chen, Chen, and Liang 2016). Recognizing as shown in this chapter, that CAPP is NP hard, Chen, Chen, and Liang (2016) applied a GA to solve it, that we name Chen GA (CGA) to easily reference it in the rest of this chapter. CGA is developed based on the general framework of GA (Holland 1973) by adapting CAPP features to it.

Table 3.1: Summary of publications on CAPP with MC=Machine process capability constraints; MD=Machine dedication constraints; R=Reticle constraints; EX=Exact optimization; SH=Simple Heuristic; LS=Local-search; CS=Constraint Satisfaction; S=Simulation; P=(Maximize) profit; C=(Minimize) cost; LL=(Minimize) load levelling; TH=(Maximize) throughput; CT=(Minimize) cycle time.

	Constraints			Solution Approach					Objective				
	MC	MD	R	EX	SH	LS	CS	S	P	C	LL	TH	CT
(Leachman and Carmon 1992)	x	-	-	x	-	-	-	-	x	-	-	-	-
(Toktay and Uzsoy 1998)	x	-	-	x	x	-	-	-	-	-	-	x	-
(Akçali and Uzsoy 2000)	x	-	x	-	x	-	-	x	-	-	-	x	-
(Akçali, Nemoto, and Uzsoy 2001)	-	x	-	-	-	-	-	x	-	-	-	-	x
(Chung, Huang, and Lee 2006)	x	x	-	x	-	-	-	-	-	-	x	-	-
(Chung, Huang, and Lee 2008)	x	x	-	-	x	-	-	-	-	-	x	-	-
(Pham et al. 2008)	-	x	-	x	-	-	-	-	-	x	-	-	-
(Kabak et al. 2013)	x	-	x	-	-	-	-	x	-	-	-	-	-
(Chen, Chen, and Liang 2016)	x	x	x	-	-	x	-	-	-	-	x	-	-
(Ghasemi, Heavey, and Kabak 2018)	x	x	-	-	-	x	-	-	-	-	x	-	-

3.3 Problem Statement

In this section, the mixed-integer mathematical model presented in (Ghasemi, Heavey, and Kabak 2018) is extended by including the maximum reticles sharing constraints in the photolithography area. Before introducing the model formulation, the following example is considered to illustrate the CAPP based on machine process capability, machine dedication and maximum reticles sharing constraints. Let I refer to the order set, L_i the layer set for order i , K the machine set and $J(i, l)$ the layer l of order i , respectively. As shown in Figure 3.1, there are NO orders with NL_i layers for order i . Take for example $J(1, 2)$, first assuming that machine 2 ($M(2)$) is available (The machine process capability constraint), we will assign $J(1, 2)$ to $M(2)$. In addition, assume that the 1st and 5th layers of order 2 are critical, which we denoted using (*). Thus, as long as the first critical layer of order 2 ($J(2, 1)^*$) is assigned to the $M(1)$, then all other critical layers of order 2 must be assigned to the $M(1)$ as well (The machine dedication constraint). Finally, assume the first reticle (r_1) is capable to be shared just between 2 different layers. Besides, $J(1, 1)$ and $J(1, 2)$ need reticle r_1 to be produced. Simultaneously, $J(1, 3)$ can be produced using both reticles r_1 and r_2 and there is no other layer that requires r_1 to be produced. Therefore, to avoid violating the maximum sharing reticles constraint which is 2 for reticle r_1 , r_1 must be assigned to produce $J(1, 1)$ and ($J(1, 2)$) (Maxi-

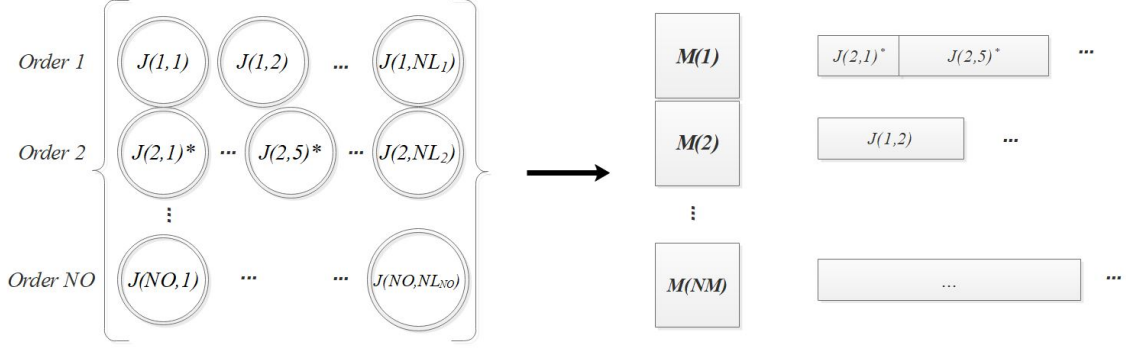


Figure 3.1: CAPPA example.

imum reticles sharing constraint).

The common notations used throughout this chapter are provided in Table 3.2. In CAPPA we consider NO orders indexed by i . Each order has a set of layers defined as L_i and NM machines indexed by k to process these layers. Besides, each layer l of order i requires special capability h to be produced defined by $CRcap_{ilh}$. In addition each machine k supports a set of capabilities (C_{kh}). To produce each layer l of order i , a related reticle r must be assigned (R_{ilr}), while the maximum number of sharing a reticle between layers is limited to δ_r . Moreover, there are different planning periods t considered, and different layers l of orders i pre planned to be produced in a certain planning period LT_{ilt} . In addition, some layers of orders are critical (CL_{il}). So, for quality reasons if the first critical layer of order i ($CLfirst_i$) is assigned to a certain machine, all other critical layers of order i must be assigned to that machine. After assigning all layers of orders to machines by considering mentioned constraints, the maximum workload between machines in each planning period t is calculated as ML_t . In fact, balancing loads between machines within the photolithography workstation enabling the highest machines utilization is essential due to the nature of photolithography machines (photolithography machines are highly expensive). Thus, in this research, the defined objective is to minimize the sum of maximum loads in all planning periods.

According to the model formulation, the objective function Equation (3.1) min-

Table 3.2: Notations table.

Indices and Sets	
$i =$	Order index, $i \in I = \{1, \dots, NO \}$
$l =$	Layer index, $l \in V = \{1, \dots, MNL \}$
$L_i =$	Layer sets for each order i , $L_i \in V$
$k =$	Machine index, $k \in K = \{1, \dots, NM \}$
$h =$	Capability index, $h \in H = \{1, \dots, NC \}$
$t =$	Planning period index, $t \in T = \{1, \dots, NT \}$
$r =$	Reticle index, $r \in RR = \{1, \dots, NR \}$
Parameters	
NO	Number of orders
MNL	Maximum number of layers considered for an order
NM	Number of machines
NC	Number of machine capabilities
NT	Number of planning periods
NR	Number of reticles
C_{kh}	If machine k has capability h then $C_{kh} = 1$, 0 otherwise
CL_{il}	If layer l of order i is a critical layer then $CL_{il} = 1$, 0 otherwise
$CRcap_{ilh}$	If layer l of order i has process capability h then $CRcap_{ilh} = 1$, 0 otherwise
LT_{ilt}	If layer l of order i is processed during period of time t then $LT_{ilt} = 1$, 0 otherwise
$CLfirst_i$	First critical layer of order i , $CLfirst_i \in L_i$
p_{ilk}	Processing time of order i and layer l on machine k
R_{ilr}	If the reticle r is capable to be used for layer l of order i Then $R_{ilr} = 1$, 0 otherwise
δ_r	Maximum number of reticle r can be shared between layers
M	A big number
Decision Variables	
dm_{ik}	If first critical layer of order i , $CLfirst_i$, is assigned to machine k , then $dm_{ik} = 1$, 0 otherwise
x_{ilkt}	If layer l of order i is assigned to machine k in planning period t then $x_{ilkt} = 1$, 0 otherwise
A_{ilr}	If reticle r is assigned to layer l of order i then $A_{ilr} = 1$, 0 otherwise
ML_t	Maximum loading level of machines at planning period t

imizes the maximum loading among machines during each period t . Equation (3.2) assigns the total of all workload to machines by considering the machine process capabilities constraint. Equation (3.3) ensures that each layer l of order i is assigned to one machine. Equation (3.5) represents the machine dedication constraint. That is, critical layers of order i are assigned to the same particular machine for process specifications. Equation (3.6) guarantees that the total of assigned workload for each machine and period can not exceed the maximum loading level. Equation (3.7) specifies the number of reticles r shared between layers. Equation (3.8) ensures that each layer l of order i is assigned to one reticle. Finally, Equations (3.9), (3.10), and (3.11) show the binary integer, and non-negativity constraints.

Mixed-Integer Mathematical Model

$$\min \sum_{t \in T} ML_t \quad (3.1)$$

Subject to

$$\sum_{t \in T} \sum_{k \in K} \sum_{h \in H} \sum_{l \in L_i} x_{ilkt} C_{kh} CRcap_{ilh} LT_{ilt} = \sum_{t \in T} \sum_{h \in H} \sum_{l \in L_i} CRcap_{ilh} LT_{ilt} \quad \forall i \in I \quad (3.2)$$

$$\sum_{k \in K} x_{ilkt} = 1 \quad \forall i \in I, l \in L_i \quad (3.3)$$

$$\sum_{t \in T} \sum_{h \in H} \sum_{l \in L_i} x_{ilkt} CL_{il} CRcap_{ilh} LT_{ilt} = \sum_{t \in T} \sum_{h \in H} \sum_{l \in L_i, l \geq CLfirst_i} dm_{ik} CL_{il} CRcap_{ilh} LT_{ilt} \quad (3.4)$$

$$\forall i \in I, k \in K \quad (3.5)$$

$$\sum_{h \in H} \sum_{i \in I} \sum_{l \in L_i} x_{ilkt} p_{il} C_{kh} CRcap_{ilh} LT_{ilt} - ML_t \leq 0 \quad \forall t \in T, k \in K \quad (3.6)$$

$$\sum_{i \in I} \sum_{l \in L_i} \sum_{k \in K} x_{ilkt} R_{ilr} A_{ilr} \leq \delta_r \quad \forall r \in RR \quad (3.7)$$

$$\sum_{r \in R} A_{ilr} = 1 \quad \forall i \in I, l \in L_i \quad (3.8)$$

$$dm_{ik} \in \{0, 1\} \quad \forall i \in I, k \in K \quad (3.9)$$

$$x_{ilkt} \in \{0, 1\} \quad \forall i \in I, l \in L_i, k \in K \quad (3.10)$$

$$ML_t \geq 0 \quad \forall t \in T \quad (3.11)$$

3.3.1 CAPP Complexity Proof

In this section, we show that the CAPP problem is strongly NP-hard by transforming it from minimizing makespan on parallel machines with machine eligibility restrictions, which is well known to be strongly NP-Hard.

Minimizing makespan on parallel machines with machine eligibility restrictions

Consider the problem of scheduling independent jobs J_1, \dots, J_n on parallel machines M_1, \dots, M_m to minimize the maximum completion time, which is commonly known as

the makespan. Each job J_j , for $j = \{1, \dots, n\}$, can be assigned to a set of eligible machines $k = \{1, \dots, m\}$ defined by an eligibility set $Eligibility = \{El_{1,1}, \dots, El_{j,k}, \dots, El_{n,m}\}$, where El_{jk} equals to 1 if machine k is eligible to produce job j and 0 otherwise. Moreover, job j has a positive integer processing time P_{kj} on machine k . No machine can process two jobs at the same time and preemption is not allowed (once the processing of a job on a machine has started, it must be completed without interruption on that machine).

The minimizing makespan on parallel machines with machine eligibility restrictions problem is well known to be NP-hard in the strong sense (Lenstra, Rinnooy Kan, and Brucker 1977; Liao and Sheen 2008).

Transformation of the Minimizing makespan on parallel machines with machine eligibility restrictions problem to CAPPA

Inspired by notations provided by Garey and Johnson (1990) and the methodology suggested by Fathi et al. (2016), consider the set

$$AC = \{ac_{111}, \dots, ac_{ilk}, \dots, ac_{NO,MNL,NM} | ac_{ilk} \in \{0, 1\}\} \quad (3.12)$$

where $ac_{ilk} = 1$ only if the machine k has the capability to produce layer l of order i , otherwise $ac_{ilk} = 0$. Moreover, set $FCL = \{fc_1, \dots, fc_i, \dots, fc_{NO}\}$ where fc_i defines the first critical layer number for order i , and set $AFCL = \{Afc_1, \dots, Afc_i, \dots, Afc_{NO}\}$ defines the assigned machine number of an arbitrary assignment of first critical layer of order i . Consequently, the assignment of all layers can be defined by

$$ACL = \{acl_{111}, \dots, acl_{ilk}, \dots, acl_{NO,MNL,NM} | acl_{ilk} \in \{0, 1\}\} \quad (3.13)$$

where $acl_{ilk} = ac_{ilk}$ for noncritical layers and $acl_{i,l,Afc_i} = 1$ and $acl_{i,l,k \neq Afc_i} = 0$ for all critical layers of order i . By considering R_{ilr} as the selected set of reticles for layer l of order i and $\sum_{r \in R} R_{ilr} \leq \delta_r$, one can show that this capacity allocation problem has a feasible solution, only and only if, the minimizing makespan on parallel machines

with machine eligibility restrictions by considering layers as processing jobs with $Eligibility = ACL$. This proves the NP-hardness, in the strong sense of finding a feasible solution to the CAPP. A.

3.4 Improved Reference Group Genetic Algorithm (IRGGA)

The GA simulates the biological evolution procedure in nature, and it explores optimal solutions using successive selection, crossover, and mutation operations. It is worth mentioning that GAs have been applied widely in nonlinear constraint problems (Zhang and Wong 2015).

Although GA has been previously used to solve CAPP (Chen, Chen, and Liang 2016), due to the complexity of this problem, improvements can be made on how GA can solve this problem. The general framework of a GA is as follows. The algorithm starts with a population of solutions called chromosomes which pass through crossover and mutation operators to get the offspring population. Environmental selection is then executed based on a fitness function to select solutions that will continue to the next generation. These steps are repeated until a stopping criterium is met. Recently, Beheshtinia, Ghasemi, and Farokhnia (2017) proposed to extend the basic GA by adding a psychological concept, named Reference-Groups. The resulting algorithm was called Reference Group GA (RGGA). This psychological concept, firstly introduced by Merton (1957), is where people try to be similar to good people in society, while differing their features from bad people. In this chapter, we improve and adapt RGGA to CAPP by adding a new heuristic inside the population initialization, the first step of the algorithm. In addition, Beheshtinia, Ghasemi, and Farokhnia (2017) firstly proposed the concept of crossover and mutation operators using imitation and distinguish procedures in the RGGA algorithm, applying the concept to a delivery problem. However, they did not introduce operators in detail. In this work, we restate this concept with regard to the CAPP

application.

3.4.1 Solution Structure and Greedy Heuristic-based Initialization

Solution Structure

GAs with random keys were first introduced by Bean (1994) for solving combinatorial optimization problems involving sequencing. In this chapter, we refer to this class of GAs as Random-Key GAs (RKGA). In a RKGA, chromosomes are represented as a string or vector of randomly generated real numbers in a predetermined interval. A deterministic algorithm called a decoder, takes as input any chromosome and associates with it a solution of the combinatorial optimization problem for which an objective value or fitness can be computed.

An example random key used in the IRGGA, assume that there are 3 orders with their specified number of layers planned to be produced. In this example, Table 3.3 shows the chromosome structure of an arbitrary production plan with 7 genes a_1, a_2, \dots, a_7 . The string is composed of 7 bounded numbers in each gene, where each number represents a machine (k) that produces a layer (l) of each order (i) from $J(i, l)$. Each selected machine k is used to produce a layer l of order i . In producing this random key, an algorithm is used that will not contravene machine dedication and machine process capability constraints, blocking generation of any infeasible solutions. Table 3.3 shows the assignment of order 1 with layer 1, and order 2 with layer 1, and order 2 with layer 2 assigned to the first machine, respectively. Order 1 with layer 2 and order 3 with layer 1 and order 3 with layer 3 are assigned to the second machine. Finally, order 3 with layer 2 is assigned to the third machine.

Table 3.3: Chromosome structure

$J(i, l)$	1,1	1,2	2,1	2,2	3,1	3,2	3,3
Select $k \in K = \{1, \dots, NM \}$	1	2	1	1	2	3	2
	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Greedy Heuristic-Based Initialization

Corne and Ross (1995) applied a heuristic technique to GAs for the set covering problem. Their study compared the performance of GAs, Simulated Annealing (SA), and Multistart Stochastic Hill-Climbing (MSSH) when starting from both random solutions and seeded solutions. This revealed that although GAs performed badly in comparison to the other two approaches when starting from random solutions, the situation changed if each of the approaches started with seeded solutions. The act of seeding the population greatly increased the performance of the GA while having very little effect on the other two approaches, resulting in the GA outperforming both SA and MSSH on these problems. So in IRGGA, as described in Algorithm 1, we first generate the initial population using RKGA concept and then seed it using a Greedy Heuristic (GH) proposed by Piersma and Dijk (1996).

Algorithm 1: Population Initialization Algorithm

Output: P' (population)**begin**

1. $P = \text{randomInitialize}();$ // Randomly generate a population of vectors considering machine dedication, machine process capability, and maximum reticles sharing constraints.
2. $\text{maxLoadEvaluate}(P);$ // Calculate the maximum load of each solution using the objective function.
3. $P' = \text{GH-based Initialization}(P);$ // improve the generated population (see Algorithm 2).
4. $RML = \{\text{The best } NR \text{ solutions based on the objective function}\}$ // $NR = \text{size of the Role Model List (RML)}$.
5. $IPL = \{\text{The worst } NI \text{ solutions based on the objective function}\}$ // $NI = \text{size of the Imperfect Model List (IML)}$.

end

After randomly initializing the population, we implement a GH-based Initialization (c.f. Algorithm 1, line 3) to seed the population by reassigning one job from a machine with the maximum load to another possible machine. Algorithm 2 describes

how the GH is implemented to improve the population in IRGGA ¹.

Algorithm 2: GH-based Initialization Algorithm

Inputs : P (population), N (population size)

Output: P' (population)

begin

for $index = 1$ **to** N **do**

1. Define the current assignment of layers to orders as S .
2. Choose machine k_{max} as the machine with the smallest index among machines with maximum load.
3. Define the set J as max of the layers assigned to machine k_{max}
4. Define j as the layer with the smallest index in J .
5. Define the set $E_{max} = \{(i, j) : i \in \{1, \dots, NM\}, i \neq k_{max} \text{ and } j \in J\}$ as all other possible assignments of the layers assigned to the machine k_{max} .
6. When the set E_{max} is empty stop. Otherwise, assign layer j to the machine i which has the smallest index among machines in E_{max} and name this assignment as g .
7. Compare the maximum load $ML(S)$ of assignment S with the maximum load of machine i in assignment g where layer j is reassigned to machine i . Move to assignment g if the maximum load of machine i , in assignment g is smaller than $ML(S)$.
Otherwise, continue with assignment S .

8. Return P' ;

3.4.2 Crossover and Mutation operators using imitation and distinguish procedures

The crossover operator (CO) generates two new individuals from the two selected parents and the mutation operator generates one new individual from a selected offspring. Both operators guarantee the reservation of good individuals and a diverse

¹Appendixes C.1 to C.7 contain the actual MATLAB codes to parametrize and initiate IRGGA.

population. Crossover and mutation are described in Algorithm 3 and Algorithm 4 respectively, and then illustrated in Figure 3.2. Crossover function occurs between all members at a predetermined rate. In other words, all chromosomes have the chance to influence each other (see Algorithm 3), which in IRGGA is considered as interactions between people in a community and how their behaviors have an influence on others ².

Algorithm 3: IRGGA Crossover Algorithm

Inputs : P (population), N (population size), P_c (crossover occurrence rate)

Output: CP (Child Population)

begin

```

for  $i = 1$  to  $[(P_c * (N/2))]$  do
  1.  $P_1 = \text{randomSelect}(P)$ ; //Choose randomly one chromosome
    from the population as the first parent.
  2.  $P_2 = \text{randomSelect}(P)$ ; //Choose randomly another different
    chromosome as second parent.
  3.  $V = \text{randomPick}(0, n - 1)$ ; //Randomly select any crossover point
     $V \in [0, \dots, n - 1]$  |  $n$  is the chromosome length.
  4.  $O_1 = [P_1(1 : V), P_2(V + 1 : n)]$ ;
  5.  $O_2 = [P_2(1 : V), P_1(V + 1 : n)]$ ;
  6.  $\text{Add}(O_1, CP)$ ; // add the first child to the child population  $CP$ 
  7.  $\text{Add}(O_2, CP)$ ; // add the second child to the child population  $CP$ 
  8. Return  $CP$ ;

```

Mutation function consists of two new procedures as follows:

- **Imitation procedure (IP):** As described in Algorithm 4, there are two chromosomes in the imitation procedure, one of them is the Role Model (RM), which is taken from the Role Model List (RML), and the other one is the Chromosome With Inherited Features (CWIF). CWIF inherits a number of RM chromosome features. In this procedure, a gene from the CWIF

²Appendixes C.8 to C.13 contain the actual MATLAB codes of main loops, fitness calculations, and recombination procedures in IRGGA.

is selected randomly. It is checked whether the gene in the RM chromosome is equal to it or not. If not equal, then the value of the gene in the CWIF chromosome turns into the RM one. Else, no change is needed.

- **Avoidance procedure (AP):** As described in Algorithm 4, there are two chromosomes in the AP, one of them is the Imperfect Model (IM), which is taken from Imperfect Model List (IML), and the other one is the Chromosome With Distinguished Features (CWDF). In this procedure, the CWDF wants to be different from the IM in some features. In this procedure, a gene from the CWDF is selected randomly. It is checked whether the gene in the influencer chromosome is equal to it or not. If the value is equal, then the value of the gene in the influencer chromosome is replaced with a new random machine assignment by considering feasibility constraints. If the value of the gene in the CWDF is not similar to the IM one, no change is needed.

It is worth mentioning here that the initial population is initiated by considering machine dedication, machine process capability, and maximum reticles sharing constraints. Thereby, all solutions would be feasible. Besides, crossover and mutation operators are designed in the way that they change allocations by guaranteeing the feasibility.

Figure 3.2 shows the objective function values for 100 different solutions in a minimizing problem. Each solution (i.e., chromosome) consists of m features, called genes. Solutions with the lowest objective function values are known as RM and the group of highest solutions are known as IM. Based on the concept of IRGGA, crossover occurs between all solutions in a population (eg. S_{38} and S_{59}). In addition, IP occurs between regular solutions and RM. Simultaneously, AP occurs between regular solutions and IM. To illustrate, IP occurs between the 44th and 66th solutions and the AP occurs between the 65th and 92th solutions.

Algorithm 4: IRGGA Mutation Algorithm

Inputs : P (Population), N (Population size), P_m (Mutation occurrence rate), IP Replication (Imitation procedure occurrence rate), AP Replication (Avoidance procedure occurrence rate)

Output: MP (Mutation Population)

begin

for $i = 1$ **to** $(P_m * N)$ **do**

1. $CWIF = \text{randomSelect}(P)$; // Randomly choose a chromosome from P .
2. $RM = \text{randomSelect}(RML)$; // Randomly choose a chromosome from RML .
3. $IM = \text{randomSelect}(IML)$; // Randomly choose a chromosome from IML .

for $j = 1$ **to** IP Replication **do**

4. $(RP, index) = \text{randomSelectGene}(CWIF)$; // returns a randomly selected gene and its position in a chromosome $CWIF$.
5. $RR = \text{randomSelectGene}(RM, index)$; // extracts the gene at position $index$ form RM .
- if** $RP \neq RR$ **then**
 6. $RR = RM.integer$ // RR turns into the RM .

for $k = 1$ **to** AP Replication **do**

7. $(RI, index) = \text{randomSelectGene}(CWDF)$;
8. $IV = \text{randomSelectGene}(IM, index)$; // extract the gene at position $index$ form IM .
- if** $RI = IV$ **then**
 9. $RI = \text{randomCreate}()$; // Randomly create RI by considering feasibility.

10. $\text{add}(CWIF_i, CWDF_i, MP)$; // Add the child to the child population MP

11. **Return** MP ;

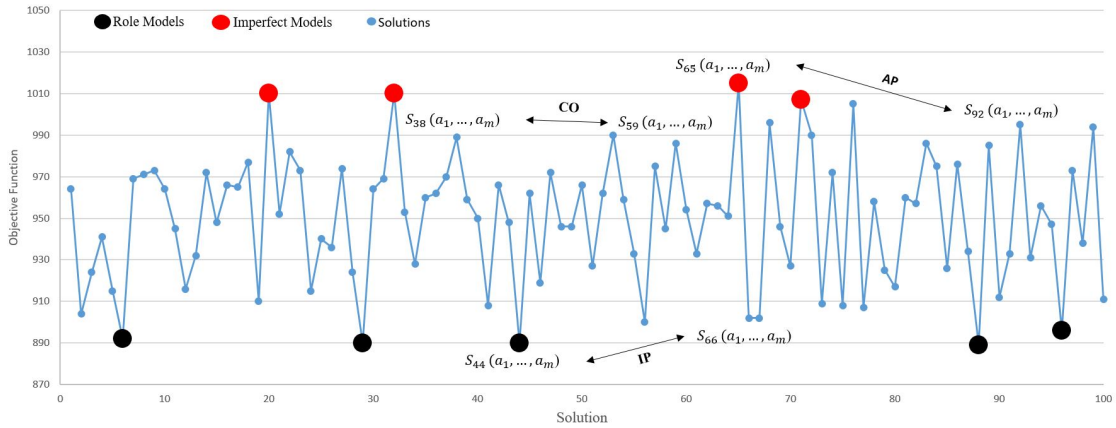


Figure 3.2: General concept of IRGGA

3.5 Experiments and Results

In this section, we describe the conducted experiments and analyse the obtained results. Firstly, IRGGA parameter settings are introduced to comprehensively describe the experiments design parameters. Then, the accuracy of IRGGA is validated against an exact solver for small-sized problems. Finally, IRGGA is compared with two other recent GAs in the CAPP literature, called CGA (Chen, Chen, and Liang 2016) and RGGA (Beheshtinia, Ghasemi, and Farokhnia 2017), using real case data sets extracted from Bosch Fab in Reutlingen.

3.5.1 Parameters Design

Parameter settings of a GA have a deep influence on its performance. Thereby, we devote this section to discuss the parameter values of the algorithms using Taguchi experimental design. This method is based on a special set of arrays called orthogonal arrays to conduct the minimum number of efficient experiments that could give insights on all factors that affect the performance measure. IRGGA has five parameters: population size (N), crossover probability (P_c), mutation probability (P_m), size of the role model list (NR) and size of the imperfect model list (NI). To perform the test, different stage values need to be selected for each parameter. Table 3.4 describes the different parameters evaluated using the Taguchi method. The main effects plots are shown in Figure 3.3 which shows how each factor affects the response characteristics (S/N ratio, means, slope, and standard deviation). A main effect exists when different levels of a factor affect the characteristic differently. For a factor with two levels, you may discover that one level increases the mean compared to the other level. This difference is a main effect.

After running a number of experiments, we picked these parameters values practically. Based on the number of parameters and their set values, Taguchi test asked 27 different test algorithm runs. The Taguchi test results are shown in Figure 3.3. After running the test, the values 100, 0.7, 0.3, 15, and 10 are selected for N , P_c , P_m , NR and NI respectively. It is worth mentioning here that similar values for

Table 3.4: Taguchi test parameters.

Parameter	1	2	3
N	50	75	100
P_c	0.5	0.6	0.7
P_m	0.2	0.3	0.4
NR	5	10	15
NI	5	10	15

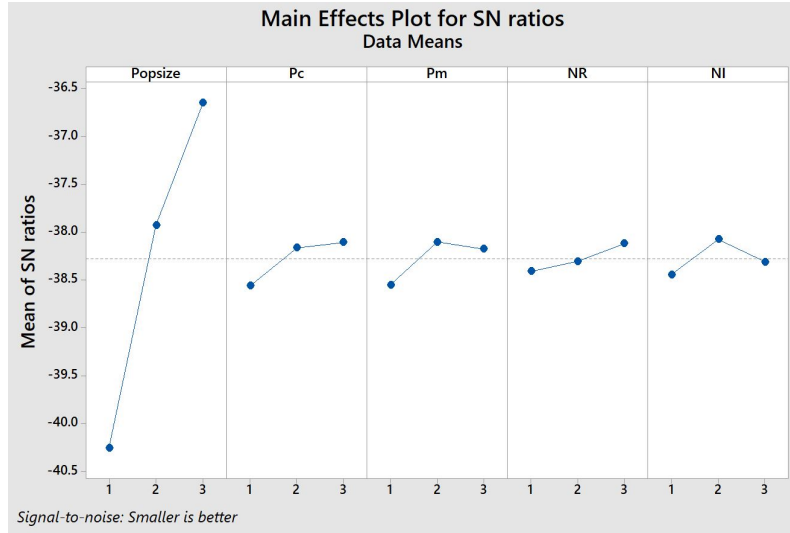


Figure 3.3: Taguchi test results.

the parameters N , P_c , and P_m will be considered for CGA and RGGA. Besides, the termination criterion for all algorithms is 20 iterations with no fitness function improvement.

3.5.2 Validation of GA Algorithms

To evaluate the efficiency and accuracy of the GA algorithms in finding the optimal solution, we compare results solved using IBM ILOG CPLEX Optimization Studio version 12. All algorithms are implemented in MATLAB released 2017, and tested on a personal computer with Intel Core i7 (2.0 gigahertz) processor and 8 gigabyte RAM. We validate all GA algorithms using CPLEX, but only show the IRGGA results here. Table 3.5 presents the obtained results (i.e., the maximum load of the found optimal solution and the CPU time) for both CPLEX and IRGGA. In Table 3.5, the *mean* and Standard Deviation (*STD*) are obtained by running the IRGGA algorithm 15 times using different seed values. Problem 1 in Table 3.5 is defined as

Table 3.5: IRGGA and CPLEX Comparison on small size CAPPAs instances (CPU.T represents CPU Time).

No	Problem Features	Result Cplex	CPU.T CPLEX	IRGGA Result (<i>mean,STD</i>)	IRGGA CPU.T (<i>mean,STD</i>)
1	5 * 7 * 3 * 3 * 4 * 20	146	3.1	(146.15,0.53)	(5.07,0.82)
2	6 * 7 * 4 * 4 * 5 * 20	172	24.2	(172.61,1.21)	(4.9,0.99)
3	6 * 10 * 5 * 4 * 5 * 20	208	73.1	(208.61,1.21)	(4.92,0.8)
4	7 * 10 * 6 * 4 * 5 * 20	212	106.3	(212.46,1.15)	(9.69,0.99)
5	8 * 10 * 6 * 4 * 5 * 20	228	163.4	(228.92,1.68)	(14.84,1.4)
6	9 * 10 * 7 * 4 * 5 * 20	204	291.7	(206.3,1.03)	(16.07,1.93)
7	10 * 10 * 8 * 4 * 6 * 20	232	542.1	(232.6,2.13)	(17.23,1.7)
8	15 * 15 * 5 * 4 * 8 * 20	-	-	(1923.92,478.6)	(44.46,2.4)
9	20 * 20 * 5 * 4 * 30 * 20	-	-	(2867.61,1.21)	(127.93,6.77)

5 * 7 * 3 * 3 * 4 * 20 which describes a problem with 5 orders, 7 layers at most for each order, 3 machines, 3 periods of time of a planning period, 4 reticles (r_1, r_2, r_3 and r_4) and each reticle can be used 20 times ($\delta_r = 20$), respectively. Based on the obtained results, it is clear that by increasing the size of the model we can see a rise in CPLEX computation time (CPU time seconds (CPU.T)). As illustrated in test problems 8 and 9, CPLEX failed to obtain a final solution in 1000 seconds. However, IRGGA in addition to its ability to find the exact optimal solutions in a reasonable time (except for test problem 6 where it found a near-optimal solution), it has an interesting scalability feature regarding problem size. One can clearly notice in Table 3.5 that, contrarily to the exact CPLEX solver, IRGGA CPU time is increasing in a reasonable way relatively to the problem size.

3.5.3 Comparison of GA algorithms

In this section, firstly we introduce the different test problems based on data sets collected from a Bosch fab in Reutlingen, Germany. Then, IRGGA, RGGA, and CGA are used to solve these problems. The results are then analyzed to evaluate the performance and efficiency of different algorithms in solving CAPPAs.

Table 3.6: Test problem instances under Number of Machines ED (Table 3.7).

Number of Orders	Number of Layers	Processing Times	Number of Machines	Planning Period
57	$U[20, 25]$	$\text{Gamma}[16.98, 2.448]$	ED	3 Periods of time

Table 3.7: Empirical distribution (ED) for number of machines.

Number of Machines	7	8	9	10	11	12
Probability	0.02	0.05	0.1	0.28	0.32	0.23
Cumulative	0.02	0.07	0.17	0.45	0.77	1

Experiments Design

As Figure 3.4 shows, a three stage data-driven experiments design method is proposed. In the first stage, to obtain problem variable distributions, the data provided from Bosch fab is analyzed. In the second stage, different problems are generated using distributions obtained in stage 1. Third stage, modifies constant factors NC , CL_{it} and δ_r . There are five parameters in CAPP: number of orders, number of layers, processing times, number of machines and the planning period. These parameters are defined based on real photolithography data sets extracted from a Bosch fab. As Table 3.6 indicates, we consider 57 orders as a sample from data sets in 3 periods of time, with the number of layers found to be a uniform distribution between 20 and 25. Figure 3.5 shows the fitted distributions on processing time data sets. Although the Normal distribution can describe features of the processing times with a good estimation, it contains negative numbers. Therefore, a Gamma distribution is used which guarantees both positive numbers for processing times and a good quality of estimation. Moreover, due to different reasons (such as the periodic maintenance, non-predicated shut downs, etc.), the number of available machines in the photolithography area is not constant. Based on the collected data, we assume this parameter to follow an Empirical Distribution (ED). This assumption enables us to consider the number of available machines as a discrete parameter of integer numbers. For this purpose, a roulette wheel data selection is designed, which is described in Table 3.7, where the number of available machines and their related probabilities are presented.³

³Appendixes B.1 to B.7 sample an actual set of data created to test IRGGA.

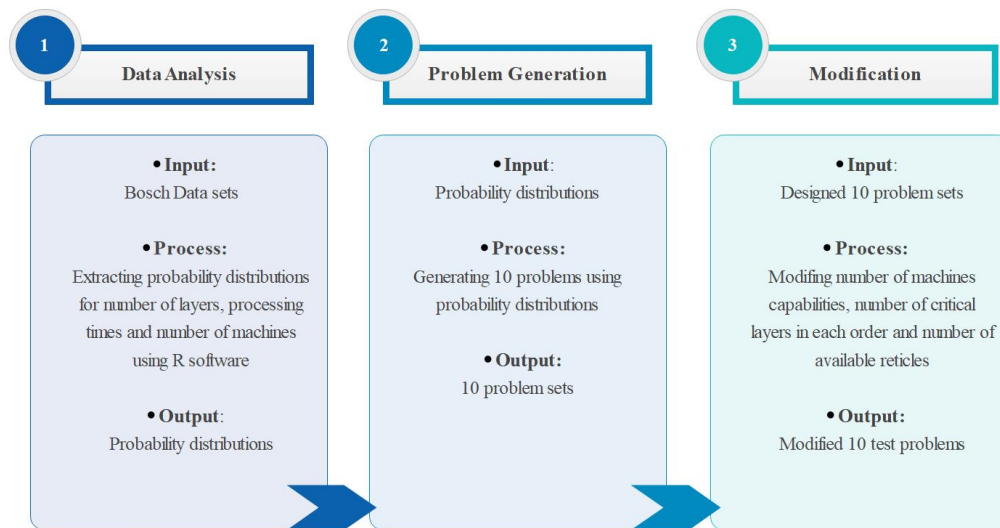


Figure 3.4: Experiments design diagram.

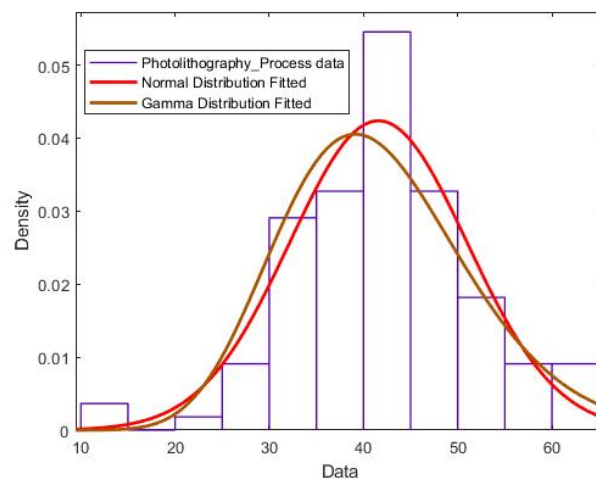


Figure 3.5: Bosch data set processing time distribution.

Comparative Experimental Results

To compare the effectiveness of IRGGA, RGGA and CGA, we conducted the designed experiments using the information presented in section 3.5.3. As previously mentioned, there are 3 different factors influencing CAPP which are process windows, machine dedication and reticle availability constraints. In this chapter, we note these factors as P_{Rate} (Process windows ratio), M_{Rate} (Machine dedication ratio) and R_{Rate} (Reticle availability ratio) for each problem. The P_{Rate} is equal to the cumulative capabilities of all machines divided by the number of capabilities if all machines are capable to produce all products. The P_{Rate} is defined as follows:

$$P_{Rate} = \frac{\sum_{k \in K} \sum_{h \in H} C_{kh}}{NM * NC} \quad (3.14)$$

where $C_{kh} = 1$ if machine k has capability h , 0 otherwise. NM and NC represent the total number of machines and capabilities, respectively.

The M_{Rate} is equal to the current number of critical layers divided by the number of all layers planned to be produced. The M_{Rate} is defined as follows:

$$M_{Rate} = \frac{\sum_{i \in I} \sum_{l \in L_i} CL_{il}}{\sum_{i \in I} NL_i} \quad (3.15)$$

where $CL_{il} = 1$ if layer l of order i is a critical layer, 0 otherwise. NL_i represents the number of layers for order i .

Finally the R_{Rate} is considered as the current total number of maximum reticles sharing between layers divided by the total number of layers. Note that the total number of maximum reticles sharing calculated by summing the maximum number of sharing between different layers for all reticles. The R_{Rate} is defined as follows:

$$R_{Rate} = \frac{\sum_{r \in R} \delta_r}{\sum_{i \in I} NL_i} \quad (3.16)$$

where δ_r is considered as the number of possible sharing of reticle r between different layers.

As shown in Table 3.6, to perform experiments both the number of layers for

each order and the number of available machines follow distributions. As factors P_{Rate} , M_{Rate} and R_{Rate} are influenced by the number of layers for each order and the number of available machines, they have probabilistic features and might vary between different problems. As shown in Table 3.8, to perform sensitivity analysis on CAPP results, we defined different rates for P_{Rate} , M_{Rate} and R_{Rate} . As previously mentioned in section 3.5.3, to have P_{Rate} , M_{Rate} and R_{Rate} fixed for a number of problems, we acted on the constant factors C_{kh} , CL_{il} and δ_r , that is, to examine influence of each factor in the CAPP objective function.

As previously mentioned, the objective function of the studied CAPP is to minimize the total maximum loads of machines in different time periods of the planning horizon. Here, we want to develop a measure that would provide an indication of how good the result is by measuring it against the load on the machine while ignoring the factors of process capability, machine dedication and maximum reticles sharing constraints. In addition, we assume that all machines are available during the planning horizon. It is an ideal overall measure of how good the maximum loads of the machines are. Machines Utilization (MU) formulation is defined as:

TITMA= Total Idle Time of Machines while they are Available.

TATM= Total Available Time of Machines.

$$MU = 1 - \frac{TITMA}{TATM} \quad (3.17)$$

When loads of machines are different, the machine(s) with the maximum load have the maximum utilization equal to 1 and all other machines have some idle time with utilization rate less than 1 in a planning period. In fact, an upper bound for MU in CAPP can be defined as the situation where all machines have the same work load, which can be defined as $MU = 1$. In other words, $MU = 1$ if and only if all machine loads are equal to the lower load bound below:

$$Lowerbound = \frac{\sum_{i \in I} \sum_{l \in L_i} P_{il}}{|NM|} \quad (3.18)$$

where p_{il} and NM define the processing time of layer l of order i and the number of machines, respectively. Consequently, for an arbitrary capacity allocation solution s , its Maximum Load (ML_s) Deviation from the *Lowerbound* ($MLDL_s$) can be defined as follows:

$$MLDL_s = \frac{ML_s}{Lowerbound} \quad (3.19)$$

Table 3.8 compares the obtained results related to the found optimal solutions and the running CPU times for all algorithms. Moreover, $MLDL$ values are presented by considering the mean ML value obtained by each algorithm and the lower bound of the related problem.

These results are the mean of values obtained in 15 runs of each algorithm on each problem. The presented $MLDL$ values indicate the quality of algorithms in being close to the *Lowerbound*. To illustrate, by solving problem $P1$ with factors $P_{Rate} = 0.69$, $M_{Rate} = 0.24$ and $R_{Rate} = 1.69$, we obtained amounts of 1.39, 1.42 and 1.42 $MLDL$ for IRGGA, RGGA and CGA, respectively.

In fact, due to the stochastic behavior of GAs, two separate runs of the same algorithm can come up with different results. Thereby, reporting results from one single run of each algorithm may mislead the drawn conclusions. Therefore, a statistical analysis is needed to make solid conclusions and to be certain that the obtained results are relevant and that the relation between compared algorithms is not reversed. In this chapter, a Statistical Hypothesis Testing (SHT) method is used to decide whether or not the difference between the indicated values (obtained results for 15 runs) for all algorithms is statistically significant. The null hypothesis for the proposed SHT is $\{H_0: \mu_1 - \mu_2 = 0\}$ and the alternative hypothesis is proposed as $\{H_1: \mu_1 - \mu_2 \neq 0\}$ where μ_1 and μ_2 are the mean values for compared algorithms. To choose the most suitable SHT method, firstly a normality test was performed.

As IRGGA, RGGA and CGA result sets for each problem were proven to follow the Normal distribution, we chose to use the t -test comparing two samples with different variances as it is the best method to perform an SHT between normal

Table 3.8: Comparative experiments results

Problem	Factor	IRGGA		RGGGA		CGA		
		Obj.	CPU.T	Obj.	CPU.T	Obj.	CPU.T	
P1	$P_{Rate} = 0.69$	<i>Mean</i>	7743.61	1429.99	7912.91	1303.17	7944.41	1526.14
	$M_{Rate} = 0.24$	<i>SD</i>	166.22	416.54	233.48	475.09	184.28	475.09
	$R_{Rate} = 1.69$	<i>MLDL</i>	1.39		1.42		1.43	
P2	$P_{Rate} = 0.69$	<i>Mean</i>	7821.14	1455.91	7992.1	1305.63	8016.63	1537.27
	$M_{Rate} = 0.36$	<i>SD</i>	221.08	432.33	186.61	226.23	257.68	486.89
	$R_{Rate} = 1.69$	<i>MLDL</i>	1.29		1.32		1.32	
P3	$P_{Rate} = 0.69$	<i>Mean</i>	7917.56	1424.22	7963.16	1338.14	7919.21	1612.03
	$M_{Rate} = 0.46$	<i>SD</i>	197.88	375.84	173.62	242.08	301.74	482.48
	$R_{Rate} = 1.69$	<i>MLDL</i>	1.45		1.46		1.45	
P4	$P_{Rate} = 0.73$	<i>Mean</i>	7706.55	1419.93	7913.03	1340.78	8007.88	1645.93
	$M_{Rate} = 0.46$	<i>SD</i>	256.29	379.97	185.41	242.88	399.92	403.83
	$R_{Rate} = 1.69$	<i>MLDL</i>	1.38		1.40		1.42	
P5	$P_{Rate} = 0.61$	<i>Mean</i>	7891.74	1423.71	7986.69	1398.27	8238.77	1581.54
	$M_{Rate} = 0.46$	<i>SD</i>	279.17	374.29	239.86	262.62	412.52	363.25
	$R_{Rate} = 1.69$	<i>MLDL</i>	1.45		1.47		1.5	
P6	$P_{Rate} = 0.69$	<i>Mean</i>	7670.87	1427.98	7920.55	1292.72	7844.37	1523.75
	$M_{Rate} = 0.24$	<i>SD</i>	180.4	418.52	190.57	216.97	234.99	335.17
	$R_{Rate} = 1.7$	<i>MLDL</i>	1.39		1.43		1.42	
P7	$P_{Rate} = 0.69$	<i>Mean</i>	7961.95	1419.7	8055.54	1342.74	8297.12	1643.85
	$M_{Rate} = 0.24$	<i>SD</i>	403.6	370.13	199.76	249.78	291.88	329.8
	$R_{Rate} = 1.5$	<i>MLDL</i>	1.47		1.47		1.51	
P8	$P_{Rate} = 0.61$	<i>Mean</i>	7767.33	1470.79	7977.16	1320.14	8006.77	1559.94
	$M_{Rate} = 0.24$	<i>SD</i>	216.52	429.31	251.6	240.9	284.8	328.27
	$R_{Rate} = 1.7$	<i>MLDL</i>	1.39		1.43		1.42	
P9	$P_{Rate} = 0.81$	<i>Mean</i>	7928.93	1415.38	8052.12	1336.51	8114.98	1658.33
	$M_{Rate} = 0.24$	<i>SD</i>	387.84	368.37	329.06	255.17	475.92	281.03
	$R_{Rate} = 1.5$	<i>MLDL</i>	1.47		1.47		1.48	
P10	$P_{Rate} = 0.79$	<i>Mean</i>	7733.39	1415.65	7800.04	1331.11	7863.33	1662.69
	$M_{Rate} = 0.24$	<i>SD</i>	408.54	360.41	314.07	257.4	361.53	341.96
	$R_{Rate} = 1.9$	<i>MLDL</i>	1.41		1.42		1.43	

samples (Gentle, Härdle, and Mori 2012). The results of the two-sample t -test under confidence levels of 95% and 97.5 % for each pair of algorithms are shown in Table 3.9 using + and – symbols which represent rejecting the null hypothesis (significant difference between algorithms results) and failing to reject the null hypothesis (non-significant difference between algorithms results), respectively.

To illustrate, instead of the obtained results from solving problems $P3$ and $P10$ all other problem results indicate the significance of the difference between the objective function values of IRGGA and RGGA (under the significance level of 95%), which shows performance of IRGGA in comparison to RGGA in 80% of problems under the confidence level of 95%.

Based on the obtained results, we can notice the competitiveness and superiority of IRGGA over RGGA and CGA in solving CAPPA on different problem instances. To clarify the reasons behind these results, here we describe some of IRGGA special features and compare it with RGGA and CGA:

1. The heuristic approach to seed the initial random population in IRGGA enables the algorithm to start with solutions with a higher quality than RGGA and CGA which are based on random initialization. Besides, the main difference between IRGGA and RGGA is the implementation of the GH in population initialization. That is showing the quality of GH in improving the initial population as well.
2. The imitation and avoidance operators in IRGGA enables it to search only in the feasible area. In fact, the first population for IRGGA, RGGA and CGA algorithms is based on feasible solutions. Then, the algorithm operators play a key role in remaining in the feasible area. The imitation operator in IRGGA changes allocations for a specific gene of each chromosome in two solutions, where the feasible allocation of layers from one solution is copied into another one. Therefore, it guarantees the feasibility for the new solution. In CGA, the solutions feasibility is also considered but it should be checked for every new chromosome due to the crossover and mutation procedures.

Table 3.9: *t*-test results

<i>Problem</i>	Algorithm	Significance level= 95%		
		IRGGA	RGGA	CGA
<i>P1</i>	IRGGA	*	+	+
	RGGA	+	*	-
	CGA	+	-	*
<i>P2</i>	IRGGA	*	+	+
	RGGA	+	*	-
	CGA	+	-	*
<i>P3</i>	IRGGA	*	-	-
	RGGA	-	*	-
	CGA	-	-	*
<i>P4</i>	IRGGA	*	+	+
	RGGA	+	*	-
	CGA	+	-	*
<i>P5</i>	IRGGA	*	+	+
	RGGA	+	*	+
	CGA	+	+	*
<i>P6</i>	IRGGA	*	+	+
	RGGA	+	*	+
	CGA	+	+	*
<i>P7</i>	IRGGA	*	+	+
	RGGA	+	*	+
	CGA	+	+	*
<i>P8</i>	IRGGA	*	+	+
	RGGA	+	*	-
	CGA	+	-	*
<i>P9</i>	IRGGA	*	+	+
	RGGA	+	*	-
	CGA	+	-	*
<i>P10</i>	IRGGA	*	-	+
	RGGA	-	*	+
	CGA	+	+	*

The checking function adds a high computational cost to the algorithm which can be considered as one of the most important disadvantages of CGA in comparison to IRGGA.

3. IRGGA saves, all over the search progress, a set of best and worst obtained solutions. However, CGA loses this information. This helps the algorithm converge to the optimal region and find the optimal solution faster.

CAPPA Sensitivity Analysis

In this section we provide some practical analysis on CAPPA sensitivity to its main constraints by considering different levels of P_{Rate} , M_{Rate} and R_{Rate} in P1. It is clearly noticed in Figure 3.6 that, by increasing the process flexibility (P_{Rate}) there is a significant improvement in the maximum load of machines since it is a relaxation of the process windows constraint. In contrast, increasing the number of critical layers (M_{Rate}) increases the maximum loads as it makes the problem more restricted. In addition, by increasing the sharing level of reticles there is a significant improvement in maximum loads of machines which is due to the increase of the number of available reticles.

Figure 3.7 plots the objective function values for problems 1 to 5 (P1,..., P5), presented in Table 3.8 with a fixed $R_{Rate} = 1.69$. As Figure 3.7 shows, in points $P4$ ($M_{Rate} = 0.46$, $P_{Rate} = 0.73$) and $P3$ ($M_{Rate} = 0.46$, $P_{Rate} = 0.69$), we achieved the lowest and highest objective function values, respectively. Simultaneously, it is clear that by increasing the P_{Rate} from 0.69 to 0.73 and increasing the M_{Rate} from 0.36 to 0.46 at the same time (Moving from $P1$ to $P4$), there is a little change in the objective function value. This is due to the fact that although increasing the number of critical layers in CAPPA is a source of reaching the maximum loads, increasing the flexibility of machines can reduce the impact of critical layers. Stated differently, in a fab increasing the eligibility of machines results in reducing the sensitivity of the photolithography to the changes in layers features.

Figure 3.8 plots the objective function values for problems 6 to 10, presented in

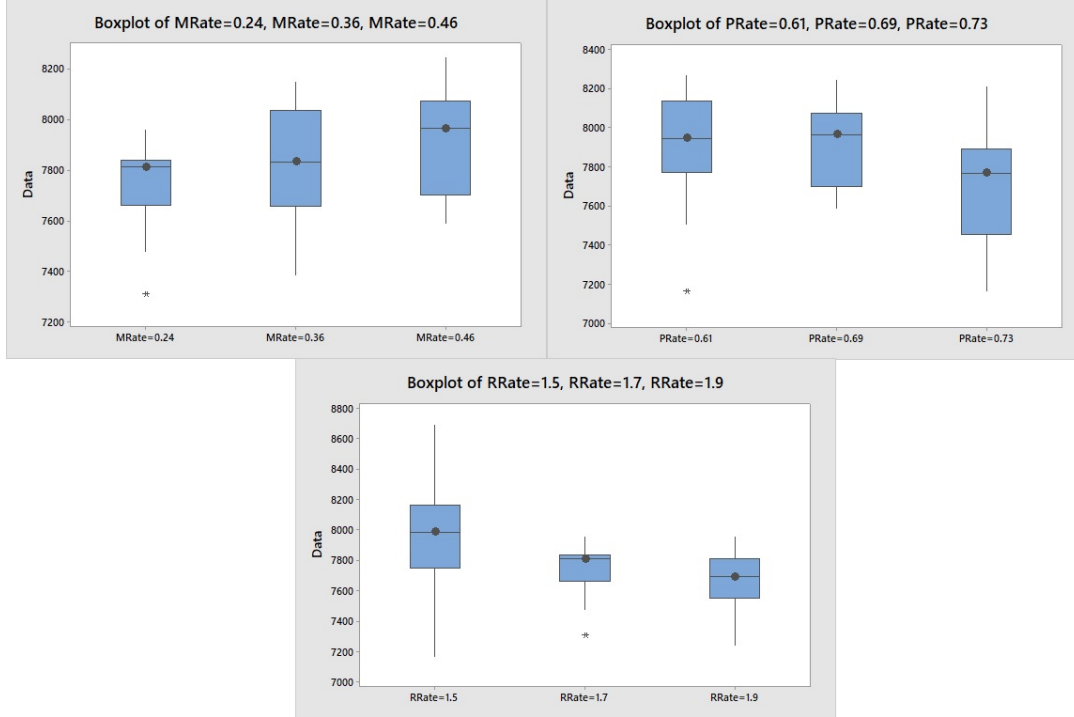


Figure 3.6: Maximum load boxplots variation with M_{Rate} , P_{Rate} and R_{Rate} changes for problem 1.

Table 3.8 with $M_{Rate} = 0.24$. As Figure 3.8 shows, in points $P10$ ($P_{Rate} = 0.79$, $R_{Rate} = 1.9$) and $P7$ ($P_{Rate} = 0.69$, $R_{Rate} = 1.5$) we achieved the lowest and highest objective function values respectively. By considering Figure 3.8, the sensitivity of the objective function to reducing maximum reticles sharing is proved to be higher than its sensitivity to increasing the machines flexibility. It is due to the fact that in modern photolithography fabs a sufficient number of reticles should be available and it is more critical than having fully flexible tool. To illustrate, by increasing the P_{Rate} from 0.61 to 0.69 and decreasing the R_{Rate} from 1.7 to 1.5 (moving from $P8$ to $P7$) there is a significant rise in the maximum loads level.

3.6 Conclusions

In this chapter, we proposed a GA to solve the strongly NP-hard capacity allocation problem, named CAPP, that considers all critical features (i.e., process windows, machine dedication and maximum reticles sharing constraints). To solve the proposed problem, we developed a GA that we called IRGGA, based on RGGA, a GA

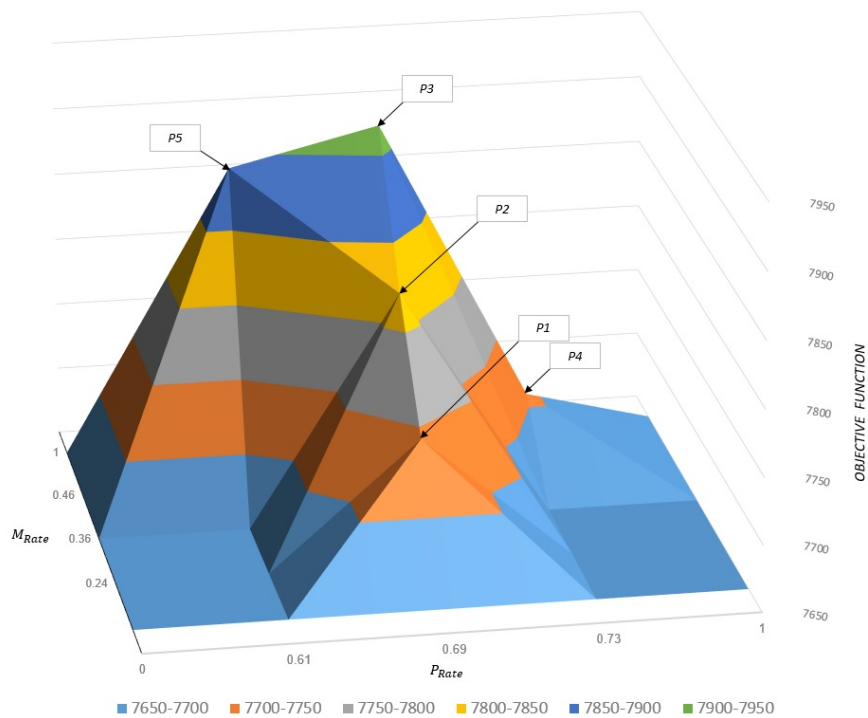


Figure 3.7: Analysis of CAPP sensitivity to P_{Rate} and M_{Rate} .

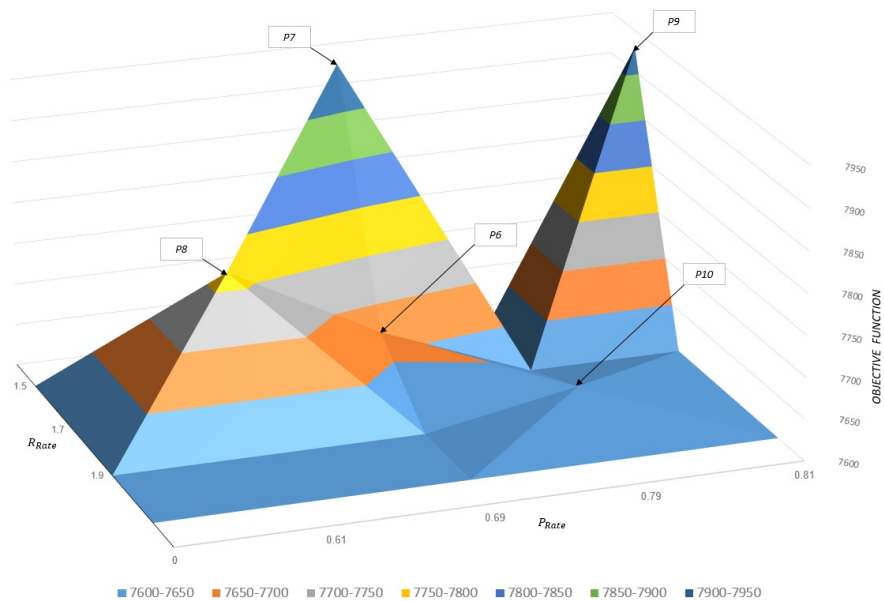


Figure 3.8: Analysis of CAPP sensitivity to P_{Rate} and R_{Rate} .

proposed by Beheshtinia, Ghasemi, and Farokhnia (2017). In IRGGA, we improved RGGA by implementing a heuristic inside the initialization procedure. Moreover, Beheshtinia, Ghasemi, and Farokhnia (2017) provided just the concept of crossover and mutation operators using imitation and distinguish procedures, while they did not provide detailed information about operator structures. We implemented these genetic operators inside IRGGA and detailed operator information was provided. To assess the quality of IRGGA, we compared it against an exact solver on small-sized problem instances. Furthermore, a comparison between IRGGA and two other GAs proposed in the literature using different sets of experiments extracted from a Bosch fab is presented. Our computational experiments have shown that IRGGA can achieve high quality solutions on both small and large sized CAPPA problem instances.

Moreover, we found that raising the process flexibilization in photolithography fabs that are producing layers with different processing times and machine capability constraints, while considering all other factors in a steady state, decreases the level of machines maximum load. As well, by increasing the sharing of reticles while all other factors are stable, we noticed a significant decrease of the maximum load level. In contrast to the mentioned factors, increasing the number of critical layers results in a rise in the maximum load of machines. Our experiments showed that although increasing process flexibilization has an impact on reducing the machines maximum loads, the maximum loads are more sensitive to reducing the maximum reticles sharing level. In other words, we found that by simultaneously increasing machine flexibilization and decreasing the maximum of reticles shared, there is a noticeable increase in machine loads. Finally, based on our experiments, we concluded that increasing the eligibility of machines (process windows constraints) in a fab results in reducing the sensitivity of the tool set to changes in jobs, such as increasing the level of critical layers.

As mentioned in a review of SO methods with applications to semiconductor operational problems presented in Ghasemi, Heavey, and Laipple (2018), recent ad-

vances in SO research and explosive growth in computing power have made it possible to optimize complex manufacturing system problems using SO. Consequently, in next chapters we would like to apply SO concepts in solving photolithography scheduling problems.

Chapter 4

Estimating Production Planning Performance in Jobshop Production Environments Under Uncertainty by Genetic Programming- Case Study: Robert Bosch

4.1 Introduction

Faced with the growing challenges of global markets, manufacturing systems must become more flexible, efficient, and reliable. To achieve this requires changes to both manufacturing plants and the information flows that support production. However, the problem of designing improved manufacturing systems is not trivial, since the task of predicting the performance of the proposed system is complex. An important tool is Discrete Event Simulation Models (DESMs) used as a predictor of performance, allowing examination of the likely behavior of the proposed manufacturing

system under experimental conditions (Trigueiro de Sousa Junior et al. 2019). While DESMs do not directly provide explanations for the observed system behavior, it is essentially a trial and error methodology, and although attention to experimental design techniques enhances its value, it does not provide a method of optimization. On the other hand, optimization techniques are key tools to improve decisions within almost all systems. Integrating simulation models with optimization methods could establish promising decision support tools benefiting from the advantages of both tools. That is the main idea to support proposing SOs for different complex and stochastic industrial problems (e.g., SJSSPs). Depending on the model being built using DESMs, carrying out a set of experiments can be potentially expensive in terms of computer time (Can and Heavey 2012). Therefore, the use of metamodels has been proposed to alleviate some of these problems (Kelleher et al. 2018).

However, to the best of our knowledge, there is no research addressing how to train a metamodel aiming to integrate within SOs and replace highly computational time-intensive simulation replications. Figure 4.1 summarises the main differences between typical evolutionary SOs and evolutionary SOs integrated with metamodels. Within typical evolutionary SO techniques, firstly the initial population consisting of pop solutions is created. Next, the fitness value for each solution is calculated using SL simulation replications. Using recombination techniques (e.g. mutations), a new set of solutions from the initial population is created. By considering the number of new solutions equal to the initial population size, $pop \times SL$ simulation replications are needed to calculate the fitness values of all solutions. Finally, using the proper *Selection* and *Stoppage* modules, this procedure is replicated until a termination criteria is achieved. The number of simulation replications (SL) needed to calculate the fitness values accurately varies based on the level of complexity and stochasticity of the domain of interest. For instance, to calculate the objective value for each SJSSP solution accurately, $SL = 10^5$ simulation replications are needed (Horng, Yang, and Lin 2012; Yang, Lv, et al. 2014). Consider just 100 as the population size (pop), 100 the number of SO iterations and 0.001 the run length of a simulation

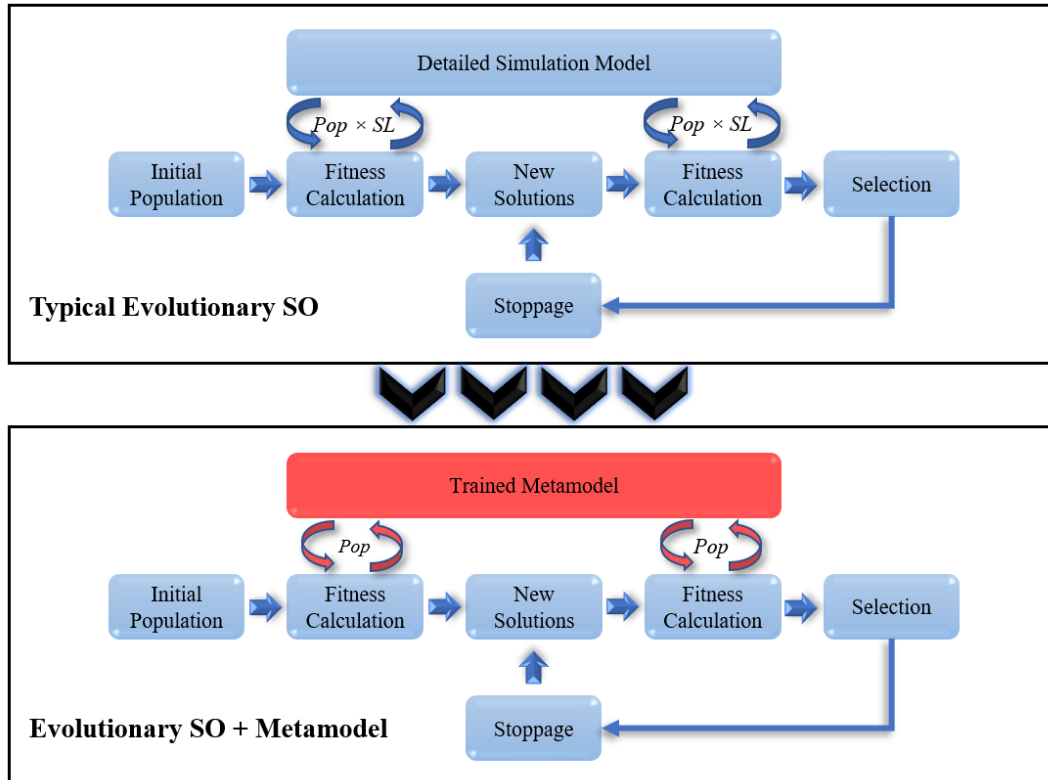


Figure 4.1: The difference between typical evolutionary SOs and evolutionary SOs integrated with metamodels.

replication in seconds, respectively. The amount of time required to run simulation optimization is equal to $((100 \times 10^5 \times 0.001) + (100 \times 100 \times 10^5 \times 0.001)) = 1010000$ seconds. Indeed, such a computation time amount is not reasonable to solve a SJSSP. This has motivated us to examine the replacement of simulation replications in evolutionary SOs by using accurate metamodels described in this research. As shown in Figure 4.1, this replacement would reduce the required computations for SO considerably. The question here is how to train and structure a metamodel to estimate fitness values accurately enough to replace simulation replications within SOs?

Scheduling jobs between a limited number of machines (sources) is one of the most challenging problems in production environments. Within the set of production environments found in the industry, many production facilities can be viewed as job shops (e.g., photolithography toolsets in the semiconductor industry) (Zhang, Ding, et al. 2019). That is why there is a considerable amount of research regarding

optimizing scheduling problems in job shops under a variety of objective functions (Vela et al. 2020). Although stochasticity has been known as a crucial part of most industrial operations, in the scheduling literature, there is little attention to solving Stochastic Job Shop Scheduling Problems (SJSSPs), while Deterministic Job Shop Scheduling Problems (DJSSPs) have been widely researched (Mokhtari and Dadgar 2015). While DJSSPs are extremely complex problems to solve, introducing stochastic parameters increases tremendously the solution space of SJSSPs. To the best of our knowledge, Simulation Optimization (SO) techniques have been an important proposed tool in the literature on optimizing SJSSPs. In general, SO methods are structured where the optimizer(s) are integrated with simulation experiments (Ghasemi, Heavey, and Laipple 2018). As mentioned before, simulation experiments require a high level of computational power, with most of the research in proposing SOs to solve SJSSPs dealing with this issue by reducing the number of simulation replications (Yang, Lv, et al. 2014). This means sacrificing some stochastic scenario explorations to solve a SJSSP in a reasonable amount of computational time, with sometimes no information about the neglected scenarios. Therefore, an efficient estimation methodology could be highly advantageous. In other words, an efficient DESM estimation method with accurate and robust objective value estimations to calculate stochastic scenarios would be advantageous.

In this chapter, our prime objective is to investigate how to best train Genetic Programming (GP)-based metamodels trained by SJSSP DESM instances to replace simulation replications within SOs. Training in statistics and machine learning refers to the discovery of predictive relationships. To develop a metamodel using GP, a set of SJSSP DESM replications are sampled to create a training set used to create training vectors. This is where the art of data translation is needed to convert raw data sets (here DESM replications results) to trainable metamodel inputs (training vectors). Then, the created training vectors are used as input to a metamodeling technique. Can and Heavey (2012) introduced GP in the metamodeling of stochastic problems to evolve symbolic expressions. Besides, they compared the quality of

GP in metamodeling simulation models in comparison with Neural Network (NN). As the GP-based metamodeling technique presented in this research is aiming to replace simulation replications within SO methods, we evaluate our metamodeling approach by inserting the GP-based metamodel within an evolutionary optimization algorithm and examining its accuracy in finding elite solutions in different algorithm generations.

The training vectors are evaluated within an evolutionary optimizer setting using representative data from the front end Bosch photolithography toolset. The toolset data set consists of 929,178 rows of data. Firstly, we retrieve important and relevant information about data using classification methods. This helped to classify data into different classes, with patterns for each data class identified. After analyzing this complex data set using Rstudio software (*RStudio | Open source & professional software for data science teams 2020*), we captured track in and track out times to the photolithography department for each job in the data set. Then distributions were fitted to SJSSP instances. The motivation for this experimentation of the GP metamodel is that the metamodel will be used to replace simulation within a SO evolutionary algorithm. Thus, we wanted to evaluate how the metamodel would operate within this optimization context.

In summary, this chapter provides:

- A new implementation of GP in metamodeling SJSSP simulation models to be used in SO techniques;
- An evaluation of three training vectors for metamodeling GP from a DESM of a SJSSP;
- An experimental study within the context of evolutionary algorithms using representative data from a real case study;
- A comprehensive sensitivity analysis of the accuracy of GP to metamodel SJSSPs within SO;

The remainder of this chapter is organized as follows. Section 4.2 provides a literature review on metamodeling DESMs primarily used in SO applied to SJSSPs. Section 4.3 presents a mathematical formulation for the problem while Section 4.4 describes the three training vectors used to create the GP metamodel. Section 4.6 describes the experiments used to evaluate the three training vectors. Finally, Section 4.7 concludes the chapter and examines further proposed research.

4.2 Literature Review

Running simulation models can be quite expensive in terms of computational time (Barton and Meckesheimer 2006). This holds especially true in the case of optimization algorithms that require a high level of computational time to replicate simulation. Simulation metamodels offer a trade-off between the accuracy and efficiency needed for simulation analysis. Simulation metamodels try to predict the input-output relations of a simulation model through another functional model to provide robust and fast decision support (Sabuncuoglu and Touhami 2002). To metamodel DESMs, researchers have presented a variety of methods and concepts, such as Radial Basis Functions (RBFs) (Hussain, Barton, and Joshi 2002), Kriging (KG) metamodeling (Kleijnen 2009), while Neural Network (NN) has been known as the predominant approach to metamodel DESMs (Dunke and Nickel 2020). Recently, Kleijnen (2017) conducted a literature review considering papers addressing Regression and Kriging metamodels mainly focused on regression and Kriging metamodeling applications in simulation aimed at sensitivity analysis and optimization of real systems.

In an earlier study, Kilmer, Smith, and Shuman (1997) studied NNs metamodeling in an inventory control problem. In another study, Yildiz and Eski (2006) developed neural network-based simulation metamodels of assembly lines to identify optimal design and operational parameters. In a later research, Kleijnen (2008) investigated Generalized Response Surface Methodology (GRSM) simulation metamodeling to extend the Response Surface Methodology (RSM) proposed by Box and

Wilson (1951). To verify their approach, they used the statistical analysis presented by Bettonvil, Castillo, and Kleijnen (2009). To investigate queuing system transient behavior, Yang and Liu (2012) implemented RBFs to metamodel a production system's DESMs. As an extension to this, Li et al. (2016) researched the input/output relationships in manufacturing systems through a regression based DESM metamodeling methodology. In another study, Chen and Zhou (2017) proposed the stochastic KG to metamodel simulation models. They attempted to design efficient sequential design strategies to implement stochastic KG for mean response prediction with a fixed simulation budget. Intrinsic KG is firstly presented by Mehdad and Kleijnen (2018) to overcome classical KG flaws such as dependence on estimation of the trend in the input-output data to metamodel a DESM. Most statistical metamodeling techniques such as KG are computationally complex and time intensive. Thus, there has been little research on their real case and large size applications. In a recent research, Dunke and Nickel (2020) applied a NN metamodeling technique to an order picking system DESM. However, they concluded that NN-based simulation metamodels are not capable of encompassing a statistically secured selection of parameters and operational control strategies within their experiments.

The use of GP for generating symbolic representations of training data was first proposed by Koza and Koza (1992). Through the use of effective convergence algorithms, GP can produce compact expressions which have excellent generalization properties (Murphy and Ryan 2008). Such representations are often mathematical expressions, i.e., functions, unlike metamodels generated by Artificial Neural Networks (ANNs). These explicit functions can provide an alternative platform to metamodel DESMs in practice when the trade-off between accuracy and computation requirements is appropriate. Can and Heavey (2012) compared both GP and ANN in metamodeling DESMs. For the case studies considered the results showed that GP outperforms ANN in metamodeling DESMs. Wu, Chou, and Su (2008) presented a GP method evolving symbolic regression expressions for generating regressive models for direct transformation from Global Positioning System (GPS)

signals to 2-D coordinates. In another study, Barmapalexis et al. (2011) used GP in the optimization of a pharmaceutical zero-order release matrix tablet. They compared their approach with an ANN. Recently, Amir Haeri, Ebadzadeh, and Folino (2017) improved the classical GP for symbolic regression by using statistical information to generate some modified subtrees.

The main use of GP in metamodeling is to find a symbolic regression tree corresponding to the mathematical formula that best fits the data according to a fitness criteria. Fitting such a model is done in an optimization framework where the error (e.g., MRE (Mean Relative Error)) of the generated symbolic trees against sample data is minimized numerically via regression. Therefore, it is inherently suitable to perform simulation-based metamodeling of complex industrial systems, yet it is rarely applied (Can and Heavey 2011).

Due to the highly complex nature of SJSSPs, with their high level of covariations between their parameters and/or variables, the following research has been carried out. Azadeh, Negahban, and Moghaddam (2012) metamodeled SJSSP using neural networks. They used the metamodel to estimate the makespan value associated with possible permutations of priority rules in order to find the set with the optimum makespan value. In another study, Vinod and Sridharan (2009) developed a regression-based metamodel for scheduling SJSSP in which setup times were sequence-dependent. For the sake of simplicity, the scheduling decision rules and the setup time relations were considered as independent variables. They used a metamodel to evaluate five new setup-oriented scheduling rules against seven rules from the literature. In other works, due to the complexity of the SJSSP DESMs, a low number of replications were used with estimates of the objective values in developing metamodels. Horng, Lin, and Yang (2012) and Yang, Lv, et al. (2014) estimated SJSSP objective values within an SO method using 368 simulation replications. However, none of these two papers addressed the accuracy of their simulation replication reduction approach in estimating the SJSSP objective values. In other words, they did not provide information on the neglected simulation scenarios

while deducting simulation replications. In a more recent research, Wu, Yu, and Li (2019) addressed the SJSSP problem where machine breakdowns occurred. They considered a data-driven response surface method to metamodel the SJSSP using a makespan objective function.

Although recent applications of SO methods in solving complex stochastic problems such as SJSSPs have provided promising outcomes, they still suffer from high computation complexity due to the use of simulation replications within their algorithm. On the other hand, GP-based simulation metamodels have been developed recently to replace simulation replications and deal with the simulation time intensity while providing accurate estimations of simulation outputs (Can and Heavey 2012). However, to the best of our knowledge, there is a limited number of GP-based simulation metamodels applications within SO techniques proposed to solve complex stochastic problems such as SJSSPs. Therefore, in this chapter, we address this gap by proposing GP-based simulation metamodels to be used within SO methods to solve SJSSPs.

4.3 Mathematical Model for Optimization of SJSSP

In this section, the mathematical model of SJSSP is detailed. Table 4.1 summarises the notation used to define the model. Before introducing the model formulation, the following example is considered to illustrate the mathematical model of the SJSSP. Let J refer to the job set, O_j the operation set for job j , with M the machine set. The vector (i, M_i, P'_i) defines the stochastic processing time P'_i of operation i on the planned machine M_i . As shown in Figure 4.2, there are $N = 3$ jobs with $NO_1 = 3$, $NO_2 = 3$, and $NO_3 = 1$ operations for job $j = 1, 2$ and 3 , respectively. Take for example vector $(5, M_1, 3)$ from problem inputs (operations environment), this refers to operation number 5, planned to be assigned to machine 1, while its processing time equals 3, noting that for illustrative purposes we use deterministic times. Operation 5 (note that we sequentially number the operations across all jobs in this chapter) is the second operation of job 2, which is assigned to the 3rd position

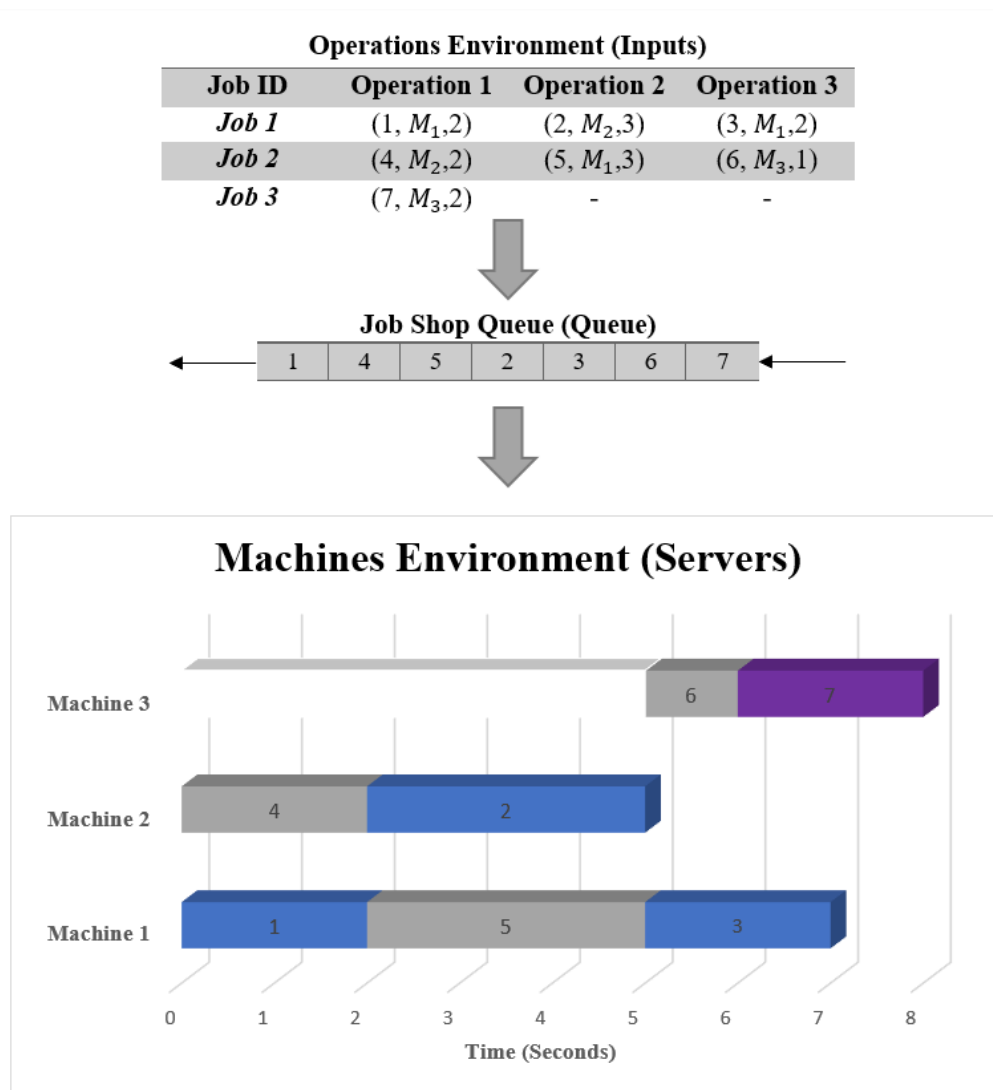


Figure 4.2: Proposed Job Shop Scheduling Problem Example.

of the queue (see Figure 4.2). Therefore, as it is the second operation of job 2, and observing precedence constraints, it must follow the first operation (4) of job 2. Thus, we denote that $X_{53} = 1$; the values we use in the mathematical formulation presented below. The “Job Shop Queue (Queue)” in Figure 4.2 are the inputs used in evaluation models.

The goal of scheduling all operations of N jobs on NM machines is to minimize the expected value of total costs defined by Equation 4.1. Where, SL defines the total number of simulation replications indexed by s . In other words, both tardiness and earliness are considered as cost sources in the proposed SJSSP. This is due to the fact that tardiness of jobs causes customer dissatisfaction, imposing CT_j units

Table 4.1: Notations table.

Indices and Sets	
$j =$	Jobs index, $j \in J = \{1, \dots, N \}$.
$i, i' =$	operations indexes, $i, i' \in V = \{1, \dots, NO \}$.
$O_j =$	operations sets for each job j , $O_j \in V$.
$m =$	Machine index, $m \in M = \{1, \dots, NM \}$.
$k =$	Queuing position index, $k \in K = \{1, \dots, NO \}$.
$\Omega =$	Precedence orders sets defining the execution precedence of operations of the same jobs.
$s =$	Simulation replication index, $s \in S = \{1, \dots, SL \}$.
Parameters	
N	Number of jobs.
NO	Total number of operations.
NO_j	Total number of operations for job j .
NM	Number of machines.
SL	Total number of simulation replications indexed by s .
P'_i	Stochastic processing time of operation i .
ϕ_i	Probability distribution of processing time of operation i .
CE_j	Earliness cost of job j caused by inventory costs.
CT_j	Tardiness cost of job j caused by tardiness in delivering the job.
Decision Variables	
X_{ik}	If operation i is assigned to the k^{th} position of the dispatching queue, then $X_{ik} = 1$, 0 otherwise.
T'_{js}	The stochastic tardiness time of job j in the simulation replication s .
E'_{js}	The stochastic earliness time of job j in the simulation replication s .
Functions	
f_s	The objective calculation function for the simulation replication s .

of cost for job j in a solution. In parallel, the earliness of jobs imposes a holding cost of CE_j in a solution for each unit time of earliness of job j .

$$\text{Min } F = \text{Min} \left(\frac{1}{SL} \sum_{s=1}^{SL} \left(\frac{1}{N} \left(\sum_{j=1}^N (T'_{js} \times CT_j) + (E'_{js} \times CE_j) \right) \right) \right) \quad (4.1)$$

Equation 4.1 is transformed into Equation 4.2, for use with evaluation models, where f_s calculates total costs of solution X_{ik} using simulation replication s . In addition, F defines the fitness value for a SJSSP solution.

$$\text{Min } F = \text{Min} \left(\frac{1}{SL} \sum_{s=1}^{SL} f_s (\{\cup_{i=1}^{NO} \cup_{k=1}^{NO} X_{ik}\}, P'_i) \right) \quad (4.2)$$

In each solution, X_{ik} , is a binary variable that denotes if operation i is assigned to the k^{th} position of the dispatching queue, with the above objectives having the following constraints. Each job should be assigned to one of the existing dispatching queue positions as follows:

$$\sum_{k=1}^{NO} X_{ik} = 1, \forall i = \{1, \dots, NO\} \quad (4.3)$$

Moreover, it must be guaranteed that for each existing position in dispatching queue k , at most one operation is assigned. Equation 4.4 defines this constraint as follows:

$$\sum_{i=1}^{NO} X_{ik} = 1, \forall k = \{1, \dots, NO\} \quad (4.4)$$

Precedence of operations of a job are defined in the set Ω . Consequently, Equations 4.5 and 4.6 construct a precedence relationship between two operations i and i' from the same job j in the SJSSP. In other words, when operation i (assigned to the position k) precedes operation i' , operation i' must be assigned to a position k' ($k' > k$) on the dispatching queue.

$$X_{ik} \geq X_{i'k'}, \forall i, i' \in \Omega, k < k' \quad (4.5)$$

$$X_{ik} - (X_{ik} X_{i'k'}) \geq X_{i'k'}, \forall i, i' \in \Omega, k > k' \quad (4.6)$$

Equation 4.7 defines the decision variable within the model:

$$X_{ik} \in \{0, 1\} \quad (4.7)$$

All in all, Equations 4.3 to 4.7 guarantee the feasibility of the queue solution X_{ik} by considering both assignment and precedence constraints.

Algorithm 5 presents the simulation model procedure to calculate the objective value f for queue solution X in a simulation replication. It is worth mentioning that as queue solution X consists of NO queue positions filled by NO operations, the problem size based on queue solutions equals $NO!$. This proves the NP-Hardness of the proposed SJSSP even without considering stochastic features (Horng, Lin, and Yang 2012). While, since the operations' processing time fluctuates stochastically

in SJSSP, the completion times of jobs cannot be defined. Even if the sequence has been predefined using the queue solution X , it would become invalid due to the change of the operations' realised process times.

Algorithm 5: Simulation algorithm.

Inputs : X (queue solution), ϕ (processing times distributions), First (all first operations of jobs set), Last (all last operations of jobs set), N , NO , CT , CE

Output: f (fitness value)

begin

for $i = 1$ **to** NO **do**

┌ $-P'_i = Rand(\phi_i)$; % Create a random value for the processing time of
└ operation i from the probability distribution ϕ_i

- Assign operations on machines based on the sequence of operations on the queue solution X and define the vector $vec_i = (i, M_i, l, P'_i)$ for the stochastic processing time P'_i of operation i on l^{th} position of the planned machine M_i , for $i = \{1, \dots, NO\}$;

for $k = 1$ **to** NO **do**

┌ $-index = find(i || X_{ik} \geq 0)$;

if $vec_{index}(3) == 1$ **then**

┌ **if** $index \in First$ **then**

└ $-completion_{index} = P'_{index}$;

else

└ $-completion_{index} = P'_{index} + completion_{index-1}$;

else

┌ **if** $index \in First$ **then**

└ $-index2 = find(i || vec_i(3) = vec_{index}(3) - 1 \& vec_i(2) = vec_{index}(2))$;

└ $-completion_{index} = P'_{index} + completion_{index2}$;

else

└ $-index3 = find(i || vec_i(3) = vec_{index}(3) - 1 \& vec_i(2) = vec_{index}(2))$;

└ $-completion_{index} =$

└ $P'_{index} + Max\{completion_{index-1}, completion_{index3}\}$;

for $j = 1$ **to** N **do**

┌ $-Jcompletion_j = completion_{Last_j}$;

- $f = \sum_{j=1}^N ((Max\{0, Jcompletion_j - d_j\} \times CT_j) + (Max\{0, d_j - Jcompletion_j\} \times CE_j))$;

- Return f ;

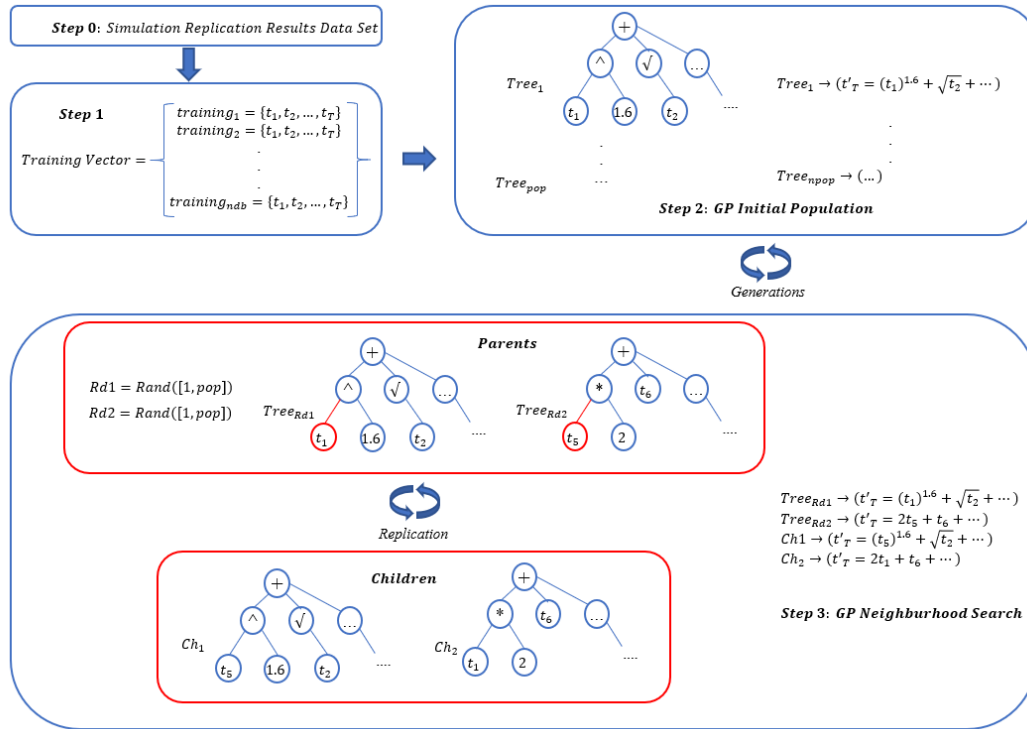


Figure 4.3: Proposed GP metamodeling procedure general framework.

4.4 SJSSP DESM Metamodeling Using Genetic Programming (GP)

As previously noted, SJSSP DESM replications are expensive in terms of computation time. Consequently, there is a necessity to introduce methods that reduce this computation time and provide accurate estimation of SJSSP DESM results. That enables researchers to replace time-intensive simulation replications in SO methods with metamodels and thus allocate computation budgets to the optimization core rather than simulation replications resulting in improved quality of SO solutions. In this chapter, we evaluate the quality of GP for metamodeling SJSSP DESMs.

The design of GP uses n -ary arithmetic functions, system decision variables, and evolutionary operators, such as crossover and mutation used to develop symbolic expressions. These symbols provide GP with the ability to learn a mathematical function that approximates the relationship between the input and the output using evolutionary mechanisms. This use of GP is referred to as symbolic regression.

Figure 4.3 shows the general framework of GP considered within this research.

In *Step 0*, *ndb* queue solutions and their objective function values are calculated using *SL* simulation replications. These values are used within Equation 4.2 to create an input data set (later we refer it as the Training Data Set (TDS)). In *Step 1*, the created data set is transformed to a set of vectors (*Training Vector*). In each *training vector*, features of a solution are defined using $T - 1$ elements (t_1, \dots, t_{T-1}). These elements of the training vector are used by GP to estimate a solution's objective value using a mathematical function. The last element of this vector (t_T) is the solution's objective value. Next, a population of random GP trees are created (*Step 2*). Note that GP uses a tree-based representation to evolve symbolic regression (e.g., $Tree_1 \rightarrow t_T = t_1^{1.6} + \sqrt{t_2} + \dots$). The quality in estimating objective values is defined by calculating the estimation error, where the estimated objective value t'_T is compared with t_T for each tree. Subsequently, through an iterative procedure (*Generations*), the initial population is improved (*Step 3*). That accrues by generating a set of offspring for the initial population using a neighbourhood search methodology in each iteration (*Replication*). In other words, in each generation of GP, a set of $2 \times Replication$ offspring trees are created from the initial population of trees. To illustrate the offspring creation procedure (see Figure 4.3) two arbitrary random solutions (parent trees) from the initial population are recombined by swapping their branches that create two new solutions named as children. This procedure replicates until $2 \times Replication$ are created in each generation. After creating a set of child solutions, all initial and child solutions are compared in terms of their error level in estimating t_T , and the best *pop* solutions are nominated as the next generation population. This procedure is replicated until a termination criteria is achieved (*Generation*).

The first element of all three training vectors are calculated similarly (t_1). In this research, t_1 is equal to the Delay Based Metric (DBM^c) for solution c . Assume that each operation has a stochastic processing time, where a sample is denoted by $D = \{d_1, d_2, \dots, d_{NO}\}$. Equation 4.8 calculates the DBM^c metric.

$$DBM^c = \frac{STD(D)}{Mean(D)} \quad (4.8)$$

The elements t_2, \dots, t_{T-1} are the core aspects evaluated within this research which are derived from the operations given in the queue and/or on machines for processing in the SJSSP (see Figure 4.2). The motivation for evaluating the three training vectors in this research are summarized as:

1. Queue Positions Based Learning Vector (QPBLV): According to the literature on SJSSP, the position of operations in the SJSSP queue provides valuable knowledge to estimate job shop performances (Hao, Lin, et al. 2013).
2. Machines Positions Based Learning Vector (MPBLV): Considering the objective value calculation procedure for SJSSP (see Algorithm 5), the position of an operation on machines has a direct impact on the completion time of both the considered operation and all its dependant operations. Therefore, in this rule we evaluate if an operation on a machine may provide sufficient knowledge to estimate the SJSSP objective value.
3. Machines Positions Interaction Based Learning Vector (MPIBLV): In SJSSP, there are two dependencies between operations: (i) between operations of the same job and (ii) between operations assigned to a same machine. There are also interactions between both dependencies. For instance, consider the SJSSP example in Figure 4.2, starting time of both operations 2 and 3 depends on the finishing time of operation 1 (dependency type (i)). Similarly, the starting time of both operations 5 and 3 depend on finishing time of operation 1 as they are all assigned to the same machine, and operation 1 is the first operation sequenced on machine 1 (dependency type (ii)). This has motivated us to examine if such dependencies can provide sufficient knowledge to predict objective values of SJSSPs.

In the following, all three mentioned training vectors are detailed.

4.4.1 Queue Positions Based Learning Vector (QPBLV)

Consider the vector of feasible queue solution c defined as $sol_c = \{a_1, a_2, \dots, a_{NO}\}$, where a_k is the operation in the k^{th} position on the queue for solution c . If a_k belongs to the first job, then $SP_{a_k}^c$ for solution c is calculated as follows.

$$SP_{a_k}^c = k \times a_k \quad (4.9)$$

If a_k is not from the first job in solution c :

$$SP_{a_k}^c = k \times \left(a_k - \sum_{j=1}^{L(a_k)-1} NL_j \right) \quad (4.10)$$

where, $L(a_k)$ defines the job type of operation a_k and NL_j the number of operations in job j . Thus, SPL_j^c for job j is calculated as follows:

$$SPL_j^c = \sum_{a_k \in Q_j} SP_{a_k}^c \quad (4.11)$$

where Q_j is the set of operations in job j . Consequently, GP is trained using the set $QPBLV_c$ as the learning set for solution c :

$$QPBLV_c = \{\{DBM^c\} \cup \{\cup_{j=1}^N SPL_j^c\} \cup F\} \quad (4.12)$$

Note that F defines the value of Equation 4.2 calculated using SL simulation replications for a solution. Considering the total number of ndb training solutions, the final training set $QPBLV$ can be defined as:

$$QPBLV = \{\cup_{c=1}^{ndb} QPBLV_c\} \quad (4.13)$$

Algorithm 6 summarises the $QPBLV$ calculation procedure. To illustrate the set of $QPBLV$ calculations, here we extend the SJSSP solution example provided in Figure 4.2. The example solution is considered as the first solution in the data set ($c = 1$). Considering that the processing time values in Figure 4.2 are samples

for all job operations, then using Equation 4.8 DBM^1 equals 0.2982.

SP values for all operations are calculated based on the queue sequence (start with operation 1, next operation 4, etc.). As operation 1 belongs to the first job, according to Equation 4.9, $SP_1^1 = 1 \times 1 = 1$. Likewise, SP_2^1 and SP_3^1 are equal to 8 and 15, respectively. As operation 4 does not belong to the first job, using Equation 4.10, $SP_4^1 = 2 \times (4 - 3) = 2$. Similarly, SP_5^1 , SP_6^1 , and SP_7^1 values are 6, 18, and 7, respectively. Then, using Equation 4.11, SPL_1^1 , SPL_2^1 , and SPL_3^1 values are equal to 24, 26, and 7, respectively. Consider that the due date of all jobs is equal to 1 and assume that both earliness and tardiness costs are equal to 1. Therefore, using Equation 4.2, the objective value F for the current example solution is equal to 18 ($SL = 1$). Finally, the $QPBLV$ training set of the proposed solution is defined in Equation 4.14.

$$QPBLV_1 = \{0.2982, 24, 26, 7, 18\} \quad (4.14)$$

4.4.2 Machines Positions Based Learning Vector (MPBLV)

As the schedule plan of operations on each machine plays a critical role in SJSSP objective values, here we present a GP training vector based on an operation's position on machines, with a distinction between operations from the first job and other jobs. This is defined using the vector $solS_k(sol_c(k), c, u, m)$, where $sol_c(k)$ is the operation in position k in the queue, c is the solution number and u is the position of $sol_c(k)$ on machine m . For all queue members, if $sol_c(k)$ belongs to the first job, then $SP_{sol_c(k)}^c$ for solution c is calculated as follows:

$$SP_{sol_c(k)}^c = solS_k(3) \times sol_c(k) \quad (4.15)$$

where $solS_k(3)$ is the 3rd position in vector $solS_k$. Then if $sol_c(k)$ is not from the first job in solution c :

Algorithm 6: QPBLV.

Inputs : ndb (total number of GP training and testing solutions), $First$ (all operations of the first job), Q (set of operations in each job), L (set of job identifications for operations), NL (a set defining number of operations in each job), N , NO , SL

Output: $QPBLV$

begin

for $c = 1$ **to** ndb **do**

- Create a vector of a random feasible queue solution (chromosome) and name it as $sol_c = \{a_1, a_2, \dots, a_{NO}\}$;
- Calculate the fitness value (F) using SL simulation replications in Equation 4.2) using the DESM of SJSSP;
- $DBM^c = \frac{STD(D)}{Mean(D)}$;

for $k = 1$ **to** NO **do**

if $a_k \in First$ **then**

$SP_{a_k}^c = k \times a_k$;

else

$SP_{a_k}^c = k \times (a_k - \sum_{j=1}^{L(a_k)-1} NL_j)$;

for $j = 1$ **to** N **do**

$SPL_j^c = \sum_{a_k \in Q_j} SP_{a_k}^c$;

$QPBLV_c = \{DBM^c\} \cup \{\cup_{j=1}^N SPL_j^c\} \cup F$;

- $QPBLV = \{\cup_{c=1}^{ndb} QPBLV_c\}$;

- Return $QPBLV$;

$$SP_{sol_c(k)}^c = solS_k(3) \times \left(sol_c(k) - \sum_{j=1}^{L(sol_c(k))-1} NL_j \right) \quad (4.16)$$

where, $L(sol_c(k))$ defines the job type of operation $sol_c(k)$ and NL_j the number of operations in job j . Thus, SPL_j^c is calculated as follows:

$$SPL_j^c = \sum_{sol_c(k) \in Q_j} SP_{sol_c(k)}^c \quad (4.17)$$

where Q_j is the set of operations in job j . Then, GP is trained using the set $MPBLV_c$ as the learning set for solution c :

$$MPBLV_c = \{ \{ DBM^c \} \cup \{ \cup_{j=1}^N SPL_j^c \} \cup F \} \quad (4.18)$$

Note again that F is defined as in Equation 4.2 and is calculated by SL simulation replications for a solution. Considering the total number of ndb training solutions, the final training set $MPBLV$ can be defined as:

$$MPBLV = \{ \cup_{c=1}^{ndb} MPBLV_c \} \quad (4.19)$$

Algorithm 7 summarises the $MPBLV$ calculation procedure. To illustrate the set of $MPBLV$ calculations here we extend the SJSSP solution example provided in Figure 4.2, with the example solution here considered as the first solution in the data set ($c = 1$). As previously mentioned, consider the processing time values in Figure 4.2 as samples, using Equation 4.8 DBM^1 equals 0.2982. The SP values for all operations are calculated based on the queue sequence, therefore operation 1 belongs to the first job, according to Equation 4.15, $SP_1^1 = 1 \times 1 = 1$ and using the same equation SP_2^1 and SP_3^1 are equal to 4 and 9, respectively. As operation 4 does not belong to the first job, using Equation 4.16, $SP_4^1 = 1 \times (4 - 3) = 1$. Similarly, SP_5^1 , SP_6^1 , and SP_7^1 values are 4, 3, and 2, respectively. Then, using Equation 4.17, SPL_1^1 , SPL_2^1 , and SPL_3^1 values are equal to 14, 8, and 2, respectively. Consider that the due date of all jobs is equal to 1. Besides, both earliness and tardiness costs

are equal to 1. Therefore, using Equation 4.2, the objective value F for the current example solution equals to 18. Finally, the $MPBLV$ training set of the proposed solution is defined in Equation 4.20.

$$MPBLV_1 = \{0.2982, 14, 8, 2, 18\} \quad (4.20)$$

Algorithm 7: MPBLV.

Inputs : ndb (total number of GP training and testing solutions), $First$ (all operations of the first job), Q (set of operations in each job), L (set of job identifications for operations), NL (a set defining number of operations in each job), N , NO

Output: $MPBLV$

begin

for $c = 1$ **to** ndb **do**

 - Create a vector of a random feasible queue solution (chromosome) and name it as $sol_c = \{a_1, a_2, \dots, a_{NO}\}$;

 - Calculate the fitness value (F using Equation 4.1) from SL replications of DESM of SJSSP;

for $k = 1$ **to** NO **do**

 - Define vector $solS_k(sol_c(k), c, u, m)$, where $sol_c(k)$ describes the operation in position k in the queue, c defines the solution, u describes the position of $sol_c(k)$ on machine m and m the machine.

 - $DBM^c = \frac{STD(D)}{Mean(D)}$;

for $k = 1$ **to** NO **do**

if $a_k \in First$ **then**

$SP_{sol_c(k)}^c = solS_k(3) \times sol_c(k)$;

else

$SP_{sol_c(k)}^c = solS_k(3) \times (sol_c(k) - \sum_{j=1}^{L(sol_c(k))-1} NL_j)$;

for $j = 1$ **to** N **do**

$SPL_j^c = \sum_{a_k \in Q_j} SP_{a_k}^c$;

$MPBLV_c = \{DBM^c\} \cup \{\cup_{j=1}^N SPL_j^c\} \cup f_s$;

 - $MPBLV = \{\cup_{c=1}^{ndb} MPBLV_c\}$;

 - Return $MPBLV$;

4.4.3 Machines Positions Interaction Based Learning Vector (MPIBLV)

This training vector combines both QPBLV and MPBLV to obtain a new training vector which we denote as MPIBLV. To develop this we use the vector $solS_k(sol_c(k), c, u, m)$ for each operation, where $sol_c(k)$ describes the operation in position k in the chromosome (queue), c as defined previously is the solution number and u is the position of $sol_c(k)$ on machine m . In addition, to calculate MPIBLV, we categorize operations into two categories; the first operations of jobs and the remaining operations. Then, $SP_{sol_c(k)}^c$ for operation $sol_c(k)$ positioned in the k^{th} position of the queue of solution c is calculated as follows. If $sol_c(k)$ is the first operation of any job j in solution c , and it is assigned to the first position of machine m then:

$$SP_{sol_c(k)}^c = 0 \quad (4.21)$$

If $sol_c(k)$ is the first operation of job j in solution c , and it is assigned to the u^{th} position ($u > 1$) of machine m :

$$SP_{sol_c(k)}^c = SP_{sol_c(k')}^c + 1, \quad (4.22)$$

where $SP_{sol_c(k')}^c$ defines the value of SP for the operation $sol_c(k')$ assigned to the position $u - 1$ on machine m and queue position k' ($k' < k$) in solution c .

If $sol_c(k)$ is not the first operation of any job j , and it is assigned to the first position of machine m in solution c :

$$SP_{sol_c(k)}^c = SP_{sol_c(k)-1}^c + 1, \quad (4.23)$$

where $SP_{sol_c(k)-1}^c$ defines the value of SP for the operation $sol_c(k) - 1$ of job j assigned to position u' on machine m' in solution c .

If $sol_c(k)$ is not the first operation of any job j , and it is assigned to the u^{th} position ($u > 1$) of machine m in solution c :

$$SP_{sol_c(k)}^c = SP_{sol_c(k)-1}^c + SP_{sol_c(k')}^c + 2, \quad (4.24)$$

where $SP_{sol_c(k)-1}^c$ defines the value of SP for the operation $sol_c(k) - 1$ of job j assigned to position u' on machine m' for solution c , and $SP_{sol_c(k')}^c$ defines the value of SP for operation $sol_c(k')$ ($k' < k$) assigned to position $u - 1$ on machine m for solution c . Consequently, SPL_j^c is defined as follows:

$$SPL_j^c = \sum_{sol_c(k) \in Q_j} SP_{sol_c(k)}^c \quad (4.25)$$

where Q_j is the set of operations for job j . Finally, GP is trained using the set $MPIBLV_c$ as the learning set for solution c .

$$MPIBLV_c = \{\{DBM^c\} \cup \{\cup_{j=1}^N SPL_j^c\} \cup F\} \quad (4.26)$$

The function F , as defined above, is obtained from Equation 4.2 using SL simulation replications. Considering the total number of ndb training solutions, the final training set $MPIBLV$ can be defined as:

$$MPIBLV = \{\cup_{c=1}^{ndb} MPIBLV_c\} \quad (4.27)$$

Algorithm 8 summarises the $MPBLV$ calculation procedure. To illustrate the set of $MPIBLV$ calculations, here we extend the example provided in Figure 4.2. To ensure the validity of all calculations, SP values for all operations are calculated based on the queue sequence (start with operation 1, next 4, etc.). As operation 1 is the first operation of job 1 and it is assigned to the first position on machine 1, using Equation 4.21, SP_1^1 equals to 0. Similarly, the SP value for operation 4 equals 0 ($SP_4^1 = 0$). Operation 5 is not the first operation of any job and it is not assigned to any first position on machines. Therefore, using Equation 4.24, $SP_5^1 = SP_4^1 + SP_1^1 + 2 = 2$. Likewise, the SP value for operations 2 and 3 are 2 and 6, respectively ($SP_2^1 = 2$ and $SP_3^1 = 6$). Operation 6 is not the first operation of

any job, and it is assigned to the first position on machine 3. Thus, using Equation 4.23, SP_6^1 equals 3 ($SP_6^1 = SP_5^1 + 1$). Operation 7 is the first operation of job 3, and it is not assigned to the first position of any machine. Therefore, using Equation 4.22, SP_7^1 equals 4 ($SP_7^1 = SP_6^1 + 1$). By aggregating the SP values of operations of each job (Equation 4.25), $SPBM_1^1$, $SPBM_2^1$ and $SPBM_3^1$ are equal to 8, 5 and 4, respectively (e.g., $SPBM_1^1 = SP_1^1 + SP_2^1 + SP_3^1$). Considering the processing time values for all job operations in Figure 4.2, using Equation 4.8, DBM^1 is equals to 0.2982. Consider the due date of all jobs is equal to 1. Besides, both earliness and tardiness costs are equal to 1. Therefore, using Equation 4.1, the objective value F for the current example solution equals to 18. Finally, GP $MPIBLV$ of the proposed solution is defined in Equation 4.28.

$$MPIBLV_1 = \{0.2982, 8, 5, 4, 18\} \quad (4.28)$$

4.5 Data Preparation For Experimentation

In this section, preparations for the experiments reported in the next section are presented. First, data obtained from an industrial setting is defined in the next subsection. Then calibrations of the GP metamodel are reported in subsection 4.5.2, and finally, the training and evaluation data sets used in the experimentation are reported in subsection 4.5.3.

4.5.1 Input Data for the SJSSP

A sample SJSSP was created with the parameters (number of jobs, number of operations, processing times, and number of machines) taken from a Bosch semiconductor fab in Germany. In semiconductor manufacturing, the photolithography tool is a SJSSP, typically seen as a bottleneck resource as products within each layer are processed by this tool (Ghasemi, Azzouz, et al. 2020). Rather than evaluate the different training vectors using simplified test data, testing was carried out on a real

Algorithm 8: MPIBLV.

Inputs : ndb (total number of GP training and testing solutions), $First$ (all first operations of jobs), N , NO

Output: $MPIBLV$ (training set)

begin

for $c = 1$ **to** ndb **do**

- Create a vector of a random feasible queue solution (chromosome) and name it as $sol_c = \{a_1, a_2, \dots, a_{NO}\}$;

for $k = 1$ **to** NO **do**

- Define vector $solS_k(sol_c(k), c, u, m)$, where $sol_c(k)$ describes the operation in position k in the queue solution, c defines the solution, u describes the position of $sol_c(k)$ on machine m and m the machine.

- Calculate the fitness value (F using Equation 4.2) from the DESM of SJSSP with SL replications;

- $DBM^c = \frac{STD(D)}{Mean(D)}$;

for $k = 1$ **to** NO **do**

if $solS_k(1) \in First$ **then**

if $solS_k(3) = 1$ **then**

| - $SP_{sol_c(k)}^c = 0$;

else

| - $SP_{sol_c(k)}^c = SP_{sol_c(k')}^c + 1$; where $SP_{sol_c(k')}^c$ defines the value of SP for the operation $sol_c(k')$ assigned to the position $u - 1$ on machine m and queue position k' in solution c .

else

if $solS_k(3) = 1$ **then**

| - $SP_{sol_c(k)}^c = SP_{sol_c(k)-1}^c + 1$;

else

| - $SP_{sol_c(k)}^c = SP_{sol_c(k)-1}^c + SP_{sol_c(k')}^c + 2$;

- $SPBM_j^c = \sum_{k \in O_j^c} SP_k^c \forall j = \{1, \dots, N\}$; % O_j^c is the set of operations for job j in solution c .

- $MPIBLV_c = \{\{DBM^c\} \cup \{\cup_{j=1}^N SPBM_j^c\} \cup f_s\}$;

- $MPIBLV = \{\cup_{c=1}^{ndb} MPIBLV_c\}$;

Table 4.2: Operations flow matrix.

Job id	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
1	3	2	1	6	4	8	5	7
2	1	2	3	5	4	7	6	8
3	1	2	3	5	4	6	8	7
4	2	1	3	4	7	5	6	8
5	4	3	2	1	7	5	6	8
6	2	3	1	5	6	4	8	7
7	1	3	4	2	5	7	8	6
8	2	1	3	4	5	7	6	8

Table 4.3: Jobs due dates.

job id	1	2	3	4	5	6	7	8
Due Date	490	510	540	500	540	470	530	560

system to better validate the training vectors.

There are four parameters in SJSSP: number of jobs, number of operations, processing times, and number of machines. As Table 4.2 indicates, we consider 8 jobs as a sample from the Bosch data, with 8 operations per job. Here, the processing time parameters are defined based on real photolithography machine data sets extracted from a Bosch fab. Figure 4.4 shows the fitted distributions on processing time data sets using the Matlab Distribution Fitting App (Mathlab Fitting App 2020). Although the Normal distribution can describe features of the processing times with a good estimation, the normal distribution has no bounds (see Figure 4.4). Therefore, a Gamma distribution with the shape parameter equal to 22.47 and the scale parameter equal to 2.43 ($Gamma(22.47, 2.43)$) is used, which guarantees both positive numbers for processing times and good estimated values. Note that the fitting quality is investigated graphically. Additionally, the number of machines is considered as a constant, which equals 8 in this sample SJSSP. The flow of operations between different machines is defined in Table 4.2. To illustrate, the first operation of job 1 is planned to be processed by machine 3 (M_3). Besides, jobs due dates are shown in Table 4.3. Note that both earliness and tardiness costs are considered equal to 1 per minute.

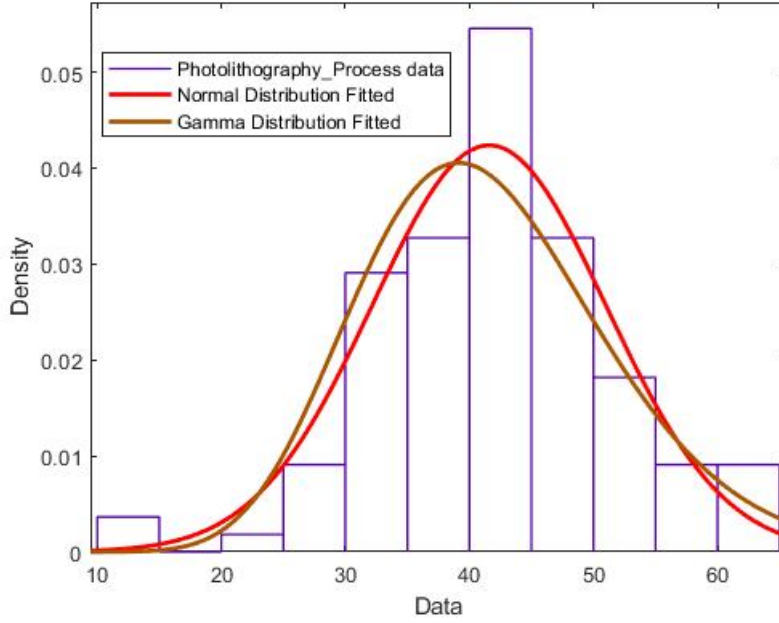


Figure 4.4: Data sets box plots.

4.5.2 GP calibration

Clearly, the parameter settings of an evolutionary algorithm have a deep influence on its performance. Thereby, in this research, we discuss the parameter values for GP by using Taguchi experimental design. This method is based on a special set of arrays called orthogonal arrays to conduct the minimum number of efficient experiments that could give insights on all factors that affect the performance measure.

In evaluating GP solutions, we used the Mean Relative Error (MRE), which is known as one of the most practical metrics for evaluating estimation techniques (Park and Stefanski 1998). The general formulation of MRE is as follows:

$$MRE = \frac{1}{ss} \times \sum_{b=1}^{ss} \frac{|Y'_b - Y_b|}{Y_b}, \quad (4.29)$$

where Y'_b defines the estimated value (here t'_T) of the factor Y_b (here t_T) in the sample b , and total number of samples equals ss .

GP used in Heuristic Lab has five parameters (HeuristicLab 2020): Population Size (PS), Mutation Probability (MP), Maximum Generations (MG), Tree Length (TL), and Tree Depth (TD). To perform the test, different stage values need to be

Table 4.4: Taguchi test parameters.

Parameter	1	2	3
<i>PS</i>	500	750	1000
<i>MP</i>	0.15	0.20	0.25
<i>MG</i>	100	250	500
<i>TL</i>	100	125	150
<i>TD</i>	20	25	30

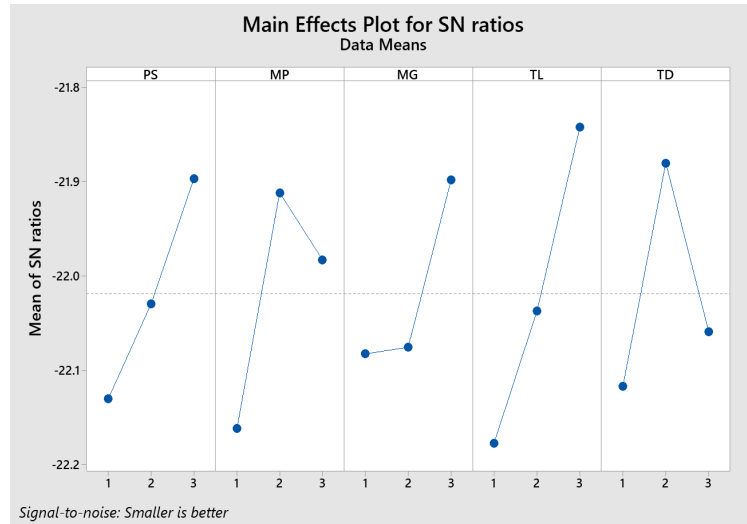


Figure 4.5: Taguchi test results.

selected for each parameter. Table 4.4 describes the different parameters evaluated using the Taguchi method. After running a number of experiments, we picked these parameter values that effected metamodel generation. The main effects plots are shown in Figure 4.5, which shows how each factor affects the response characteristics. A main effect exists when different levels of a factor affect the characteristics differently.

Based on the number of parameters and their set values, the Taguchi test required 27 different GP test runs. The Taguchi test results are shown in Figure 4.5. After running the test, the values 1000, 0.20, 500, 150, and 25 are selected for *PS*, *MP*, *MG*, *TL* and *TD*, respectively.

4.5.3 Training and Evaluation Data Sets Design

To create SJSSP data sets, an Evolutionary Improvement Procedure (EIP) is used for two reasons. Firstly, to obtain robustness against the intrinsically enlargement

of SJSSP solution space in our experiments by exploring the whole solution space. Secondly, to provide sensitivity analysis by examining the influence of data sets on the GP estimation accuracy. This research focuses on the evaluation of a metamodel for use within an evolutionary SO method. Therefore, we generate similar data that would be generated within such an optimization context. To do this, we use the EIP shown in Figure 4.6 to generate what we term: Low Quality Data Sets (LQDSs), Medium Quality Data Sets (MQDSs) and High Quality Data Sets (HQDSs). Note that LQDS is not generated using any evolutionary operators, it is created using a random selection of data for the SJSSP. For the MQDS and HQDS data sets, the quality of the data sets, it is inferred, is dependent on the number of evolutionary operators used to obtain each data set. An overview of the EIP is provided here with further details in subsection 4.6.1.

Figure 4.6 demonstrates the EIP used to create data sets in this research. Firstly, a set of ndb (ndb will be defined later) random SJSSP solutions are created. Then, using SL (SL will be defined later) simulation replications for each solution, objective values of the solutions are calculated and the Data Set DS is stored as LQDS. Next, using the mutation function, an offspring is created for each solution in DS. Again, using SL simulation replications objective values are calculated for offspring solutions, where the DS and offspring solutions are stored in DSO (DSO refers to an arbitrary structure array of solutions used in EIP (see Figure 4.6)). Next, all solutions in DSO are ranked based on their objective values and the best ndb are selected as the new DS. Here the procedure of mutation and selection is considered as a generation. Thus, after 50 generations, DS is stored as MQDS. Similarly, after 100 generations, HQDS is created. In another EIP run, a sample from LQDS, MQDS, and HQDS are captured to generate a Mixed Data Set (MDS). In other words, SJSSP solutions from LQDS, MQDS, and HQDS are sampled (equal number of solutions from each category) to create MDS.

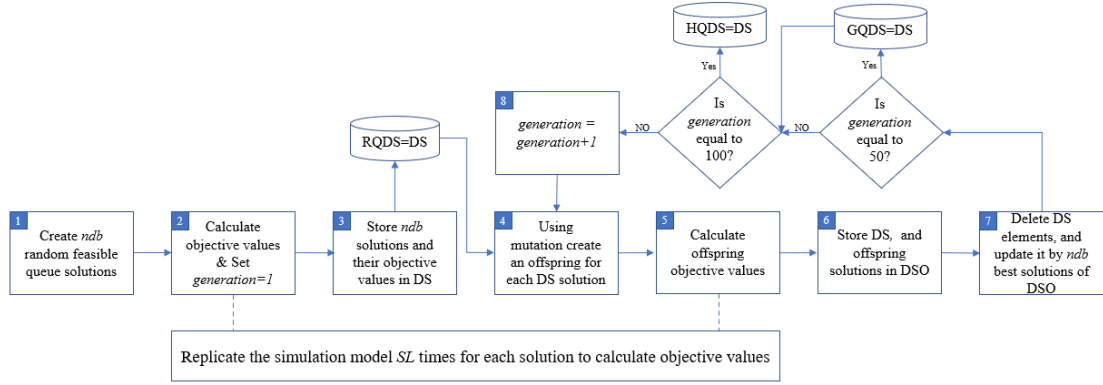


Figure 4.6: EIP to obtain SJSSP data sets.

4.6 Experiments

The main objective of this research is to examine the possibility of replacing simulation replications within SO methods with simulation metamodels built using GP. Thereby, considering the three steps to prepare the SJSSP simulation metamodel, Figure 4.7 provides a research framework to examine the possibility of replacing simulation replications in SO methods with a SJSSP GP metamodel. Firstly, a set of SJSSP solutions and their objective values calculated using simulation replications are defined, called Training Data Set (TDS) (see Figure 4.7). TDSs are then transformed into training vectors to train GP models. The trained GP's accuracy is examined by comparing its objective estimations (t'_T) of an Evaluation Data Set (EDS), and finally, these are compared with estimations from simulation results (t_T) to evaluate the effectiveness of GP metamodels.

In this evaluation, it will be shown that both TDS quality and the selected training vector have a direct impact on the GP-based SJSSP simulation metamodel accuracy in estimating SJSSP objective values. Thus, three different training vectors (QPBLV, MPBLV, and MPIBLV) and twelve TDSs with different features are considered in this experimental analysis. This section contains two subsections. The first subsection will address the following question: What is the best training vector to transform different SJSSP solution data sets into GP metamodels to replace simulation replications within an evolutionary SO procedure? Subsection

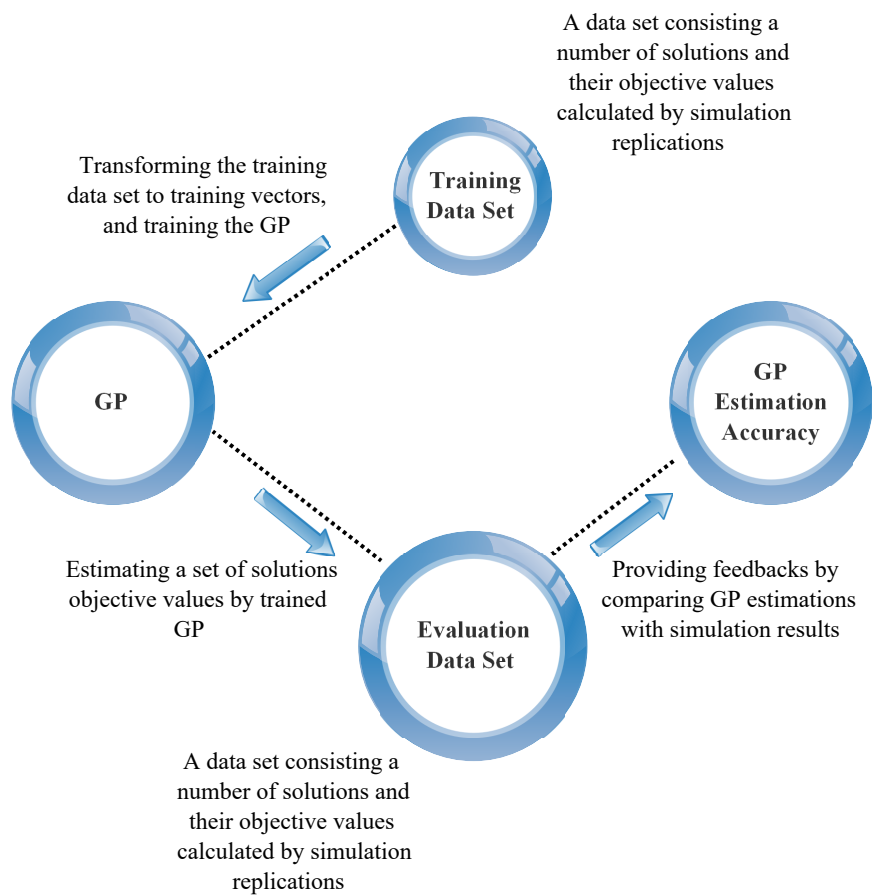


Figure 4.7: The general SJSSP metamodeling accuracy analysis structure.

Table 4.5: Training Data Sets Statistical Analysis.

Data Set	TDS/EDS	Quality	ndb	SL	Mean	StDev	Min	Q1	Med	Q3	Max	Range
DS1	TDS	LQDS	5000	1	6761.7	1124.2	3358.9	5978.6	6705	7868.4	11429.8	8070.9
DS2	TDS	MQDS	5000	1	5479.8	1051	3008.2	4712.3	5369.2	6093.3	10242.6	7234.4
DS3	TDS	HQDS	5000	1	3980.7	817.4	2018.9	3410.8	3861.6	4432.7	8471.5	6452.6
DS4	EDS	MDS	10000	100	5390.5	1699.8	2018.9	3895.5	5239	6720.2	12482.9	10464

4.6.2 will evaluate the following question: Does the quality of the input data sets and simulation replications impact GP estimation accuracy?

4.6.1 Evaluation of Training Vectors

Table 4.5 describes four SJSSP solution data sets, with DS1 generated using LQDS, DS2 using MQDS, and DS3 using HQDA. Each one of these data sets is generated using a single replication (i.e., $SL = 1$) and will be used to train a GP metamodel. The MDS is an evaluation data set (EDS), which is denoted DS4, used to compare the generated GP models using data sets DS1, DS2, and DS3. While the TDSs were generated using a single replication, 100 replications ($SL = 100$) are used to evaluate objective values accurately using DS4 (EDS). Each data set solution is analyzed statistically in Table 4.5 with the table showing Mean, Standard Deviation (StDev), Minimum (Min), First Quartile (Q1), Median (Med), Third Quartile (Q3), Maximum (Max), and Range of solutions objective values in each data set calculated using SL Simulation replications. Table 4.5 also shows the total number of solutions in a data set (ndb).

Figure 4.8 shows the box plots of the above-mentioned data sets in Table 4.5. Examining these objective value solutions of all four data sets, they vary between 2018.9 to 1282.9. Indeed, there is a significant quality improvement from DS1 to DS2 (LQDS to MQDS) as well as DS2 to DS3 (MQDS to HQDS), with $ndb = 5000$, as is expected, as 50 EIP generations have been applied to MQDS and 100 to HQDS. However, DS4 labeled EDS, which is used to evaluate the three training vectors, has a wider range of SJSSP solutions, again this is expected.

As stated previously, SO methods use an optimizer integrated with simulation experiments to achieve optimal solutions. For example, methods that follow this SO

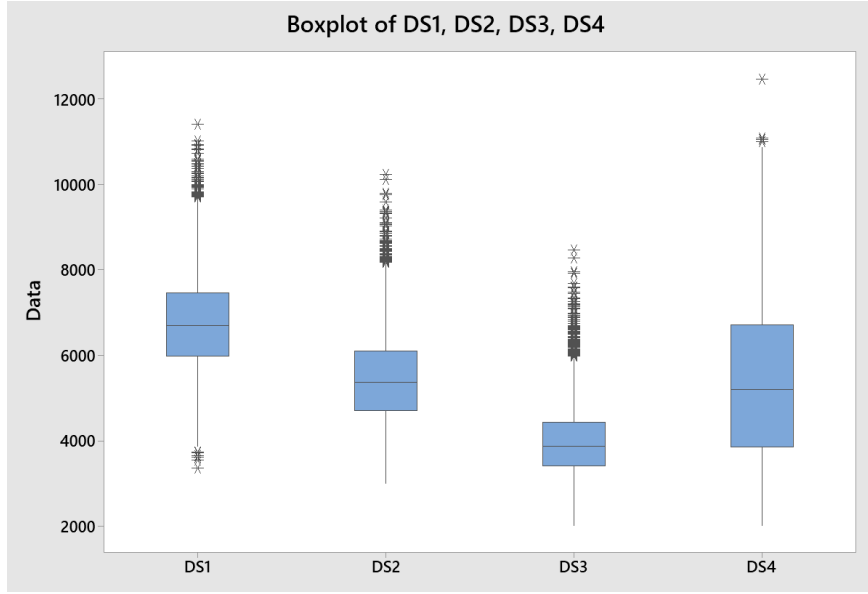


Figure 4.8: Data sets box plots of DS1, DS2, DS3, and DS4.

procedure were proposed by Ho (1999), Gu et al. (2010), Yang, Lv, et al. (2014), among others. Within population-based evolutionary SO methods, during each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through an objective-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as this process may be very time-consuming (Ma et al. 2019). Thus, any estimation strategy intended to replace simulation-based fitness calculations should guarantee the detection of promising solution(s). In this chapter, the GP metamodel is evaluated to examine if it can guide the solutions towards promising candidates in a SO framework and replace simulation replications. Therefore, we require that the GP metamodel be evaluated for predicting the location of solutions in different objective value regions. This is accomplished by examining if the GP metamodel can predict the correct quartile within a very large solution space for stochastic optimization problems, and specifically here for SJJSP. For instance, consider 1000 SJJSP solutions sorted from 1 to 1000 based on their objective values calculated by *SL* simulation replications for each solution. In this research, it is

assumed that a GP metamodel is accurate enough to replace simulation replications in a SO method, if it can detect accurately solutions in quartiles (e.g., solutions 1 to 250) of the sorted solution space.

Consider $DS = \{sol_1, sol_2, \dots, sol_{ndb}\}$ as the evaluation data set containing the total number of ndb SJSSP feasible solutions, where sol_c defines solution c in the SJSSP evaluation data set. Moreover, the objective value of each solution sol_c calculated by SL number of simulation replications are defined by $Obsol_c^{SL}$. Besides, EDS objective value estimations of GP trained by training vector w and training data set v' are defined by $GObsol_c^{v',w}$. As a result, $Sim_{SL}^{lb,ub}$ is calculated as follows.

$$Sim_{SL}^{lb,ub} = \left\{ \cup_{c=lb}^{ub} sol_c \mid Obsol_1^{SL} \leq Obsol_2^{SL} \leq \dots \leq Obsol_{ndb}^{SL} \right\} \quad (4.30)$$

where lb and ub are integer numbers and $1 \leq lb \leq ub \leq ndb$. Similarly, $GP_{v',w}^{lb,ub}$ is calculated as follows:

$$GP_{v',w}^{lb,ub} = \left\{ \cup_{c=lb}^{ub} sol_c \mid GObsol_1^{v',w} \leq GObsol_2^{v',w} \leq \dots \leq GObsol_{ndb}^{v',w} \right\} \quad (4.31)$$

Finally, the Solution Detection Factor ($SDF_{v',w,SL}^{lb,ub}$) used to evaluate similarity of solutions sorted by $GP_{v',w}^{lb,ub}$ and $Sim_{SL}^{lb,ub}$ is defined as follows:

$$SDF_{v',w,SL}^{lb,ub} = \frac{Size(GP_{v',w}^{lb,ub} \cap Sim_{SL}^{lb,ub})}{ndb} \times 100 \quad (4.32)$$

To clarify, $GP_{v',w}^{lb,ub}$ defines the set of solutions between lb and ub in the evaluation data set after ranking and sorting of solutions based on the objective values calculated by GP trained using training vector w and training data set v' . Moreover, $Sim_{SL}^{lb,ub}$ defines the set of solutions between lb and ub in the evaluation data set after ranking and sorting of solutions based on objective values calculated by SL number of simulation replications. Clearly, $SDF_{v',w,SL}^{lb,ub}$ shows the percentage of solutions in the EDS within bounds lb to ub sorted by SL simulation replications detected by a GP trained by training vector w and data set v' .

Clearly, optimization is all about comparing different options and selecting the best one(s). Thus, not only is SDF a metric to evaluate the accuracy of any metamodel in ranking a population of solutions based on their estimated objective value(s), but also it examines the possibility of replacing simulation replications with a certain metamodel.

Table 4.6 illustrates the performance of GP trained by DS1, DS2, and DS3 using learning vectors QPBLV, MPBLV, and MPIBLV in metamodeling the proposed SJSSP. Moreover, it provides SDF values for GPs trained by different training vectors and data sets in estimating objective values when compared with DS4.

As the first comparison metric, the MRE values are compared for training vectors. It is evident that whereas MPIBLV trained GP from all three data sets with low values of MRE (between 0.08 and 0.09), other training vectors resulted in training errors mostly over 2 times more than MPIBLV (QPBLV even could not perform a feasible metamodel from DS1).

To demonstrate more clearly on the quality of MPIBLV in training GP, Figure 4.9 shows Heuristic Lab results for the training and testing of the the GP metamodel from DS1. In Figure 4.9, the axis labeled “X009” provides the objective values of solutions with the other axis showing the solutions. Figure 4.9 also shows an extracted portion that demonstrates the quality of the metamodel. In the GP metamodel shown 3500 solutions are used for training and 1500 data are used for testing. Target values, training values by GP, and the obtained test values by GP are shown in blue, yellow, and red, respectively.

Table 4.6 provides SDF values. To illustrate, the SDF value of GP trained by QPBLV and DS2 in detecting solutions between 0 to 2500 (evaluated against 0% to 25% of the evaluation data set DS4) is equal to 57 ($SDF_{2,3,1,100}^{0,2500} = 57$). That indicates 57% of solutions ranked and sorted within bounds 1 to 2500 for 100 simulation replications are detected by GP trained using QPBLV from DS2.

Considering the information provided above, it is quite clear that a GP trained using MPIBLV has improved quality in predicting a solution’s objective value. This

Table 4.6: GP training vectors comparison.

Learning Vector	Training Data	MRE	SDF evaluated against DS4			
			0-25%	25-50%	50-75%	75-100%
QPBLV	DS1	<i>inf</i>	—	—	—	—
	DS2	0.17	57	47	42	47
	DS3	0.17	58	48	44	51
MPBLV	DS1	1	39	42	37	58
	DS2	0.17	56	40	40	38
	DS3	0.16	61	50	44	51
MPIBLV	DS1	0.09	77	64	65	79
	DS2	0.08	78	64	63	75
	DS3	0.08	81	68	64	76



Figure 4.9: GP training by MPIBLV line chart.

is shown by the MRE values provided in Table 4.6, with the accuracy demonstrated in Figure 4.9.

Besides, Table 4.6 makes it obvious that there is a significant difference between *SDF* values in all DS4 quartiles performed by MPIBLV against other training vectors. For instance, 81 percent of solutions in the first quartile of DS4 is detected by the GP trained using MPIBLV and DS3. However, this value is just 58 and 61 percent using the same data set for QPBLV and MPBLV, respectively.

Overall, MPIBLV evidently outperforms both other training vectors. To answer other defined questions, in the following, we extend our analysis by focusing on MPIBLV and sensitivity analysis of GP estimations towards changes in training data sets features.

4.6.2 Evaluation of Quality of Data Sets and Simulation Replications

Table 4.7 presents data on 10 SJSSP solution data sets (DS5, DS6,...,DS14). Each data set is analyzed statistically based on the objective values of its comprising solutions. Similar to the previous analysis, to obtain quality robust solutions on the intrinsically enlarged SJSSP solution space, two data sets (MQDS and HQDS) are generated using EIP to evaluate if these improve the accuracy of the generated GP metamodel. DS5, DS6, and DS7 are LQDSs which are randomly generated, while DS8, DS9, and DS10 are MQDSs, and DS11, DS12, and DS13 are HQDSs are generated using EIP. Results for DS14 (MDS), which combines LQDS, MQDS and HQDS data sets is also shown in Table 4.7.

Figure 4.10 illustrates the data in Table 4.7 graphically. Objective values of solutions in all 10 data sets vary between 2090.6 to 11783. Clearly, there is a significant quality improvement trend from the first group to the second group (LQDSs to MQDSs) as well as the second group to the third group (MQDSs to HQDSs). Note that, objective values in data sets DS5, DS8, and DS11 are calculated using 1 simulation replication, DS6, DS9, and DS12 calculated using 20 simulation replications

Table 4.7: Training data sets statistical analysis for MPIBLV.

Group	Data Set	TDS/EDS	Quality	ndb	SL	Mean	StDev	Min	Q1	Med	Q3	Max	Range
Group 1	DS5	TDS	LQDS	5000	1	6788.3	1136.9	3481.2	5971.6	6726.5	7506.3	11783	8301.8
	DS6	TDS	LQDS	5000	20	6783.6	1131.8	3379.5	5974.2	6688.4	7514.1	11718.5	8339
	DS7	TDS	LQDS	5000	50	6778.9	1136.4	3410	5989.2	6707.4	7515.2	11704.3	8294.3
Group 2	DS8	TDS	MQDS	5000	1	5501	1011.6	2952.3	4741.2	5398.9	6125.2	10391.3	7439
	DS9	TDS	MQDS	5000	20	5513.4	1009.1	2416.9	4798.9	5407.1	6121.6	10385.1	7968.1
	DS10	TDS	MQDS	5000	50	5503	1007.3	2734.3	4766.6	5398.8	6137	10048.4	7314.1
Group 3	DS11	TDS	HQDS	5000	1	4162.6	843.1	2279	3552.4	4029.2	4653.9	8497.1	6218
	DS12	TDS	HQDS	5000	20	4161.4	847.3	2242.8	3555.9	4026.9	4651.9	8358.1	6115.3
	DS13	TDS	HQDS	5000	50	4164.1	850.3	2280.6	3547.9	4040.3	4658.9	8700.9	6420.3
	DS14	EDS	MDS	10000	100	5228.2	1794.2	2090.6	3607.2	5018	6678.8	11338.2	9247.7

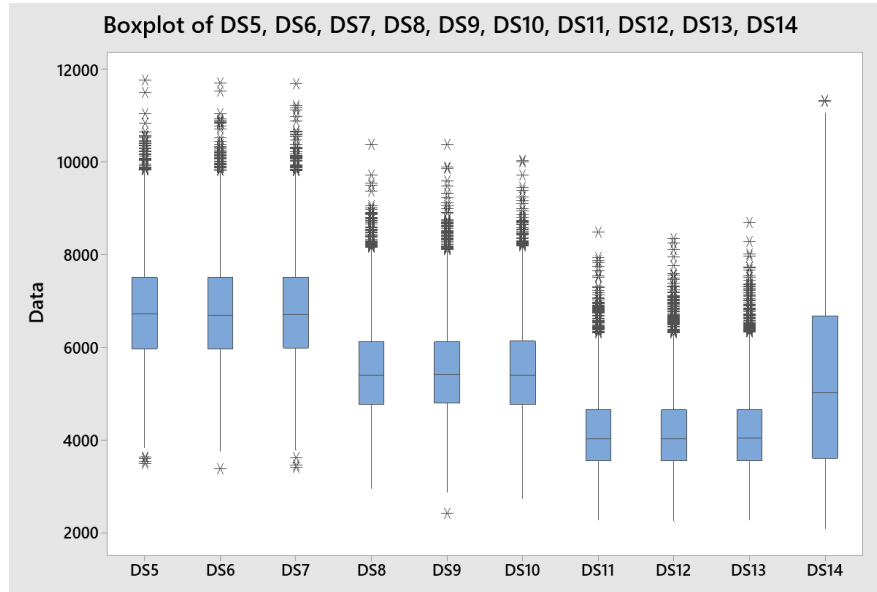


Figure 4.10: Data sets box plots for MPIBLV.

and DS7, DS10, and DS12 calculated using 50 simulation replications.

Table 4.8 provides SDF values that evaluate the GP trained objective values for data sets DS5, DS6, ..., DS13 using the MPIBLV training vector against DS14, the EDS. Different simulation replications in TDSs are considered in this phase ($SL = 1, 20, \text{ and } 50$). To illustrate, the SDF value for DS6 is 78 in detecting solutions between 0 to 2500 (0% to 25% of DS14) ($SDF_{14,6,3,100}^{0,2500} = 78$). That indicates that 78% of solutions ranked and sorted between 1 to 2500 of the GP trained metamodel for DS6 can detect a value compared with DS14. DS14 was generated using 100 simulation replications.

Figure 4.11 compares different GP input training data sets (LQDS, MQDS, HQDS are shown using blue, orange, and gray lines, respectively) and different simulation replications using the SDF metric. The figure shows that the GP metamodels

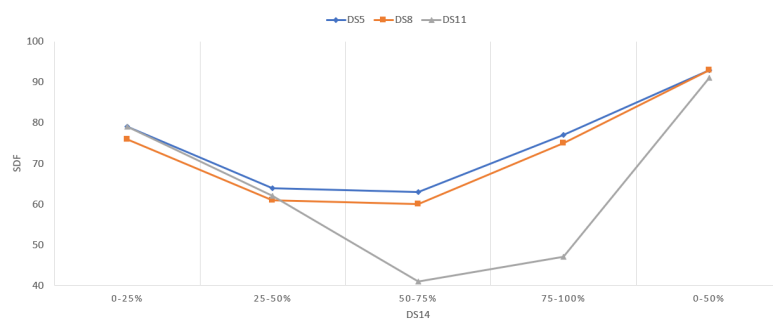
Table 4.8: Algorithms comparison.

Learning Vector	Training Data	SL	<i>SDF</i> evaluated against DS14				
			0-25%	25-50%	50-75%	75-100%	0-50%
MPIBLV	DS5	1	79	64	63	77	93
	DS6	20	78	64	62	76	93
	DS7	50	78	62	59	74	92
	DS8	1	76	61	60	75	93
	DS9	20	79	63	59	75	92
	DS10	50	80	64	60	73	92
	DS11	1	79	62	41	47	91
	DS12	20	80	64	58	72	92
	DS13	50	80	64	48	55	92

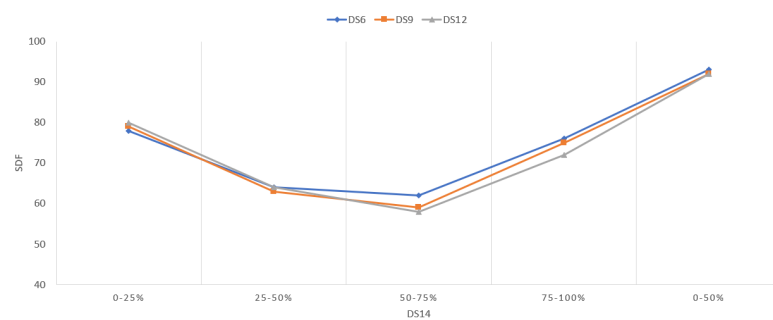
trained using MPIBLV detected the correct half of the solution space providing the lowest *SDF* of 91% when evaluated against DS14 for all replications and all qualities of input data. Figure 4.11 shows that there is a small difference between the different qualities of data sets of LQDS and MQDS over all simulation replications (1-50) evaluated with quartile 0-25% varying between 76-79, 25-50% between 61-64, 50-75% between 59-63, and finally, 75-100% between 75-77. These are small variations when the randomness of the input data is considered. However, HQDS is shown to have low values of *SDF*, 41 and 47 in Figure 4.11a and 48 and 55 in Figure 4.11c for quartiles 50-75% and 75-100%, respectively. Therefore, a recommendation from the presented results are to use LQDS, which is a random set of data (see Figure 4.6) to train GP metamodels. Applying the evolutionary process may provide low *SDF* values as demonstrated in Figure 4.11a and Figure 4.11c.

4.7 Conclusions

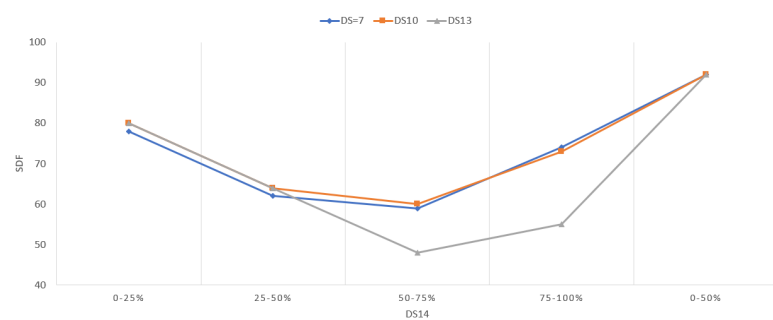
Although SO techniques provide promising solutions for large and complex stochastic problems, simulation model execution is potentially expensive in terms of computation time. Thereby, in this chapter, we attempted to replace simulation replications with a GP-based metamodel in the case of SJSSP. This provides the opportunity to implement evolutionary SO methods for complex optimization problems without extending extensive computer resources for simulation replications. More-



(a) *SDF* line chart for GPs trained using $SL = 1$.



(b) *SDF* line chart for GPs trained using $SL = 20$.



(c) *SDF* line chart for GPs trained using $SL = 50$.

Figure 4.11: *SDF* line charts based on number of simulation replications and the different quality of input data.

over, this replacement enables SO methods to allocate more computation budgets on strengthening the optimization core rather than on simulation replications.

The major contributions of this chapter lie in presenting a new application of GP in metamodeling SJSSP simulation models, an evaluation of three training vectors for metamodeling SJSSP simulation model using GP, experimenting the proposed approach using real data sets from a Bosch photolithography area, and comprehensive sensitivity analysis on the SJSSP simulation metamodels accuracy. The results of our study show that the training vector MPIBLV is much superior to QPBLV and MPBLV. Future evaluation of MPIBLV showed that LQDS, which is a random data set, has very similar SDF values with MQDS across different simulation replications. HQDS did experience much lower SDF values for certain quartiles. The results show that MPIBLV should be used with LQDS and single replication to train GP metamodels and that GP provides an accurate metamodel to replace simulation replications within SO methods in solving SJSSP.

In fact, the GP-based metamodeling procedure presented in this chapter can be integrated into various evolutionary optimization techniques to solve SJSSPs. Thus, in the next chapter it will be integrated with evolutionary SOs to solve SJSSPs.

Chapter 5

Evolutionary Learning Based Simulation Optimization for Stochastic Job Shop Scheduling Problems

5.1 Introduction

Digitization of decision making in manufacturing has increased in the last decade, which for complex products in most industries, requires a higher level of decision-making support processes. Scheduling, as a decision-making process, plays a key role in the most complex manufacturing environments (Pinedo 2016). Within the set of scheduling problems found in industry, many can be viewed as job shop scheduling problems (Zhang, Ding, et al. 2019). Although stochasticity has been known as a crucial part of most industrial operations in the scheduling literature, there is little attention to solving Stochastic Job Shop Scheduling Problems (SJSSPs), while Deterministic Job Shop Scheduling Problems (DJSSPs) have been widely researched (Winands, Adan, and Houtum 2011). While DJSSPs are extremely complex problems to solve, introducing stochastic parameters increases tremendously the solution

space of SJSSPs (Gao et al. 2016).

In general, a job shop scheduling problem is where a set of jobs, with each job consisting of a set of operations, are scheduled on different sources (machines) with fixed routes, but the route typically differs for each job. Based on this description, a SJSSP is defined as a job shop scheduling problem with stochastic problem parameters and/or variables, where typical operation processing times are considered as the stochastic parameter of SJSSPs (Ferreira, Figueira, and Amorim 2020).

SJSSPs with stochastic processing times have been solved using simulation as the evaluative tool and metaheuristics as the generative tool (Akker, Blokland, and Hoogeveen 2013). However, most of these approaches applied to solve SJSSPs suffer long execution solution times (Horng and Lin 2015).

A key feature of simulation based optimization that makes it difficult to implement within complex problems such as SJSSPs is the need to address the search versus selection trade-off. Given a limited computing budget, how should that budget be allocated between searching over the feasible space for (potentially) better solutions, and determining which of the solutions that have been examined are actually good? (Boesel, Nelson, and Kim 2003). Ordinal Optimization (OO) provides a means to overcome this issue, where, instead of optimizing problems in the global solution space, Θ , OO is divided up into two phases (Horng, Yang, and Lin 2012). In the first phase a sample set of solutions Ψ is selected from the global space with their solutions ranked using a low cost computational method. Then in phase 2 a smaller set of solutions is selected from Ψ and optimized using a method that has a higher computational cost. The aim is to find the optimal, or, if not, the best set of solutions from Θ within a given computational cost.

For example, in phase 1, Horng, Lin, and Yang (2012) used a reduced number of replications using a coarse simulation model (in terms of accuracy of results) and an Evolutionary Strategy (ES) optimizer to obtain Ψ and, in phase 2, used replications from a more detailed simulation model to find the optimal or best solution. Horng, Lin, and Yang (2012) called the method Evolutionary Strategy in Ordinal Opti-

mization (ESOO). Latterly, Yang, Lv, et al. (2014) improved ESOO by optimizing simulation run allocations to elite solutions in a new approach called Evolutionary Strategy in Ordinal Optimization Optimal Computation Budget Allocation (ESOO-OCBA). In their approach in phase 2, simulation replications are allocated between solutions unequally, with more replications allocated to solutions with better fitness function values. Although these approaches present a new structure for tackling SJSSPs, they use a lot of CPU time to find a solution to the given problem, due to the required number of simulation replications.

The high level of CPU time within the ESOO and ESSO-OCBA approaches has motivated the development of methods that allow the creation of approximate models, i.e., metamodels of systems which sacrifice accuracy for computational gain. A metamodel refers to an approximate predictive model of system performance which depends on decision variables. Can and Heavey (2012) introduced Genetic Programming (GP) in metamodeling of stochastic problems to evolve symbolic expressions. In addition, they compared the quality of GP in metamodeling simulation models in comparison with Neural Network (NN).

In this research, a Simulation Optimization (SO) model for the SJSSP is proposed. A new two-phase OO, denoted as the Evolutionary Learning Based Simulation Optimization (ELBSO) method, to solve SJSSPs is introduced. In the ELBSO method, instead of replicating the simulation model many times, a GP estimates the value of a fitness function by metamodeling the simulation model. In the first phase of OO GP is used with GA to evaluate the Ψ set from Θ . In the second phase of ELBSO, a Simulation Budget Allocation (SBA) procedure executes a set of solutions, Ψ , from the first phase. The ELBSO method allows SJSSPs to be solved in a short period of time and with good accuracy when compared to other available methods. However, the expertise of the ELBSO modeler plays a key role in its quality, due to its dependence on both an accurate simulation model and the setting of parameters for the quality of the GP metamodel and GA algorithm.

In summary, this chapter provides,

- A new method for solving SJSSPs that reduces significantly computational time while producing reasonably high-quality solutions;
- A new metamodel training method for scheduling problems;
- A new application of GP in metamodeling for SJSSPs;
- A comparison of test problems presented by Horng, Lin, and Yang (2012), Yang, Lv, et al. (2014) and Shen and Zhu (2016);
- A comparative analysis of ELBSO with ESSO and several dispatching rules in terms of accuracy and computing budget, where the number of machines, operations and jobs are varied including the cost of earliness and tardiness of jobs.

The remainder of this chapter is organized as follows. Section 5.2 provides a literature review in solving the job shop scheduling problem and, more specifically, the SJSSP problem. Section 5.3 presents a mathematical formulation for the problem, while Section 5.4 describes the proposed ELBSO. Section 5.5 calibrates the GP model. In Section 5.6, the results of the conducted experiments are reported while Section 5.7 provides a discussion of results. Finally, Section 5.8 concludes the chapter and looks at possible follow-on studies.

5.2 Literature Review

Over recent years, a large body of research has been published on Job Shop Scheduling Problems (JSSP), which are one of the basic models used in manufacturing. JSSP is one of the famous mathematical optimization problems that has been proved to be NP-hard (Horng, Lin, and Yang 2012).

JSSP deals with the sequencing of a series of operations on fixed machines. Every job is assumed to have a different processing time and each job must go through a number of operations performed in a specific order, due to precedence constraints on different machines. The problem is to schedule the operations on the machine

considering the constraints to minimize the total duration of time it takes for all jobs to be processed (Mohan, Lanka, and Rao 2019).

Generally, JSSP aims at minimizing the makespan with the consideration of precedence and resource constraints. For a conventional JSSP model, an assumption that is typically made is that all time parameters are known exactly, i.e., deterministic values. However, in manufacturing systems, uncertainties are often encountered (Wang et al. 2018).

Job shop scheduling problems can be static or dynamic. In dynamic models, for the duration of the scheduling problem, stochastic events can occur (i.e., machine breakdowns, changes in due dates and/or processing times (Kim and Bobrowski 1997)), while in static models no new events occur (Ramasesh 1990). Here we focus on static stochastic job shop scheduling problems, or SJSSP. Only a few researchers have addressed SJSSP (Gen, Hao, and Zhang 2017).

Uncertainties in parameters can originate for different reasons, e.g., uncertainty due to processing time. For example, Yang, Lv, et al. (2014), used a Monte Carlo-based simulation technique for solving stochastic job shop problems with random process times. They combined the evolutionary strategy with OO to enhance the performance evaluation process. In a similar study, Hao, Gen, et al. (2017) presented a Multi Objective Estimation of Distribution Algorithm (MoEDA) to solve the stochastic job shop problem.

The uncertainty of other parameters such as lot size and priority of the jobs is discussed in the literature. For example, Petrovic et al. (2008) developed a fuzzy rule-based approach to determine lot sizes which were modeled using fuzzy sets. Hasan, Sarker, and Essam (2011) supposed different scenarios of machine unavailability and breakdown, and proposed a modified GA to make the stochastic job shop problem more reliable. In the following section, a deeper insight is provided on the literature that describes methods for solving SJSSPs.

5.2.1 Solutions of Stochastic Job Shop Scheduling Problems

Due to the uncertainty of parameters in real manufacturing systems, there is a significant gap between scheduling in theory and practice. Most scheduling problems with a large number of machines are proven to be NP hard. As the parameters of the problem increase, feasible solutions rise to an exponential order and providing “good” solutions is extremely difficult. As a result, it is impractical to use exact methods to solve SJJSPs Mohan, Lanka, and Rao 2019.

At present, the common mathematical methods for modeling scheduling problems with uncertainties are stochastic programming, fuzzy programming, rough sets, grey programming, and interval theory (Gen, Hao, and Zhang 2017). Here we review the most important studies related to stochastic job shop scheduling.

In an early study, Golenko-Ginzburg, Kesler, and Landsman (1995) proposed two dispatching rules to solve SJSSP with uncertain job processing times. They solved a SJSSP with 10 jobs and 5 operations per job in less than 1 second. In later research, Sakawa and Mori (1999) proposed a Genetic Algorithm (GA) to solve job shop scheduling problems with fuzzy processing times and due dates. Their GA is compared with a Simulated Annealing (SA) algorithm to solve an instance of 10 jobs and 10 machines. Moreover, it is reported that their GA solved their proposed problem in 220.5 seconds. Sakawa and Mori (1999) research is extended in Lei (2008) by proposing a Pareto Particle Swarm Optimization (PSO) algorithm to solve a similar fuzzy job shop scheduling problem. They reported that their algorithm can solve a 10 job and 10 machine problem in approximately 11 seconds and outperforms two other PSO methods in the literature in solving the proposed fuzzy job shop problems.

Gu et al. (2010) combined the exploration and exploitation mechanism of Quantum Genetic Algorithm (QGA) with the idea of multi-population in competitive co-evolutionary search mechanisms. The new method, called Competitive Co-evolutionary Quantum Genetic Algorithm (CCQGA), was compared with QGA and standard GAs in solving standard SJSSPs Fisher 1963. These are deterministic problems

so the authors used the mean of the normal distribution for the processing time of the deterministic benchmark problem, and the variance was generated from the uniform distribution $U[0, 1]$. They used 30 simulation replications to evaluate each solution to solve small (6 jobs and 6 machines) to large (20 jobs and 5 machines) sized problems, with CPU times between 636 to 1717 seconds. In another study, Lei (2011) presented a Simplified Multi-objective Genetic Algorithm (SMGA) for the SJSSP with exponential processing time to minimize the total tardiness ratio and the makespan. They validated their proposed algorithm by comparing it with Pareto Archived Simulated Annealing (PASA) and a multi-objective evolutionary algorithm from the literature in solving several small size deterministic job shop instances. They attempted to incorporate uncertainty in the deterministic test problem by the addition of the exponential λ factor.

To the best of our knowledge, Horng, Yang, and Lin (2012) attempted, for the first time, to solve complex stochastic problems using a SO method that used a high number of simulation replications to provide optimization with high accuracy. They considered an evolutionary strategy within OO for solving stochastic and hard optimization problems, a hotel booking limit problem, which involved a large discrete solution space. They used a Radical Basis Function (RBF) to develop an approximate solution in the first phase of OO and, in the second phase, they used OCBA. This method is named ESOO. ESOO was then adapted to solve a SJSSP with random processing times to minimize both earliness and tardiness costs (Horng, Lin, and Yang 2012). They firstly used a GA procedure in phase 1 of OO, using 368 simulation replications to calculate the objective function for each solution in this stage. Then the best solutions from phase 1 are analysed further in phase 2 where 10^5 simulation replications are executed. They validated ESOO by comparing the results with 5 typical dispatching rules for problems with 8 jobs, operations and machines and differently distributed machine processing time (truncated normal, uniform, and exponential distributions). Moreover, they reported that ESOO solves the SJSSP instances in approximately 350 seconds. ESOO was later modified by

Yang, Lv, et al. (2014), and termed ESOO-OCBA, where they added a budget experiment allocation procedure to the second phase of ESOO, and solved similar SJSSP instances as in Horng, Lin, and Yang (2012). ESOO-OCBA performed similarly to ESOO in terms of quality of solution and computational time.

Zhang, Song, and Wu (2013) presented a hybrid Differential Evolution (DE) algorithm for solving SJSSP with uncertain processing times to minimize the total tardiness. Using a K-armed bandit method and 20 simulation replications, they estimated the objective values for each solution. The proposed DE algorithm was verified by comparing it with a hybrid Particle Swarm Optimization-Simulated Annealing (PSO-SA) method in solving both small and large size SJSSPs. In further work published in 2013 Akker, Blokland, and Hoogeveen (2013) combined SA as the local search technique with a simulation method to find robust solutions for a stochastic job shop scheduling problem. To reduce the time intensity of their proposed method, one simulation replication is considered to calculate the objective value for each solution, and then 1000 simulation replications are executed on the best solution of each iteration. Sharma and Jain (2015) developed a discrete event simulation model of a stochastic dynamic job shop (SDJS) scheduling problem while considering sequence-dependent setup times. They compared 9 different dispatching rules in solving SDJS while minimizing the makespan. To evaluate the objective value for each solution, they compared their results with 30 simulation replications. Kemmoe-Tchomte, Lamy, and Tchernev (2015) proposed a meta-heuristic approach with simulation for a stochastic job-shop optimization problem. They combined a greedy randomized adaptive search procedure (GRASP) with an evolutionary local search (ELS) to solve the problem. Using the mentioned method, they firstly created 5 optimal solutions in a deterministic environment and then 1000 simulation replications were executed to evaluate each solution in a stochastic environment.

Horng and Lin (2015) integrated the Ant Colony System and Ordinal Optimization (ACSOO) for solving the stochastic job shop scheduling problem with random process times and compared this with conventional dispatching rules. In the first

stage of ACSOO they used an extreme learning machine to find 10 optimal solutions. Then, in the second phase, 29412 simulation replications were run to calculate the solution objective values. They showed that their method solves a SJSSP with 6 jobs and 6 machines with random processing times in around 100 seconds. In another study, Shen and Zhu (2016) presented a chance constraint approach for SJSSP by considering 10 jobs with 5 operations per job to be executed on 5 different machines. In their study, the processing times are stochastic, and the objective function is to minimize both earliness and tardiness costs. In their method, firstly using a chance constraint technique, the SJSSP was converted to a deterministic model with 0.8 level of confidence. Then the deterministic problem is solved by GA, PSO, and Firefly Algorithm (FA).

Shoval and Efatmaneshnik (2018) proposed a heuristic algorithm for a stochastic job shop problem considering the probability of success in a manufacturing process. They showed the probability of success related to the type of job, the tolerances, and the resources assigned to the job.

More recent works are Jamili (2019), and Lin et al. (2019). Jamili (2019) presented a new robust mathematical model for job shop scheduling problems and used a branch and bound algorithm and particle swarm optimization to solve the problem. To deal with uncertainty, using robust optimization techniques, the SJSSP model is transferred to a robust deterministic mathematical model. The author showed the effectiveness of the proposed algorithm by comparing it with two heuristic algorithms based on beam search. Moreover, the authors showed that the branch and bound method solves SJSSP problems with 12 jobs and 10 machines in about 24 minutes, while this amount is less than 4 minutes for the proposed PSO. Lin et al. (2019) transformed the stochastic job shop problem into a multi-state job shop production network. Then, they developed the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) technique for the order of preference by similarity to an ideal solution using the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) to maximize the network reliability and minimize the cost at

the same time. In their research, the main focus is on optimizing available machines in a job shop environment rather than scheduling jobs on machines.

Table 5.1 presents an overview of publications for solving the SJSSP. Most publications in solving SJSSP have considered either makespan or tardiness/earliness minimization objective functions. On the other hand, the total cost minimization objective function has been recently implemented in SJSSP. From the uncertainty modeling prospective, the uncertainty in a majority of SJSSP models has been dealt with using stochastic/probabilistic variables and/or parameters. However, there is little research on the robust estimation of these stochastic/probabilistic features of SJSSP models. Moreover, job processing times have been considered as the main uncertainty parameter by most researchers. Metaheuristics have been the main methodology applied to solve SJSSPs in the literature. To calculate the SJSSP objective values, most optimization algorithms have been integrated with a simulation model. However, the objective calculation using simulation replications has a high computation cost. Therefore, most researchers have applied three main techniques to replace the high number of simulation replications. One approach used by several researchers, such as Horng, Lin, and Yang (2012) and Yang, Lv, et al. (2014), and Akker, Blokland, and Hoogeveen (2013), is to use a small number of simulation replications executed during the search phase. For instance, although it is reported by Horng, Lin, and Yang (2012) that 10^5 simulation replications are enough to ensure the accuracy of the objective values, they performed 368 simulation replications to calculate the objective values of the proposed SJSSP (in the exploration phase, i.e., phase 1 of OO). A second approach is where researchers such as Shen and Zhu (2016) and Jamili (2019), convert SJSSP mathematical models to deterministic ones where a level of robustness and/or confidence is achieved when optimized. Typically, evolutionary methods are proposed to solve the deterministic models. A third approach is where metamodels are used. This was reported in Horng, Yang, and Lin (2012) where an RBF metamodel was used in phase 1 of OO for the optimization of a hotel booking limits problem.

Table 5.1: The relevant published journal papers of job shop problem under uncertainty; Notations: MS=(Minimize) Make-Span, TE=(Minimize) Tardiness and/or Earliness, TC=(Minimize) Total Cost, C-FT=(Minimize) Completion-Flow Time, TR=(Maximize) Total Revenue, RB=Robust, ST-P=Stochastic-Probabilistic, FU=Fuzzy, UP=Uncertain Parameter, DR=Dispatching Rules, HE=Heuristic, MH=Meta Heuristic, SI=Simulation, MM=Metamodel.

References	Objectives					Uncertainty				Methodology				
	MS	TE	TC	C-FT	TR	RB	ST-P	FU	UP	DR	HE	MH	SI	MM
Golenko-Ginzburg, Kesler, and Landsman (1995)	x	-	-	-	-	-	x	-	Processing Times	-	x	-	-	-
Lei (2008)	-	-	-	x	-	-	-	x	Processing Times	-	-	x	-	-
Gu et al. (2010)	x	-	-	-	-	-	x	-	Processing Times	-	-	x	x	-
Hasan, Sarker, and Essam (2011)	x	-	-	-	-	-	-	x	Machines Availability	-	-	x	-	-
Lei (2011)	x	x	-	-	-	-	x	-	Processing Times	-	-	x	x	-
Hornig, Lin, and Yang (2012)	-	x	-	-	-	-	x	-	Processing Times	-	-	x	x	-
Zhang, Song, and Wu (2013)	-	x	-	-	-	-	x	-	Processing Times	-	-	x	x	-
Akker, Blokland, and Hoogeveen (2013)	x	-	-	-	-	-	x	-	Processing Times	-	-	x	x	-
Yang, Lv, et al. (2014)	-	x	-	-	-	-	x	-	Processing Times	-	-	x	x	-
Sharma and Jain (2015)	x	-	-	-	-	-	x	-	Processing Times	x	-	-	x	-
Kemmoe-Tchomte, Lamy, and Tchernev (2015)	x	-	-	-	-	-	x	-	Processing Times	-	-	x	x	-
Hornig and Lin (2015)	x	-	-	-	-	-	x	-	Processing Times	-	-	x	x	x
Shen and Zhu (2016)	-	x	-	-	-	-	x	-	Processing Times	-	-	x	x	-
Hao, Gen, et al. (2017)	x	x	-	-	-	-	x	-	Processing Times	-	-	x	x	-
Shoval and Efatmaneshnik (2018)	-	-	x	-	-	-	x	-	Processing Times	-	x	-	x	-
Jamili (2019)	-	x	-	-	-	-	x	-	Processing Times	-	-	x	-	-
Lin et al. (2019)	-	-	x	-	-	-	x	-	Machines Availability	-	-	x	-	-

To address these gaps, this research presents a new method for solving SJSSPs that reduces computational time significantly while producing reasonably high quality solutions. To reduce the large computation budget allocations used in simulation replications, we propose an efficient SJSSP Discrete Event Simulation Model (DESM) metamodeling structure using GP. The proposed method then is compared with ESOO and ESOO-OCBA and typical dispatching rules (Hornig, Lin, and Yang 2012; Yang, Lv, et al. 2014) from the first category, and against GA, PSO, and FA (Shen and Zhu 2016) from the second category. A comparative analysis of ELBSO with ESOO and several dispatching rules is then presented where the input parameters are varied. Finally, a Time Intensity (TI) factor is introduced to evaluate the time efficiency of SO techniques.

5.3 Mathematical Model for Optimization of SJSSP

In this section, the mathematical model of SJSSP is detailed. Table 5.2 summarises the notation used in this section. According to the scheduling environments notation provided by Pinedo (2016) and Graham et al. (1979), we study a $J_{NM}|Range(p_i)|(\alpha \times T) + (\beta \times E)$ problem. That defines a job shop problem (J)

with NM machines under condition of random processing times, which are between a specified range, and with the goal of optimizing the total weighted expected tardiness (T) and earliness (E).

According to Shen and Zhu (2016), in SJSSP the goal of scheduling all operations of N jobs on NM machines is to minimize the expected value of total costs. Therefore, the objective function for SJSSP is as follows:

$$MinF = Min \left(\sum_{j=1}^N ((Max\{0, C_{j,NM} - d_j\} \times CT_j) + (Max\{0, d_j - C_{j,NM}\} \times CE_j)) \right) \quad (5.1)$$

Shen and Zhu (2016) also provided the following sets of constraints for SJSSP:

Sequence constraints: a job on a machine can start processing after completing its previous processing procedure,

$$S_{jw} \geq S_{j,w-1} + t'_{wj}, j \in \{1, \dots, N\}; w \in \{1, \dots, NM\} \quad (5.2)$$

Resource constraints: a job on a machine can start processing after the completion of the previous job,

$$O_{ml} \geq O_{m,l-1} + t'_{v_{m,l-1},w}, \text{ where } b_{v_{m,l-1},w} = m, w \in \{1, \dots, NM\}; l \in \{1, \dots, N\} \quad (5.3)$$

Time constraints: each job can be available at time zero,

$$S_{jw} \geq 0, j \in \{1, \dots, N\}; w \in \{1, \dots, NM\} \quad (5.4)$$

Table 5.2: Notations table.

Indices and Sets	
$j =$	Jobs index, $j \in \{1, \dots, N \}$.
$i, i' =$	operations ids, $i, i' \in \{1, \dots, NO \}$.
$w =$	operations indices, $w \in \{1, \dots, NM \}$.
$m =$	Machine index, $m \in \{1, \dots, NM \}$.
$l =$	positions on machines index, $l \in \{1, \dots, N \}$.
$k =$	Queuing position index, $k \in \{1, \dots, NO \}$.
$\Omega =$	Precedence orders sets defining the execution precedence of operations of the same jobs.
$s =$	Simulation replication index, $s \in \{1, \dots, SL \}$.
Parameters	
N	Number of jobs.
NO	Total number of operations.
NO_j	Total number of operations for job j .
NM	Number of machines.
SL	Total number of simulation replications indexed by s .
d_j	Due date of job j .
P'_i	Stochastic processing time of operation id i .
ϕ_i	Probability distribution of processing time of operation id i .
t'_{wj}	Stochastic processing time of operation w of job j .
CE_j	Earliness cost of job j caused by inventory costs.
CT_j	Tardiness cost of job j caused by tardiness in delivering the job.
$V = (v_{ml})_{NM \times N}$	The process matrix, where $v_{ml} \in \{1, \dots, N \}$ denotes that job v_{ml} is processed at machine m in position l .
$B = (b_{jw})_{N \times NM}$	The operation matrix, where $b_{jw} \in \{1, \dots, NM \}$ denotes a machine. Elements $\{b_{j1}, b_{j2}, \dots, b_{jNO_j}\}$ in the j^{th} row represent operations of job j . That is the job j is processed orderly on machines $b_{j1}, b_{j2}, \dots, b_{jNO_j}$.
Decision Variables	
S_{jw}	The starting time of operation w of job j .
O_{ml}	The starting time of l^{th} job processed on machine m .
C_{jw}	The completion time of w^{th} operation of job j .
Q_{ml}	The completion time of a job processed on l^{th} position of machine m .
X_{ik}	If operation id i is assigned to the k^{th} position of the dispatching queue, then $X_{ik} = 1$, 0 otherwise.
T'_{js}	The stochastic tardiness time of job j in the simulation replication s .
E'_{js}	The stochastic earliness time of job j in the simulation replication s .
Functions	
f_s	The objective calculation function for the simulation replication s .

In this research, the above model is converted to a SO model, where, during the optimization procedure, values will be obtained from an evaluation model, which will be a metamodel in phase 1 of OO or a simulation model in phase 2. Before introducing the model formulation, the following example is considered to illustrate the SO mathematical model of the SJSSP. Consider the vector $vec = (i, M_i, P'_i)$ that defines the stochastic processing time P'_i of operation i on the planned machine M_i . As shown in Figure 5.1, there are $N = 3$ jobs with $NO_1 = 3$, $NO_2 = 3$, and $NO_3 = 1$ operations for job $j = 1, 2$ and 3 , respectively. Take, for example the vector $(5, M_1, 4)$ from the problem input (operation environment), this refers to operation number 5, planned to be assigned to machine 1, while its processing time equals to 4, noting that for illustrative purposes we use deterministic times. Operation 5 (note that we sequentially number the operations across all jobs in this chapter denoted as operations ids) is the second operation of job 2, which is assigned to the 3rd position of the queue (see Figure 5.1). Therefore, as it is the second operation of job 2, and observing precedence constraints, it must follow the first operation (4) of

job 2. Thus, we denote that $X_{53} = 1$; values we use in the mathematical formulation presented below. The “Job Shop Queue (Queue)” in Figure 5.1 are the inputs used in the evaluation models (metamodel in phase 1 and simulation model in phase 2).

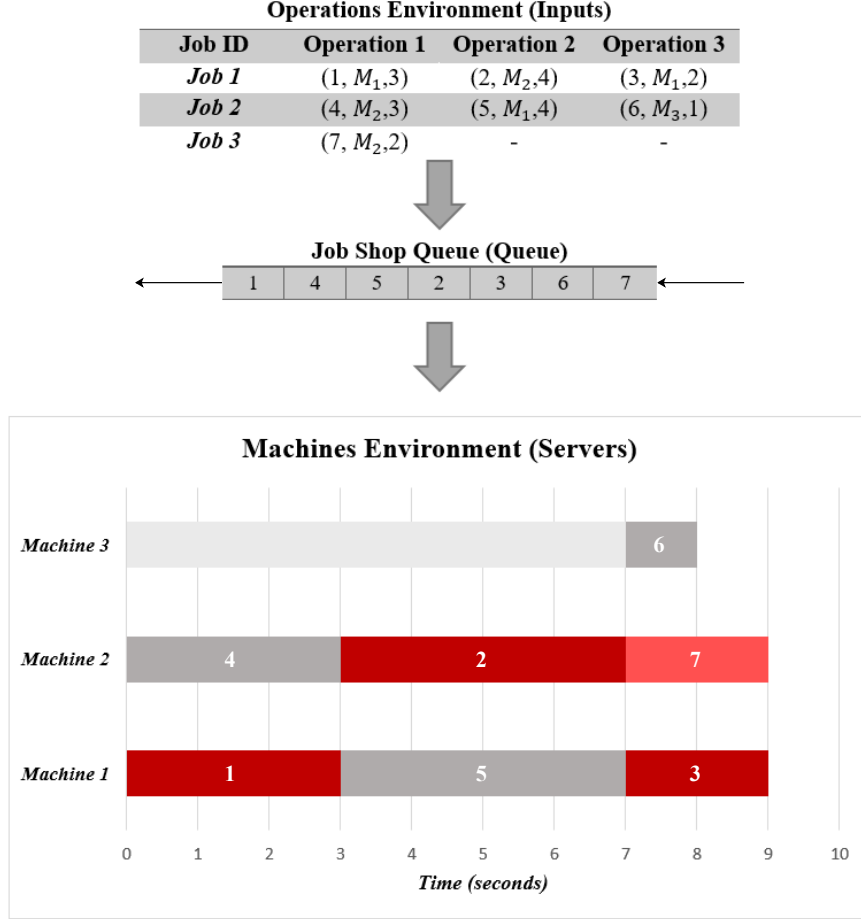


Figure 5.1: Proposed job shop scheduling problem example.

Equation 5.1 is transformed into Equation 5.5, for use with the evaluation model, where f_s calculates total costs of solution X_{ik} by the simulation replication s . Besides, F defines the fitness value for a SJSSP solution.

$$Min F = Min \left(\frac{1}{SL} \sum_{s=1}^{SL} f_s (\{\cup_{i=1}^{NO} \cup_{k=1}^{NO} X_{ik}\}, P'_i) \right) \quad (5.5)$$

In each solution, X_{ik} , is a binary variable that denotes if operation i is assigned to the k^{th} position of the dispatching queue, with the above objectives having the following constraints. Each job should be assigned to one of the existing dispatching

queue positions as follows:

$$\sum_{k=1}^{NO} X_{ik} = 1, \forall i \in \{1, \dots, NO\} \quad (5.6)$$

Moreover, it must be guaranteed that for each existing position in dispatching queue k , at most one operation is assigned. Equation 5.7 defines this constraint as follows:

$$\sum_{i=1}^{NO} X_{ik} = 1, \forall k \in \{1, \dots, NO\} \quad (5.7)$$

Precedence of operations of a job are defined in the set Ω . Consequently, Equations 5.8 and 5.9 construct a precedence relationship between two operations i and i' from the same job j in the SJSSP. In other words, when operation i (assigned to the position k) precedes operation i' , operation i' must be assigned to a position k' ($k' > k$) on the dispatching queue.

$$X_{ik} \geq X_{i'k'}, \forall i, i' \in \Omega, \forall k, k' \in \{1, \dots, NO\}, k < k' \quad (5.8)$$

$$X_{ik} - (X_{ik}X_{i'k'}) \geq X_{i'k'}, \forall i, i' \in \Omega, \forall k, k' \in \{1, \dots, NO\}, k > k' \quad (5.9)$$

Equation 5.10 defines the decision variable feature in the model:

$$X_{ik} \in \{0, 1\}, \forall i, k \in \{1, \dots, NO\} \quad (5.10)$$

All in all, Equations 5.6 to 5.10 guarantee the feasibility of queue solution X_{ik} by considering both assignment and precedence constraints.

Algorithm 9 presents the simulation model procedure to calculate the objective value f for queue solution X in a simulation replication.¹ It is worth mentioning that as queue solution X consists of NO queue positions filled by NO operations,

¹Appendix D.7 and D.8 define the actual MATLAB codes utilized to create the simulation model.

the problem size based on queue solutions equals *NO!*. This proves the NP-Hardness of the proposed SJSSP even without considering stochastic features Horng, Lin, and Yang 2012. While, since the operation processing times fluctuate stochastically in SJSSP, the completion times of jobs cannot be defined. Even if the sequence has been predefined using queue solution X , it would become invalid due to the change of the operation realised process time.

Algorithm 9: Simulation algorithm.

Inputs : X (queue solution), ϕ (processing times distributions), First (all first operations of jobs set), Last (all last operations of jobs set), N , NO , CT , CE

Output: f (fitness value)

begin

for $i = 1$ **to** NO **do**

┌ $-P'_i = Rand(\phi_i)$; % Create a random value for the processing time of
└ operation i from the probability distribution ϕ_i

- Assign operations to machines based on the sequence of operations on the queue solution X and define the vector $vec_i = (i, M_i, l, P'_i)$ for the stochastic processing time P'_i of operation i on l^{th} position of the planned machine M_i , for $i = \{1, \dots, NO\}$;

for $k = 1$ **to** NO **do**

┌ $-index = find(i || X_{ik} \geq 0)$;

if $vec_{index}(3) == 1$ **then**

┌ **if** $index \in First$ **then**

└ $-completion_{index} = P'_{index}$;

else

└ $-completion_{index} = P'_{index} + completion_{index-1}$;

else

┌ **if** $index \in First$ **then**

└ $-index2 = find(i || vec_i(3) = vec_{index}(3) - 1 \& vec_i(2) = vec_{index}(2))$;

└ $-completion_{index} = P'_{index} + completion_{index2}$;

else

└ $-index3 = find(i || vec_i(3) = vec_{index}(3) - 1 \& vec_i(2) = vec_{index}(2))$;

└ $-completion_{index} =$

└ $P'_{index} + Max\{completion_{index-1}, completion_{index3}\}$;

for $j = 1$ **to** N **do**

┌ $-Jcompletion_j = completion_{Last_j}$;

- $f = \sum_{j=1}^N ((Max\{0, Jcompletion_j - d_j\} \times CT_j) + (Max\{0, d_j - Jcompletion_j\} \times CE_j))$;

- Return f ;

5.4 Evolutionary Learning Based Simulation Optimization (ELBSO) Method

Given the high complexity of SJSSP for large problems Pinedo 1982, we focus our attention on the development of a 2-phase ELBSO approach consisting of an evolutionary algorithm embedded in an OO structure for solution improvement. The OO concept was first introduced by Ho, Zhao, and Jia (2008). Recently, in a survey on simulation optimization, the quality of OO in solving problems with a large solution space is referenced in Liu, Xie, et al. (2018). In the OO methodology, firstly a subset of good quality solutions (Ψ) from the main solution space (Θ) is detected. Researchers have embedded OO within SO concepts (e.g., (Horng, Lin, and Yang 2012; Yang, Lv, et al. 2014)), but they require a high number of simulation replications which are well known to be computation intensive. In the ELBSO method presented here, a novel learning metamodel is used to reduce the computational overhead in solving the SJSSP.

After developing a DESM of the proposed SJSSP, Figure 5.2 shows the general structure of ELBSO, with fuller details described in the following subsections. In the GP preparation phase of ELBSO, a data set is produced using the DESM of the SJSSP for different replications and scenarios. Then, using this data set, a GP model is trained to estimate the SJSSP objective value. In phase 1, an initial random set pop of population size $popsize$ is generated. Next, a set $npop$ of population size $npopsize$ of offsprings is created using a Neighbourhood Search Function (NSF) (Beyer and Schwefel 2002). All solution objective values in both pop and $npop$ sets are evaluated using GP in the selection module, and $popsize$ best solutions are selected for the next generation. These steps are replicated until *termination* is achieved. Subsequently, phase 2 is started by running the simulation model R_p times for each solution $p \in pop$ (R_p defines the number of simulation replications in phase 2 for each solution). In this phase, we use a method in ELBSO which we call Simulation Budget Allocation (SBA), however other procedures such as OCBA or

rank and selection could be used. To perform the SBA procedure, *popsize* is deducted by the specified *didrate* rate, which is enabled in ELBSO to reduce the population size iteratively and as a result allocate more simulation replications on elite solutions. Then, using the objective value obtained from $\sum_{p \in pop} R_p$ replications, where *popsize* is reduced by *didrate*, the best solutions are selected. Phase 2 is replicated until *termination* is achieved.

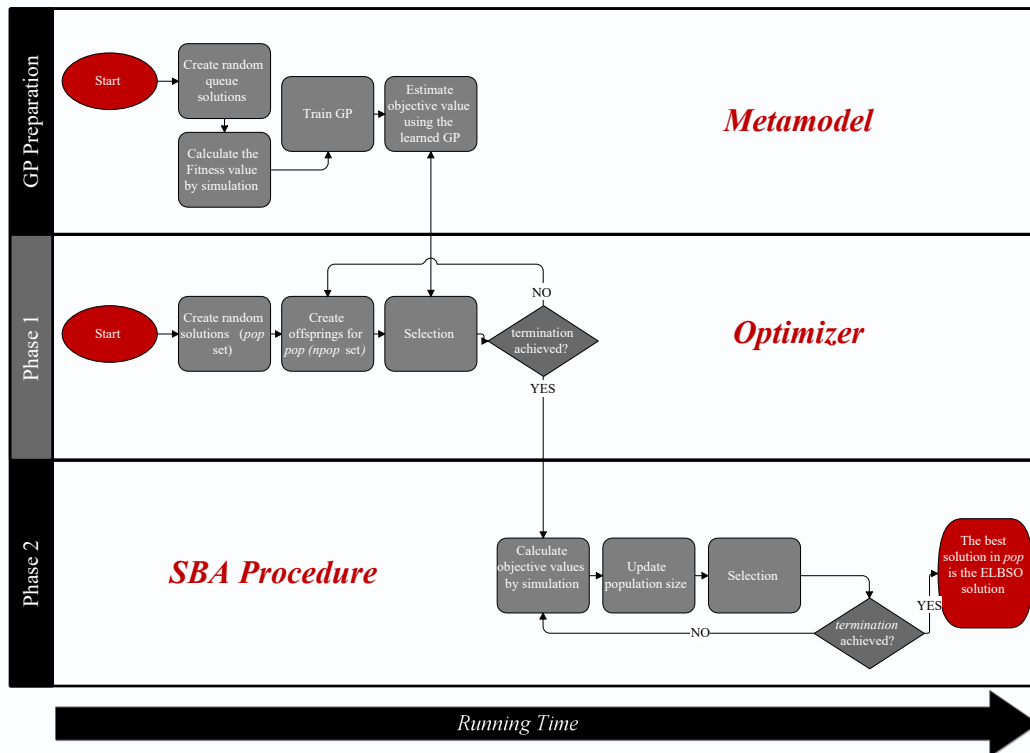


Figure 5.2: The general framework of ELBSO method.

5.4.1 ELBSO Solution Structure

Similar to most evolutionary algorithms, ELBSO defines solutions in a chromosome structure. In ELBSO, chromosomes are represented as a string or vector of randomly generated integer numbers in a predetermined interval. A decoder takes as input any chromosome and associates with it a solution of the combinatorial optimization problem for which an objective value or fitness can be computed.

Consider the SJSSP solution example in Figure 5.1. The chromosome structure for this solution in ELBSO is shown in Figure 5.3. In other words, SJSSP queue

solutions are ELBSO chromosomes. To illustrate, operations 1 to 7 are scheduled on machines based on their priorities a_1 to a_7 ($a_1 = 1^{st}$ priority, $a_2 = 2^{nd}$ priority and the same for the rest of the priorities). It is worth mentioning here that to guarantee the feasibility of each solution, precedence constraints must be considered. For instance, in Figure 5.3 operation 4 has higher priority than operation 5 (a_2 and a_3) because operation 4 is the first operation of job 2 and operation 5 is the second operation of the same job.

1	4	5	2	3	6	7
a_1	a_2	a_3	a_4	a_5	a_6	a_7

Figure 5.3: Solution structure in ELBSO.

5.4.2 GP Preparation: Metamodeling the SJSSP DESM using GP in ELBSO

As noted before, simulation replications are highly time-intensive. Furthermore, it is increasingly necessary to design larger scale systems, using hierarchical models of entire production systems, to achieve the necessary major gains in performance, and, as the models become larger, they become prohibitively demanding of computer memory. Metamodeling provides one approach to statistically summarize simulation results, allowing some extrapolation from the simulated range of system conditions, and therefore, potentially offering some assistance in optimization (Amir Haeri, Ebadzadeh, and Folino 2017).

Most metamodeling techniques proposed within the literature are mainly driven by Graph-Based Machine Learning (GBML) concepts. The architecture of these methods consists of nodes and edges describing the metamodeled system's features and interactions between features, respectively (Jin, Chen, and Han 2017). The metamodeling structure of NNs presented by Dunke and Nickel (2020) and gene regulatory network reconstruction based on fuzzy cognitive maps proposed by Liu, Chi, et al. (2019) are two recent cases of such ML approaches. As an alternative,

GP evolving programs in a domain via symbolic regression has been known as one of the most efficient metamodeling tools (Koza 1994). To metamodel a DESM, Can and Heavey (2012) compared GP with GBML (NN) in terms of the estimation accuracy. Their result shows the superior quality of GP in metamodeling DESMs of production systems.

The design of GP requires certain components to be defined to emulate the evolutionary process. These involve n -ary arithmetic functions, system decision variables, and evolutionary operators that support crossover and mutation, to allow the generation of symbolic expressions. GP attempts to learn a mathematical function approximating the relationship between the input and the output through the use of evolutionary mechanisms. This use of GP is referred to as symbolic regression (Koza 1994). Evolutionary algorithms are powerful techniques to find solutions to many difficult real-world search and optimization problems. New offspring solutions are sought by information exchange among a population of previously discovered parent solutions. This is performed using operators inspired by evolution, e.g., crossover, mutation. The distinct feature of evolutionary algorithms is their robustness in adapting to a certain problem through the effective use of encoding to represent a solution with flexibility in the search as a consequence of parallel processing of individual solutions within a population. For instance, the solutions in GP are generally defined in the form of trees. Figure 5.4 illustrates tree-based representation by arbitrary symbolic regression functions. It can be seen that the solution is formed by several arithmetic operators and problem variables. In the context of symbolic regression, the former set is referred to as functional primitives and the latter as the terminal set. While the functional primitives are common operators like $\sin, \log, \exp, x^3 \dots$, the terminal set includes the variables and constants of the problem. It is worth mentioning that GP evolves symbolic regression models without prior assumptions on the interaction of variables. In contrast, GP is allowed to discover such interactions itself and to evolve new functions based on the performance of the solutions on the training data.

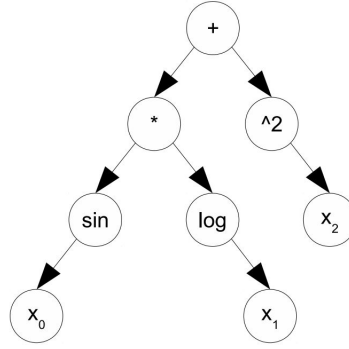


Figure 5.4: Tree-based encoding of a function, e.g., $(\sin(x_0) \times \log(x_1)) + x_2^2$, in GP for symbolic regression illustrating functional primitives and terminal set (Can and Heavey 2011).

Figure 5.5 shows the general framework of GP considered within this research. Firstly, *ndb* queue input data solutions and their objective function values are calculated using *SL* simulation replications using Equation 5.5. Then the created data set is transformed to a set of vectors (*Training Vector* see Figure 5.5). In each vector ($training_i$), features of a solution are defined using $T - 1$ elements (t_1, \dots, t_{T-1}) . These elements of the training vector are used by GP to estimate a solution's objective value, the details of which will be defined later. The last element of this vector (t_T) is the solution's objective value. Next, a population of random GP trees are created (*GP Initial Population*). Note that GP uses a tree-based representation to evolve symbolic regression (e.g., $Tree_1 \rightarrow t_T = t_1^{1.6} + \sqrt{t_2} + \dots$). The quality in estimating objective values is defined by calculating the estimation error, where the estimated objective value t'_T is compared with t_T for each tree. Subsequently, through an iterative procedure (*Generations*), the initial population is improved. This accrues by generating a set of offspring for the initial population using a neighbourhood search methodology in each iteration (*Replication*). In other words, in each generation of GP, a set of $2 \times Replication$ offspring trees are created from the initial population of trees. To illustrate the offspring creation procedure (see Figure 5.5) two arbitrary random solutions (*Parents*) from the initial population are recombined by swapping their branches, this creates two new solutions (*Children*). This procedure replicates until $2 \times Replication$ trees are created in each generation.

After creating a set of child solutions, all initial and child solutions are compared in terms of their error level in estimating T_t , and the best *pop* solutions are nominated as the next generation initial population. This procedure is replicated until termination is achieved (*Generations*).

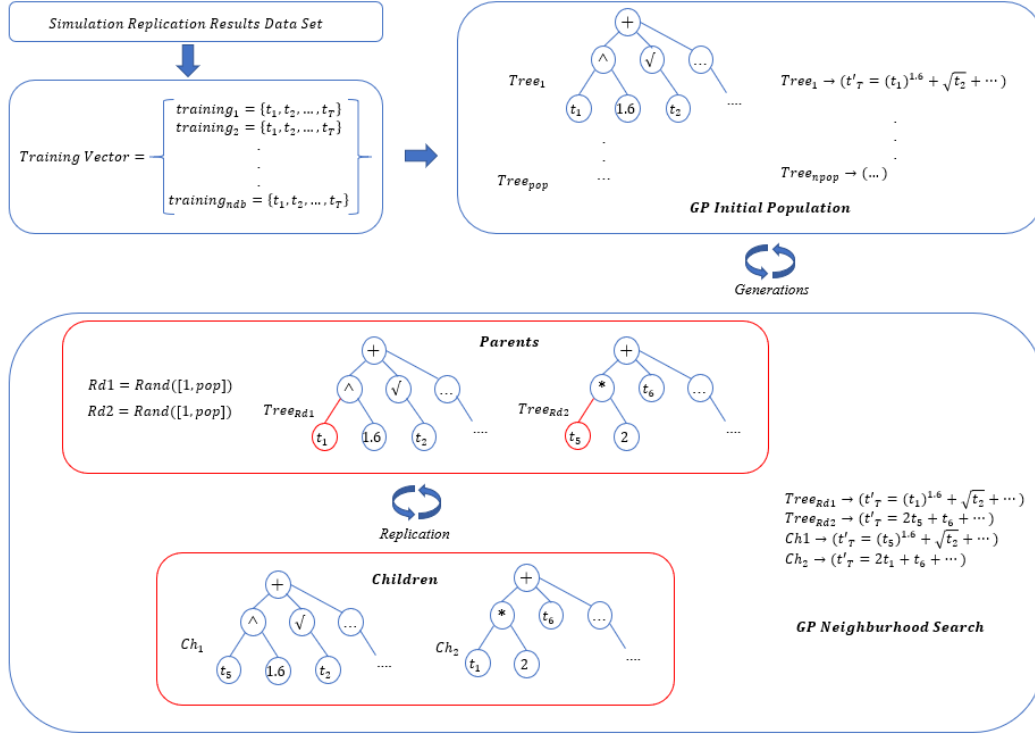


Figure 5.5: Proposed GP metamodeling procedure general framework.

In the literature, GP is mostly evolved for dispatching rules or priority rules (Nguyen, Mei, and Zhang 2017). However, there is little research addressing GP for performance estimation of production systems (Can and Heavey 2011). In this research, we propose a novel methodology to evolve queue performance estimation by GP. As Figure 5.2 shows, GP is used in the selection module of the first phase of ELBSO. In other words, instead of replicating the simulation model to evaluate each solution’s quality, GP estimates the objective function value for each solution of the population. The question here is: What metrics of the solution are used to train the GP to be capable of estimating the objective function of SJSSPs?

To answer this, we defined 2 types of metrics: Delay Based Metrics (DBMs) and Sequencing Priority Based Metrics (SPBMs). DBMs and SPBMs are based

on SJSSPs inputs and chromosome structures, respectively. Equation 5.11 defines the DBM^c metric proposed to train GP, where c refers to solution c . Assume that each operation has a stochastic processing time, where a sample is denoted by $D = \{d_1, d_2, \dots, d_{NO}\}$. Equation 5.11 calculates the DBM^c metric, which is used with the SPBM metrics (described below), for solution c used to train the GP metamodel.

$$DBM^c = \frac{STD(D)}{Mean(D)} \quad (5.11)$$

As well as the DBM metric, SPBM metrics are also used to train GP. The SPBM metric is based on the sequencing priorities of the SJSSP chromosome. To develop this, we use the vector $solS_k(sol_c(k), c, u, m)$ for each operation, where $sol_c(k)$ describes the operation in position k in the chromosome (queue), c as defined previously is the solution number and u is the position of $sol_c(k)$ on machine m . In addition, to calculate the training vector (*Learning*), we categorize the operations into two categories; the first, operations of jobs, and the second, operations. Then, $SP_{sol_c(k)}^c$ for operation $sol_c(k)$ positioned in the k^{th} position of the queue of solution c is calculated as follows. If $sol_c(k)$ is the first operation of any job j in solution c , and it is assigned to the first position of machine m then:

$$SP_{sol_c(k)}^c = 0 \quad (5.12)$$

If $sol_c(k)$ is the first operation of job j in solution c , and it is assigned to the u^{th} position ($u > 1$) of machine m :

$$SP_{sol_c(k)}^c = SP_{sol_c(k')}^c + 1, \quad (5.13)$$

where $SP_{sol_c(k')}^c$ defines the value of SP for the operation $sol_c(k')$ assigned to the position $u - 1$ on machine m and queue position k' ($k' < k$) in solution c .

If $sol_c(k)$ is not the first operation of any job j , and it is assigned to the first position of machine m in solution c :

$$SP_{sol_c(k)}^c = SP_{sol_c(k)-1}^c + 1, \quad (5.14)$$

where $SP_{sol_c(k)-1}^c$ defines the value of SP for the operation $sol_c(k) - 1$ of job j assigned to position u' on machine m' in solution c .

If $sol_c(k)$ is not the first operation of any job j , and it is assigned to the u^{th} position ($u > 1$) of machine m in solution c :

$$SP_{sol_c(k)}^c = SP_{sol_c(k)-1}^c + SP_{sol_c(k')}^c + 2, \quad (5.15)$$

where $SP_{sol_c(k)-1}^c$ defines the value of SP for the operation $sol_c(k) - 1$ of job j assigned to position u' on machine m' for solution c , and $SP_{sol_c(k')}^c$ defines the value of SP for operation $sol_c(k')$ ($k' < k$) assigned to position $u - 1$ on machine m for solution c . Consequently, $SPBM_j^c$ is defined as follows:

$$SPBM_j^c = \sum_{sol_c(k) \in Y_j} SP_{sol_c(k)}^c, \quad (5.16)$$

where Y_j is the set of operations for job j . Finally, GP is trained using the set $Learning_c$ as the learning set for solution c .

$$Learning_c = \{\{DBM^c\} \cup \{\cup_{j=1}^N SPBM_j^c\} \cup F\} \quad (5.17)$$

Note that F defines the value of Equation 5.5 using SL simulation replications. Considering the total number of ndb training solutions, the final training set $Learning$ can be defined as:

$$Learning = \{\cup_{c=1}^{ndb} Learning_c\} \quad (5.18)$$

In this research, a GP is trained by using the set defined as $Learning$. Using the HeuristicLab (2020), a GP is created to metamodel the SJSSP DESM. Algorithm 10 summarises the GP preparation phase in ELBSO.²

²Appendix D.9 defines the actual MATLAB codes utilized to create the training vector in ELBSO.

Algorithm 10: GP Preparation (the training set creation).

Inputs : ndb (total number of GP training and testing solutions), $First$ (all first operations of jobs set), N , NO

Output: $Learning$ (training set)

begin

for $c = 1$ **to** ndb **do**

- Create a vector of a random feasible queue solution (chromosome) and name it as $sol_c = \{a_1, a_2, \dots, a_{NO}\}$;

for $k = 1$ **to** NO **do**

- Define vector $solS_k(sol_c(k), c, u, m)$, where $sol_c(k)$ describes the operation in position k in the chromosome, c defines the solution, u describes the position of $sol_c(k)$ on machine m .

- Calculate the fitness value (F using Equation 5.11) from the DESM of SJSSP using SL replications;

- $DBM^c = \frac{STD(D)}{Mean(D)}$;

for $k = 1$ **to** NO **do**

if $solS_k(1) \in First$ **then**

if $solS_k(3) = 1$ **then**

| - $SP_{sol_c(k)}^c = 0$;

else

| - $SP_{sol_c(k)}^c = SP_{sol_c(k')}^c + 1$; where $SP_{sol_c(k')}^c$ defines the value of SP for the operation $sol_c(k')$ assigned to the position $u - 1$ on machine m in solution c .

else

if $solS_k(3) = 1$ **then**

| - $SP_{sol_c(k)}^c = SP_{sol_c(k)-1}^c + 1$;

else

| - $SP_{sol_c(k)}^c = SP_{sol_c(k)-1}^c + SP_{k'}^c + 2$;

- $SPBM_j^c = \sum_{sol_c(k) \in Y_j} SP_{sol_c(k)}^c, \forall j = \{1, \dots, N\}$; % Y_j is the set of operations for job j .

- $Learning_c = \{\{DBM^c\} \cup \{\cup_{j=1}^N SPBM_j^c\} \cup F\}$;

- $Learning = \{\cup_{c=1}^{ndb} Learning_c\}$;

To illustrate the set of *learning* calculations here we extend the example provided in Figure 5.1. To ensure calculation validity, SP values for all operations are calculated based on the queue sequence (start with operation 1, next 4 and etc.). As operation 1 is the first operation of job 1 and it is assigned to the first position on machine 1, using Equation 5.12, SP_1^1 equals to 0. Similarly, the SP value for operation 4 equals to 0 ($SP_4^1 = 0$). Operation 5 is not the first operation of any job and it is not assigned to any first position on a machine. Therefore, using Equation 5.15, $SP_5^1 = SP_4^1 + SP_1^1 + 2 = 2$. Likewise, the SP value for operations 2 and 3 are 2 and 6, respectively ($SP_2^1 = 2$ and $SP_3^1 = 6$). Operation 6 is not the first operation of any job, and it is assigned to the first position on machine 3. Thus, using Equation 5.14, SP_6^1 equals to 3 ($SP_6^1 = SP_5^1 + 1$). Operation 7 is the first operation of job 3, and it is not assigned to the first position of any machine. Therefore, using Equation 5.13, SP_7^1 equals to 3 ($SP_7^1 = SP_2^1 + 1$). By aggregating the SP values of operations of each job (Equation 5.16), $SPBM_1^1$, $SPBM_2^1$ and $SPBM_3^1$ are equal to 8, 6 and 3, respectively (e.g., $SPBM_1^1 = SP_1^1 + SP_2^1 + SP_3^1$). Considering processing time values for all job operations in Figure 5.1, using Equation 5.11, DBM^1 equals to 0.379532. Consider the due date of all jobs is equal to 1. Besides, both earliness and tardiness costs are equal to 1. Therefore, using Equation 5.1, the objective value F for the current example solution equals to 23. Finally, the GP training set of the proposed solution is defined in Equation 5.19.

$$Learning_1 = \{0.379532, 8, 6, 3, 23\} \quad (5.19)$$

5.4.3 Phase 1 of ELBSO algorithm

In the first phase of ELBSO, a set of pop random initialized solutions are improved through a number of generations. In each generation, $npop$ neighbourhoods are created from the main population. Finally, pop solutions created from the $pop+npop$ solutions are selected for the next generation based on their objective function value estimated by the trained GP.

A neighbourhood search operation is considered as the primary process for exploration in evolutionary algorithms. Neighbourhood search is done by recombining the information of two parents to provide a powerful exploration capability. There are many neighbourhood search operators such as discrete recombination, intermediate recombination, line recombination, and binary valued recombination (Beyer and Schwefel 2002). In our approach, both crossover and mutation operators are used to perform neighbourhood search, which uses discrete recombination for crossover and random insertion for mutation. In crossover, two new chromosomes known as children, are created from two different individuals known as parents, with the probability of being a parent for all solutions in the population being equal. Due to precedence constraints, after implementing a crossover operator, some chromosomes may be infeasible. Therefore, a repairing procedure is implemented on infeasible solutions. In the mutation operator, a new chromosome (child) is created from a parent chromosome. To illustrate, consider 3 jobs in which there are 3, 2, and 2 operations for jobs 1, 2, and 3, respectively. Figure 5.6 shows the crossover procedure in ELBSO for two arbitrary chromosomes considered as parents (P1 and P2). By considering a cut point after the fourth gene in both parents, CH1 and CH2 are created and known as children. Consequently, some duplication may occur after implementing the crossover operator (e.g., operations numbers 4 and 7 in CH1) causing infeasibility. Therefore, as shown in RCH1 and RCH2 known as repaired chromosomes, the last duplicated genes are removed, where unallocated operations are inserted randomly in the chromosomes by considering sequence dependent constraints to guarantee the feasibility of each chromosome (e.g., operations 2 and 3 in FCH1). Similarly, Figure 5.7 illustrates the mutation operator in ELBSO for the example provided. In the mutation operator, a gene in the parent chromosome is selected, deleted, and reinserted in the parent chromosome. Here, operation 2 is firstly deleted from the parent (P). Then, considering the feasibility, it is inserted in between positions 3 and 4 in the chromosome. Algorithm 11 summarises the first

phase of ELBSO.³

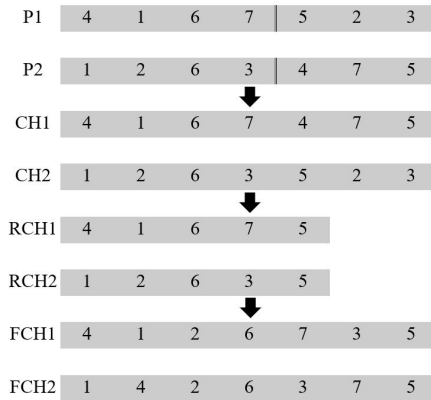


Figure 5.6: Crossover in ELBSO.

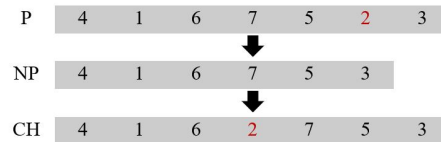


Figure 5.7: Mutation in ELBSO.

5.4.4 Phase 2 of ELBSO algorithm

In the second phase of ELBSO a SBA procedure is implemented. As Algorithm 12 shows, there are *termination* iterations in the second phase of ELBSO. In each iteration, the total number of Rp simulation replications are executed. Therefore, we consider $SL = \frac{Rp}{popsize}$ number of simulation replications for each solution, where *popsize* refers to the number of solutions in *pop*. Then, all solutions in *pop* are ranked based on their fitness value f_j (from Equation 5.5). At the end of each iteration, *popsize* is reduced by $popsize \times didrate$, where *didrate* defines the population size deduction rate per iteration. Finally, we rank the updated *popsize* solutions according to their fitness values. As Rp is a constant parameter, reducing *popsize* increases SL . Therefore, more simulation replications are used for elite solutions. This procedure is executed until the required number of iterations equal to *termination* value

³Appendix D.1 to D.5 define the actual MATLAB codes utilized to create the first phase of ELBSO.

Algorithm 11: Phase 1 of ELBSO algorithm.

Inputs : *popsiz*e (population size), *npopsiz*e (neighbourhood solutions size), *termination* (number of iterations)

Output: *pop*

begin

- for** $i = 1$ **to** *popsiz*e **do**
 - Create a random chromosome;
 - Calculate the fitness value using the learned GP;
 - Store the created chromosome in *pop*;
- for** $i = 1$ **to** *termination* **do**
 - for** $j = 1$ **to** *npopsiz*e : *increment by 2* **do**
 - Create two new solutions using the crossover operator from two selected random solutions in *pop*;
 - Implement the mutation operator on both new solutions;
 - Calculate the fitness value using the learned GP for both new solutions;
 - Store both *npopsiz*e created solutions and *pop* in *npop1*;
 - $pop = \{\}$;
 - Store *popsiz*e best solutions of *npop1* based on their GP estimated fitness value in *pop*;
- Return *pop*;

is achieved. After finishing all iterations, the best ranked solution in *pop* is named as the best solution from the ELBSO algorithm.⁴

5.5 GP Calibration

Parameter settings of an evolutionary algorithm have a deep influence on its performance. Therefore, we calibrate the parameter values for GP by using Taguchi experimental design. This method is based on a special set of arrays called orthogonal arrays to conduct the minimum number of efficient experiments that could give insights on all factors that affect the performance measure.

In evaluating GP solutions, we used the Mean Relative Error (MRE) which is known as one of the most practical metrics for evaluating estimation techniques (Park and Stefanski 1998). The general formulation of MRE is as follows:

⁴Appendix D.6 defines the actual MATLAB codes utilized to create the second phase of ELBSO.

Algorithm 12: Phase 2 of ELBSO algorithm.

Inputs : *popsize* (population size), *pop* (population set), *termination* (number of iterations), *Rp* (total number of simulation replications), *didrate* (population size reduction rate)

Output: *Sol* (best solution from ELBSO)

begin

- for** $i = 1$ **to** *termination* **do**
 - for** $j = 1$ **to** *popsize* **do**
 - Run $SL = \frac{Rp}{popsize}$ simulation replication on each solution in *pop* and calculate the fitness value f_j for each solution by Equation 5.5;
 - Update the *popsize* using *didrate* by setting a new $popsize = popsiz \times didrate$;
 - Rank solutions based on their fitness values, by using the new *popsize* best solutions, and delete the other solutions;
- Return *Sol*=the best ranked solution in *pop*;

$$MRE = \frac{1}{ss} \times \sum_{b=1}^{ss} \frac{|Y'_b - Y_b|}{Y_b}, \quad (5.20)$$

where Y'_b defines the estimated value of the factor Y_b in the sample b , and total number of samples equals ss .

GP used in HeuristicLab has five parameters: Population Size (*PS*), Mutation Probability (*MP*), Maximum Generations (*MG*), Tree Length (*TL*), and Tree Depth (*TD*). To perform the test, different stage values need to be selected for each parameter. Table 5.3 describes the different parameters evaluated using the Taguchi method. After running a number of preliminary experiments of GP in HeuristicLab, we picked these parameter values as these affected metamodel generation. The main effects plots are shown in Figure 5.8, which shows how each factor affects the response characteristics. A main effect exists when different levels of a factor affect the characteristics differently.

Based on the number of parameters and their set values, the Taguchi test required 27 different GP test runs. After running the test, the values 1000, 0.20, 500, 150 and 25 are selected for *PS*, *MP*, *MG*, *TL* and *TD*, respectively.

Using the HeuristicLab tool version 3.3 (HeuristicLab 2020) we designed a GP metamodel with the discussed parameters. To demonstrate the accuracy of the

Table 5.3: Taguchi test parameters.

Parameter	1	2	3
<i>PS</i>	500	750	1000
<i>MP</i>	0.15	0.20	0.25
<i>MG</i>	100	250	500
<i>TL</i>	100	125	150
<i>TD</i>	20	25	30

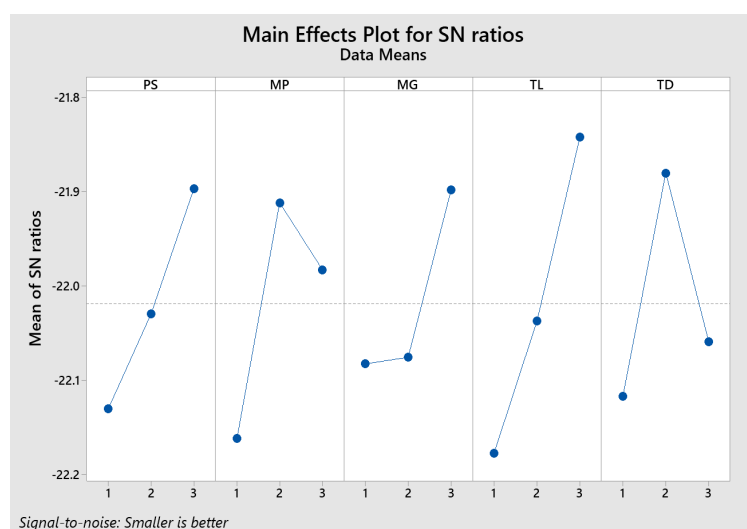


Figure 5.8: Taguchi test results.

metamodel, Figure 5.9 shows example results for the training and testing of the the GP metamodel. In Figure 5.9, the axis labeled “X009” provides the objective values with the other axis showing the solutions. Figure 5.9 also shows an extracted portion that demonstrates the quality of the metamodel. In the training of the GP metamodel, 2000 solution where used and 1000 solution samples for testing. Target values, training values by GP, and obtained test values by GP are shown in blue, yellow, and red, respectively. It is quite clear that GP has a good quality not only in predicting solution fluctuations, but also their objective values. Furthermore, we achieved the value of $MRE=0.09$ showing the efficacy of the trained GP in estimating SJSSP objective values.

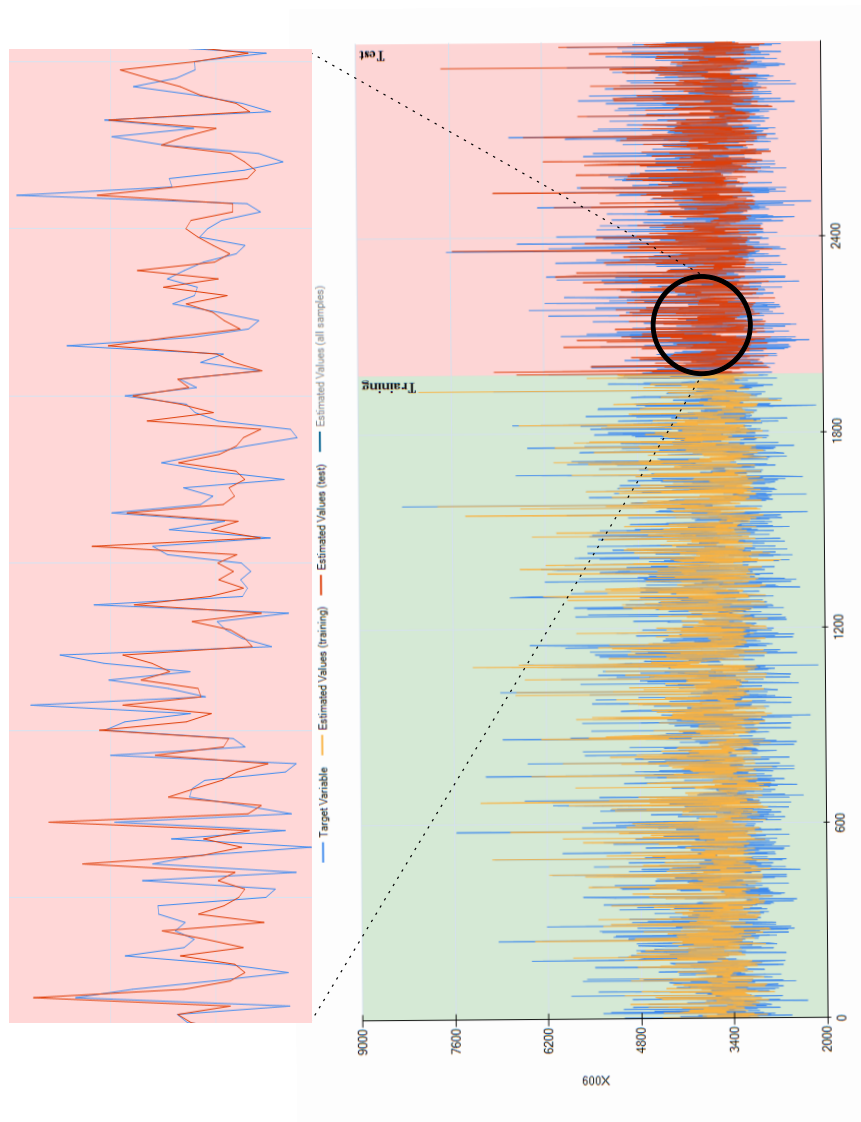


Figure 5.9: GP line chart result.

5.6 Experiment Results and Sensitivity Analysis

The main objective of this research is to present a new learning-based SO method to solve SJSSPs fast and accurately. Thus, in this section, ELBSO, which is proposed in Section 5.4, will be compared with the published results and a comparative analysis is carried out. Horng, Lin, and Yang (2012) proposed an evolutionary SO method embedded in OO denoted as ESOO to solve SJSSP. In a follow-on study, Yang, Lv, et al. (2014) modified ESOO by applying OCBA which they called ESOO-OCBA. In subsection 5.6.1, we will compare ELBSO with these results. More recently, Shen and Zhu (2016) addressed SJSSP by converting the stochastic mathematical model to a deterministic one using the chance constraint technique. They solved SJSSPs using the chance constrained model with 0.8 confidence level using Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Firefly Algorithm (FA). In Subsection 5.6.2, ELBSO is compared against these results. A comparative analysis where the input parameters are varied is provided in Subsection 5.6.3 against ELBSO and our own re-implementation of ESOO and four dispatching rules, Critical Ratio (CR), Earliest Due Date (EDD), Shortest Processing Time first (SPT), and Longest Processing Time first (LPT), which have been widely applied to SJSSPs (Ferreira, Figueira, and Amorim 2020). Lastly, in Subsection 5.6.4 we compare the computation time efficiency of ELBSO with ESOO and four typical dispatching rules.

5.6.1 Comparison with ESOO Horng, Lin, and Yang (2012) and ESOO-OCBA Yang, Lv, et al. (2014)

In this subsection, ELBSO is compared with two alternative SO solution methods presented in Horng, Lin, and Yang (2012) and Yang, Lv, et al. (2014). The input data presented in Horng, Lin, and Yang (2012) considers a SJSSP with 8 jobs with 8 operations per job to be executed on 8 different machines Horng, Lin, and Yang 2012. Table 5.4 shows vector (i, μ, σ^2) of the test problem environment, where

Table 5.4: The test problem environment presented by Horng, Lin, and Yang (2012).

Job id	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
1	3, 70, 140	2, 80, 160	1, 90, 180	6, 50, 100	4, 40, 80	8, 60, 120	5, 70, 140	7, 50, 100
2	1, 80, 160	2, 40, 80	3, 50, 100	5, 90, 180	4, 40, 80	7, 50, 100	6, 60, 120	8, 40, 80
3	1, 50, 100	2, 40, 80	3, 80, 160	5, 60, 120	4, 70, 140	6, 40, 80	8, 40, 80	7, 70, 140
4	2, 60, 120	1, 50, 100	3, 60, 120	4, 70, 140	7, 80, 160	5, 40, 80	6, 50, 100	8, 80, 160
5	4, 50, 100	3, 50, 100	2, 70, 140	1, 40, 80	7, 50, 100	5, 60, 120	6, 90, 180	8, 60, 120
6	2, 60, 120	3, 80, 160	1, 90, 180	5, 70, 140	6, 50, 100	4, 40, 80	8, 80, 160	7, 90, 180
7	1, 40, 80	3, 60, 120	4, 40, 80	2, 80, 160	5, 60, 120	7, 70, 140	8, 50, 100	6, 60, 120
8	2, 90, 180	1, 70, 140	3, 50, 100	4, 60, 120	5, 90, 180	7, 80, 160	6, 40, 80	8, 40, 80

Table 5.5: Job due dates in the test problem presented by Horng, Lin, and Yang (2012).

Job id	1	2	3	4	5	6	7	8
Due Date	490	510	540	500	540	470	530	560

i refers to the operation id, μ denotes the mean and σ^2 the variance of stochastic processing times. Table 5.5 shows the due date time for all jobs. Moreover, both tardiness and earliness costs are equal to 1. Three test problem sets were constructed from Table 5.4, using the truncated normal distribution (mean= μ and variance= σ^2), uniform distribution ($\mu - 3\sigma, \mu + 3\sigma$) and the exponential distribution with mean= μ . In ELBSO, similar settings to ESOO and ESOO-OCBA are used, where *popsiz*e, *npopsiz*e, *SL*, *iteration*, and *nSub* are equal to 1000, 2000, 10^5 , 100, and 6, respectively.

Using the trained GP in ELBSO, Table 5.6 compares the results of solving the presented experiment set by ELBSO, ESOO, and ESOO-OCBA. Moreover, each algorithm is replicated 20 times. Note that the results for ESOO and ESOO-OCBA in solving the mentioned SJSSP instances are reported by Horng, Lin, and Yang (2012) and Yang, Lv, et al. (2014), and here we compare ELBSO results with these reported results.

As it is shown in Table 5.6, ELBSO provides quite similar solutions to the reported results for ESOO and ESOO-OCBA in terms of quality.

It is worth mentioning that all experiments using ELBSO reported in this section were executed on a cloud computer provided by Amazon Web Services (AWS) with the following features: c4.xlarge instance, vCPU2, 7.5 Mem (GiB), 750 Dedicated

Table 5.6: Algorithms comparison.

Algorithm	ELBSO Result	ELBSO CPU	ESOO	ESOO-OCBA
Normal	(2269.56, 113.5)	(613.82, 59.14)	2280	2089
Uniform	(2518.7, 119.12)	(561.27, 46.11)	2778	2452
Exponential	(2550.71, 143.18)	(542.11, 23.16)	2683	2590

EBS Bandwidth (Mbps), and Intel Xeon E5-2666 v3 (Haswell) processor.

5.6.2 Comparing ELBSO with GA, PSO, and FA proposed by Shen and Zhu (2016)

Shen and Zhu (2016) presented a chance constraint approach for SJSSP by considering 10 jobs with 5 operations per job to be executed on 5 different machines. The processing times follow the linear distribution $L(j, i)$ (j and i refer to job and operation indexes, respectively), with the chance constrained model using a confidence level equal to 0.8. The processing time of each operation is calculated as follows:

$$P_{j,i} = (0.2 \times j) + (0.8 \times i) \quad (5.21)$$

Table 5.7 presents the SJSSP environment modeled by Shen and Zhu (2016). Each element of the table refers to the processing time of the five machines for the given job identifications. To illustrate, the second operation of job 1 is assigned to machine 4 (M_4), and its processing time equals to 1.8 (shown by: 2, 1.8 where $1.8 = (0.2 \times 1) + (0.8 \times 2)$). Moreover, Table 5.8 shows the job due dates, CE , and CT values considered by Shen and Zhu (2016).

As mentioned, using the chance constraint technique, Shen and Zhu (2016) converted the stochastic processing times in SJSSP to deterministic values using the 80 percent of confidence level to satisfy constraints. In this case, just one simulation replication is sufficient to calculate the objective value for each SJSSP solution exactly. Thus, the first phase of ELBSO combined with a simulation replication for each solution is considered for comparison with GA, PSO, and FA solutions proposed by Shen and Zhu (2016) in solving the SJSSP. To provide a fair comparison,

Table 5.7: Test problem environment presented by Shen and Zhu (2016).

Job id	M_1	M_2	M_3	M_4	M_5
1	5, 4.2	1, 1	3, 2.6	2, 1.8	4, 3.4
2	4, 3.6	3, 2.8	1, 1.4	5, 4.4	2, 2
3	2, 2.2	4, 3.8	5, 4.6	3, 3	1, 1.4
4	1, 1.6	5, 4.8	4, 4	2, 2.4	3, 3.2
5	2, 2.6	4, 4.2	5, 5	1, 1.8	3, 3.4
6	3, 3.6	1, 2.8	5, 5.2	4, 4.4	2, 2.8
7	1, 2.2	4, 4.6	2, 3	5, 5.6	3, 3.8
8	2, 3.2	5, 5.6	3, 4	1, 2.4	4, 4.8
9	5, 5.8	1, 2.6	3, 4.2	2, 3.4	4, 5
10	4, 5.2	5, 6	3, 4.4	2, 3.6	1, 2.8

Table 5.8: Job due dates, CE , and CT values in test problem presented by Shen and Zhu (2016).

Job id	1	2	3	4	5	6	7	8	9	10
Due Date	30	35	30	25	35	40	35	30	20	30
CE	20	15	10	15	25	20	10	15	20	10
CT	15	10	30	25	20	20	10	20	25	30

all ELBSO parameters are considered equal to the GA parameters used by Shen and Zhu (2016). Therefore, the solution population size ($popsiz$), offspring solution population size ($npopsiz$), and total number of iterations ($iteration$) are equal to 30, 30, and 10000, respectively.

Table 5.9 compares the results of solving the test problem using ELBSO and the other three algorithms, GA, PSO, and FA, as presented by Shen and Zhu (2016). As algorithm results vary for different replications, Shen and Zhu (2016) suggested 10 replications for each algorithm. Thus, we consider the same number of replications for ELBSO. In Table 5.9, the mean values and the best results are provided. In addition, the CPU computation times for ELBSO were provided while in Shen and Zhu (2016) no computational times were provided. As shown in Table 5.9, ELBSO outperforms all three other algorithms in solving the test problem and in a reasonable amount of time.

Table 5.9: Jobs due dates, CE , and CT values in test problem 1 presented by Shen and Zhu (2016)

ELBSO Mean	ELBSO Best	ELBSO CPU	GA Mean	GA Best	PSO Mean	PSO Best	FA Mean	FA Best
1103	904	106.54	2334	2030	1869	1761	1631	1547

5.6.3 Comparative Analysis of ELBSO with ESOO Solution Algorithms and Dispatching Rules

In this subsection, we present a comparative analysis between ELBSO and ESOO solution algorithms and well known dispatching rules. We develop and implement the ESOO algorithm based on step-by-step instructions provided in Horng, Lin, and Yang (2012), which was validated against the truncated normal example in Table 5.6. A value of 2398 for a time duration of 15,000 seconds was obtained, indicating the accuracy of our ESOO implementation.

Table 5.10 presents a vector (i, μ, σ^2) of test problems considered here, where i refers to the operation id, μ denotes the mean and σ^2 the variance of stochastic processing times, where processing times follow the truncated normal distribution with μ and σ . Table 5.11 shows the due date time for all jobs. In these test problems, we considered 8 different configurations created by combining different SJSSP problem dimensions defined by **number of jobs** \times **number of machines** \times **number of operations for each job** and CEs , and CTs (see Table 5.12 on page 145). To illustrate, in each experiment, firstly a problem is considered from Table 5.12, where based on the problem size, SJSSP parameters are obtained from Tables 5.10 and 5.11. For example, if Problem id 1 is selected from Table 5.12 then machines, jobs, and operations 1 to 10 are used.

Four dispatching rules are also included in this comparative analysis, SPT, LPT, CR and EDD. Note that as suggested by Horng, Lin, and Yang (2012) in each typical dispatching rule result a solution is created based on a sample of 10^5 SJSSP inputs. To illustrate, to generate a solution for SPT, 10^5 SJSSP processing times using the distributions defined in Table 5.10 are generated. Then, based on the mean of these operation processing times, SPT creates a solution. Finally, the objective value for this solution is calculated using 10^5 simulation replications.

In this research, to construct a fair comparison, we consider all ELBSO parameters similar to those proposed by Horng, Lin, and Yang (2012) for ESOO. Thus, for ELBSO and ESOO, the input parameters are the same, that is, the solution popu-

Table 5.10: Operations flow and their processing times in test problems set 2.

Job id	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
1	3, 70, 140	2, 80, 160	1, 90, 180	5, 50, 100	4, 40, 80	8, 60, 120	6, 70, 140	7, 50, 100
2	1, 80, 160	2, 40, 80	3, 50, 100	5, 90, 180	4, 40, 80	7, 50, 100	6, 60, 120	8, 40, 80
3	1, 50, 100	2, 40, 80	3, 80, 160	5, 60, 120	4, 70, 140	10, 40, 80	8, 40, 80	7, 70, 140
4	2, 60, 120	1, 50, 100	3, 60, 120	4, 70, 140	5, 80, 160	7, 40, 80	10, 50, 100	8, 80, 160
5	4, 50, 100	3, 50, 100	2, 70, 140	1, 40, 80	5, 50, 100	7, 60, 120	9, 90, 180	10, 60, 120
6	2, 60, 120	3, 80, 160	1, 90, 180	5, 70, 140	4, 50, 100	6, 40, 80	9, 80, 160	10, 90, 180
7	1, 40, 80	3, 60, 120	4, 40, 80	2, 80, 160	5, 60, 120	7, 70, 140	8, 50, 100	6, 60, 120
8	2, 90, 180	1, 70, 140	3, 50, 100	4, 60, 120	5, 90, 180	7, 80, 160	6, 40, 80	10, 40, 80
9	5, 80, 160	4, 60, 120	3, 50, 100	2, 60, 120	1, 60, 120	7, 70, 140	10, 40, 80	8, 40, 80
10	2, 80, 160	1, 70, 140	3, 50, 100	4, 70, 140	5, 90, 180	8, 70, 140	6, 50, 100	7, 40, 80
11	5, 60, 120	1, 80, 160	3, 50, 100	4, 60, 120	2, 80, 160	10, 50, 100	6, 40, 80	8, 50, 100
12	3, 60, 120	1, 60, 120	2, 50, 100	5, 90, 180	4, 70, 140	10, 70, 140	9, 40, 80	8, 40, 80
13	1, 90, 180	2, 60, 120	3, 70, 140	4, 90, 180	5, 90, 180	6, 60, 120	7, 40, 80	8, 40, 80
14	2, 80, 180	1, 70, 140	3, 60, 120	4, 70, 140	5, 90, 180	7, 70, 140	8, 50, 100	6, 50, 100
15	2, 90, 180	1, 70, 140	3, 50, 100	4, 60, 120	5, 90, 180	7, 80, 160	6, 40, 80	8, 40, 80
	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	
1	10, 70, 140	9, 80, 160	11, 90, 180	13, 50, 100	12, 40, 80	15, 60, 120	14, 70, 140	
2	10, 80, 160	9, 40, 80	15, 50, 100	12, 90, 180	11, 40, 80	13, 50, 100	14, 60, 120	
3	6, 50, 100	9, 40, 80	13, 80, 160	12, 60, 120	11, 70, 140	15, 40, 80	14, 40, 80	
4	9, 60, 120	6, 50, 100	15, 60, 120	11, 70, 140	13, 80, 160	12, 40, 80	14, 50, 100	
5	6, 50, 100	8, 50, 100	11, 70, 140	12, 40, 80	15, 50, 100	14, 60, 120	13, 90, 180	
6	7, 60, 120	8, 80, 160	15, 90, 180	14, 70, 140	13, 50, 100	12, 40, 80	11, 80, 160	
7	9, 40, 80	10, 60, 120	11, 40, 80	12, 80, 160	14, 60, 120	13, 70, 140	15, 50, 100	
8	8, 90, 180	9, 70, 140	12, 50, 100	13, 60, 120	15, 90, 180	11, 80, 160	14, 40, 80	
9	9, 60, 180	6, 60, 120	14, 50, 100	15, 70, 140	11, 60, 120	12, 80, 160	13, 40, 80	
10	9, 80, 160	10, 60, 120	11, 60, 120	12, 60, 120	13, 80, 160	14, 80, 160	15, 60, 120	
11	7, 60, 120	9, 60, 120	14, 60, 120	15, 60, 120	13, 60, 120	12, 80, 160	11, 60, 120	
12	7, 50, 100	6, 70, 140	13, 50, 100	14, 80, 160	15, 80, 160	11, 80, 160	12, 60, 120	
13	9, 60, 120	10, 70, 140	11, 50, 100	12, 50, 100	13, 80, 160	14, 70, 140	15, 50, 100	
14	10, 90, 180	9, 60, 120	15, 60, 120	14, 60, 120	13, 90, 180	11, 80, 160	12, 40, 80	
15	9, 90, 180	10, 70, 140	11, 50, 100	12, 60, 120	13, 90, 180	14, 80, 160	15, 40, 80	

Table 5.11: Jobs due dates in test problems set 2.

Job id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Due Date	290	310	440	300	440	470	530	560	640	610	550	700	760	740	820

Table 5.12: Test problems set 2 instances.

Problem id	1	2	3	4	5	6	7	8
Size	$10 \times 10 \times 10$	$10 \times 10 \times 10$	$10 \times 10 \times 10$	$10 \times 10 \times 10$	$15 \times 15 \times 15$	$15 \times 15 \times 15$	$15 \times 15 \times 15$	$15 \times 15 \times 15$
CT	1	1	5	5	1	1	5	5
CE	1	5	1	5	1	5	1	5

lation size ($popsize = 1000$), offspring solution population size ($npopsize = 2000$), total number of simulation replications to calculate each solution's objective value ($SL = 10^5$), total number of algorithm iterations ($iteration = 100$), and as both algorithms are OO-based evolutionary SO methods, in the second phase of all algorithms the total number of sub phases executed simulation experiments ($nSub = 6$). As each method does not provide similar solutions in any algorithm execution, 20 replications of each algorithm are used.

We consider 30 minutes to be the upper limit in computational time for a scheduling tool to be used within operations or short-term planning. Therefore, we considered 2000 seconds execution time as the termination criteria for all algorithms. The results for all experiments reported in this subsection were obtained using a cloud computer provided by Amazon Web Services (AWS) with the following features: c4.xlarge instance, vCPU2, 7.5 Mem (GiB), 750 Dedicated EBS Bandwidth (Mbps), and Intel Xeon E5-2666 v3 (Haswell) processor.

Table 5.13 provides a comparison of the best solutions obtained for the 2000 seconds execution time, showing the mean and standard deviation (STD) of the values obtained in 20 runs of each algorithm for each problem. Note that when executing the ESOO algorithm in 2000 seconds, it never enters the second phase.

Due to the stochastic behavior of the algorithms in this study, two separate runs of the same algorithm can come up with different results. Therefore, reporting results from one single run of each algorithm may mislead the drawn conclusions. Consequentially, a statistical analysis is needed to make solid conclusions and to be certain that the obtained results are relevant and that the relation between the compared algorithms is not reversed. In this chapter, a Statistical Hypothesis Testing (SHT) method is used to decide whether or not the difference between the indicated values (obtained results for 20 runs) for all algorithms is statistically significant. The null hypothesis for the proposed SHT is $\{H_0: \mu_1 - \mu_2 = 0\}$ and the alternative hypothesis is proposed as $\{H_1: \mu_1 - \mu_2 \neq 0\}$ where μ_1 and μ_2 are the mean values for the compared algorithms. To choose the most suitable SHT

Table 5.13: Test problem set 2 results

Problem id	1	2	3	4	5	6	7	8
ELBSO Mean	9106.61	9119.52	44258.7	45606.82	35761.3	35900.3	180486.46	181767.82
ELBSO STD	219.41	218.49	1396.38	1589.31	1380.49	1604.22	5604.32	7655.91
ESOO Mean	9443.05	9473.31	47005.7	47840.51	37858.27	37952.31	184344.22	185448.52
ESOO STD	221.65	163.78	1165.9	1226.3	1087.81	2284.29	3667.85	7248.7
SPT Mean	12752.2	13635.67	61993.68	64997.61	49178.65	50663.05	238771.1	233923.79
SPT STD	97.94	678.51	5003.26	6334.86	2916.52	1255.86	10262.32	10769.38
LPT Mean	14071.6	14582.48	78382.28	73101.54	52834.56	52118.53	252368.24	256213.69
LPT STD	898.18	818.62	8445.8	3860.63	2288.24	3580.9	10833.36	11176.73
CR Mean	13235.3	14127.6	65197.94	68264.86	50769.87	52047.22	251862.71	258698.9
CR STD	164.49	689.18	3341.23	6019.21	2370.54	1814.47	10833.36	8381.78
EDD Mean	13363.1	15230.78	65691.29	69391.36	51468.99	52455.76	260059.26	258631.1
EDD STD	242.9	778.15	4853.51	7381.27	2188.52	1956.77	9507.29	7563.11

method, firstly a normality test was performed.

As all algorithm result sets for each problem were proven to follow the Normal distribution, we chose to use the t -test comparing two samples with different variances as it is the best method to perform an SHT between normal samples Gentle, Härdle, and Mori 2012. The results of the two-sample t -test under confidence level of 95% for each pair of algorithms are shown in Table 5.14 using + and – symbols which represent rejecting the null hypothesis (significant difference between algorithm results) and failing to reject the null hypothesis (non-significant difference between algorithms results), respectively.

To illustrate, the results obtained from solving Problem 1 by all algorithms indicate the significance of the difference between the objective function values of ELBSO and all other methods (under the significance level of 95%), which shows that the performance of ELBSO in comparison to other algorithms in solving Problem 1 under a confidence level of 95%. Note that the algorithm provides better results in comparison with other methods when its results mean value is less and the differences between results are proven to be significant. Results indicate that ELBSO outperforms all other methods in solving the test problems in 2000 seconds. Moreover, among the typical dispatching rules evaluated here, SPT provides the best performance.

Table 5.14: *t*-test results

Problem id	Algorithm	Significance level= 95%				
		ESOO	CR	EDD	SPT	LPT
1	ELBSO	+	+	+	+	+
	ESOO		+	+	+	+
	CR			-	+	+
	EDD				+	+
	SPT					+
2	ELBSO	+	+	+	+	+
	ESOO		+	+	+	+
	CR			+	+	+
	EDD				+	+
	SPT					+
3	ELBSO	+	+	+	+	+
	ESOO		+	+	+	+
	CR			+	+	+
	EDD				-	+
	SPT					+
4	ELBSO	+	+	+	+	+
	ESOO		+	+	+	+
	CR			-	+	+
	EDD				+	+
	SPT					+
5	ELBSO	+	+	+	+	+
	ESOO		+	+	+	+
	CR			-	-	+
	EDD				-	-
	SPT					+
6	ELBSO	+	+	+	+	+
	ESOO		+	+	+	+
	CR			-	+	-
	EDD				+	-
	SPT					+
7	ELBSO	+	+	+	+	+
	ESOO		+	+	+	+
	CR			-	+	-
	EDD				+	-
	SPT					+
8	ELBSO	+	+	+	+	+
	ESOO		+	+	+	+
	CR			+	+	+
	EDD				+	-
	SPT					+

5.6.4 Runtime Comparison of ELBSO and ESSO

A question may be posed as to how each algorithm would perform if there is no execution time constraint as a termination criterion? To answer, firstly note that each typical dispatching rule provides the final result in less than 150 seconds, so the results shown are the best of these methods. To answer the question for both ELBSO and ESSO algorithms, the following Time Intensity (TI) factor is introduced. It is important to note that this compares the simulation time of ELBSO and ESSO. TI does not include other computational time for the algorithms in which these simulation models or metamodels are embedded within, i.e., selection, mutation, etc.

Consider the number of SL simulation replications to calculate objective value F for a SJSSP solution in a SO method. Considering CPT_s as the CPU execution time to run each simulation replication s , the total CPT to evaluate each solution can be defined as follows:

$$CPT = \sum_{s=1}^{SL} CPT_s \quad (5.22)$$

where CPT_s is the time to execute a single replication on the computer listed above. Furthermore, the CPT number for a population of solutions with the size of $popsize$ is equal to:

$$CPT = popsize \times \left(\sum_{s=1}^{SL} CPT_s \right) \quad (5.23)$$

The evolutionary algorithms considered here consist of both initial and neighbourhood solution populations. In other words, in each iteration of the algorithms, a set of neighbourhood solutions is created and added to the initial population. Therefore, the TI factor for a SO algorithm is defined as follows:

$$\begin{aligned}
TI = & \left(popsize \times \left(\sum_{s=1}^{SL} CPT_s \right) \right) + \left(iteration \times npopsize \times \left(\sum_{s=1}^{SL} CPT_s \right) \right) \\
& + (Rp \times CPT_s);
\end{aligned} \tag{5.24}$$

where *iteration* equals the total number of generations and *Rp* defines the total number of simulation replications in a SO method. To clarify, TI calculates the simulation replication CPU execution time for any evolutionary SO algorithms considered here. As discussed, simulation replications are highly time intensive, so TI can be used as a metric to analyse the time efficiency of different SO methods.

As previously mentioned, there are 6 different parameters (*popsize*, *npopsize*, *CPT_s*, *SL*, *iteration* and *Rp*) influencing the TI factor in the SO methods considered here. To construct a fair TI factor comparison, we consider the same parameter values for ELBSO and ESOO as detailed above, i.e., *SL*, *popsize*, *npopsize*, *iteration* and *Rp* for both algorithms are equal to 368, 1000, 2000, 100 and 10⁵, respectively (Horng, Lin, and Yang 2012). Note that after running 10 replications of the simulation model for Problem id 1 in Table 5.12 on the above mentioned cloud computer, a mean of 0.001029 for *CPT_s* was achieved (see Table 5.15 on page 151).

When compared with the ESOO SO solution method, ELBSO has an added computation phase which is the metamodeling phase using GP (GP preparation). Therefore, to calculate the TI factor for ELBSO, the GP Preparation time (*GPT*) is added to the TI formulation as shown below:

$$TI = GPT + (popsize \times CPT_m) + (iteration \times npopsize \times CPT_m) + (Rp \times CPT_s) \tag{5.25}$$

The metamodel created by GP in the first phase of ELBSO is used instead of simulation replications to calculate the solution objective values. Examining Table 5.15, the *SL* value in ELBSO is equal to 1, i.e., declaring the single usage of

Table 5.15: Algorithms TI analysis.

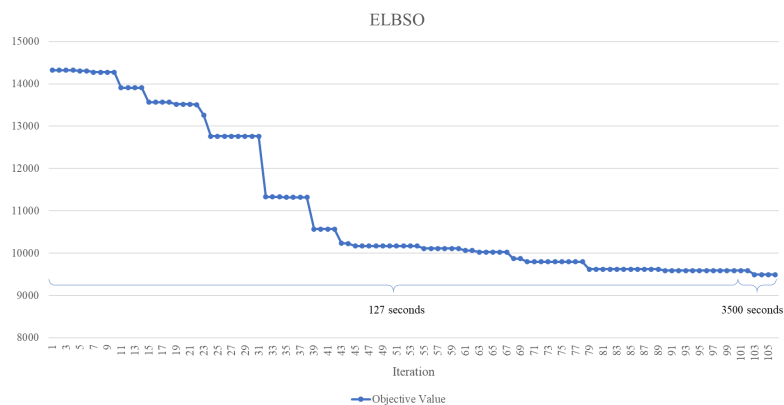
Algorithm	ELBSO	ESOO
<i>popsize</i>	1000	1000
<i>npopsize</i>	2000	2000
CPT_s	0.001029	0.001029
CPT_m	0.011347	–
GPT	270	–
SL	1	368
<i>iteration</i>	100	100
Rp	10^5	10^5
TI	2373	76215

the metamodel to evaluate the objective value for each solution in the first phase. ELBSO requires GPT amount of time for the GP preparation. After replicating the GP training phase for 10 different instances for Problem id 1 in Table 5.12, the mean GPT value obtained is 270 seconds. Moreover, the CPU execution time to calculate the objective value using the metamodel equals $CPT_m = 0.011347$ CPU seconds, again for Problem id 1, but this time will not vary due to the size of the problem solved.

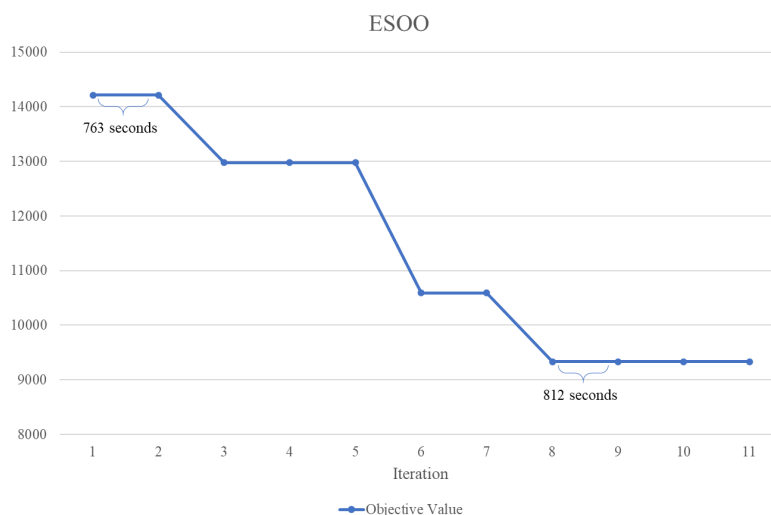
Examining Table 5.15 and using Equation 5.25, which is an extension of Equation 5.24 for the ELBSO solution method, the TI factor for ELBSO equals 2373.647 CPU seconds for solving Problem id 1 and the time to solve the same problem using ESoo gives $TI=7.6215 \times 10^4$ (See Equation 5.24). Regarding the TI factor, depending on the problem size, ELBSO could be executed by SJSSP to support operational or short-term planning. The TI factor difference between ELBSO and ESoo is dramatically high, which significantly proves the quality of ELBSO over ESoo, especially in terms of CPU solution time. In carrying out the above experiments, it was found that CE and CT had no effect on the time to solve SJSSP. However, the size of the problem in terms of jobs, operations, and machines has a major effect.

To analyse the performance of both ELBSO and ESoo for longer periods of time than 2000 seconds, Figure 5.10 traces one replication of both algorithms in solving Problem 1 using a 10000 seconds termination criteria. For ELBSO, each of the first 100 iterations takes about 1 second, and from iterations 101 to 106 it takes about

3500 seconds. For ESOO, the running time for each iteration is between 750 to 850 seconds. It is evident that after 8 iterations which takes about 6400 seconds (107 minutes) for ESSO, ESOO provides better solutions than ELBSO optimal solution.



(a) ELBSO results.



(b) ESOO results.

Figure 5.10: 10000 seconds termination criteria.

5.7 Discussion

In section 5.6 we provided a comparative analysis of the proposed algorithm against known published results for the SJSSP. In subsection 5.6.1 we compared the proposed algorithm with the results published by Horng, Lin, and Yang (2012) and Yang, Lv, et al. (2014). ELBSO provided similar results to these two algorithms. Furthermore, in subsection 5.6.2 we compared the ELBSO algorithm with the chance

constraint approach where optimization was carried out using GA, PSO and FA with ELBSO providing superior results to these solutions. Within subsection 5.6.3, to carry out a comparative analysis of ELBSO and ESOO, we re-implemented ESOO. Within this comparative analysis, we varied the number of machines, jobs, and operations solving problems with each input varying from 10 to 15. In addition, we also varied the values of CT and CE. The results show, using statistical analysis, that ELBSO provides superior results to ESOO and four other dispatching rules when we limit the duration of execution to 2000 seconds. Thus, we feel these tools could be used for short-term decision making within companies. Lastly, in subsection 5.6.4, we analysed the results of our implementation of ESOO and ELBSO when the termination was set to 10,000 seconds (166.667 hours). The results show, for all cases, that ESOO will only provide better results after a long execution period, which in the case presented executed for 6000-6800 seconds or 100-113.333 hours. This, we feel, is an infeasible amount of time to implement an optimization algorithm for short-term planning.

5.8 Conclusions

Simulation models have been used as one of the strongest decision making tools to analyze real industrial systems by considering stochastic parameters and/or variables of different systems. On the other hand, optimization techniques are key tools to improve decisions within almost all systems. Integrating simulation models with optimization methods could establish promising decision support tools benefiting from the advantages of both tools. That is the main idea to support proposing SOs for different complex and stochastic industrial problems (e.g., SJSSPs). However, simulation models of such problems are highly time-intensive, causing SO implementations to be infeasible for real and large-sized industrial problems. Thus, accurate and fast simulation metamodel implementations are necessary to be integrated within SOs.

In this chapter, we proposed a new SO method, denoted ELBSO, that consists of

a learning procedure using GP and an evolutionary optimization structure embedded in OO to solve the SJSSP. To solve the proposed problem, we firstly developed a SO mathematical definition for SJSSP. Next, a novel GP training strategy to learn from SJSSP is developed to metamodel the SJSSP DESM. Then, a two-phase ELBSO algorithm is presented which is an evolutionary SO based method with reduced computations due to the use of GP in estimating the objective values of solutions. Finally, using sets of standard SJSSP experiments, the developed ELBSO is compared with several published results in the literature. Our computational experiments have shown that ELBSO can achieve high quality solutions on the test problems evaluated with low computational cost.

Presenting ELBSO for SJSSP promises the possibility of introducing real-time methods to solve practical complex production system planning and scheduling problems. Furthermore, complex production systems such as semiconductor manufacturing, are suffering from the lack of existing real-time scheduling methods to address their SJSSPs (Ghasemi, Heavey, and Laipple 2018; Ghasemi, Azzouz, et al. 2020). Consequently, ELBSO can be the start of using embedding metamodeling concepts in SO methods to provide real-time decision support tools for such complex production systems by reducing DESM replications that are computationally time intensive.

Chapter 6

Conclusion

6.1 Introduction

This chapter reviews the results of the previous chapters and it also discusses the concluding remarks of the thesis. It highlights the major outcomes and contributions to the literature. In Section 6.2, the main conclusions which are drawn from the thesis are summarized and in Section 6.3, recommendations for further research are given.

6.2 Summary of Main Conclusions

Designing efficient simulation, optimization, and Machine Learning (ML) based Decision Support Tools (DSTs) for Production Planning and Scheduling (PP&S) is investigated in this thesis. Semiconductor manufacturing is selected as the case research in this thesis due to its obvious importance in the era of Industry 4.0. Within semiconductor manufacturing, photolithography is a well-known bottleneck process. Thus, for the case research, data sets were obtained from a Bosch photolithography toolset. Capacity Allocation Problem in a Photolithography Area (CAPPA) and Stochastic Job Shop Scheduling Problem (SJSSP) are two main PP&S problems within photolithography areas. Therefore, the designed DSTs are implemented to both problems.

To answer the first research question in this thesis (How to design an optimization-based framework to solve CAPPa accurately while considering all critical constraints?), a MILP model integrating all critical constraints was formulated, a CPLEX solver was used to solve small size CAPPa problems, and a new GA named IRGGA was developed (Chapter 3). The findings on solving CAPPa are:

- It is crucial to consider all CAPPa critical constraints together rather than considering them independently (which is the case for current research in the literature).
- IRGGA provides high-quality solutions in a reasonable time for CAPPa derived from Bosch photolithography data sets.

To answer the second research question in this thesis (What are the influencing factors in obtaining near optimal solutions in CAPPa problems?), a comprehensive sensitivity analysis is provided. The analysis showed that the sensitivity of the objective function to reducing the maximum reticles sharing was proved to be higher than its sensitivity to an increase in the machines flexibility. It is due to the fact that in modern photolithography fabs a sufficient number of reticles should be available and it is more critical than having a fully flexible toolset. Moreover, the analysis showed that although increasing the number of critical layers in CAPPa is a means of reaching the maximum loads, increasing the flexibility of machines can reduce the impact of critical layers. Stated differently, in a fab, increasing the eligibility of machines results in reducing the sensitivity of the photolithography to the changes in layers features.

To answer the third research question of this thesis (How to design and train a metamodel to replace a DESM in a SJSSP?), a GP-based DESM metamodel consisting of three new training vectors (MPBLV, QPBLV, and MPIBLV) was developed. Using several sets of data extracted from Bosch data sets, the GP-based metamodel and its training vectors were examined (Chapter 4). The main findings of this analysis are:

- The MPIBLV produces the most promising training results.
- Random quality data sets provide the best knowledge to train metamodels.
- The estimation accuracy of metamodels are independent from the number of simulation runs in the training data set.

To answer the last research question (How to design and implement evolutionary SO integrated with metamodels to solve SJSSPs?), in Chapter 5, a novel evolutionary SO method integrated with GP-based metamodels is proposed (ELBSO). ELBSO is then compared with several heuristics and SO methods from the literature in solving different SJSSPs. To the best of my knowledge, ELBSO is the first integration of GP-based metamodels with evolutionary SO techniques. When an optimization problem's space is complex, stochastic, and large (e.g., SJSSPs), it is not possible to calculate the objective values precisely in a reasonable time. This is due to the time-intensive nature of simulation models designed for these problems. Thus, most researchers who proposed SO methods for such problems have integrated optimization modules with a variety of techniques estimating stochastic objective values. In chapter 5, GP-based metamodels are used as an alternative for objective value estimation within SOs. As discussed, this implementation has strengthened ELBSO to solve SJSSPs accurately in a reasonable time. Such an implementation is highly valuable since industrial companies very often want improved feasible solutions to increase productivity with fast execution times, rather than optimal algorithms.

6.3 Recommendations for Further Research

There are several directions for future work. The IRGGA can be applied to a broad range of problems in the field of job scheduling by modifying the chromosome structures. The GP-based metamodeling procedure presented in this research can be integrated into various evolutionary optimization techniques to solve SJSSPs.

Moreover, this approach can be implemented in other complex stochastic optimization problems. In fact, metamodeling is one of the core innovations in this thesis, as it significantly reduces the computation time required to replicate simulation models. Therefore, we are interested in investigating the methodology used to train the GP metamodel to explore if it is the best approach used within an evolutionary algorithm. Moreover, we considered a simple evolutionary structure within ELBSO which can be changed with new efficient evolutionary algorithms (Ghasemi, Heavey, and Kabak 2018). There are several sources of uncertainty in practice, in, for example, semiconductor manufacturing such as reentries, mask setups, and precedence delays. However, in this research, we just considered stochastic processing times. Thus, embedding other sources of uncertainty in the proposed SO model can be another direction of future research. Finally, solving real case problems is always an interesting application of the presented ELBSO. We used the data set obtained from a Bosch photolithography area to design the GP-based metamodel and to solve the CAPP. However, the data set has not been used in ELBSO. Thus, in future, we would like to implement ELBSO on such real industrial data sets.

Bibliography

- Akçalı, E., K. Nemoto, and R. Uzsoy (Feb. 2001). “Cycle-time improvements for photolithography process in semiconductor manufacturing”. In: *IEEE Transactions on Semiconductor Manufacturing* 14.1, pp. 48–56. ISSN: 0894-6507. DOI: 10.1109/66.909654.
- Akçalı, E. and R. Uzsoy (2000). “A sequential solution methodology for capacity allocation and lot scheduling problems for photolithography”. In: *Twenty Sixth IEEE/CPMT International Electronics Manufacturing Technology Symposium (Cat. No.00CH37146)*, pp. 374–381. DOI: 10.1109/IEMT.2000.910749.
- Akker, Marjan van den, Kevin van Blokland, and Han Hoogeveen (2013). “Finding Robust Solutions for the Stochastic Job Shop Scheduling Problem by Including Simulation in Local Search”. en. In: *Experimental Algorithms*. Ed. by David Hutchison et al. Vol. 7933. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 402–413. ISBN: 978-3-642-38526-1 978-3-642-38527-8. DOI: 10.1007/978-3-642-38527-8_35. URL: http://link.springer.com/10.1007/978-3-642-38527-8_35 (visited on 01/17/2020).
- Akyol, Derya Eren and G. Mirac Bayhan (Aug. 2007). “A review on evolution of production scheduling with neural networks”. en. In: *Computers & Industrial Engineering* 53.1, pp. 95–122. ISSN: 03608352. DOI: 10.1016/j.cie.2007.04.006. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0360835207000666> (visited on 11/14/2020).
- Amir Haeri, Maryam, Mohammad Mehdi Ebadzadeh, and Gianluigi Folino (Nov. 2017). “Statistical genetic programming for symbolic regression”. en. In: *Applied*

- Soft Computing* 60, pp. 447–469. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2017.06.050. URL: <http://www.sciencedirect.com/science/article/pii/S1568494617303939> (visited on 04/11/2020).
- Azadeh, A., A. Negahban, and M. Moghaddam (Jan. 2012). “A hybrid computer simulation-artificial neural network algorithm for optimisation of dispatching rule selection in stochastic job shop scheduling problems”. In: *International Journal of Production Research* 50.2. Publisher: Taylor & Francis, pp. 551–566. ISSN: 0020-7543. DOI: 10.1080/00207543.2010.539281. URL: <https://doi.org/10.1080/00207543.2010.539281> (visited on 05/23/2020).
- Azadeh, Ali, Mohsen Moghaddam, et al. (Sept. 2010). “A flexible artificial neural network–fuzzy simulation algorithm for scheduling a flow shop with multiple processors”. en. In: *The International Journal of Advanced Manufacturing Technology* 50.5-8, pp. 699–715. ISSN: 0268-3768, 1433-3015. DOI: 10.1007/s00170-010-2533-6. URL: <http://link.springer.com/10.1007/s00170-010-2533-6> (visited on 11/14/2020).
- Barpalexis, P. et al. (May 2011). “Symbolic regression via genetic programming in the optimization of a controlled release pharmaceutical formulation”. en. In: *Chemometrics and Intelligent Laboratory Systems* 107.1, pp. 75–82. ISSN: 0169-7439. DOI: 10.1016/j.chemolab.2011.01.012. URL: <http://www.sciencedirect.com/science/article/pii/S0169743911000153> (visited on 04/20/2020).
- Barton, Russell R. and Martin Meckesheimer (Dec. 2006). “Chapter 18 Metamodel-Based Simulation Optimization”. English (US). In: *Handbooks in Operations Research and Management Science* 13.C. Publisher: North-Holland Publ Co, pp. 535–574. ISSN: 0927-0507. DOI: 10.1016/S0927-0507(06)13018-2. URL: <https://pennstate.pure.elsevier.com/en/publications/chapter-18-metamodel-based-simulation-optimization> (visited on 05/22/2020).
- Bean, James C. (May 1994). “Genetic Algorithms and Random Keys for Sequencing and Optimization”. In: *ORSA Journal on Computing* 6.2, pp. 154–160. ISSN:

- 0899-1499. DOI: 10.1287/ijoc.6.2.154. URL: <https://pubsonline.informs.org/doi/abs/10.1287/ijoc.6.2.154> (visited on 09/20/2018).
- Beheshtinia, Mohammad Ali, Amir Ghasemi, and Moein Farokhnia (Dec. 21, 2017). “Supply chain scheduling and routing in multi-site manufacturing system (case study: a drug manufacturing company)”. In: *Journal of Modelling in Management* 13.1, pp. 27–49. ISSN: 1746-5664. DOI: 10.1108/JM2-10-2016-0094. URL: <https://www.emeraldinsight.com/doi/abs/10.1108/JM2-10-2016-0094> (visited on 07/16/2018).
- Bettonvil, Bert, Enrique del Castillo, and Jack P. C. Kleijnen (Dec. 2009). “Statistical testing of optimality conditions in multiresponse simulation-based optimization”. en. In: *European Journal of Operational Research* 199.2, pp. 448–458. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2008.11.050. URL: <http://www.sciencedirect.com/science/article/pii/S0377221708010321> (visited on 06/22/2020).
- Beyer, Hans-Georg and Hans-Paul Schwefel (Mar. 1, 2002). “Evolution strategies – A comprehensive introduction”. In: *Natural Computing* 1.1, pp. 3–52. ISSN: 1572-9796. DOI: 10.1023/A:1015059928466. URL: <https://doi.org/10.1023/A:1015059928466> (visited on 01/13/2020).
- Boesel, Justin, Barry L. Nelson, and Seong-Hee Kim (Oct. 2003). “Using Ranking and Selection to “Clean Up” after Simulation Optimization”. en. In: *Operations Research* 51.5, pp. 814–825. ISSN: 0030-364X, 1526-5463. DOI: 10.1287/opre.51.5.814.16751. URL: <http://pubsonline.informs.org/doi/abs/10.1287/opre.51.5.814.16751> (visited on 03/28/2021).
- Box, G. E. P. and K. B. Wilson (1951). “On the Experimental Attainment of Optimum Conditions”. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 13.1. Publisher: [Royal Statistical Society, Wiley], pp. 1–45. ISSN: 0035-9246. URL: <https://www.jstor.org/stable/2983966> (visited on 06/22/2020).
- Can, Birkan and Cathal Heavey (Oct. 2011). “Comparison of experimental designs for simulation-based symbolic regression of manufacturing systems”. en.

- In: *Computers & Industrial Engineering* 61.3, pp. 447–462. ISSN: 0360-8352. DOI: 10.1016/j.cie.2011.03.012. URL: <http://www.sciencedirect.com/science/article/pii/S036083521100088X> (visited on 05/24/2020).
- Can, Birkan and Cathal Heavey (2012). “A comparison of genetic programming and artificial neural networks in metamodeling of discrete-event simulation models”. In: *Computers and Operations Research* 39.2, pp. 424–436. DOI: 10.1016/j.cor.2011.05.004. URL: <http://dx.doi.org/10.1016/j.cor.2011.05.004>.
- Çatay, Bülent, Ş. Selçuk Erengüç, and Asoo J. Vakharia (Aug. 2003). “Tool capacity planning in semiconductor manufacturing”. In: *Computers & Operations Research* 30.9, pp. 1349–1366. ISSN: 0305-0548. DOI: 10.1016/S0305-0548(02)00075-8. URL: <http://www.sciencedirect.com/science/article/pii/S0305054802000758> (visited on 01/08/2019).
- Chen, James C., Yin-Yann Chen, and Yu Liang (Oct. 2016). “Application of a genetic algorithm in solving the capacity allocation problem with machine dedication in the photolithography area”. en. In: *Journal of Manufacturing Systems* 41, pp. 165–177. ISSN: 02786125. DOI: 10.1016/j.jmsy.2016.08.010. URL: <http://linkinghub.elsevier.com/retrieve/pii/S0278612516300577> (visited on 05/10/2018).
- Chen, Xi and Qiang Zhou (Oct. 2017). “Sequential design strategies for mean response surface metamodeling via stochastic kriging with adaptive exploration and exploitation”. en. In: *European Journal of Operational Research* 262.2, pp. 575–585. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2017.03.042. URL: <http://www.sciencedirect.com/science/article/pii/S0377221717302643> (visited on 06/22/2020).
- Chong, C.S., Appa Iyer Sivakumar, and R. Gay (2003). “Simulation-based scheduling for dynamic discrete manufacturing”. en. In: *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*. New Orleans, LA, USA: IEEE, pp. 1465–1473. ISBN: 978-0-7803-8131-5. DOI:

- 10.1109/WSC.2003.1261590. URL: <http://ieeexplore.ieee.org/document/1261590/> (visited on 11/14/2020).
- Chung, S. H., C. Y. Huang, and A. H. I. Lee (Oct. 2006). “Capacity allocation model for photolithography workstation with the constraints of process window and machine dedication”. In: *Production Planning & Control* 17.7, pp. 678–688. ISSN: 0953-7287. DOI: 10.1080/09537280600901145. URL: <https://doi.org/10.1080/09537280600901145> (visited on 07/16/2018).
- Chung, Shu-Hsing, Chun-Ying Huang, and Amy H. I. Lee (June 2008). “Heuristic algorithms to solve the capacity allocation problem in photolithography area (CAPPA)”. en. In: *OR Spectrum* 30.3, pp. 431–452. ISSN: 0171-6468, 1436-6304. DOI: 10.1007/s00291-007-0093-4. URL: <https://link.springer.com/article/10.1007/s00291-007-0093-4> (visited on 07/16/2018).
- Corne, Dave and Peter Ross (1995). “Some combinatorial landscapes on which a Genetic Algorithm outperforms other Stochastic iterative methods”. en. In: *Evolutionary Computing*. Ed. by Terence C. Fogarty. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 1–13. ISBN: 978-3-540-47515-6.
- Coughlan, Paul and David Coughlan (Jan. 2002). “Action research for operations management”. In: *International Journal of Operations & Production Management* 22.2. Publisher: MCB UP Ltd, pp. 220–240. ISSN: 0144-3577. DOI: 10.1108/01443570210417515. URL: <https://doi.org/10.1108/01443570210417515> (visited on 01/19/2021).
- Dolgui, Alexandre et al. (Jan. 2019). “Scheduling in production, supply chain and Industry 4.0 systems by optimal control: fundamentals, state-of-the-art and applications”. In: *International Journal of Production Research* 57.2. Publisher: Taylor & Francis _eprint: <https://doi.org/10.1080/00207543.2018.1442948>, pp. 411–432. ISSN: 0020-7543. DOI: 10.1080/00207543.2018.1442948. URL: <https://doi.org/10.1080/00207543.2018.1442948> (visited on 01/03/2021).
- Dunke, Fabian and Stefan Nickel (Feb. 2020). “Neural networks for the metamodeling of simulation models with online decision making”. en. In: *Simulation Mod-*

- elling Practice and Theory* 99, p. 102016. ISSN: 1569-190X. DOI: 10.1016/j.simpat.2019.102016. URL: <http://www.sciencedirect.com/science/article/pii/S1569190X19301479> (visited on 05/22/2020).
- Dutton, David M. and Gerard V. Conroy (Dec. 1997). “A review of machine learning”. en. In: *The Knowledge Engineering Review* 12.4, pp. 341–367. ISSN: 0269-8889, 1469-8005. DOI: 10.1017/S026988899700101X. URL: https://www.cambridge.org/core/product/identifier/S026988899700101X/type/journal_article (visited on 01/05/2021).
- ELINFOR (2019). *Electronic & lightning infor storage information website*. URL: <https://www.elinfor.com/> (visited on 09/27/2019).
- Fathi, Masood et al. (Feb. 2016). “A modified particle swarm optimisation algorithm to solve the part feeding problem at assembly lines”. en. In: *International Journal of Production Research* 54.3, pp. 878–893. ISSN: 0020-7543, 1366-588X. DOI: 10.1080/00207543.2015.1090032. URL: <http://www.tandfonline.com/doi/full/10.1080/00207543.2015.1090032> (visited on 09/27/2019).
- Fazel Zarandi, Mohammad Hossein et al. (Jan. 2020). “A state of the art review of intelligent scheduling”. en. In: *Artificial Intelligence Review* 53.1, pp. 501–593. ISSN: 0269-2821, 1573-7462. DOI: 10.1007/s10462-018-9667-6. URL: <http://link.springer.com/10.1007/s10462-018-9667-6> (visited on 11/15/2020).
- Ferreira, Cristiane, Gonalo Figueira, and Pedro Amorim (2020). “Optimizing Dispatching Rules for Stochastic Job Shop Scheduling”. In: *Hybrid Intelligent Systems*. Ed. by Ana Maria Madureira et al. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, pp. 321–330. ISBN: 978-3-030-14347-3. DOI: 10.1007/978-3-030-14347-3_31.
- Figueira, Gonalo and Bernardo Almada-Lobo (Aug. 2014). “Hybrid simulation-optimization methods: A taxonomy and discussion”. In: *Simulation Modelling Practice and Theory*. Simulation-Optimization of Complex Systems: Methods and Applications 46, pp. 118–134. ISSN: 1569-190X. DOI: 10.1016/j.simpat.

- 2014.03.007. URL: <http://www.sciencedirect.com/science/article/pii/S1569190X14000458> (visited on 05/04/2018).
- Fisher, Henry (1963). “Probabilistic learning combinations of local job-shop scheduling rules”. In: *Industrial scheduling*, pp. 225–251.
- Forza, Cipriano (Jan. 2002). “Survey research in operations management: a process-based perspective”. In: *International Journal of Operations & Production Management* 22.2. Publisher: MCB UP Ltd, pp. 152–194. ISSN: 0144-3577. DOI: 10.1108/01443570210414310. URL: <https://doi.org/10.1108/01443570210414310> (visited on 01/19/2021).
- Gao, Kai Zhou et al. (Oct. 1, 2016). “Artificial bee colony algorithm for scheduling and rescheduling fuzzy flexible job shop problem with new job insertion”. In: *Knowledge-Based Systems* 109, pp. 1–16. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2016.06.014. URL: <http://www.sciencedirect.com/science/article/pii/S0950705116301794> (visited on 01/27/2020).
- Garey, Michael R. and David S. Johnson (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co. ISBN: 978-0-7167-1045-5.
- Gen, Mitsuo, Xinchang Hao, and Wenqiang Zhang (2017). “Advances in Hybrid Metaheuristics for Stochastic Manufacturing Scheduling: Part I Models and Methods”. en. In: *Proceedings of the Tenth International Conference on Management Science and Engineering Management*. Ed. by Jiuping Xu et al. Vol. 502. Singapore: Springer Singapore, pp. 1063–1077. ISBN: 978-981-10-1836-7 978-981-10-1837-4. DOI: 10.1007/978-981-10-1837-4_88. URL: http://link.springer.com/10.1007/978-981-10-1837-4_88 (visited on 01/17/2020).
- Gentle, James E., Wolfgang Karl Härdle, and Yuichi Mori (July 2012). *Handbook of Computational Statistics: Concepts and Methods*. en. Google-Books-ID: aSv09LwmuRYC. Springer Science & Business Media. ISBN: 978-3-642-21551-3.
- Ghasemi, Amir, Radhia Azzouz, et al. (Jan. 2020). “Optimizing capacity allocation in semiconductor manufacturing photolithography area – Case study: Robert

- Bosch”. en. In: *Journal of Manufacturing Systems* 54, pp. 123–137. ISSN: 0278-6125. DOI: 10.1016/j.jmsy.2019.11.012. URL: <http://www.sciencedirect.com/science/article/pii/S0278612519301153> (visited on 12/20/2019).
- Ghasemi, Amir, Cathal Heavey, and Kamil Erkan Kabak (2018). “Implementing a New Genetic Algorithm to Solve the Capacity Allocation Problem in the Photolithography Area”. In: *Proceedings of the 2018 Winter Simulation Conference. WSC '18*. event-place: Gothenburg, Sweden. Piscataway, NJ, USA: IEEE Press, pp. 3696–3707. URL: <http://dl.acm.org/citation.cfm?id=3320516.3320956> (visited on 12/12/2019).
- Ghasemi, Amir, Cathal Heavey, and Georg Laipple (2018). “A Review of Simulation-optimization Methods with Applications to Semiconductor Operational Problems”. In: *Proceedings of the 2018 Winter Simulation Conference. WSC '18*. event-place: Gothenburg, Sweden. Piscataway, NJ, USA: IEEE Press, pp. 3672–3683. URL: <http://dl.acm.org/citation.cfm?id=3320516.3320954> (visited on 12/04/2019).
- Golenko-Ginzburg, Dimitri, Shmuel Kesler, and Zinoviy Landsman (Aug. 1995). “Industrial job-shop scheduling with random operations and different priorities”. en. In: *International Journal of Production Economics* 40.2, pp. 185–195. ISSN: 0925-5273. DOI: 10.1016/0925-5273(95)00078-8. URL: <http://www.sciencedirect.com/science/article/pii/0925527395000788> (visited on 08/25/2020).
- Graham, R. L. et al. (Jan. 1979). “Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey”. en. In: *Annals of Discrete Mathematics*. Ed. by P. L. Hammer, E. L. Johnson, and B. H. Korte. Vol. 5. Discrete Optimization II. Elsevier, pp. 287–326. DOI: 10.1016/S0167-5060(08)70356-X. URL: <https://www.sciencedirect.com/science/article/pii/S016750600870356X> (visited on 04/06/2021).
- Gu, Jinwei et al. (May 2010). “A novel competitive co-evolutionary quantum genetic algorithm for stochastic job shop scheduling problem”. en. In: *Computers*

- Operations Research* 37.5, pp. 927–937. ISSN: 03050548. DOI: 10.1016/j.cor.2009.07.002. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0305054809001841> (visited on 01/17/2020).
- Hao, Xinchang, Mitsuo Gen, et al. (Mar. 2017). “Effective multiobjective EDA for bi-criteria stochastic job-shop scheduling problem”. en. In: *Journal of Intelligent Manufacturing* 28.3, pp. 833–845. ISSN: 0956-5515, 1572-8145. DOI: 10.1007/s10845-014-1026-0. URL: <http://link.springer.com/10.1007/s10845-014-1026-0> (visited on 01/24/2020).
- Hao, Xinchang, Lin Lin, et al. (Jan. 2013). “Effective Estimation of Distribution Algorithm for Stochastic Job Shop Scheduling Problem”. en. In: *Procedia Computer Science*. Complex Adaptive Systems 20, pp. 102–107. ISSN: 1877-0509. DOI: 10.1016/j.procs.2013.09.246. URL: <http://www.sciencedirect.com/science/article/pii/S1877050913010466> (visited on 06/11/2020).
- Hasan, S.M. Kamrul, Ruhul Sarker, and Daryl Essam (Aug. 2011). “Genetic algorithm for job-shop scheduling with machine unavailability and breakdowns”. en. In: *International Journal of Production Research* 49.16, pp. 4999–5015. ISSN: 0020-7543, 1366-588X. DOI: 10.1080/00207543.2010.495088. URL: <http://www.tandfonline.com/doi/abs/10.1080/00207543.2010.495088> (visited on 01/24/2020).
- HeuristicLab (2020). *HeuristicLab*. URL: <https://dev.heuristiclab.com/trac.fcgi/> (visited on 01/21/2020).
- Ho, Yu-Chi (Feb. 1999). “An explanation of ordinal optimization: Soft computing for hard problems”. In: *Information Sciences* 113.3, pp. 169–192. ISSN: 00200255. DOI: 10.1016/S0020-0255(98)10056-7. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0020025598100567> (visited on 10/07/2019).
- Ho, Yu-Chi, Qian-Chuan Zhao, and Qing-Shan Jia (Jan. 23, 2008). *Ordinal Optimization: Soft Optimization for Hard Problems*. Google-Books-ID: CzDUiD-EDmHAC. Springer Science & Business Media. 325 pp. ISBN: 978-0-387-68692-9.

- Holland, J. (1973). “Genetic Algorithms and the Optimal Allocation of Trials”. In: *SIAM Journal on Computing* 2.2, pp. 88–105. DOI: 10.1137/0202009. eprint: <https://doi.org/10.1137/0202009>. URL: <https://doi.org/10.1137/0202009>.
- Horn, Shih-Cheng and Shieh-Shing Lin (Feb. 2015). “Integrating Ant Colony System and Ordinal Optimization for Solving Stochastic Job Shop Scheduling Problem”. en. In: *2015 6th International Conference on Intelligent Systems, Modelling and Simulation*. Kuala Lumpur, Malaysia: IEEE, pp. 70–75. ISBN: 978-1-4799-8258-5. DOI: 10.1109/ISMS.2015.9. URL: <http://ieeexplore.ieee.org/document/7311212/> (visited on 01/20/2020).
- Horn, Shih-Cheng, Shieh-Shing Lin, and Feng-Yi Yang (Feb. 2012). “Evolutionary algorithm for stochastic job shop scheduling with random processing time”. In: *Expert Systems with Applications* 39.3, pp. 3603–3610. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2011.09.050. URL: <http://www.sciencedirect.com/science/article/pii/S0957417411013595> (visited on 10/07/2019).
- Horn, Shih-Cheng, Feng-Yi Yang, and Shieh-Shing Lin (Aug. 2012). “Embedding evolutionary strategy in ordinal optimization for hard optimization problems”. en. In: *Applied Mathematical Modelling* 36.8, pp. 3753–3763. ISSN: 0307904X. DOI: 10.1016/j.apm.2011.11.013. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0307904X11007013> (visited on 01/17/2020).
- Hung, Yi-Feng and Gia-Jin Cheng (Feb. 2002). “Hybrid capacity modeling for alternative machine types in linear programming production planning”. en. In: *IIE Transactions* 34.2, pp. 157–165. ISSN: 0740-817X, 1573-9724. DOI: 10.1023/A:1011943930465. URL: <https://link.springer.com/article/10.1023/A:1011943930465> (visited on 07/16/2018).
- Hussain, Mohammed F., Russel R. Barton, and Sanjay B. Joshi (Apr. 2002). “Meta-modeling: Radial basis functions, versus polynomials”. en. In: *European Journal of Operational Research* 138.1, pp. 142–154. ISSN: 0377-2217. DOI: 10.1016/

- S0377-2217(01)00076-5. URL: <http://www.sciencedirect.com/science/article/pii/S0377221701000765> (visited on 04/20/2020).
- Jamili, A. (Feb. 2019). “Job shop scheduling with consideration of floating breaking times under uncertainty”. en. In: *Engineering Applications of Artificial Intelligence* 78, pp. 28–36. ISSN: 09521976. DOI: 10.1016/j.engappai.2018.10.007. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0952197618302197> (visited on 01/17/2020).
- Jin, S., W. Chen, and J. Han (Nov. 2017). “Graph-based machine learning algorithm with application in data mining”. In: *2017 Third International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, pp. 269–272. DOI: 10.1109/ICRCICN.2017.8234519.
- Kabak, K. E. et al. (Feb. 2013). “Impact of Recipe Restrictions on Photolithography Toolsets in an ASIC Fabrication Environment”. In: *IEEE Transactions on Semiconductor Manufacturing* 26.1, pp. 53–68. ISSN: 0894-6507. DOI: 10.1109/TSM.2012.2220572.
- Kelleher, Clayton T. et al. (Jan. 2018). “Using dynamic Bayesian networks as simulation metamodels based on bootstrapping”. en. In: *Computers & Industrial Engineering* 115, pp. 595–602. ISSN: 0360-8352. DOI: 10.1016/j.cie.2017.11.033. URL: <http://www.sciencedirect.com/science/article/pii/S0360835217305661> (visited on 12/28/2020).
- Kemmoe-Tchomte, Sylverin, Damien Lamy, and Nikolay Tchernev (Oct. 2015). “A metaheuristic based on simulation for stochastic Job-shop optimization”. en. In: *2015 International Conference on Industrial Engineering and Systems Management (IESM)*. Seville, Spain: IEEE, pp. 108–116. ISBN: 978-2-9600532-6-5. DOI: 10.1109/IESM.2015.7380144. URL: <http://ieeexplore.ieee.org/document/7380144/> (visited on 01/20/2020).
- Kilmer, R. A., A. E. Smith, and L. J. Shuman (1997). “An emergency department simulation and a neural network metamodel”. In: *Journal of the Society for Health Systems* 5.3, pp. 63–79. ISSN: 1043-1721.

- Kim, S. C. and P. M. Bobrowski (Aug. 1997). “Scheduling jobs with uncertain setup times and sequence dependency”. en. In: *Omega* 25.4, pp. 437–447. ISSN: 0305-0483. DOI: 10.1016/S0305-0483(97)00013-3. URL: <http://www.sciencedirect.com/science/article/pii/S0305048397000133> (visited on 04/06/2020).
- Kleijnen, Jack P. C. (Jan. 2008). “Response surface methodology for constrained simulation optimization: An overview”. en. In: *Simulation Modelling Practice and Theory* 16.1, pp. 50–64. ISSN: 1569-190X. DOI: 10.1016/j.simpat.2007.10.001. URL: <http://www.sciencedirect.com/science/article/pii/S1569190X07001256> (visited on 06/21/2020).
- (Feb. 2009). “Kriging metamodeling in simulation: A review”. en. In: *European Journal of Operational Research* 192.3, pp. 707–716. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2007.10.013. URL: <http://www.sciencedirect.com/science/article/pii/S0377221707010090> (visited on 04/20/2020).
- (Jan. 2017). “Regression and Kriging metamodels with their experimental designs in simulation: A review”. en. In: *European Journal of Operational Research* 256.1, pp. 1–16. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2016.06.041. URL: <http://www.sciencedirect.com/science/article/pii/S0377221716304623> (visited on 06/21/2020).
- Koza, John R. (June 1994). “Genetic programming as a means for programming computers by natural selection”. en. In: *Statistics and Computing* 4.2, pp. 87–112. ISSN: 1573-1375. DOI: 10.1007/BF00175355. URL: <https://doi.org/10.1007/BF00175355> (visited on 12/18/2019).
- Koza, John R. and John R. Koza (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. en. Google-Books-ID: Bhtxo60BV0EC. MIT Press. ISBN: 978-0-262-11170-6.
- Leachman, Robert C. and Talif F. Carmon (Sept. 1992). “On Capacity Modeling for Production Planning with Alternative Machine Types”. In: *IIE Transactions*

- 24.4, pp. 62–72. ISSN: 0740-817X. DOI: 10.1080/07408179208964234. URL: <https://doi.org/10.1080/07408179208964234>.
- Lee, Young Hoon and Byungjin Lee (Dec. 2003). “Push-pull production planning of the re-entrant process”. en. In: *Int J Adv Manuf Technol* 22.11, pp. 922–931. ISSN: 1433-3015. DOI: 10.1007/s00170-003-1653-7. URL: <https://doi.org/10.1007/s00170-003-1653-7> (visited on 03/22/2019).
- Lei, Deming (Apr. 2008). “Pareto archive particle swarm optimization for multi-objective fuzzy job shop scheduling problems”. en. In: *The International Journal of Advanced Manufacturing Technology* 37.1-2, pp. 157–165. ISSN: 0268-3768, 1433-3015. DOI: 10.1007/s00170-007-0945-8. URL: <http://link.springer.com/10.1007/s00170-007-0945-8> (visited on 01/24/2020).
- (Dec. 2011). “Simplified multi-objective genetic algorithms for stochastic job shop scheduling”. en. In: *Applied Soft Computing* 11.8, pp. 4991–4996. ISSN: 15684946. DOI: 10.1016/j.asoc.2011.06.001. URL: <https://linkinghub.elsevier.com/retrieve/pii/S1568494611002195> (visited on 01/17/2020).
- Lenstra, J. K., A. H. G. Rinnooy Kan, and P. Brucker (Jan. 1977). “Complexity of Machine Scheduling Problems”. In: *Annals of Discrete Mathematics*. Ed. by P. L. Hammer et al. Vol. 1. Studies in Integer Programming. Elsevier, pp. 343–362. DOI: 10.1016/S0167-5060(08)70743-X. URL: <http://www.sciencedirect.com/science/article/pii/S016750600870743X> (visited on 08/20/2018).
- Leonard-Barton, Dorothy (Aug. 1990). “A Dual Methodology for Case Studies: Synergistic Use of a Longitudinal Single Site with Replicated Multiple Sites”. In: *Organization Science* 1.3. Publisher: INFORMS, pp. 248–266. ISSN: 1047-7039. DOI: 10.1287/orsc.1.3.248. URL: <https://pubsonline.informs.org/doi/abs/10.1287/orsc.1.3.248> (visited on 01/10/2021).
- Levinson, Harry J. (2010). *Principles of lithography*. en. 3rd ed. Press monograph 198. Bellingham, Wash: SPIE Press. ISBN: 978-0-8194-8324-9.
- Li, Minqi et al. (Jan. 2016). “A metamodel-based Monte Carlo simulation approach for responsive production planning of manufacturing systems”. en. In: *Journal of*

- Manufacturing Systems* 38, pp. 114–133. ISSN: 0278-6125. DOI: 10.1016/j.jmsy.2015.11.004. URL: <http://www.sciencedirect.com/science/article/pii/S0278612515001168> (visited on 12/12/2019).
- Liao, Lu-Wen and Gwo-Ji Sheen (Jan. 2008). “Parallel machine scheduling with machine availability and eligibility constraints”. en. In: *European Journal of Operational Research* 184.2, pp. 458–467. ISSN: 03772217. DOI: 10.1016/j.ejor.2006.11.027. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0377221706011623> (visited on 09/26/2019).
- Lin, Yi-Kuei et al. (July 2019). “Bi-objective optimization for a multistate job-shop production network using NSGA-II and TOPSIS”. en. In: *Journal of Manufacturing Systems* 52, pp. 43–54. ISSN: 02786125. DOI: 10.1016/j.jmsy.2019.05.004. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0278612519300238> (visited on 01/19/2020).
- Liu, J., Y. Chi, et al. (2019). “Ensemble multi-objective evolutionary algorithm for gene regulatory network reconstruction based on fuzzy cognitive maps”. In: *CAAI Transactions on Intelligence Technology* 4.1. Conference Name: CAAI Transactions on Intelligence Technology, pp. 24–36. ISSN: 2468-2322. DOI: 10.1049/trit.2018.1059.
- Liu, Ran, Xiaolei Xie, et al. (Apr. 2018). “A survey on simulation optimization for the manufacturing system operation”. en. In: *International Journal of Modelling and Simulation* 38.2, pp. 116–127. ISSN: 0228-6203, 1925-7082. DOI: 10.1080/02286203.2017.1401418. URL: <https://www.tandfonline.com/doi/full/10.1080/02286203.2017.1401418> (visited on 10/05/2019).
- Liu, Yongkui, Lin Zhang, et al. (May 2017). “Resource service sharing in cloud manufacturing based on the Gale–Shapley algorithm: advantages and challenge”. In: *International Journal of Computer Integrated Manufacturing* 30.4-5, pp. 420–432. ISSN: 0951-192X. DOI: 10.1080/0951192X.2015.1067916. URL: <https://doi.org/10.1080/0951192X.2015.1067916> (visited on 08/02/2019).

- Low, Chor Ping and Can Fang (2005). “On the Load-Balanced Demand Points Assignment Problem in Large-Scale Wireless LANs”. en. In: *Information Networking. Convergence in Broadband and Mobile Networking*. Ed. by Cheeha Kim. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 21–30. ISBN: 978-3-540-30582-8.
- Ma, Haiping et al. (Feb. 2019). “Multi-population techniques in nature inspired optimization algorithms: A comprehensive survey”. en. In: *Swarm and Evolutionary Computation* 44, pp. 365–387. ISSN: 2210-6502. DOI: 10.1016/j.swevo.2018.04.011. URL: <http://www.sciencedirect.com/science/article/pii/S2210650217306363> (visited on 11/18/2020).
- Martínez-Costa, Carme et al. (July 2014). “A review of mathematical programming models for strategic capacity planning in manufacturing”. en. In: *International Journal of Production Economics* 153, pp. 66–85. ISSN: 09255273. DOI: 10.1016/j.ijpe.2014.03.011. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0925527314000917> (visited on 03/20/2019).
- Mathlab Fitting App (2020). en-US. URL: <https://uk.mathworks.com/help/stats/fit-a-distribution-using-the-distribution-fitting-app.html> (visited on 06/24/2020).
- Mehdad, Ehsan and Jack P. C. Kleijnen (2018). “Stochastic intrinsic Kriging for simulation metamodeling”. en. In: *Applied Stochastic Models in Business and Industry* 34.3. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/asmb.2300>, pp. 322–337. ISSN: 1526-4025. DOI: 10.1002/asmb.2300. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/asmb.2300> (visited on 06/22/2020).
- Merton, Robert K. (1957). “The Role-Set: Problems in Sociological Theory”. In: *The British Journal of Sociology* 8.2, pp. 106–120. ISSN: 0007-1315. DOI: 10.2307/587363. URL: <http://www.jstor.org/stable/587363> (visited on 07/16/2018).
- Missbauer, Hubert and Reha Uzsoy (2011). “Optimization Models of Production Planning Problems”. en. In: *Planning Production and Inventories in the Extended Enterprise: A State of the Art Handbook, Volume 1*. Ed. by Karl G. Kempf,

- Pinar Keskinocak, and Reha Uzsoy. International Series in Operations Research & Management Science. New York, NY: Springer US, pp. 437–507. ISBN: 978-1-4419-6485-4. DOI: 10.1007/978-1-4419-6485-4_16. URL: https://doi.org/10.1007/978-1-4419-6485-4_16 (visited on 01/05/2021).
- Mohan, Jatoth, Krishnanand Lanka, and A. Neelakanteswara Rao (2019). “A Review of Dynamic Job Shop Scheduling Techniques”. en. In: *Procedia Manufacturing* 30, pp. 34–39. ISSN: 23519789. DOI: 10.1016/j.promfg.2019.02.006. URL: <https://linkinghub.elsevier.com/retrieve/pii/S2351978919300368> (visited on 01/17/2020).
- Mokhtari, Hadi and Mehrdad Dadgar (Sept. 2015). “Scheduling optimization of a stochastic flexible job-shop system with time-varying machine failure rate”. en. In: *Computers & Operations Research* 61, pp. 31–45. ISSN: 0305-0548. DOI: 10.1016/j.cor.2015.02.014. URL: <http://www.sciencedirect.com/science/article/pii/S0305054815000453> (visited on 12/31/2020).
- Mönch, Lars, J. W. Fowler, and Scott J. Mason (2013). *Production planning and control for semiconductor wafer fabrication facilities: modeling, analysis, and systems*. en. Operations research/computer science interfaces series vol. 52. New York: Springer. ISBN: 978-1-4614-4471-8 978-1-4614-4472-5.
- Mönch, Lars, Reha Uzsoy, and John W. Fowler (July 2018). “A survey of semiconductor supply chain models part I: semiconductor supply chains, strategic network design, and supply chain simulation”. en. In: *International Journal of Production Research* 56.13, pp. 4524–4545. ISSN: 0020-7543, 1366-588X. DOI: 10.1080/00207543.2017.1401233. URL: <https://www.tandfonline.com/doi/full/10.1080/00207543.2017.1401233> (visited on 12/08/2019).
- Moore, G.E. (Jan. 1998). “Cramming More Components Onto Integrated Circuits”. en. In: *Proceedings of the IEEE* 86.1, pp. 82–85. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.1998.658762. URL: <http://ieeexplore.ieee.org/document/658762/> (visited on 01/05/2021).

- Mouelhi-Chibani, Wiem and Henri Pierreval (Mar. 2010). “Training a neural network to select dispatching rules in real time”. en. In: *Computers & Industrial Engineering* 58.2, pp. 249–256. ISSN: 03608352. DOI: 10.1016/j.cie.2009.03.008. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0360835209000953> (visited on 11/15/2020).
- Murphy, Gearoid and Conor Ryan (2008). “A Simple Powerful Constraint for Genetic Programming”. en. In: *Genetic Programming*. Ed. by Michael O’Neill et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 146–157. ISBN: 978-3-540-78671-9. DOI: 10.1007/978-3-540-78671-9_13.
- Nguyen, Su, Yi Mei, and Mengjie Zhang (Mar. 2017). “Genetic programming for production scheduling: a survey with a unified framework”. en. In: *Complex & Intelligent Systems* 3.1, pp. 41–66. ISSN: 2199-4536, 2198-6053. DOI: 10.1007/s40747-017-0036-x. URL: <http://link.springer.com/10.1007/s40747-017-0036-x> (visited on 11/09/2019).
- Park, Heungsun and L. A. Stefanski (Oct. 15, 1998). “Relative-error prediction”. In: *Statistics & Probability Letters* 40.3, pp. 227–236. ISSN: 0167-7152. DOI: 10.1016/S0167-7152(98)00088-1. URL: <http://www.sciencedirect.com/science/article/pii/S0167715298000881> (visited on 03/13/2020).
- Petrovic, Sanja et al. (Mar. 2008). “Fuzzy job shop scheduling with lot-sizing”. en. In: *Annals of Operations Research* 159.1, pp. 275–292. ISSN: 0254-5330, 1572-9338. DOI: 10.1007/s10479-007-0287-9. URL: <http://link.springer.com/10.1007/s10479-007-0287-9> (visited on 01/24/2020).
- Pham, H. N. A. et al. (Nov. 2008). “Scheduling for Dedicated Machine Constraint Using Integer Programming”. In: *2008 20th IEEE International Conference on Tools with Artificial Intelligence*. Vol. 1, pp. 499–506. DOI: 10.1109/ICTAI.2008.85.
- Piersma, N. and W. van Dijk (Nov. 1996). “A local search heuristic for unrelated parallel machine scheduling with efficient neighborhood search”. en. In: *Mathematical and Computer Modelling* 24.9, pp. 11–19. ISSN: 08957177. DOI: 10.

1016/0895-7177(96)00150-1. URL: <https://linkinghub.elsevier.com/retrieve/pii/0895717796001501> (visited on 06/10/2019).

Pinedo, Michael (1982). “On the Computational Complexity of Stochastic Scheduling Problems”. In: *Deterministic and Stochastic Scheduling*. Ed. by M. A. H. Dempster, J. K. Lenstra, and A. H. G. Rinnooy Kan. NATO Advanced Study Institutes Series. Dordrecht: Springer Netherlands, pp. 355–365. ISBN: 978-94-009-7801-0. DOI: 10.1007/978-94-009-7801-0_21.

Pinedo, Michael L. (2016). *Scheduling*. en. Cham: Springer International Publishing. ISBN: 978-3-319-26578-0 978-3-319-26580-3. DOI: 10.1007/978-3-319-26580-3. URL: <http://link.springer.com/10.1007/978-3-319-26580-3> (visited on 01/04/2021).

Productive4.0 - A European co-funded innovation and lighthouse project on Digital Industry (2021). en-US. URL: <https://productive40.eu/> (visited on 01/10/2021).

Ramasesh, R (Jan. 1990). “Dynamic job shop scheduling: A survey of simulation research”. en. In: *Omega* 18.1, pp. 43–57. ISSN: 0305-0483. DOI: 10.1016/0305-0483(90)90017-4. URL: <http://www.sciencedirect.com/science/article/pii/0305048390900174> (visited on 04/06/2020).

Renna, Paolo and Pierluigi Argoneto (Apr. 2010). “A game theoretic coordination for trading capacity in multisite factory environment”. en. In: *Int J Adv Manuf Technol* 47.9, pp. 1241–1252. ISSN: 1433-3015. DOI: 10.1007/s00170-009-2254-x. URL: <https://doi.org/10.1007/s00170-009-2254-x> (visited on 08/02/2019).

RStudio / Open source & professional software for data science teams (2020). Library Catalog: rstudio.com. URL: <https://rstudio.com/> (visited on 06/04/2020).

Sabuncuoglu, Ihsan and Souheyl Touhami (Jan. 2002). “Simulation metamodelling with neural networks: An experimental investigation”. In: *International Journal of Production Research* 40.11. Publisher: Taylor & Francis, pp. 2483–2505. ISSN:

- 0020-7543. DOI: 10.1080/00207540210135596. URL: <https://doi.org/10.1080/00207540210135596> (visited on 05/22/2020).
- Sakawa, Masatoshi and Tetsuya Mori (Apr. 1999). “An efficient genetic algorithm for job-shop scheduling problems with fuzzy processing time and fuzzy due date”. en. In: *Computers & Industrial Engineering* 36.2, pp. 325–341. ISSN: 0360-8352. DOI: 10.1016/S0360-8352(99)00135-7. URL: <http://www.sciencedirect.com/science/article/pii/S0360835299001357> (visited on 08/25/2020).
- Schmidt, Rainer et al. (2015). “Industry 4.0 - Potentials for Creating Smart Products: Empirical Research Results”. en. In: *Business Information Systems*. Ed. by Witold Abramowicz. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, pp. 16–27. ISBN: 978-3-319-19027-3. DOI: 10.1007/978-3-319-19027-3_2.
- Seok, Hyesung and Shimon Y. Nof (Jan. 2014). “Dynamic coalition reformation for adaptive demand and capacity sharing”. In: *International Journal of Production Economics* 147, pp. 136–146. ISSN: 0925-5273. DOI: 10.1016/j.ijpe.2013.09.003. URL: <http://www.sciencedirect.com/science/article/pii/S0925527313003897> (visited on 08/02/2019).
- Shahzad, Atif and Nasser Mebarki (Sept. 2012). “Data mining based job dispatching using hybrid simulation-optimization approach for shop scheduling problem”. en. In: *Engineering Applications of Artificial Intelligence* 25.6, pp. 1173–1181. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2012.04.001. URL: <http://www.sciencedirect.com/science/article/pii/S0952197612000899> (visited on 01/05/2021).
- Sharma, Pankaj and Ajai Jain (Aug. 2015). “Performance analysis of dispatching rules in a stochastic dynamic job shop manufacturing system with sequence-dependent setup times: Simulation approach”. en. In: *CIRP Journal of Manufacturing Science and Technology* 10, pp. 110–119. ISSN: 17555817. DOI: 10.1016/j.cirpj.2015.03.003. URL: <https://linkinghub.elsevier.com/retrieve/pii/S1755581715000140> (visited on 01/20/2020).

- Shen, Jiayu and Yuanguo Zhu (June 2016). “Chance-constrained model for uncertain job shop scheduling problem”. en. In: *Soft Computing* 20.6, pp. 2383–2391. ISSN: 1432-7643, 1433-7479. DOI: 10.1007/s00500-015-1647-z. URL: <http://link.springer.com/10.1007/s00500-015-1647-z> (visited on 07/09/2020).
- Shoval, Shraga and Mahmoud Efatmaneshnik (2018). “A probabilistic approach to the Stochastic Job-Shop Scheduling problem”. en. In: *Procedia Manufacturing* 21, pp. 533–540. ISSN: 23519789. DOI: 10.1016/j.promfg.2018.02.154. URL: <https://linkinghub.elsevier.com/retrieve/pii/S235197891830194X> (visited on 01/20/2020).
- Takeda-Berger, Satie L. et al. (2020). “Machine Learning in Production Scheduling: An Overview of the Academic Literature”. en. In: *Dynamics in Logistics*. Ed. by Michael Freitag et al. Lecture Notes in Logistics. Cham: Springer International Publishing, pp. 409–419. ISBN: 978-3-030-44783-0. DOI: 10.1007/978-3-030-44783-0_39.
- Toktay, L. Beril and Reha Uzsoy (Aug. 1998). “A capacity allocation problem with integer side constraints”. In: *European Journal of Operational Research* 109.1, pp. 170–182. ISSN: 0377-2217. DOI: 10.1016/S0377-2217(98)80011-8. URL: <http://www.sciencedirect.com/science/article/pii/S0377221798800118> (visited on 03/20/2019).
- Trigueiro de Sousa Junior, Wilson et al. (Feb. 2019). “Discrete simulation-based optimization methods for industrial engineering problems: A systematic literature review”. en. In: *Computers & Industrial Engineering* 128, pp. 526–540. ISSN: 0360-8352. DOI: 10.1016/j.cie.2018.12.073. URL: <http://www.sciencedirect.com/science/article/pii/S036083521830682X> (visited on 12/14/2019).
- Uzsoy, Reha, Chung-Yee Lee, and Louis A. Martin-Vega (Sept. 1992). “A Review of Production Planning and Scheduling Models in the Semiconductor Industry Part I: System Characteristics, Performance Evaluation and Production Planning”. In: *IIE Transactions* 24.4, pp. 47–60. ISSN: 0740-817X. DOI: 10.1080/

07408179208964233. URL: <https://doi.org/10.1080/07408179208964233>
(visited on 03/22/2019).

Vela, Camino R. et al. (July 2020). “Evolutionary tabu search for flexible due-date satisfaction in fuzzy job shop scheduling”. en. In: *Computers & Operations Research* 119, p. 104931. ISSN: 0305-0548. DOI: 10.1016/j.cor.2020.104931. URL: <http://www.sciencedirect.com/science/article/pii/S0305054820300484>
(visited on 12/31/2020).

Vinod, V. and R. Sridharan (Mar. 2009). “Simulation-based metamodels for scheduling a dynamic job shop with sequence-dependent setup times”. In: *International Journal of Production Research* 47.6. Publisher: Taylor & Francis _eprint: <https://doi.org/10.1080/00207540701486082>, pp. 1425–1447. ISSN: 0020-7543. DOI: 10.1080/00207540701486082. URL: <https://doi.org/10.1080/00207540701486082>
(visited on 05/23/2020).

Voss, Chris, Nikos Tsikriktsis, and Mark Frohlich (Jan. 2002). “Case research in operations management”. In: *International Journal of Operations & Production Management* 22.2. Publisher: MCB UP Ltd, pp. 195–219. ISSN: 0144-3577. DOI: 10.1108/01443570210414329. URL: <https://doi.org/10.1108/01443570210414329> (visited on 01/10/2021).

Wang, Bing et al. (Jan. 2018). “A hybrid local-search algorithm for robust job-shop scheduling under scenarios”. en. In: *Applied Soft Computing* 62, pp. 259–271. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2017.10.020. URL: <http://www.sciencedirect.com/science/article/pii/S1568494617306257> (visited on 12/05/2019).

Westbrook, Roy (Jan. 1995). “Action research: a new paradigm for research in production and operations management”. In: *International Journal of Operations & Production Management* 15.12. Publisher: MCB UP Ltd, pp. 6–20. ISSN: 0144-3577. DOI: 10.1108/01443579510104466. URL: <https://doi.org/10.1108/01443579510104466> (visited on 01/19/2021).

- Weyer, Stephan et al. (Jan. 2015). “Towards Industry 4.0 - Standardization as the crucial challenge for highly modular, multi-vendor production systems”. en. In: *IFAC-PapersOnLine*. 15th IFAC Symposium on Information Control Problems in Manufacturing 48.3, pp. 579–584. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2015.06.143. URL: <http://www.sciencedirect.com/science/article/pii/S2405896315003821> (visited on 01/03/2021).
- Will M. Bertrand, J. and Jan C. Fransoo (Jan. 2002). “Operations management research methodologies using quantitative modeling”. In: *International Journal of Operations & Production Management* 22.2. Publisher: MCB UP Ltd, pp. 241–264. ISSN: 0144-3577. DOI: 10.1108/01443570210414338. URL: <https://doi.org/10.1108/01443570210414338> (visited on 01/19/2021).
- Winands, E.M.M., I.J.B.F. Adan, and G.J. van Houtum (Apr. 2011). “The stochastic economic lot scheduling problem: A survey”. en. In: *European Journal of Operational Research* 210.1, pp. 1–9. ISSN: 03772217. DOI: 10.1016/j.ejor.2010.06.011. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0377221710004170> (visited on 03/21/2020).
- Wu, Chih-Hung, Hung-Ju Chou, and Wei-Han Su (Dec. 2008). “Direct transformation of coordinates for GPS positioning using the techniques of genetic programming and symbolic regression”. en. In: *Engineering Applications of Artificial Intelligence* 21.8, pp. 1347–1359. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2008.02.001. URL: <http://www.sciencedirect.com/science/article/pii/S0952197608000079> (visited on 04/20/2020).
- Wu, S. David, Murat Erkoc, and Suleyman Karabuk (Apr. 2005). “Managing Capacity in the High-Tech Industry: A Review of Literature”. In: *The Engineering Economist* 50.2, pp. 125–158. ISSN: 0013-791X. DOI: 10.1080/00137910590949887. URL: <https://doi.org/10.1080/00137910590949887> (visited on 03/20/2019).
- Wu, Zigao, Shaohua Yu, and Tiancheng Li (June 2019). “A Meta-Model-Based Multi-Objective Evolutionary Approach to Robust Job Shop Scheduling”. en. In: *Mathematics* 7.6. Number: 6 Publisher: Multidisciplinary Digital Publishing

- Institute, p. 529. DOI: 10.3390/math7060529. URL: <https://www.mdpi.com/2227-7390/7/6/529> (visited on 05/22/2020).
- Xu, Jie et al. (Nov. 2016). “Simulation optimization in the era of Industrial 4.0 and the Industrial Internet”. en. In: *Journal of Simulation* 10.4, pp. 310–320. ISSN: 1747-7778, 1747-7786. DOI: 10.1057/s41273-016-0037-6. URL: <https://www.tandfonline.com/doi/full/10.1057/s41273-016-0037-6> (visited on 11/15/2020).
- Yang, Feng and Jingang Liu (Nov. 2012). “Simulation-based transfer function modeling for transient analysis of general queueing systems”. en. In: *European Journal of Operational Research* 223.1, pp. 150–166. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2012.05.040. URL: <http://www.sciencedirect.com/science/article/pii/S0377221712004298> (visited on 06/22/2020).
- Yang, Hong-an, Yangyang Lv, et al. (2014). *Optimal Computing Budget Allocation for Ordinal Optimization in Solving Stochastic Job Shop Scheduling Problems*. en. Research article. DOI: 10.1155/2014/619254. URL: <https://www.hindawi.com/journals/mpe/2014/619254/abs/> (visited on 10/07/2019).
- Yildiz, Gokalp and Ozgur Eski (2006). “An Artificial Neural Network Based Simulation Metamodeling Approach for Dual Resource Constrained Assembly Line”. en. In: *Artificial Neural Networks – ICANN 2006*. Ed. by Stefanos Kollias et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 1002–1011. ISBN: 978-3-540-38873-9. DOI: 10.1007/11840930_104.
- Zhang, Jian, Guofu Ding, et al. (Apr. 1, 2019). “Review of job shop scheduling research and its new perspectives under Industry 4.0”. In: *Journal of Intelligent Manufacturing* 30.4, pp. 1809–1830. ISSN: 1572-8145. DOI: 10.1007/s10845-017-1350-2. URL: <https://doi.org/10.1007/s10845-017-1350-2> (visited on 01/27/2020).
- Zhang, Luping and T. N. Wong (July 2015). “An object-coding genetic algorithm for integrated process planning and scheduling”. In: *European Journal of Operational Research* 244.2, pp. 434–444. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2015.

01.032. URL: <http://www.sciencedirect.com/science/article/pii/S0377221715000521> (visited on 08/21/2018).

Zhang, Rui, Shiji Song, and Cheng Wu (Mar. 2013). “A hybrid differential evolution algorithm for job shop scheduling problems with expected total tardiness criterion”. en. In: *Applied Soft Computing* 13.3, pp. 1448–1458. ISSN: 15684946. DOI: 10.1016/j.asoc.2012.02.024. URL: <https://linkinghub.elsevier.com/retrieve/pii/S1568494612001172> (visited on 01/17/2020).

Zhou, K., Taigang Liu, and Lifeng Zhou (Aug. 2015). “Industry 4.0: Towards future industrial opportunities and challenges”. In: *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pp. 2147–2152. DOI: 10.1109/FSKD.2015.7382284.

Appendix A

Glossary of Acronyms

Table A.1: List of Acronyms.

Abbreviation	Explanation	Abbreviation	Explanation
AP	Avoidance procedure	IT	Information Technology
AR	Action Research	JSSP	Job Shop Scheduling Problem
CAPPA	Capacity Allocation Problem in Photolithography Area	KG	Kriging
CBA	Computation Budget Allocation	MES	Manufacturing Execution System
CPS	Cyber-Physical Systems	MIP	Mixed Integer Programming
DESM	Discrete Event Simulation Model	MILP	Mixed Integer Linear Programming
DJSSP	Deterministic Job Shop Scheduling Problem	ML	Machine Learning
DST	Decision Support Tool	MPBLV	Machines Positions Based Learning Vector
ED	Empirical Distribution	MPIBLV	Machines Positions Interaction Based Learning Vector
ELBSO	Evolutionary Learning Based Simulation Optimization	MRE	Mean Relative Error
FA	Firefly Algorithm	OO	Ordinal Optimization
GA	Genetic Algorithm	PP&S	Production Planning & Scheduling
GBML	Graph-Based Machine Learning	PSO	Particle Swarm Optimization
GH	Greedy Heuristic	QPBLV	Queue Positions Based Learning Vector
GP	Genetic Programming	RBF	Radial Basis Function
GPS	Global Positioning System	RGGA	Reference Group GA
GRSM	Generalized Response Surface Methodology	RSM	Response Surface Methodology
IoT	Internet of Things	SA	Simulated Annealing
IoS	Internet of Services	SJSSP	Stochastic Job Shop Scheduling Problem
IP	Imitation procedure	SO	Simulation Optimization
IRGGA	Improved Reference Group Genetic Algorithm	STD	Standard Deviation

Appendix B

Data sets to evaluate IRGGA

In this appendix, a sample of data sets used to examine IRGGA in Section 3.5.3 is presented. Figure B.1 shows the capability of machines (m1, m2, etc.) in processing each certain process (cap1, cap2, etc.), while the rest of figures describe the layers of orders environment consisting of processing time (Time) of each layer of orders planned to be produced in a certain week (Week) and needs certain reticle (Reticle) and process capability (Capability), while it is critical or not.

	cap1	cap2	cap3	cap4	cap5	cap6	cap7	cap8	cap9	cap10	cap11	cap12
m1	1	1	0	1	1	1	1	0	1	0	1	0
m2	1	1	1	0	1	1	1	1	1	0	0	1
m3	0	0	0	0	1	1	1	0	1	1	1	1
m4	0	1	1	0	1	0	1	0	0	0	1	1
m5	1	1	1	1	1	1	1	1	1	1	1	1
m6	1	1	1	0	1	1	1	1	1	0	1	0
m7	1	0	1	0	1	0	1	0	0	1	0	1
m8	1	0	1	1	1	0	0	0	1	0	1	1
m9	1	1	0	0	1	1	0	0	1	1	1	1
m10	1	0	1	1	1	0	0	0	1	1	1	1
m11	1	1	0	0	1	1	0	0	1	1	1	1
m12	1	1	1	1	1	1	1	1	1	1	1	1

Figure B.1: CAPPA machines input, where in each row the capability of each machine in running a certain process (cap1, cap2, etc.) is showed by 1, and 0 otherwise.

Order	Layer	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	Time	46.5	59.52	37.2	64.17	63.24	53.94	40.92	56.73	55.8	42.78	50.22	54.87	50.22	62.31	39.99
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	5	8	4	10	3	4	9	12	9	3	6	3	12	2	1
	Critical	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
	Reticle	3	2	15	12	10	6	15	9	14	7	6	9	10	20	14
2	Time	53.94	43.71	37.2	43.71	52.08	51.15	49.29	57.66	46.5	57.66	53.94	63.24	50.22	59.52	47.43
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	6	1	8	7	10	5	8	2	10	6	10	11	3	8	8
	Critical	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
	Reticle	7	12	9	13	14	13	6	10	15	15	4	4	6	1	2
3	Time	39.06	38.13	62.31	62.31	64.17	39.99	51.15	39.99	53.01	50.22	42.78	41.85	53.94	43.71	55.8
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	6	1	7	12	12	9	6	1	1	1	5	11	10	12	11
	Critical	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0
	Reticle	10	6	18	16	13	16	11	4	1	16	9	18	2	12	6
4	Time	57.66	53.01	42.78	39.99	45.57	62.31	57.66	62.31	55.8	49.29	54.87	55.8	44.64	52.08	38.13
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	5	7	2	3	7	7	7	4	4	4	8	4	11	6	5
	Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	Reticle	1	2	11	16	13	8	10	13	10	19	16	9	9	12	17
5	Time	56.73	53.01	64.17	62.31	43.71	49.29	43.71	62.31	64.17	54.87	60.45	47.43	39.06	60.45	56.73
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	11	1	7	6	4	6	11	12	5	6	5	4	12	8	9
	Critical	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	Reticle	2	5	15	7	3	8	10	8	5	9	15	19	4	18	20
6	Time	63.24	47.43	49.29	39.99	61.38	40.92	59.52	56.73	63.24	56.73	55.8	65.1	50.22	60.45	57.66
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	4	2	12	4	4	11	3	11	4	7	11	9	12	11	5
	Critical	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	Reticle	14	5	20	19	5	11	5	8	10	9	1	20	13	15	20
7	Time	47.43	65.1	65.1	64.17	49.29	44.64	39.06	45.57	43.71	59.52	50.22	40.92	47.43	58.59	44.64
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	6	11	3	7	12	11	1	5	3	7	1	2	10	3	8
	Critical	0	1	0	1	1	0	0	0	0	0	0	0	0	0	1
	Reticle	5	2	15	8	7	1	8	20	12	18	16	17	5	17	9
8	Time	63.24	39.99	54.87	62.31	62.31	41.85	39.99	50.22	55.8	45.57	59.52	43.71	59.52	45.57	44.64
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	1	3	5	9	6	1	9	7	2	5	5	5	1	2	5
	Critical	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
	Reticle	9	18	16	12	5	16	9	12	13	16	4	11	20	13	16
9	Time	47.43	43.71	47.43	61.38	55.8	59.52	60.45	39.99	62.31	39.06	47.43	45.57	56.73	42.78	53.01
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	7	5	4	7	6	11	4	6	10	5	10	6	12	3	11
	Critical	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	Reticle	17	11	5	20	5	4	2	6	12	1	12	13	10	12	1
10	Time	43.71	39.06	50.22	56.73	55.8	63.24	53.94	48.36	41.85	54.87	60.45	64.17	54.87	61.38	59.52
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	10	12	6	1	11	9	8	12	5	9	11	12	1	2	6
	Critical	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	Reticle	17	9	6	4	16	18	11	19	3	17	7	4	2	4	20

Figure B.2: CAPP inputs sample 1 showing the processing time (Time) of each layer of orders planned to be produced in a certain week (Week) and needs certain reticle (Reticle) and process capability (Capability), while it is critical (Critical=1) or not (Critical=0).

11	Time	51.15	53.94	65.1	60.45	37.2	38.13	56.73	54.87	60.45	44.64	49.29	49.29	54.87	49.29	58.59
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	5	11	2	11	7	10	3	5	10	10	9	3	4	2	1
	Critical	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
12	Reticle	9	6	9	9	13	13	11	1	9	12	20	15	16	15	13
	Time	39.99	39.06	64.17	51.15	62.31	46.5	49.29	59.52	64.17	58.59	39.06	56.73	37.2	65.1	41.85
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	11	3	8	4	7	10	7	8	7	9	2	9	8	6	3
13	Critical	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	Reticle	20	11	5	18	12	18	18	7	12	2	20	9	20	14	18
	Time	40.92	49.29	48.36	39.06	56.73	64.17	58.59	43.71	42.78	56.73	42.78	63.24	37.2	38.13	41.85
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
14	Capability	3	9	11	11	6	7	12	6	5	2	2	1	10	2	2
	Critical	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1
	Reticle	12	13	10	8	4	16	7	9	11	1	16	2	17	17	2
	Time	49.29	52.08	60.45	45.57	39.99	59.52	53.01	39.99	48.36	52.08	56.73	54.87	48.36	54.87	50.22
15	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	11	9	9	12	7	7	8	9	10	1	8	9	1	4	8
	Critical	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1
	Reticle	4	2	11	4	7	3	19	15	9	8	10	20	18	4	18
16	Time	46.5	51.15	39.06	40.92	49.29	62.31	51.15	58.59	54.87	39.99	49.29	54.87	65.1	37.2	47.43
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	4	10	5	8	10	4	3	8	12	10	1	7	12	5	7
	Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	Reticle	3	4	12	6	4	15	16	10	6	19	19	4	6	2	8
	Time	49.29	59.52	43.71	38.13	62.31	48.36	41.85	65.1	50.22	54.87	55.8	39.06	60.45	64.17	63.24
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	3	7	7	9	2	9	5	6	7	4	9	8	12	4	7
18	Critical	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1
	Reticle	5	6	9	11	19	13	5	18	17	10	7	1	20	9	6
	Time	55.8	50.22	40.92	40.92	60.45	39.99	47.43	61.38	44.64	52.08	60.45	39.06	51.15	57.66	55.8
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
19	Capability	6	4	7	5	6	1	4	12	9	11	5	4	6	7	12
	Critical	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
	Reticle	1	6	20	15	3	13	12	6	16	18	2	12	2	10	11
	Time	37.2	58.59	48.36	47.43	52.08	53.01	50.22	41.85	55.8	50.22	57.66	53.01	38.13	64.17	62.31
20	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	9	9	1	11	1	11	8	8	10	5	4	9	10	11	1
	Critical	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
	Reticle	2	7	6	9	14	12	13	18	3	19	16	5	18	19	20
19	Time	43.71	57.66	40.92	39.06	65.1	56.73	62.31	40.92	43.71	48.36	50.22	49.29	41.85	47.43	46.5
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	8	8	11	9	1	5	6	1	7	12	4	2	11	12	9
	Critical	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
20	Reticle	5	9	13	8	18	7	3	4	16	7	3	13	19	16	6
	Time	53.94	65.1	64.17	45.57	47.43	53.94	56.73	64.17	56.73	59.52	38.13	50.22	40.92	39.99	40.92
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	5	11	10	7	8	6	1	12	3	7	6	7	1	1	2
20	Critical	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
	Reticle	11	7	1	10	11	2	13	14	19	4	3	11	15	19	6

Figure B.3: CAPPA inputs sample 2.

21	Time	53.94	41.85	47.43	59.52	62.31	49.29	48.36	56.73	49.29	38.13	54.87	45.57	59.52	63.24	64.17
	Week	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2
	Capability	4	8	2	1	2	9	3	4	8	3	9	6	5	9	9
	Critical	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
22	Reticle	19	3	8	12	11	13	17	8	16	8	20	16	18	10	5
	Time	59.52	37.2	54.87	53.94	39.99	63.24	48.36	37.2	37.2	52.08	46.5	39.06	41.85	42.78	55.8
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	4	1	9	8	10	7	2	11	9	9	2	3	1	5	12
23	Critical	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1
	Reticle	17	17	2	13	11	5	8	7	20	2	7	17	18	10	5
	Time	52.08	43.71	42.78	40.92	53.01	50.22	45.57	46.5	39.06	42.78	61.38	62.31	53.94	61.38	46.5
	Week	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2
24	Capability	12	1	5	2	6	4	12	6	7	12	11	8	7	10	5
	Critical	1	1	0	0	0	0	0	0	0	0	0	1	0	1	1
	Reticle	10	10	1	18	14	7	15	17	11	4	2	7	3	13	10
	Time	44.64	63.24	39.99	45.57	49.29	50.22	56.73	56.73	56.73	59.52	45.57	39.06	53.01	52.08	49.29
25	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	11	3	5	8	1	10	7	9	3	9	7	7	4	1	4
	Critical	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	Reticle	18	15	4	3	16	19	20	11	19	13	7	10	16	20	9
26	Time	63.24	60.45	60.45	53.94	53.01	54.87	50.22	49.29	43.71	37.2	46.5	55.8	39.99	42.78	47.43
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	6	2	8	9	2	7	3	4	3	11	2	1	2	6	7
	Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
27	Reticle	7	9	9	2	18	8	9	19	7	19	14	2	1	4	10
	Time	45.57	57.66	63.24	42.78	56.73	46.5	37.2	60.45	50.22	59.52	37.2	51.15	50.22	49.29	58.59
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	12	6	10	4	3	5	11	12	2	5	5	11	5	11	10
28	Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	Reticle	11	6	19	7	1	15	4	16	16	7	14	1	19	18	12
	Time	64.17	40.92	41.85	40.92	38.13	64.17	41.85	55.8	42.78	45.57	60.45	60.45	48.36	42.78	38.13
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
29	Capability	2	12	5	6	1	4	9	10	4	7	8	2	6	6	3
	Critical	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	Reticle	20	20	15	3	12	7	11	6	11	5	13	11	12	14	17
	Time	42.78	46.5	39.06	53.01	39.06	50.22	42.78	50.22	39.99	44.64	49.29	42.78	60.45	62.31	47.43
30	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	11	11	7	3	10	1	3	1	10	3	5	5	5	12	6
	Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	Reticle	13	9	1	3	10	7	9	19	18	19	14	3	4	19	10
31	Time	49.29	59.52	39.99	62.31	48.36	42.78	49.29	53.94	50.22	39.99	64.17	60.45	39.99	58.59	48.36
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	3	4	9	10	5	1	4	2	3	5	4	12	2	5	4
	Critical	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
32	Reticle	16	1	20	13	16	20	15	12	14	8	3	12	13	17	15
	Time	43.71	59.52	55.8	44.64	57.66	58.59	49.29	50.22	62.31	47.43	55.8	59.52	62.31	38.13	55.8
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	2	5	7	9	1	5	10	5	4	6	10	1	10	1	2
33	Critical	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reticle	9	14	5	20	1	19	6	15	14	10	3	9	2	8	20

Figure B.4: CAPPA input sample 3.

31	Time	53.01	50.22	51.15	46.5	48.36	38.13	62.31	47.43	54.87	43.71	48.36	52.08	43.71	53.94	57.66
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	11	4	1	2	12	2	11	1	3	9	8	11	12	9	6
	Critical	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	Reticle	19	20	2	9	17	17	18	1	14	11	9	19	11	8	17
32	Time	45.57	41.85	43.71	49.29	39.99	39.06	48.36	52.08	65.1	64.17	54.87	37.2	63.24	53.94	46.5
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	9	7	5	11	3	6	8	3	9	8	4	7	5	7	12
	Critical	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0
	Reticle	1	5	7	20	12	16	11	8	16	9	4	8	8	19	17
33	Time	58.59	55.8	41.85	64.17	64.17	43.71	40.92	65.1	42.78	44.64	63.24	47.43	48.36	47.43	52.08
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	3	11	5	12	6	5	10	3	11	4	5	3	3	1	1
	Critical	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1
	Reticle	16	17	18	8	12	15	18	1	13	3	1	7	7	14	9
34	Time	38.13	63.24	57.66	38.13	41.85	39.99	43.71	52.08	56.73	44.64	40.92	57.66	40.92	48.36	54.87
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	8	6	9	6	2	11	12	11	3	8	7	3	12	1	9
	Critical	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reticle	1	13	4	9	1	6	18	20	5	5	5	15	17	5	10
35	Time	44.64	48.36	38.13	59.52	41.85	41.85	63.24	61.38	57.66	62.31	46.5	55.8	59.52	60.45	40.92
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	4	12	7	2	11	8	6	5	5	3	8	1	3	12	6
	Critical	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
	Reticle	17	14	20	16	2	5	8	9	13	5	11	10	10	20	13
36	Time	65.1	60.45	55.8	50.22	37.2	54.87	39.06	62.31	53.94	50.22	45.57	55.8	63.24	49.29	60.45
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	4	11	1	12	6	11	10	6	4	4	9	6	4	7	1
	Critical	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	Reticle	14	9	19	19	3	15	1	11	17	4	12	19	17	17	5
37	Time	50.22	42.78	56.73	46.5	64.17	50.22	61.38	63.24	62.31	56.73	63.24	55.8	52.08	58.59	47.43
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	9	2	3	7	2	5	4	6	9	9	4	5	4	11	12
	Critical	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reticle	5	19	10	10	8	1	11	10	2	1	8	15	8	14	2
38	Time	56.73	47.43	53.94	57.66	50.22	48.36	50.22	42.78	54.87	39.99	59.52	50.22	54.87	50.22	58.59
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	9	10	8	2	2	4	1	1	3	10	6	9	12	3	8
	Critical	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	Reticle	8	1	18	16	4	1	3	17	19	9	19	11	5	18	2
39	Time	62.31	58.59	43.71	51.15	38.13	52.08	51.15	63.24	63.24	63.24	37.2	65.1	41.85	50.22	39.06
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	11	3	1	1	10	5	8	10	5	7	6	8	11	9	5
	Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reticle	9	10	17	18	7	10	4	10	13	5	18	4	4	7	1
40	Time	61.38	47.43	40.92	49.29	59.52	44.64	37.2	60.45	57.66	55.8	58.59	65.1	48.36	48.36	38.13
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	9	9	2	1	3	11	10	10	11	7	1	7	1	9	12
	Critical	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	Reticle	19	11	5	12	12	8	19	4	2	2	1	15	7	6	18

Figure B.5: CAPPA inputs sample 4.

41	Time	56.73	63.24	42.78	39.06	56.73	63.24	56.73	38.13	65.1	63.24	55.8	42.78	64.17	65.1	47.43
	Week	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2
	Capability	8	9	10	3	6	12	8	5	9	3	8	2	7	1	3
	Critical	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Reticle	11	7	7	3	16	19	19	1	15	2	9	20	9	7	8	
42	Time	45.57	51.15	57.66	40.92	42.78	53.94	54.87	53.94	60.45	57.66	61.38	37.2	49.29	48.36	46.5
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	2	8	4	11	11	10	12	12	9	3	10	4	2	6	3
	Critical	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Reticle	2	10	2	19	3	4	4	12	14	2	19	17	16	1	15	
43	Time	62.31	40.92	48.36	43.71	51.15	45.57	50.22	39.06	52.08	43.71	52.08	55.8	55.8	41.85	44.64
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	11	1	9	2	12	8	7	3	4	8	2	1	9	7	11
	Critical	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reticle	5	2	10	8	12	6	7	5	16	13	19	10	8	16	20	
44	Time	53.94	65.1	37.2	42.78	39.06	58.59	45.57	59.52	64.17	41.85	43.71	47.43	39.99	62.31	47.43
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	5	6	5	5	12	10	5	3	1	12	9	12	11	12	4
	Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reticle	20	17	18	6	12	16	10	20	5	8	12	20	8	10	8	
45	Time	63.24	44.64	38.13	58.59	43.71	40.92	45.57	53.94	62.31	42.78	49.29	42.78	39.99	48.36	54.87
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	3	3	4	5	9	4	4	4	10	9	1	8	3	8	5
	Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Reticle	7	1	6	3	8	12	17	15	3	16	7	1	6	16	10	
46	Time	49.29	57.66	65.1	38.13	45.57	53.94	53.01	54.87	53.01	50.22	47.43	39.99	50.22	56.73	56.73
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	6	2	2	6	8	7	2	3	11	6	3	8	11	12	9
	Critical	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
Reticle	2	16	12	19	8	18	18	20	9	15	10	12	5	20	11	
47	Time	65.1	56.73	55.8	57.66	60.45	47.43	50.22	53.01	57.66	43.71	39.99	45.57	38.13	47.43	57.66
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	6	6	9	3	11	11	6	9	2	5	3	5	8	4	11
	Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reticle	3	7	13	7	20	8	10	2	14	8	11	3	11	2	1	
48	Time	39.99	42.78	44.64	58.59	49.29	61.38	38.13	42.78	49.29	39.99	62.31	45.57	57.66	42.78	40.92
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	3	1	5	11	11	4	8	5	12	6	12	11	12	2	6
	Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reticle	2	15	8	4	15	6	12	20	12	20	14	6	13	12	7	
49	Time	60.45	48.36	47.43	40.92	51.15	60.45	62.31	39.99	42.78	48.36	45.57	46.5	65.1	44.64	46.5
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	3	9	8	7	10	6	9	5	5	1	5	9	4	12	3
	Critical	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Reticle	12	4	20	4	9	18	5	15	19	4	19	18	6	2	8	
50	Time	64.17	56.73	43.71	56.73	61.38	54.87	47.43	57.66	44.64	38.13	44.64	49.29	42.78	48.36	52.08
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	2	7	5	9	4	11	10	8	7	11	9	3	3	7	2
	Critical	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Reticle	16	4	17	14	18	18	2	17	3	15	1	20	1	9	11	

Figure B.6: CAPPA inputs sample 5.

51	Time	42.78	43.71	65.1	61.38	49.29	59.52	59.52	49.29	40.92	60.45	56.73	38.13	61.38	53.94	43.71
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	3	8	12	1	10	4	2	8	5	1	4	7	10	9	2
	Critical	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reticle	18	18	12	16	11	13	11	9	8	16	4	19	8	6	9
52	Time	59.52	46.5	51.15	60.45	41.85	47.43	48.36	56.73	43.71	43.71	57.66	50.22	53.01	60.45	65.1
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	1	12	11	4	5	10	12	9	9	11	5	12	12	3	1
	Critical	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	Reticle	15	3	18	1	18	7	1	15	14	9	7	18	18	2	7
53	Time	40.92	51.15	37.2	64.17	38.13	50.22	55.8	50.22	45.57	38.13	64.17	42.78	60.45	65.1	64.17
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	3	7	3	3	9	3	7	9	6	10	6	6	4	10	9
	Critical	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reticle	18	1	7	9	17	16	2	2	15	16	1	18	10	7	6
54	Time	62.31	64.17	39.99	45.57	42.78	53.01	63.24	49.29	61.38	40.92	38.13	43.71	41.85	54.87	57.66
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	10	11	3	9	7	12	9	2	1	2	4	1	3	3	9
	Critical	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
	Reticle	7	13	9	9	18	19	5	11	8	5	14	4	14	14	11
55	Time	40.92	39.06	38.13	39.06	50.22	47.43	37.2	59.52	53.94	54.87	44.64	46.5	64.17	46.5	49.29
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	5	6	1	5	4	11	1	4	1	7	10	8	4	11	3
	Critical	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	Reticle	17	14	2	15	15	20	19	13	19	11	5	17	16	17	18
56	Time	46.5	53.01	51.15	49.29	43.71	37.2	37.2	46.5	46.5	52.08	43.71	53.01	40.92	38.13	56.73
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	9	12	8	1	7	7	6	7	12	2	8	9	3	1	4
	Critical	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
	Reticle	18	4	12	2	8	1	1	13	12	1	7	20	9	4	20
57	Time	39.99	50.22	53.01	58.59	55.8	41.85	57.66	39.99	62.31	57.66	62.31	50.22	46.5	63.24	63.24
	Week	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
	Capability	9	8	7	7	8	5	6	9	9	6	11	8	12	7	1
	Critical	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
	Reticle	2	12	4	15	6	19	15	17	1	20	2	16	12	2	20

Figure B.7: CAPPA inputs sample 6.

Appendix C

IRGGA MATLAB Codes

This appendix shows the actual MATLAB codes of pseudocodes defined in algorithms 1, 2, 3, and 4.

```
clc
clear
close all
format shortG
spmd

%% parametrs setting

ordersize=57;
popsize=75;
maxiter=2000;
nweek=3;
nmachin=12;
patsize=5;
upatsize=10;
crosr=1;
mutr=0.2;
npat=0.6;
nmpat=0.2;
ncros=round(crosr*popsize);
nmut=round(mutr*popsize);

process=repmat([],ordersize);

machinc=[1 1 0 1 1 1 1 0 1 0 1 0
1 1 1 0 1 1 1 1 1 0 0 1
0 0 0 0 1 1 1 0 1 1 1 1
0 1 1 0 1 0 1 0 0 0 1 1
1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 0 1 0
1 0 1 0 1 0 1 0 0 1 0 1
1 0 1 1 1 0 0 0 1 0 1 1
1 1 0 0 1 1 0 0 1 1 1 1
1 0 1 1 1 0 0 0 1 1 1 1
1 1 0 0 1 1 0 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1];
```

Figure C.1: IRGGA (assigning parameters 1).

```

input=[46.5 59.52 37.2 64.17 63.24 53.94 40.92 56.73 55.8 42.78 50.22 54.87 50.22 62.31 39.99 46.5 53.01 49.29 46.5 39.
1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 0 0 0 0
5 8 4 10 3 4 9 12 9 3 6 3 12 2 1 3 9 5 3 1 10 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 0
3 2 15 12 10 6 15 9 14 7 6 9 10 20 14 11 15 2 13 13 9 0 0 0
53.94 43.71 37.2 43.71 52.08 51.15 49.29 57.66 46.5 57.66 53.94 63.24 50.22 59.52 47.43 54.87 50.22 65.1 51.15 64.17
1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3
6 1 8 7 10 5 8 2 10 6 10 11 3 8 8 3 7 12 2 2 9 3 8 3 9
0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 1 0 0 1 0 0 0
7 12 9 13 14 13 6 10 15 15 4 4 6 1 2 1 18 16 12 7 19 19 16 15 8
39.06 38.13 62.31 62.31 64.17 39.99 51.15 39.99 53.01 50.22 42.78 41.85 53.94 43.71 55.8 51.15 37.2 39.06 39.06 49.29
1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 0 0 0
6 1 7 12 12 9 6 1 1 5 11 10 12 11 4 8 6 4 9 12 2 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 0
10 6 18 16 13 16 11 4 1 16 9 18 2 12 6 18 14 14 10 1 18 15 0 0 0
57.66 53.01 42.78 39.99 45.57 62.31 57.66 62.31 55.8 49.29 54.87 55.8 44.64 52.08 38.13 64.17 41.85 48.36 48.36 59.52
1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3
5 7 2 3 7 7 7 4 4 4 8 4 11 6 5 9 5 3 9 8 1 10 11 10 5
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0
1 2 11 16 13 8 10 13 10 19 16 9 9 12 17 2 7 1 1 7 14 14 18 20 12
56.73 53.01 64.17 62.31 43.71 49.29 43.71 62.31 64.17 54.87 60.45 47.43 39.06 60.45 56.73 65.1 42.78 40.92 53.94 37.2
1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3
11 1 7 6 4 6 11 12 5 6 5 4 12 8 9 4 3 3 12 10 11 3 12 3 6
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 1
2 5 15 7 3 8 10 8 5 9 15 19 4 18 20 10 9 7 18 7 11 18 9 5 7
63.24 47.43 49.29 39.99 61.38 40.92 59.52 56.73 63.24 56.73 55.8 65.1 50.22 60.45 57.66 39.06 44.64 60.45 40.92 55.8
1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 0 0 0
4 2 12 4 4 11 3 11 4 7 11 9 12 11 5 7 8 5 6 2 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 0 0 0
14 5 20 19 5 11 5 8 10 9 1 20 13 15 20 13 6 8 8 6 0 0 0 0
47.43 65.1 65.1 64.17 49.29 44.64 39.06 45.57 43.71 59.52 50.22 40.92 47.43 58.59 44.64 47.43 50.22 58.59 42.78 53.94
1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 0
6 11 3 7 12 11 1 5 3 7 1 2 10 3 8 3 3 9 8 1 9 4 2 0 0
0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 0 0
5 2 15 8 7 1 8 20 12 18 16 17 5 17 9 4 3 2 3 18 5 17 15 0 0

```

Figure C.2: IRGGA main code (assigning parameters 2).

```

vkk=1;
for i=1:ordersize

process(i).layert=input(vkk,:);
vkk=vkk+1;

process(i).layerw=input(vkk,:);
vkk=vkk+1;

process(i).layercp=input(vkk,:);
vkk=vkk+1;

process(i).layercr=input(vkk,:);
vkk=vkk+1;

process(i).reticle=input(vkk,:);
vkk=vkk+1;
end

for i=1:ordersize
ra=[];
ca=[];
[ra,ca]=find(process(i).layert>0);
process(i).layert2=process(i).layert(ca);

process(i).layerw2=process(i).layerw(ca);

process(i).layercp2=process(i).layercp(ca);

process(i).layercr2=process(i).layercr(ca);

end

for i=1:ordersize

process(i).layernum=size(process(i).layert2,2);

end

```

Figure C.3: IRGGA main code (assigning parameters 3).

```

%% initial population
tic

emp.varocr=[];
emp.vari=[];
emp.varde=[];
emp.mach=[];
emp.varo=[];
emp.varl=[];
emp.fitw=[];
emp.fitml=[];
emp.fitT=[];
pop= repmat(emp,popsize,nweek);
mmfit.a=repmat([],popsize,nweek);

%first population
tic
for j=1:popsize
    critical=zeros(ordersize,1);
    for w=1:nweek
        cal=1;
        for i=1:ordersize
            for s=1:process(i).layernum
                if process(i).layerw2(s)==w
                    pop(j,w).varo(cal)=i;
                    pop(j,w).varl(cal)=s;
                    if process(i).layercr2(s)==1

                        pop(j,w).varocr(cal)=1;
                    else
                        pop(j,w).varocr(cal)=0;
                    end
                    f=size(alow(i,s).layer,1);
                    rd=randi([1,f]);
                    pop(j,w).vari(cal)=alow(i,s).layer(rd);
                    pop(j,w).varde(cal)=rand;
                    cal=cal+1;
                end
            end
        end
    end
end
end

```

Figure C.4: IRGGA main code (initial population 1).

```

for i=1:size(pop(j,w).vari,2)

    if pop(j,w).varcr(i)>0 && critical(pop(j,w).varo(i))>0

        pop(j,w).vari(i)=critical(pop(j,w).varo(i));

    else
        if pop(j,w).varcr(i)>0
            critical(pop(j,w).varo(i))=pop(j,w).vari(i);
        end
    end
end

end

bbb=pop(j,w).vari;

for s=1:nmachin
    t=1;
    for n=1:size(pop(j,w).vari,2)

        if bbb(n)==s
            pop(j,w).mach(s,t,1)=pop(j,w).varo(n);
            pop(j,w).mach(s,t,2)=pop(j,w).varl(n);
            pop(j,w).mach(s,t,3)=pop(j,w).varde(n);
            pop(j,w).mach(s,t,4)=n;
            t=t+1;
        end
    end
end

n=size(pop(j,w).mach,2);
for s=1:size(pop(j,w).mach,1)
    for n1=1:n-1
        for n2=n1+1:n
            if pop(j,w).mach(s,n1,1)>0 && pop(j,w).mach(s,n2,1)>0
                if pop(j,w).mach(s,n2,3)<pop(j,w).mach(s,n1,3)

```

Figure C.5: IRGGA main code (initial population 2).

```

        if pop(j,w).mach(s,n2,3)<pop(j,w).mach(s,n1,3)

            c1=pop(j,w).mach(s,n2,1);
            d1=pop(j,w).mach(s,n2,2);
            e1=pop(j,w).mach(s,n2,3);
            f1=pop(j,w).mach(s,n2,4);

            c2=pop(j,w).mach(s,n1,1);
            d2=pop(j,w).mach(s,n1,2);
            e2=pop(j,w).mach(s,n1,3);
            f2=pop(j,w).mach(s,n1,4);

            pop(j,w).mach(s,n1,1)=c1;
            pop(j,w).mach(s,n1,2)=d1;
            pop(j,w).mach(s,n1,3)=e1;
            pop(j,w).mach(s,n1,4)=f1;

            pop(j,w).mach(s,n2,1)=c2;
            pop(j,w).mach(s,n2,2)=d2;
            pop(j,w).mach(s,n2,3)=e2;
            pop(j,w).mach(s,n2,4)=f2;

            end
        end
    end
end

%fitness
[pop(j,w).fitw,mmfit(j,w).a,mmfit(j,w).b] =cappafit( pop(j,w),process);

fitt(w)=pop(j,w).fitw;

%[pop(j,w)] =GH( pop(j,w),mmfit);

```

Figure C.6: IRGGA main code (initial population 3).

```

%fitness
[pop(j,w).fitw,mmfit(j,w).a,mmfit(j,w).b] =cappafit( pop(j,w),process);

fitt(w)=pop(j,w).fitw;

%[pop(j,w)] =GH( pop(j,w),mmfit);

[maximum,~]=max(mmfit(j,w).b(:));
[raw,col]=find(mmfit(j,w).b==maximum,1,'first');
order=pop(j,w).mach(raw,col,1);
layer=pop(j,w).mach(raw,col,2);
num=pop(j,w).mach(raw,col,4);
[mat]=alow(order,layer).layer;
[~,~]=size(mat);
%pop(j,w).vari(num)=alow(order,layer).layer(randi([1,1]));

    end

    pop(j).fitT=sum(fitt);

end

[~,index]=min([pop.fitT]);
gpop=pop(index,:);
[value,index]=sort([pop.fitT]);
    pop=pop(index,:);
    patern=pop(1:patsize,:);
    l=popsiz-upatsize;

```

Figure C.7: IRGGA main code (initial population 4).

```
%% main loop...
BEST=zeros(maxiter,1);
tt=0;
for iter=1:maxiter
cpop=repmat(emp,ncros,nweek);
mpop=repmat(emp,nmut,nweek);

for j=1:ncros
critical=zeros(ordersize,1);
for w=1:nweek
z=randi(patsize);
vv=randi(popsize);
cpop(j,w)=patcappa(patern(z,w),pop(vv,w),npat);

for i=1:size(cpop(j,w).vari,2)

if cpop(j,w).varcr(i)>0 && critical(cpop(j,w).varo(i))>0

cpop(j,w).vari(i)=critical(cpop(j,w).varo(i));

else
if cpop(j,w).varcr(i)>0
critical(cpop(j,w).varo(i))=cpop(j,w).vari(i);
end
end

end

bbb=ccpop(j,w).vari;

for s=1:nmachin
t=1;
for n=1:size(cpop(j,w).vari,2)
```

Figure C.8: IRGGA main code (main loop 1).

```

if bbb(n)==s
cpop(j,w).mach(s,t,1)=cpop(j,w).varo(n);
cpop(j,w).mach(s,t,2)=cpop(j,w).varl(n);
cpop(j,w).mach(s,t,3)=cpop(j,w).varde(n);
t=t+1;
end
end
end

n=size(cpop(j,w).mach,2);
for s=1:size(cpop(j,w).mach,1)
for nl=1:n-1
for n2=nl+1:n
if cpop(j,w).mach(s,nl,1)>0 && cpop(j,w).mach(s,n2,1)>0
if cpop(j,w).mach(s,n2,3)<cpop(j,w).mach(s,nl,3)
c1=cpop(j,w).mach(s,n2,1);
d1=cpop(j,w).mach(s,n2,2);
e1=cpop(j,w).mach(s,n2,3);
c2=cpop(j,w).mach(s,nl,1);
d2=cpop(j,w).mach(s,nl,2);
e2=cpop(j,w).mach(s,nl,3);

cpop(j,w).mach(s,nl,1)=c1;
cpop(j,w).mach(s,nl,2)=d1;
cpop(j,w).mach(s,nl,3)=e1;

cpop(j,w).mach(s,n2,1)=c2;
cpop(j,w).mach(s,n2,2)=d2;
cpop(j,w).mach(s,n2,3)=e2;

end
end
end
end
end
end

```

Figure C.9: IRGGA main code (main loop 2).

```

if bbb(n)==s
cpop(j,w).mach(s,t,1)=cpop(j,w).varo(n);
cpop(j,w).mach(s,t,2)=cpop(j,w).varl(n);
cpop(j,w).mach(s,t,3)=cpop(j,w).varde(n);
t=t+1;
end
end
end

n=size(cpop(j,w).mach,2);
for s=1:size(cpop(j,w).mach,1)
for n1=1:n-1
for n2=n1+1:n
if cpop(j,w).mach(s,n1,1)>0 && cpop(j,w).mach(s,n2,1)>0
if cpop(j,w).mach(s,n2,3)<cpop(j,w).mach(s,n1,3)
c1=cpop(j,w).mach(s,n2,1);
d1=cpop(j,w).mach(s,n2,2);
e1=cpop(j,w).mach(s,n2,3);
c2=cpop(j,w).mach(s,n1,1);
d2=cpop(j,w).mach(s,n1,2);
e2=cpop(j,w).mach(s,n1,3);

cpop(j,w).mach(s,n1,1)=c1;
cpop(j,w).mach(s,n1,2)=d1;
cpop(j,w).mach(s,n1,3)=e1;

cpop(j,w).mach(s,n2,1)=c2;
cpop(j,w).mach(s,n2,2)=d2;
cpop(j,w).mach(s,n2,3)=e2;

end
end
end
end
end
end

```

Figure C.10: IRGGA main code (main loop 3).

```
    npop=[pop;cpop];
    [value,index]=sort([npop.fitT]);
    npop=npop(index,:);
    pop=npop(1:popsizе,:);

    [value2,index2]=min([pop.fitT]);
    if pop(index2,1).fitT<gpop(1,1).fitT
gpop=pop(index2);
    end
    BEST(iter)=gpop(1,1).fitT;

disp([' Iter = ' num2str(iter) ' BEST = ' num2str(BEST(iter))]);
if iter>2
if BEST(iter-1)==BEST(iter)
    tt=tt+1;
else
    tt=0;
end
if tt==20
    break
end
end
end

%% results

disp([' Best Fitness = ' num2str(gpop(1,1).fitT)])
disp([' Time = ' num2str(toc)])

end
```

Figure C.11: IRGGA main code (main loop 4).

```

function [fitness,afitness,cfitness]= cappafit( pop,process)

l=size(pop.mach,1);
m=size(pop.mach,2);

fit=zeros(1,1);

for v=1:l
    st=1;
    for i=1:m

        if pop.mach(v,i,1)>0

            fit(v,i)=process(pop.mach(v,i,1)).layert2(pop.mach(v,i,2));

        else
            break
        end
    end
end

afitness=fit;

fitt=cumsum(fit,2);
cfitness=fitt;

total=max(fitt(:));
fitness=total;

end

```

Figure C.12: IRGGA Fitness Calculation Code.

```

function np=create_new_popl(p,supsz,nvar)

x=p.vars;

j1=randi([1 nvar-1]);
j2=randi([j1+1 nvar]);
j3=randi([1 supsz-1]);
j4=randi([j3+1 supsz]);

nj1=x(:,j1);
nj2=x(:,j2);
nj3=x(j3,:);
nj4=x(j4,:);

x(:,j1)=nj2;
x(:,j2)=nj1;
x(j3,:)=nj3;
x(j4,:)=nj4;

np.vars=x;

end

```

Figure C.13: IRGGA Recombination Code.

Appendix D

ELBSO MATLAB Codes

This appendix shows the actual MATLAB codes of pseudocodes defined in algorithms 5, 10, 11, and 12.

```
clc
clear
close all
format shortG

%% parametrs setting
tic
popsize=1000;
maxiter=100;
newpopsize=2000;
nsubphase=6;
NRep=1;
lastsubpop=7;
SimL=10^5;
RR=round((popsize-lastsubpop)/(nsubphase-1));
mutrate=2;
M=5;
NL=5;
ne=randi([5,5],1,NL);
NE=5;
n=sum(ne);
UC=n-M+1;
st=zeros(NL,1);
se=zeros(NL,1);
st(1)=1;
se(1)=st(1)+ne(1)-1;
for i=2:NL
    st(i)=st(i-1)+ne(i-1);
    se(i)=st(i)+ne(i)-1;
end
allow=zeros(NL,NL);
```

Figure D.1: ELBSO main code (parameters assignment 1).

```

%% Initial population
pop= repmat([], popsize);
learn=zeros(popsize,NL+1);

for ps=1:popsize

    pop(ps).var=randperm(n);

    for i=1:n
        pop(ps).varj(i)=floor(pop(ps).var(i)/NL)+1;
        if rem(pop(ps).var(i),NL)==0
            pop(ps).varj(i)=pop(ps).varj(i)-1;
        end
    end

    for i=1:n-1
        for j=i+1:n

            if pop(ps).varj(i)==pop(ps).varj(j) && pop(ps).var(i)>pop(ps).var(j)

                swa=pop(ps).var(i);
                pop(ps).var(i)=pop(ps).var(j);
                pop(ps).var(j)=swa;

            end

        end
    end

    cc=randi([1,1],M,1);

    for i=1:n
        niw=allow(pop(ps).var(i));
        pop(ps).mach(niw,cc(niw))=pop(ps).var(i);
        pop(ps).machj(niw,cc(niw))=pop(ps).varj(i);
        cc(niw)=cc(niw)+1;
    end
end

```

Figure D.2: ELBSO Main Code (initial population 1).

```

[learn(ps,:)] = learningELBSO(n,pop(ps),st,se,NL,Due,U,W,NRep);
pop(ps).ESfit = ESFittELBSO5(learn(ps,1),learn(ps,2),learn(ps,3),learn(ps,4),learn(ps,5),learn(ps,6));
end

%writematrix(learn,'learnELBSO5.xls');

[~,index]=min([pop.ESfit]);
gpop=pop(index);

%% Neighbourhood search

npop=repmat([],newpopsize);
BEST=zeros(1,maxiter);
for iter=1:maxiter
for nps=1:2:newpopsize

rand1=randi([1,popsize]);
rand2=randi([1,popsize]);

[npop(nps).var,npop(nps).varj,npop(nps+1).var,npop(nps+1).varj] = crossfs(pop(rand1),pop(rand2),NL,NE,mtrate,st,se);

ccn=randi([1,1],M,1);
for i=1:n
niwn=allow(npop(nps).var(i));
npop(nps).mach(niwn,ccn(niwn))=npop(nps).var(i);
npop(nps).machj(niwn,ccn(niwn))=npop(nps).varj(i);
ccn(niwn)=ccn(niwn)+1;
end
end

```

Figure D.3: ELBSO Main Code (initial population 2).

```

[learn(ps,:)] = learningELBSO(n,pop(ps),st,se,NL,Due,U,W,NRep);
pop(ps).ESfit = ESFittELBSO5(learn(ps,1),learn(ps,2),learn(ps,3),learn(ps,4),learn(ps,5),learn(ps,6));
end

%writematrix(learn,'learnELBSO5.xls');

[~,index]=min([pop.ESfit]);
gpop=pop(index);

%% Neighbourhood search

npop=repmat([],newpopsize);
BEST=zeros(1,maxiter);
for iter=1:maxiter
for nps=1:2:newpopsize

rand1=randi([1,popsize]);
rand2=randi([1,popsize]);

[npop(nps).var,npop(nps).varj,npop(nps+1).var,npop(nps+1).varj] = crossfs(pop(rand1),pop(rand2),NL,NE,mtrate,st,se);

ccn=randi([1,1],M,1);
for i=1:n
niwn=allow(npop(nps).var(i));
npop(nps).mach(niwn,ccn(niwn))=npop(nps).var(i);
npop(nps).machj(niwn,ccn(niwn))=npop(nps).varj(i);
ccn(niwn)=ccn(niwn)+1;
end
end

```

Figure D.4: ELBSO Main Code (neighborhood search 1).

```

for i=1:n
    niwn=allow(npop(nps).var(i));
    npop(nps).mach(niwn,ccn(niwn))=npop(nps).var(i);
    npop(nps).machj(niwn,ccn(niwn))=npop(nps).varj(i);
    ccn(niwn)=ccn(niwn)+1;
end

ccn2=randi([1,1],M,1);

for i=1:n
    niwn2=allow(npop(nps+1).var(i));
    npop(nps+1).mach(niwn2,ccn2(niwn2))=npop(nps+1).var(i);
    npop(nps+1).machj(niwn2,ccn2(niwn2))=npop(nps+1).varj(i);
    ccn2(niwn2)=ccn2(niwn2)+1;
end

[nlearn(nps,:)] = learningELBSO(n,npop(nps),st,se,NL,Due,U,W,NRep);
npop(nps).ESfit = ESFitELBSO5(nlearn(nps,1),nlearn(nps,2),nlearn(nps,3),nlearn(nps,4),nlearn(nps,5),nlearn(nps,6));

[nlearn(nps+1,:)] = learningELBSO(n,npop(nps+1),st,se,NL,Due,U,W,NRep);
npop(nps+1).ESfit = ESFitELBSO5(nlearn(nps+1,1),nlearn(nps+1,2),nlearn(nps+1,3),nlearn(nps+1,4),nlearn(nps+1,5),nlearn(nps+1,6));

end

newpop=[pop,npop];
[value,index]=sort([newpop.ESfit]);
newpop=newpop(index);
pop=newpop(1:popsiz);

[value2,index2]=min([pop.ESfit]);
if pop(index2).ESfit<gpops(1).ESfit
end
BEST(iter)=gpops.ESfit;

disp([' Iter = ' num2str(iter) ' BEST = ' num2str(BEST(iter))]);

```

Figure D.5: ELBSO Main Code (neighborhood search 2).

```

%% Simulation Phase
NSsp=[1000,368,135,50,18,7];
Lr=[1000,2718,7389,20085,54598,10^5];
disp(' Second Phase Started');
for S=1:nsubphase
    for ps=1:NSsp(S)
        % NRep=round(SimL/NSsp);
        [pop(ps).fitT] = ELBSOFit5(pop(ps),n,NL,Due,st,se,Lr(S),U,W);
    end
    % if S<nsubphase
    %NSsp=NSsp-RR;
    %end
    [~,index]=min([pop.fitT]);
    gpops=pop(index);
    if S<nsubphase
    pop=pop(1:NSsp(S+1));
    end
    disp([' Iter = ' num2str(S) ' BEST = ' num2str(gpops.fitT)]);

toc

if toc>2000

[gpops(1).fitT] = ELBSOFit5(gpops(1),n,NL,Due,st,se,10^5,U,W);

disp([' Final Result....' ' BEST = ' num2str(gpops.fitT)]);
break
end
end

```

Figure D.6: ELBSO Main Code (simulation replications).

```

function [outputArg4] = ELBSOFit5(pop,n,NL,Due,st,se,NRep,U,W)

pop.machc=zeros(NL,NL);

fitness=zeros(1,NRep);

for uu=1:NRep

F=[normrnd(70,sqrt(140)) normrnd(80,sqrt(160)) normrnd(90,sqrt(180)) normrnd(50,sqrt(100)) normrnd(40,sqrt(80))
normrnd(80,sqrt(160)) normrnd(40,sqrt(80)) normrnd(50,sqrt(100)) normrnd(90,sqrt(180)) normrnd(40,sqrt(80))
normrnd(50,sqrt(100)) normrnd(40,sqrt(80)) normrnd(80,sqrt(160)) normrnd(60,sqrt(120)) normrnd(70,sqrt(140))
normrnd(60,sqrt(120)) normrnd(50,sqrt(100)) normrnd(60,sqrt(120)) normrnd(70,sqrt(140)) normrnd(80,sqrt(160))
normrnd(50,sqrt(100)) normrnd(50,sqrt(100)) normrnd(70,sqrt(140)) normrnd(40,sqrt(80)) normrnd(50,sqrt(100))];

NP1=F(1:NL,1:NL);
for i=1:NL
for j=1:NL
if NP1(i,j)<=0
NP1(i,j)=1;
end
end
end
row4=find(NP1>0);
NP=NP1(row4);

for i=1:n

[row,cal]=find(pop.mach==pop.var(i));
if cal==1

if pop.var(i)==st(pop.varj(i))

pop.machc(row,cal)=NP(pop.var(i));

else

[row2,cal2]=find(pop.mach==pop.var(i)-1);
pop.machc(row,cal)=pop.machc(row2,cal2)+NP(pop.var(i));

end
else

```

Figure D.7: Simulation Function in ELBSO.

```
    else
        if pop.var(i)==st(pop.varj(i))
            pop.machc(row,cal)=pop.machc(row,cal-1)+NP(pop.var(i));
        else
            [row2,cal2]=find(pop.mach==pop.var(i)-1);
            pop.machc(row,cal)=max(pop.machc(row2,cal2),pop.machc(row,cal-1))+NP(pop.var(i));
        end
    end
end
end

for i=1:NL
    [row3,cal3]=find(pop.mach==se(i));
    pop.fit1(i)=pop.machc(row3,cal3)-Due(i);
    if pop.fit1(i)>0
        pop.fit2(i)=pop.fit1(i)*U(i);
    else
        pop.fit2(i)=abs(pop.fit1(i))*W(i);
    end
end

pop.fitT=sum(pop.fit2(:));

fitness(1,uu)=pop.fitT;
```

Figure D.8: Simulation Function in ELBSO 2.

```
function [outputArg1] = learningELBSO(n,pop,st,se,NL,Due,U,W,NRep)

learn(1,1)=1;

for i=1:n

    [row35,cal35]=find(pop.mach==pop.var(i));

    if pop.var(i)==st(pop.varj(i))

        if cal35==1

            pop.ja(row35,cal35)=0;

        else

            pop.ja(row35,cal35)=pop.ja(row35,cal35-1)+1;

        end

    else

        [row36,cal36]=find(pop.mach==pop.var(i)-1);

        if cal35==1

            pop.ja(row35,cal35)=pop.ja(row36,cal36)+1;

        else

            pop.ja(row35,cal35)=pop.ja(row35,cal35-1)+pop.ja(row36,cal36)+2;

        end

    end

end

end
```

Figure D.9: Training Function in ELBSO.