EINDHOVEN
UNIVERSITY OF
TECHNOLOGY

Optimization of product flows in flexible manufacturing systems

Citation for published version (APA):
van Pinxten, J. H. H. (2018). *Optimization of product flows in flexible manufacturing systems*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Electrical Engineering]. Technische Universiteit Eindhoven.

Document status and date:
Published: 20/12/2018

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 04. Oct. 2023

OPTIMIZATION OF PRODUCT FLOWS IN FLEXIBLE MANUFACTURING SYSTEMS

JOOST VAN PINXTEN

Optimization of Product Flows in Flexible Manufacturing Systems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een
commissie aangewezen door het College voor
Promoties, in het openbaar te verdedigen op
donderdag 20 december 2018 om 13:30 uur

door

Joost Hendrikus Hubertus van Pinxten

geboren te Sint-Michielsgestel

Dit proefschrift is goedgekeurd door de promotor en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr.ir. J.H. Blom	
1e promotor:	prof.dr.ir. T. Basten	
co-promotor:	dr.ir. M.C.W. Geilen	
leden:	prof.dr. Sebastian Engell	(Technische Universität Dortmund)
	prof.dr. Frits Vaandrager	(Radboud Universiteit Nijmegen)
	prof.dr.ir. Jeroen Voeten	
	dr. Lou Somers	

Het onderzoek dat in dit proefschrift wordt beschreven is uitgevoerd
in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Optimization of Product Flows in Flexible Manufacturing Systems

J.H.H. van Pinxten

Advanced School for Computing and Imaging

This work was carried out in the ASCI graduate school.
ASCI dissertation series number 402.

NWO Applied and Engineering Sciences

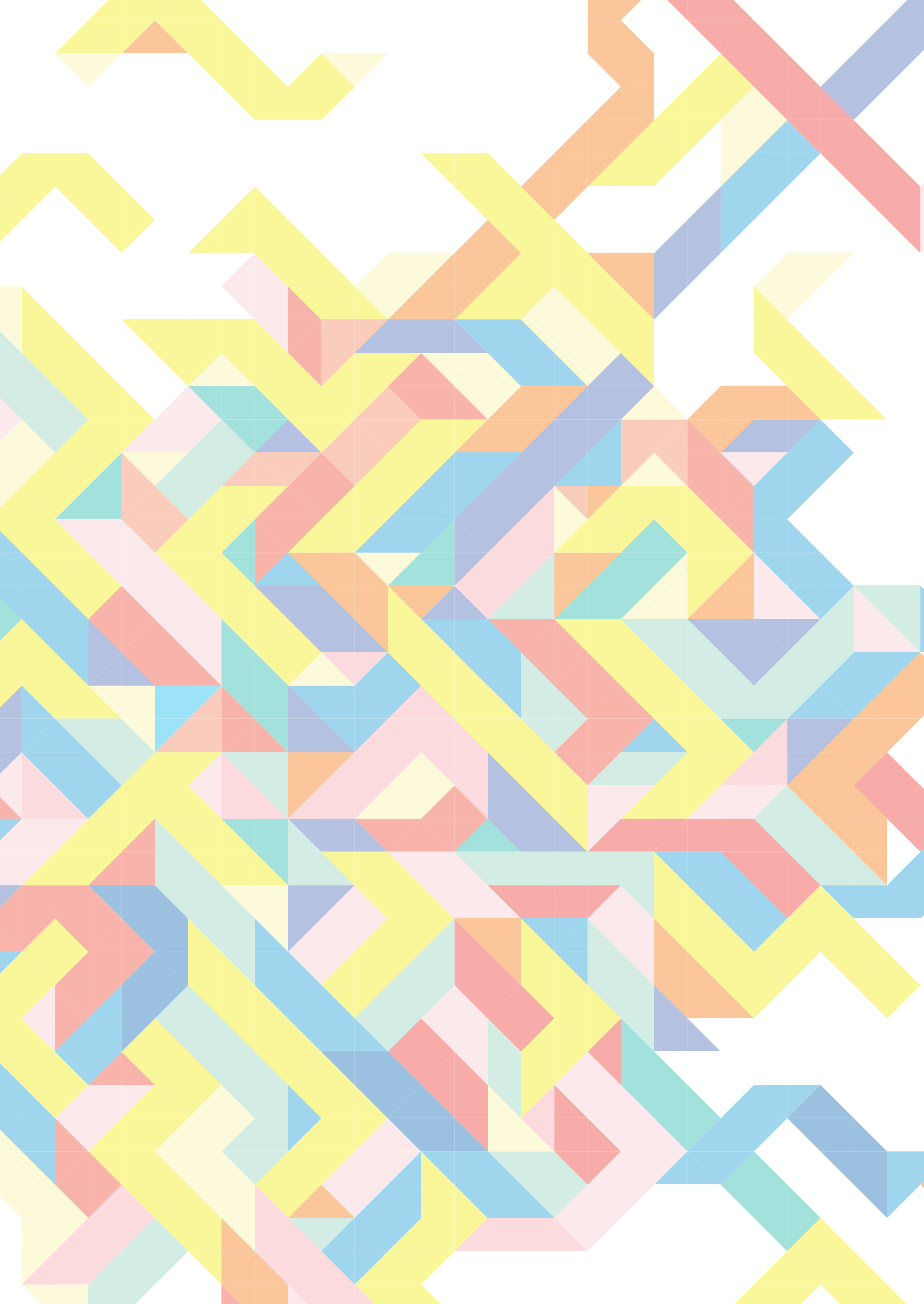
This work is part of the research program Integrated Scheduling and Control with project number 12693, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO).

Copyright © 2018, J.H.H. van Pinxten.
All rights reserved. Reproduction in part or in whole is prohibited without the written consent of the copyright owner.

Printed by Proefschriftmaken, The Netherlands

A catalogue record is available from the Eindhoven University of Technology Library. ISBN 978-94-6380-165-2

To my family and our purrito’s.



Summary

Optimization of Product Flows in Flexible Manufacturing Systems

Wafer scanners, robotic assembly lines, and industrial printers are examples of flexible manufacturing systems (FMSs) that manufacture customized products. Wafer scanners expose silicon wafers with UV-light in a process to create computer chips, and industrial printers transfer digital images onto sheets of paper. Each product may have different requirements or characteristics, which can lead to different timing constraints between operations. The productivity of FMSs is limited by cyber constraints, such as the computation time of an on-line scheduling algorithm, as well as physical constraints, such as limitations on the movement of a product. The interaction between components in FMSs as well as their productivity requirements are ever increasing. In this thesis, we provide techniques to perform on-line scheduling of the product flows of such cyber-physical FMSs, and to analyse the impact of system design parameters on total system performance.

FMSs can take many forms. We focus on scheduling algorithms for re-entrant FMSs, i.e., where the product re-enters one of the machines. Such a use case is found in industrial printing, where a sheet (i.e., the product) needs to be printed on both sides, by passing through an image transfer station twice. The media types and output order of the sheets are determined by the customer, and are not allowed to be changed. After processing a sheet, the image transfer station may need to be reconfigured before it can process the next sheet. The reconfiguration times typically depend on the differences in length, thickness, coating, and other media characteristics. The characteristics for a particular product, and therefore their timing requirements, are known only moments before the FMS must receive the processing instructions. Therefore, a scheduler must quickly calculate efficient timing instructions for each operation. The challenge is to merge the streams of unprinted sheets and re-entering printed sheets without collisions and without violating reconfiguration times, while optimizing the performance of the system. The sequence of the products at the image transfer station plays an important role in the performance of industrial printers, since it directly impacts the reconfiguration times between products. The performance is therefore determined by the interplay of user input, the physical parameters of the system components, and the scheduling decisions.

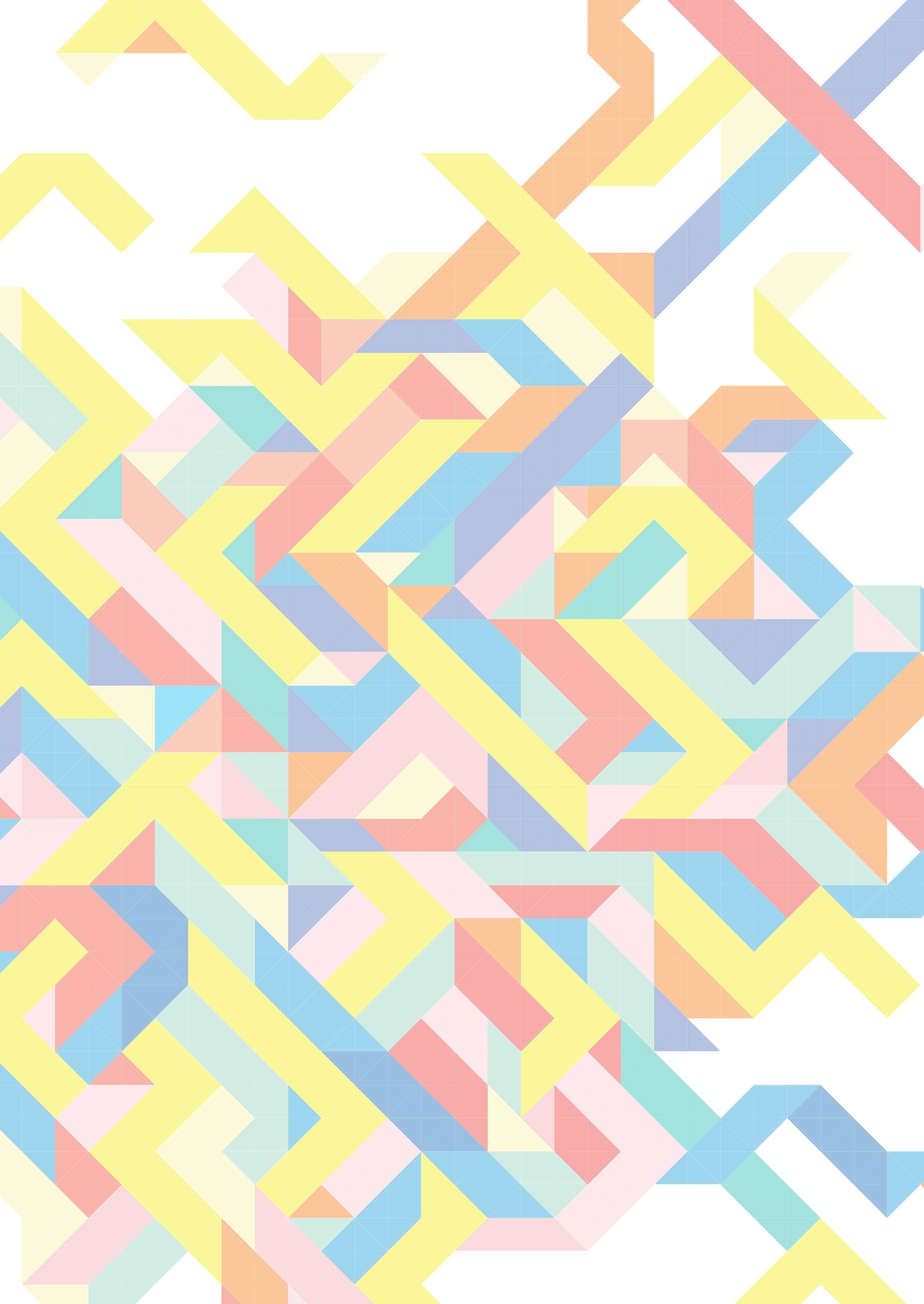
First, we present an efficient on-line sheet scheduling algorithm for industrial printers. The problem is modelled using a re-entrant flow shop with set-up times, due dates, and fixed output order. Given a particular sequence of products (i.e., a print job), the sheet scheduling algorithm determines a feasible and efficient sequence of operations at the re-entrant machine (i.e., the image transfer station). The timing instructions for each of the operations on the sheets can then be efficiently found. Our heuristic algorithm uses multi-objective optimization to explore intermediate sequencing options without performing backtracking. The heuristic leverages several properties of the scheduling problem, such as the output ordering of the sheets and limitations on the number of products that are partially processed. For the type of FMS considered, the output order of products within a batch (e.g., a print job is a batch of sheets) is fixed, but the batches still need to be ordered.

The second contribution is a scheduling heuristic that determines the orders of batches and selects system settings for each batch, such that Pareto-optimal productivity and quality trade-offs are found. Optimizing input order is particularly relevant for FMSs that may need reconfiguration between different products, such as industrial printers. The problem is modelled as a Multi-objective Generalized Traveling Salesman Problem (MOGTSP) and the ordering algorithm is an on-line, multi-objective heuristic that yields good results in less computation time than the state-of-the-art approaches for MOGTSP.

Third, we contribute a parametric critical path analysis for design-time analysis. During the design phase, a designer needs to create a physical layout that realizes a particular topology, i.e., a set of machines and their interrelations. Such a layout is constrained by limitations from many different domains. Constraints may, for example, come from minimal and maximal heating times, minimal segment lengths, or maximal velocities. The designer needs to choose a realization that simultaneously optimizes the total system behaviour and cost. Design parameters such as segment length or travelling velocity directly influence the total system behaviour, as e.g. processing a product becomes faster or slower. Our parametric critical path analysis identifies relationships between (physical) parameters and the productivity of an FMS. The scheduling graphs used in the sheet scheduling heuristic are annotated with affine expressions that relate timing constraints to design parameters. The analysis identifies bottlenecks in the productivity of an FMS in terms of affine expressions for different sets of parameter combinations. This analysis enables designers to identify how productivity depends on combinations of parameters.

As a fourth contribution, we present an algorithm that determines the impact of certain parameters on scheduling. Changes in the parameters may lead to selecting different scheduling choices, and therefore creating different schedules. We show that for some classes of schedulers it is possible to retrieve information about which scheduling decisions are chosen for different parameter combinations. This information can be obtained by performing symbolic scheduling. We can then apply the parametric critical path analysis to each of the different schedules. The total system productivity is then investigated in terms of these parameters, also considering the scheduling decisions.

This dissertation provides designers and engineers with several tools to analyse and optimize the system behaviour of FMSs. Trade-offs between productivity and quality can be decided at run-time by effectively ordering the received batches. The on-line sheet scheduler increases productivity due to more effective exploration of scheduling options for 2-re-entrant flow shops. The input optimization, parametric critical path analysis, and parametric scheduling characterization are applicable to systems having events with minimal and maximal time lags.



Contents

1 Introduction **1**
1.1 Manufacturing systems 4
1.2 Flexible manufacturing systems 6
1.3 Industrial printers 7
1.4 Flexible manufacturing system challenges 9
1.5 Flexible manufacturing system design trends 10
1.6 Research aims and contributions 12

2 Online Scheduling of Re-entrant Flexible Manufacturing Systems **15**
2.1 Scheduling product flows in re-entrant FMSs 17
2.2 Problem definition 19
2.3 Scheduling approach 23
2.4 Exploring trade-offs in scheduling decisions 31
2.5 Experimental evaluation 36
2.6 Related work 41
2.7 Conclusion 43

3 Multi-objective Optimization of Product Batches **45**
3.1 Introduction 47
3.2 Multi-Objective Generalized TSP 49
3.3 Related work 50
3.4 2PPLS for MO-GTSP 53
3.5 CPMH applied to MO-GTSP 57
3.6 Benchmarks and experimental evaluation 61
3.7 Conclusion 68

4 Parametric Critical Path Analysis **69**
4.1 Introduction 71
4.2 Event networks 72
4.3 Parametric critical path analysis 74
4.4 Case studies 85
4.5 Related work 94
4.6 Conclusion 97

5 Parametric Scheduler Characterization **98**
5.1 Symbolic scheduling 100
5.2 Running example 101
5.3 Parametrized schedulers 102
5.4 Exploring parameter combinations 109
5.5 Experimental evaluation 113
5.6 Related work 121
5.7 Conclusion 122

6 Conclusions and Future Work **124**
6.1 Conclusions 125
6.2 Future work 127

Acronyms **129**

Appendices **133**

A Bellman-Ford-Moore longest-path algorithm **134**

B Proofs for convex polyhedra **140**

Bibliography **147**

Samenvatting **160**

Acknowledgements **164**

Curriculum Vitae **169**

1

Introduction

Through the 19th and 20th century, the majority of the global work force shifted from performing agricultural and manual labour to manufacturing goods in factories and assembly lines [26, 124]. Research on factories and manufacturing lines greatly improved the efficiency and productivity of manufacturing resources. Increased efficiency of goods manufacturing through the late 20th century and early 21st century allowed the work force to shift from goods manufacturing to providing services instead [124]. Many goods are now produced using partially or fully automated manufacturing systems [24, 85]. In this thesis, we contribute research into the optimization and analysis of flexible manufacturing systems (FMSs), i.e., manufacturing lines that can easily adapt to produce customer-specific products. Such manufacturing lines make use of robots and automated processing stations to execute operations on (raw) input materials to produce goods.

Manufacturing systems that use technologies such as robots and automated processing stations benefit from technological improvements. A technological improvement enables a higher quality of the final product, extending the range of products, and/or improving the productivity of the machine. The productivity and performance of automated manufacturing systems continues to improve by taking advantage of technological improvements in areas such as computers, robots, and artificial intelligence [24]. Global efforts to reduce time-to-market and production costs are leading to ever higher demands on the performance of machines. To cope with such demands, however, manufacturing systems must operate very close to physical limitations. In addition, more coordination is needed between different components, typically realized in the form of embedded systems. As a result, the complexity of manufacturing systems has been increasing together with the quality, flexibility and productivity requirements.

The productivity of manufacturing lines has soared in the 19th and 20th century due to investments in technological developments and research. The current challenge is to develop systems that are as productive, but are more flexible in nature. Increased flexibility makes it possible to produce ‘one-off’ customized products. Requirements on such products are often known only moments before production. The processing and set-up times of the processing stations are derived from the product requirements. More dynamic and complex product flows lead to more interaction between products, and create a necessity to perform online scheduling. Designing more flexible systems in such a way that their productivity is retained is more complicated due to the interactions that occur between products. We first illustrate the trends and challenges in manufacturing systems, before the contributions are outlined.

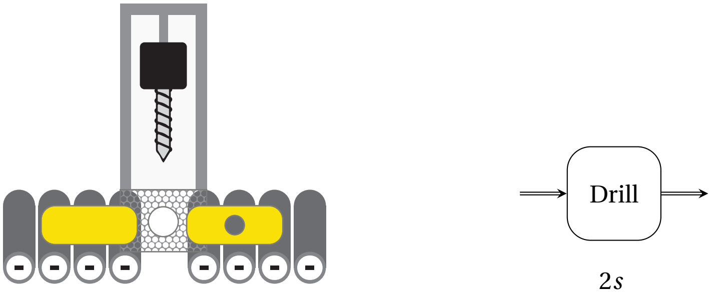
1.1 Manufacturing systems

The simplest manufacturing system consists of a single processing machine with associated transport mechanisms. A drilling system with roller conveyors is illustrated as an example in Figure 1.1a. Its associated product flow is shown in Figure 1.1b. If a hypothetical drilling station can process a product in two seconds, then the maximum throughput is limited to half a product per second, as shown in Figure 1.3a. Suppose a faster drill is available but requires that the products are drilled under stricter conditions, e.g. at a particular temperature. The manufacturing system's topology (i.e., machines and their connections) can be extended with pre-processing and post-processing steps (Figure 1.2a and 1.2b) such that the product meets the required conditions for the new drilling station. If both the pre-processing and post-processing steps cost one second, and all steps can be performed in parallel with each other, then the system is capable of delivering (in steady state) one product per second. However, the time to manufacture one product (i.e., the product flow time) is increased from two to three seconds. The pipelining effect is visualized in Figure 1.3b.

This type of extension is typical for systems where a particular machine is the bottleneck. The productivity is increased, but so are the footprint and product flow time of the system. Therefore, a design-time trade-off between system characteristics needs to be made.

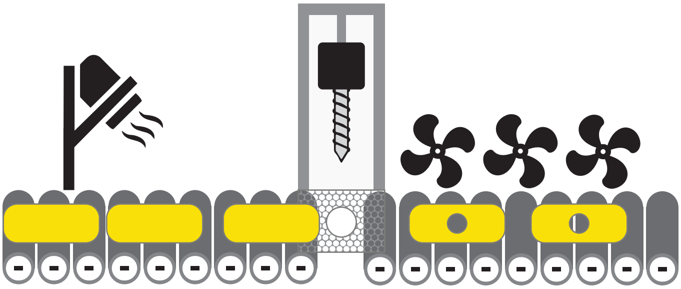
A similar change has occurred in microprocessor design several decades ago. Increasing the clock frequency means that more work can be performed per time unit. However, the clock frequency of a microprocessor is limited by the longest delay that can occur between two clock edges. This delay is necessary to ensure that the transistors have produced correct output values given their new inputs. The switching delay of individual transistors has a fundamental physical limit, which imposes a lower bound on the delay. System performance could only be improved by parallelizing the execution of instructions.

During the design of complex systems like FMSs, trade-offs are explored and one of the trade-offs that satisfies particular goals is selected. For example, the latency, throughput, footprint, and power consumption of a system are all impacted by these design decisions. Creating a processor that has a high throughput will typically also yield a high power consumption. The same is true for manufacturing systems. High productivity may lead to a higher latency and cost due to pipelining and parallelization, and a high product quality may need more processing time reducing productivity. This thesis provides new computer-aided design, analysis, and optimization techniques for trade-offs in flexible manufacturing systems.

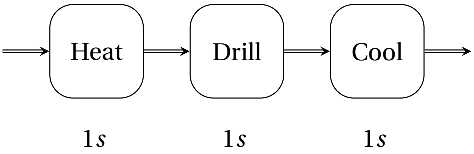


(a) A simple drilling manufacturing line. (b) Abstract representation of a simple manufacturing line.

Figure 1.1: A simple drilling system.



(a) A drilling manufacturing line with pre-processing (left, heating) and post-processing (right, cooling).



(b) Abstract representation of the extended manufacturing system's linear topology.

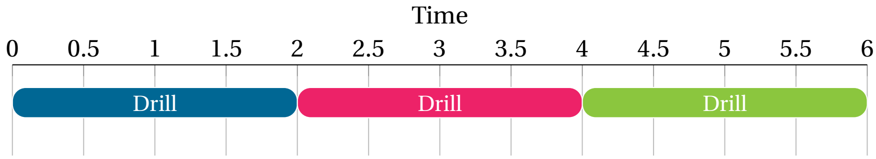
Figure 1.2: An extended drilling system.

1.2 Flexible manufacturing systems

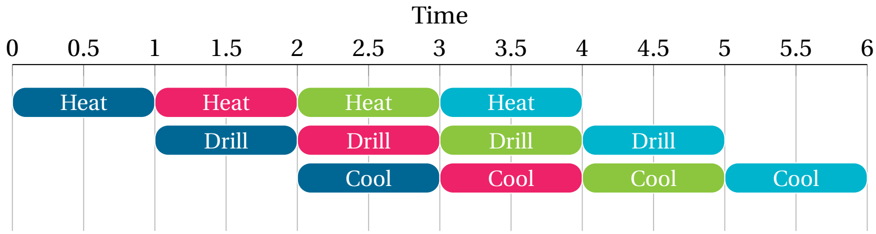
FMSs are systems consisting of interacting machines that can manufacture a range of products. These machines perform operations according to customer specifications such that specific (customized) products are manufactured from unprocessed input material. Automotive assembly lines, industrial printers, wafer scanners, and automated robotic packing systems are examples of FMSs.

Machines can range from conveyor belts, pinches, processing stations such as a print head or a lithography exposure stage, to robot arms drilling holes at the right location, picking up components, and finally assembling them into a finished product. In this thesis, we focus on FMSs that have machine and/or operation flexibility; i.e., the ability to perform the same operation on different types of unprocessed components without manual reconfiguration of the machine, or to apply variations of operations on the same type of unprocessed component.

Most products are manufactured according to a particular product flow, i.e., a certain sequence of specific operations needs to be performed to manufacture a product. For industrial printers, the actions in this product flow are loading, printing, heating, cooling and unloading either a simplex or a duplex sheet. A simplex sheet is printed only on one side, while a duplex sheet is printed on both sides. For wafer scanners, a wafer must be exposed to light in several steps according to a specific sequence of masks. Either type of FMS leaves no freedom in the order of operations for a *single product.* However, multiple products are simultaneously inside the pipeline of the machine, in different stages of their product flow. Each product may require a different operating state of the machines in the system. In industrial printers, for example, the print head height must change between thicker and



(a) Activity trace for simple drill system.



(b) Activity trace for extended drill system.

Figure 1.3: Activity traces for manufacturing lines.

thinner sheets. The machines require time to change from one state to another, which gives rise to sequence-dependent machine reconfiguration times.

The requirements for the products may not be known long in advance of manufacturing. For this reason, a scheduler must be able to make decisions for a horizon that is continuously updated. The computation of these decisions must not take too long, or it starts to hinder the productivity of the system. For many FMSs an online scheduler is required to optimize the product flow of different products such that each machine in the FMS is instructed in time when to perform which operation on which product.

1.3 Industrial printers

The research in this thesis focuses on optimizing product flows in flexible manufacturing systems. This research has been carried out in collaboration with Océ

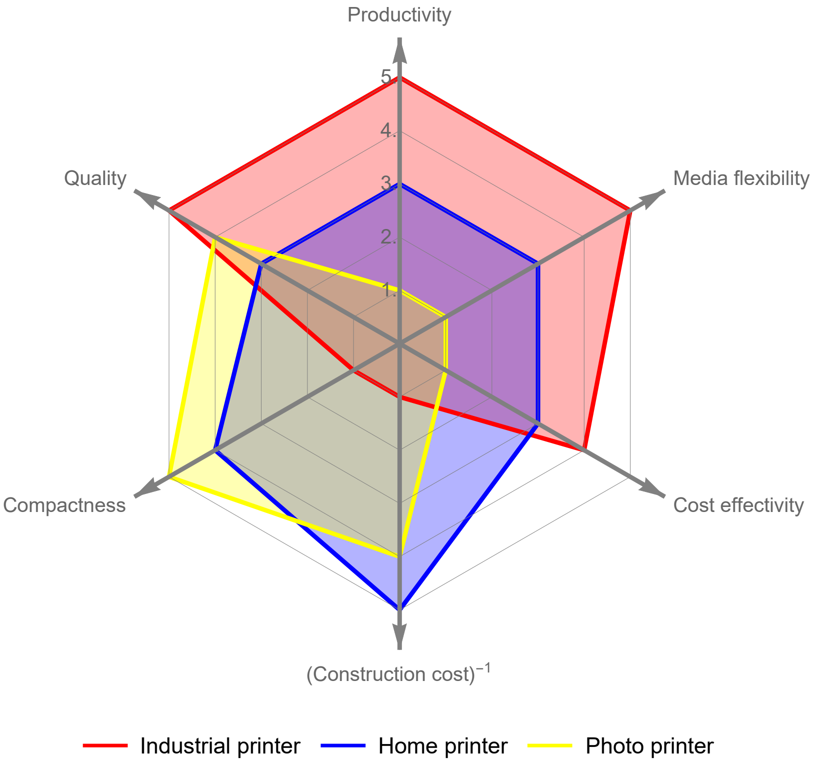


Figure 1.4: Radar chart comparing three types of printers; higher scores are better.

Technologies [91], a company that mainly creates printing and work flow management solutions. Océ Technologies and Canon, its parent company, provide a wide range of printers and printing solutions, ranging from printers for homes and small offices to multi-functional copiers, wide-format printers, to industrial printers. Each of these printers targets a specific market segment and has unique features to fit within that segment. Different market segments have different requirements on productivity (i.e., pages per minute), print quality, construction costs, running costs (i.e., cost effectiveness), media flexibility, and system size (i.e., compactness). An example qualitative comparison between three printer types is shown in Figure 1.4. Even though each of these characteristics has improved over the years, it is not possible to create a single small, productive, cheap printer that can print on all media types and sizes at a high quality. An industrial printer has high productivity and high print quality but also has a large footprint and high construction cost; the machines that perform the actions are large and expensive. In other words, a single printer cannot simultaneously dominate on all six characteristics.

The industrial cut-sheet printers were the main driving case for the research in this thesis. Figure 1.5 shows an example of an industrial cut-sheet printer, the Océ VarioPrint i300. The main challenge for this printer is to maintain a high throughput for a wide range of media, while producing a high quality print. This printer contains a long paper path that transports sheets through several machines that perform operations, aiming at a throughput of transferring 300 images per minute to sheets, over a wide range of media.

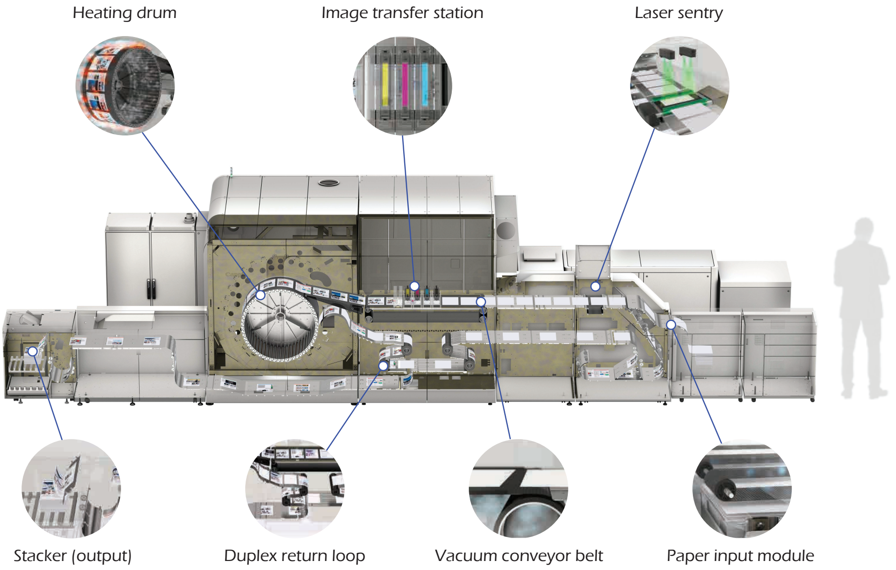


Figure 1.5: A cross section of the paper path of the Varioprint i300.

The printer consists of a paper input module (PIM), which separates sheets and loads them into the metal track of the paper path. The sheets are transported by pinches and conveyor belts. Before a sheet gets printed, a laser sentry scans the sheet to determine whether it can be printed. If the laser sentry measures an acceptable deformation of the sheet, the sheet continues to the image transfer station (ITS), otherwise it rejects the sheet. The conveyor belt under the ITS moves at a fixed speed, so that the frequency of the nozzles releasing ink drops is matched to the required dots-per-inch (DPI) of the requested image. The ink is water-based and therefore sheets need to be dried after an image is transferred to it. A heating drum is used to evaporate the water from the sheets while at the same time transporting them to the cooling section. After a sheet has cooled down, it either exits the printer through the paper finishing module (FIN), or it returns through the duplex return loop (DL) and the turn track (TT) to the merge point (MP). It is then transported to the ITS to be printed again. The sheet is turned in the turn station of the duplex return loop. The paper path shown in Figure 1.5 is also visualized in a more abstract way in Figure 1.6.

1.4 Flexible manufacturing system challenges

Processing and transporting individual products becomes harder as the performance pushes the execution towards the boundaries of the laws of physics. For wafer scanners, the quality is limited by the feature size that can be achieved, while the productivity is limited by the time it takes to achieve accurate positioning. For printing, the quality depends heavily on the correct interaction of the ink or toner with the sheets. Strong mono-disciplinary knowledge is required to push the performance of the individual machines, i.e., the processing stations and transport mechanisms, to their physical boundaries.

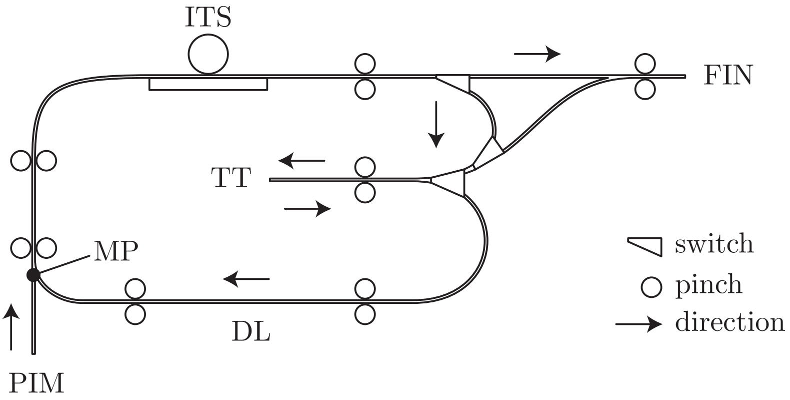


Figure 1.6: A schematic representation of the paper path of an industrial printer [111].

Besides the laws of physics, the economical impact of building and running the machine must also be taken into account. The cost of a machine should be as low as possible to improve the return-on-investment of such devices. Topologies with minimal use of resources and maximal productivity are preferred. The total system productivity can be affected when the number of processing stations is altered. A change in topology may introduce additional complexity in the decision making process. For example, the topology shown in Figure 1.6 is significantly different from a linear topology, as it allows a product to re-enter the ITS for re-processing, at the cost of additional travelling time. Re-entrant topologies require schedulers that can interleave product flows in an effective manner [121]. In an industrial printer, for example, the stream of sheets that has not been processed at all (i.e., unprocessed sheets) must be merged productively with the stream of sheets that has been printed once (re-entrant sheets).

Each machine is tuned such that a desirable trade-off between quality and productivity is achieved. The total system behaviour, and therefore its performance, does not depend only on how well each individual machine is optimized, but also on how well the timing constraints of several machines combine. The interaction between machines needs to be managed by the designers, and by a supervisory controller for online decision making. It is a major challenge in high-end FMSs to schedule the product flows such that the system productivity is maximized [122]. If the decision making process takes too much time, it defeats the purpose of optimizing the system, as it will itself become a performance bottleneck.

Multi-disciplinary models and knowledge are required to understand the complex interactions between the machines in an FMS. Each product flow can impose different requirements on the processing stations, which may need to be reconfigured between different products to achieve the required operation. It is difficult to pinpoint the performance bottlenecks in a system where many product flows interact with each other due to reconfiguration requirements. Product flow optimization becomes more complex and time consuming due to more and more re-use of machines and higher demands on productivity.

1.5 Flexible manufacturing system design trends

Virtual prototyping has become more prominent in the design of FMSs [60, 61, 86, 88]. The cost and complexity of building and maintaining multiple physical prototypes has driven the creation of visualizations and associated testing tools such as software-in-the-loop and hardware-in-the-loop testing environments [57, 100]. Early validation of design decisions can then be tested by simulating the design [111]. The effects of the real-time embedded platform [10] and the mechatronics transporting the products can be explored without building a physical prototype. At a different level of abstraction, some parts of the FMS' environment can be modelled, such as the FMS operator's instructions, the room temperature, or raw product conditions, to show the impact on the system's behaviour.

One of the major trends in the design of FMS is the increased use of Model-Based Design tools [108, 111]. The effort to create specific tooling that allows early investigation of particular aspects of a system has initially given rise to simulations that are dedicated to a single prototype [8]. Such dedicated simulators are now being replaced by configurable simulators. These configurable simulators can then be re-used for new versions of products in the same product line. Model-Based Design languages such as UML and SysML [108] are used to model, document and generate design artifacts and system components. More recently, Domain-Specific Languages and their corresponding editors (such as MPS [1], MetaEdit+ [2], and Eclipse [3]), are for example used to create configurations for early prototypes, and generate specialized pieces of code. Although initial investment of creation and adoption of such models can be relatively high, it yields flexibility in developing products and faster understanding of the core issues when correctly applied.

The models, like the simulators, are becoming more and more reusable and adaptable to new situations. This leads to better re-use of previously engineered solutions and less re-work [113, 114]. At the same time, the models are synchronised with each other to catch inconsistencies early on in the design process. Models that generate run-time components can be integrated in the design process to simulate the behaviour of the FMS in a more accurate manner.

Designers need to be more and more aware of trade-offs in their system. They need to use manual or automated tools to explore alternatives that lead to efficient designs, i.e., design-space exploration (DSE). A system model may be composed of several modular models, each of which has different performance characteristics. The total system may then be simulated to find the trade-offs in performance and cost. Different types of models are necessary to describe different aspects of the system; e.g., continuous time controllers and finite-element models are used to model the transportation of sheets in the printer, while discrete-event models are used to determine the relevant scheduling events in a printer.

The impact of a technological or design change can be difficult to assess from a total system perspective. In this thesis, we investigate whether the productivity of FMSs can be analysed automatically. This allows answering questions such as: does reducing the size of a machine also reduce the efficiency of a scheduler, or can the selected scheduler benefit from it? What is the difference in system productivity if one of its components is slowed down or sped up? Such questions are difficult to answer, as it is already difficult for a single person to maintain an overview of the system. A designer should have the tools to make informed decisions about such complex systems. We draw inspiration and build upon the ideas from the many researchers that have already disseminated on such topics, originating from integrated circuit design, project planning, and operations research perspective.

---

[1] https://www.jetbrains.com/MPS

[2] https://www.metacase.com/mep/

[3] https://www.eclipse.org/

1.6 Research aims and contributions

The problem statement driving the research aims of this thesis is:

Constraints from many domains impact the productivity of product flows in an FMS. *The goal is to automatically optimize and evaluate the productivity of re-entrant* FMSs *and to trace productivity bottlenecks back to their original domain.*

The research presented in this thesis aims to improve the productivity of FMSs by (1) performing online scheduling of re-entrant product flows, (2) relating performance of individual components to the system performance. In particular, we focus on online schedulers for re-entrant FMSs and on the relation between productivity, cost, and design-time structural parameters.

In this thesis, we contribute online schedulers and productivity analysis techniques for FMSs. In Chapter 2, we show a prototype scheduler for industrial printers that optimizes sheet sequences online by selecting between re-entering sheets and unprocessed sheets, while guaranteeing a given output order.

Optimizing the product flows on-line is non-trivial due to the wide range of products that the system supports, leading to a wide range of set-up times at different stations. Effectively interleaving the product flows is required to effectively utilize the system. However, re-entering products have processing deadlines, determined by the physical layout of the FMS. Locally productive choices may lead to infeasibility of future interleaving options, forcing the system to wait. Leaving too much room for future choices under-utilizes the system as well. The combination of set-up times and deadlines makes this a particularly challenging problem.

Our approach uses a graph-based algorithm to interleave product flows such that the productivity of the printer is optimized while meeting all timing requirements. This work is based on:

1. Joost van Pinxten, Umar Waqas, Marc Geilen, Twan Basten, and Lou Somers. Online scheduling of 2-re-entrant flexible manufacturing systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):20, October 2017

The scheduler in Chapter 2 does not change the system and product settings, as these have been pre-determined by the operator. An FMS can operate in multiple modes, leading to differences in productivity and quality. The performance of the scheduler can be non-trivial to predict, due to the flexibility of the input jobs. The transitions between jobs may lead to productivity gains or penalties. Optimizing sequences of batches can lead to higher system utilization.

In Chapter 3 we introduce a technique to explore productivity-quality trade-offs by re-ordering batches of products and selecting product and system settings. An FMS operator can then choose the product and system settings that best fit the current situation. This work is based on:

2. Joost van Pinxten, Marc Geilen, Twan Basten, Umar Waqas, and Lou Somers. Online heuristic for the multi-objective generalized traveling salesman problem. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*, pages 822–825. IEEE, March 2016

It is useful for an FMS designer to understand what the productivity bottlenecks are before the FMS is realized. However, a designer needs to take into account aspects from the control domain, the mechanical and physical domain, and the scheduler domain. We present a technique to characterize the performance of a system in terms of several parameters, such as machine length, travelling speed, or conditioning time. The first step towards such parametric analysis of a system is presented in Chapter 4 where we introduce a bottleneck analysis that can evaluate how sensitive a given schedule is for changes in the structural parameters of an FMS. The result is a method that identifies relations between the system bottlenecks and the parameters from multiple domains. This work is based on:

3. Joost van Pinxten, Marc Geilen, Martijn Hendriks, and Twan Basten. Parametric critical path analysis for event networks with minimal and maximal time lags. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 2697 – 2708, October 2018

As the parameter values change, the scheduler may take different decisions, and therefore create different dependencies between operations in the product flows. The system productivity is heavily influenced by the decisions that a scheduler takes. It is, however, not trivial to determine the impact of the parameters from multiple domains on the available scheduling choices and decisions.

We apply symbolic scheduling to keep track of relations between parameters and scheduling decisions. The resulting information is used in a new approach that finds which schedules are generated for which parameter combinations. This technique (Chapter 5) shows that information can be extracted from such a symbolic scheduler to relatively quickly cover all parameter combinations. This work is based on:

4. Joost van Pinxten, Marc Geilen, and Twan Basten. Characterising parametric schedulers. [submitted]

The combination of the scheduler characterization and parametric critical path analysis allows exploration of the behaviour of schedulers and their associated system productivity. The relationships between the contributions and the context is shown in Figure 1.7. Domain models capture the timing constraints between system events based on the information from the multi-domain design artefacts. These artefacts are parametrized, and can be used to generate code that represents the timing constraint models. These timing constraint models can be queried by the sheet scheduler during run-time.

Chapter 6 gives the final conclusions and directions for future work. Figure 1.7 shows an example iterative design method. The relationships between tools are

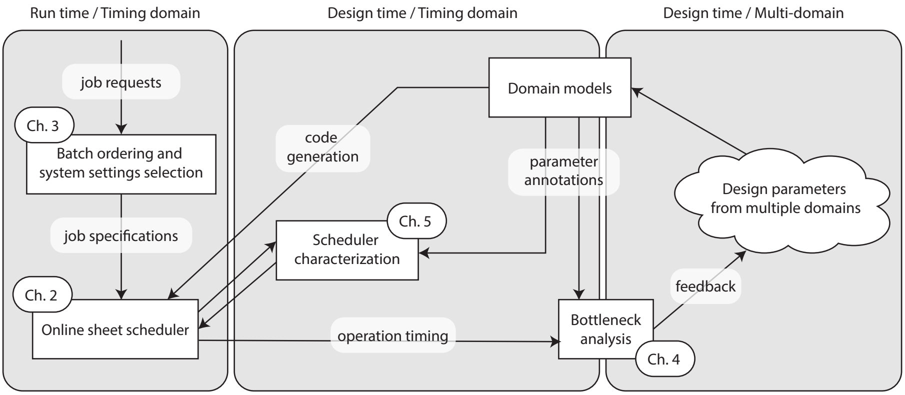


Figure 1.7: Overview of the relation between domains, run-time and design-time artefacts and the chapters of this thesis.

denoted by arrows, annotated by the information that is communicated. The tools in the flow are annotated by the respective chapters in which they are discussed.

2

Online Scheduling of Re-entrant Flexible Manufacturing Systems

Flexible manufacturing systems (FMSs) require online scheduling as manufacturing requests arrive on the fly [19, 29, 122]: the timing requirements of a particular operation are only known seconds before the scheduling decision needs to be executed. We focus on re-entrant product flows; i.e., where a product is processed more than once by the same machine.

The challenge for a scheduler is to effectively interleave re-entering products with unprocessed products. A scheduling algorithm needs to compute such decisions in an online fashion. An online scheduler must instruct the FMS, such as an industrial printer, in time. If it takes too long to compute, it may become a bottleneck for productivity. It is therefore essential to find the *best possible* schedule of interleaved product flows *in time*.

In this chapter, we contribute such an online scheduler, particularly targeted at product flows with a single re-entrant machine. Even if the timing constraints between operations of a single product flow are fixed, there is considerable room to optimize the interleaving of product flows. The product flexibility leads to product-dependent set-up times, while the minimum traveling speed leads to due dates on a product re-entering the system. Greedily filling the re-entrant loop leads to over-filling it. This would mean that the loop needs to be completely emptied often when there are no interleaving options available for new products to enter the FMS. Leaving too much room lowers the productivity too much. We contribute an online scheduling algorithm that automatically finds a good balance between filling the loop, and leaving space for future products.

2.1 Scheduling product flows in re-entrant FMSs

Several different aspects are used to classify FMSs [17, 105]. In this thesis, we focus mainly on FMSs with *operational/machine flexibility*, i.e., the flexibility to create different products with the same machine. For these types of systems, online scheduling for the product flows is necessary as the operation requirements of a

---

The content of this chapter is an adaptation of the work published in CODES-ISSS 2017 [120].

product are known only moments in advance. In this chapter, we focus on re-entrant FMSs, which are systems that need to process products multiple times on the same processing station. Typically, the time between two passes of the same product is considerably higher than the processing time, due to the conditioning steps required to achieve the proper quality. The flow of the products is fixed, the products need to be produced at the output in a given order, and reordering at the output is not possible.

A particular instance of such an FMS is an industrial printer [122]. The type of industrial printer that we consider in this thesis prints simplex and duplex sheets that need to be processed once respectively twice by the same print head. The print head is the most costly component of the system, and it is therefore not duplicated. Industrial printers are FMSs because they process many different kinds of media, and the image on each print is unique. This leads to varying processing and set-up times, which need to be taken into account during online scheduling. A print head can print hundreds of pages per minute. The instructions for how the sheet should flow through the printer need to be provided well within a second.

Other examples of re-entrant FMSs include lacquer [11] and mirror production [21], and lithography machines [98]. Several production steps are required to produce the final product. The products are transported through the manufacturing system to undergo several chemical and physical processes. These processes need a certain minimum and maximum time between two subsequent processing steps on the same product/material. The FMSs under consideration have significant, but limited buffers for re-entrant products.

In industrial printers the re-entrancy buffer is the return loop within the printer that prepares sheets for printing the opposite side of the sheet. Each sheet has a minimum and maximum travel time determined by physical constraints on heating, cooling, length, acceleration and velocity. Reconfiguration times are significant and have a strong impact on system productivity.

The scheduling freedom in the considered FMSs consists of deciding the order in which new and returning products are processed on the same machine. A schedule needs to meet timing requirements that consist of the travel times between processing stations, and reconfiguration times of the station between processing different products. Once a processing order has been chosen, the most efficient timing can be efficiently derived from the timing requirements [122] using the Bellman-Ford-Moore (BFM) longest-path algorithm [12, 41]. This computation method is described in Appendix A.

The goal of the scheduling algorithm is to find a feasible and maximally productive schedule; the throughput of the system should be as high as possible. Moreover, the timing of the operations for a product need to be provided while the schedule is not completed yet. Operation timing instructions should be computed within the time budget available for scheduling a product. However, assessing the impact of sequencing options is not trivial. Interaction between different kinds of products flowing through the FMS makes this problem complex to tackle.

To address this scheduling problem, we model the FMS as a re-entrant flow shop. This chapter improves upon the greedy scheduling mechanism of the Heuristic Constraint-based Scheduler (HCS) [122] for online performance by eliminating many infeasible scheduling options and by applying a generic multi-dimensional meta-heuristic. First, we introduce the Bounded HCS (BHCS) which improves the greedy scheduling mechanism of HCS by deriving a scheduling horizon from limitations imposed by the re-entrancy buffer. Defining this scheduling horizon allows BHCS to (a) efficiently remove many a priori infeasible sequencing options, (b) calculate timing information only within the scheduling horizon. This leads to much faster evaluation of sequencing options. This speed up allows us to increase the search space within the time budget. We apply a multi-dimensional meta-heuristic, similar to the Compositional Pareto-algebraic Heuristic (CPH) [106], to create the Multi-dimensional BHCS (MD-BHCS). MD-BHCS simultaneously explores multiple sequencing options, without resorting to backtracking. This leads to higher quality schedules.

MD-BHCS achieves a better worst-case time complexity, a better average running time and schedules with shorter makespans, on average, compared to the original HCS algorithm. The algorithms presented in this chapter focus on 2-re-entrant flow shops with bounded travel times and reconfiguration times that are significantly larger than the processing times. Such instances are found in the industrial printing problem, in which context we evaluate our solutions.

The next section precisely defines the problem we address. Section 2.3 presents the BHCS heuristic. Section 2.4 combines BHCS with a multi-dimensional meta-heuristic. Section 2.5 contains the experimental evaluation. Section 2.6 discusses related work. In Section 2.7 we conclude this chapter.

2.2 Problem definition

We first define re-entrant flow shops and introduce terminology and notation. We then relate scheduling of the FMS to re-entrant flow shop optimization.

2.2.1 Re-entrant flow shops

Manufacturing lines are commonly modelled through a variant of a job shop scheduling problem, such as a flow shop. In a job shop, each of a set of jobs needs to be allocated to a resource, while minimizing the total execution time. A flow shop is a variant of a job shop for which each job must be processed according to a given flow of operations.

In this chapter, we use re-entrant flow shops with sequence-dependent set-up times and relative due dates to model re-entrant FMSs. A *re-entrant flow shop* is a flow shop where a sequence $J = \langle j_1, \ldots, j_n \rangle$ of n jobs are processed by a set M of machines. Every job $j \in J$ is a sequence of r operations $\langle o_{j,1}, \ldots, o_{j,r} \rangle$ that needs to be executed. We use $o_{j,i}$ to denote the i-th operation of the job j. The operations O of the flow shop are the union of the operations of its jobs: $O =$

$\{o_{j,k} | j \in J, k \in \{1,\ldots,r\}\}$. A re-entrant flow vector $\phi = \langle \mu_1,\ldots,\mu_r \rangle$ specifies that the i-th operation $o_{j,i}$ of job j must be executed on machine $\phi(i) = \mu_i \in M$. A re-entrant flow vector contains at least one machine that is visited more than once. The first and second time an operation for a job is executed on machine μ are respectively called the first and second pass of that job j on machine μ, and so forth. This is denoted respectively by $o_{j,\mu,1}$, $o_{j,\mu,2}$, and $o_{j,\mu,i}$. We say that $o_{j,\mu,x}$ is a *higher pass* or *lower pass* operation than $o_{k,\mu,y}$ when $x > y$ or $x < y$ respectively.

The processing times of operations are denoted by a function $P : O \to \mathbb{R}_{>0}$. The processing times indicate the time required to finish an operation once it has started. Once started, operations must continue their execution until completion. The set-up times are reconfiguration and travelling times that occur regardless of the sequence of operations on a machine. The set-up times are defined as a partial function $S : O \times O \mapsto \mathbb{R}_{\geq 0}$, where $S(o_x, o_y)$ is the minimal time needed between the completion of o_x and the beginning of o_y. If $S(o_x, o_y)$ is undefined, then S does not impose any restriction between the completion of o_x and the beginning of o_y. We assume that the very first operation on each machine does not require reconfiguration times. The sequence-dependent set-up times occur only when two operations are executed one immediately after the other on the same machine and are defined as a partial function $SS : O \times O \mapsto \mathbb{R}_{\geq 0}$, where $SS(o_x, o_y)$ is the minimal time from completion of o_x to beginning of o_y which is needed to reconfigure a machine. The relative due dates impose a maximum time delay between the begin times of two operations, and are denoted by a partial function $D : O \times O \mapsto \mathbb{R}_{>0}$. If $D(o_x, o_y)$ is undefined, then there is no restriction on the maximum time from the beginning of o_x to that of o_y. The algorithms in this chapter assume due dates only occur within jobs, i.e., from $o_{x,i}$ to $o_{x,j}$ with $i < j$.

Definition 1 (schedule). *A schedule $B : O \to \mathbb{R}_{\geq 0}$ for a flow shop describes begin times for all operations; $C(o) = B(o) + P(o)$ denotes the time that operations complete. The following constraints need to be satisfied:*

- *Every machine $\mu \in M$ can execute at most one operation at a time. That is:*

$$\begin{aligned}\left(\forall o_{x,i}, o_{y,j} \in O\right) \big[\left(\phi(i) = \phi(j) \wedge o_{x,i} \neq o_{y,j}\right) \\ \implies \left(C(o_{x,i}) \leq B(o_{y,j}) \vee C(o_{y,j}) \leq B(o_{x,i})\right)\big]\end{aligned}$$

- *The sequence-independent set-up times of S between each pair of operations o_x and o_y in the domain of S must be met, if $S(o_x, o_y)$ is defined, then $B(o_y) \geq C(o_x) + S(o_x, o_y)$.*

- *If no other operation begins between the begin time of $o_{x,i}$ and $o_{y,j}$ on the same machine, then the sequence-dependent set-up time between them must hold:*

$$\begin{aligned}\left(\forall o_{x,i}, o_{y,j} \in O, \nexists o_{z,k} \in O\right) \big[\big(\phi(i) = \phi(j) \wedge \phi(j) = \phi(k) \\ \wedge \left(B(o_{x,i}) < B(o_{z,k}) \wedge B(o_{z,k}) < B(o_{y,j})\right)\big) \\ \implies B(o_{y,j}) \geq C(o_{x,i}) + SS(o_{x,i}, o_{y,j})\big]\end{aligned}$$

- *If a due date* $D(o_x, o_y)$ *from* o_x *to* o_y *is defined, then it imposes a due date on* o_y *such that* $B(o_y) \leq B(o_x) + D(o_x, o_y)$.

A schedule is feasible if all of these constraints are satisfied, and infeasible otherwise.

Definition 2 (makespan)**.** *The makespan of a schedule* B *is the latest completion time of any operation:*

$$makespan(B) = \max_{o \in O}(C(o)) = \max_{o \in O}(B(o) + P(o))$$

As the processing and set-up times are all non-negative, the order of the jobs and operations impose that the completion time of the last operation of the last job $o_{n,r}$ determines the makespan: $makespan(B) = B(o_{n,r}) + P(o_{n,r})$.

Definition 3 (re-entrant flow shop optimization problem)**.** *An optimal solution to a given flow shop problem is a feasible schedule* B *with the smallest possible makespan.*

In the algorithms presented in this chapter, we consider a single re-entrant machine. The job output order is fixed, i.e., $(\forall j_x, j_y \in J)[x > y \implies B(o_{x,r}) > B(o_{y,r})]$. Products are travelling on the same physical track, which means that the order in which products leave a machine, is also the order in which they arrive at the next machine. As there is only one track between each pair of machines, this leads to the following constraint:

$$\begin{aligned}(\forall o_{j,k}, o_{j,k+1}, o_{l,m}, o_{l,m+1} \in O)[\big(\phi(k) = \phi(m) \wedge \phi(k+1) = \phi(m+1)\\ \wedge B(o_{j,k}) > B(o_{l,m})\big)\\ \implies B(o_{j,k+1}) > B(o_{l,m+1})]\end{aligned}$$

In other words, products are not allowed to overtake each other. The topology of the industrial printer has only one physical track between the different stages in the flow vector. This means that the operation following in the flow vector ϕ cannot begin before all its previous job's operations up to that operation have been executed[1]. For example, it is not allowed that $o_{2,2}$ is executed before $o_{2,1}$ or $o_{1,2}$ (see Figure 2.1). Different passes of different jobs such as $o_{2,2}$ and $o_{1,3}$ do not have a particular order enforced.

2.2.2 FMS as re-entrant flow shops

The re-entrant flow shops we study are derived from FMSs. The resulting flow shops may have multiple machines, and one machine that processes a product multiple times. Each processing station of the FMS is transformed into a machine,

---

[1] When the product flow vectors differ (e.g., both simplex and duplex sheets are printed), then there are fewer restrictions on the operations before the re-entrant machine [115]

and operations on the products in the processing stations are encoded as operations of the flow shop. The minimum time between two processing steps, due to, e.g., travelling time, reconfiguration or cleaning, is captured by a (sequence-dependent) set-up time between operations. In this chapter we focus on FMSs that have bounded-time buffering capabilities for a re-entrant machine, which manifest as relative due dates on operations of the same job.

For example, an industrial printer can be modelled as a 3-machine flow shop with four operations per job, re-entrance vector $\langle \mu_1, \mu_2, \mu_2, \mu_3 \rangle$, and one job for each sheet that needs to be printed [122]. An example with five jobs is shown in Figure 2.1. Machine μ_1 (first row, yellow) loads a sheet, μ_2 (second row, cyan) prints one side of a sheet, after which it is turned in the re-entrancy buffer, before entering μ_2 again (third row, cyan) to print the other side. It is unloaded by machine μ_3 (fourth row, magenta). The load and unload operations must be executed in order of the print sequence. The operations on machine μ_2 need to be ordered such that the system is maximally productive, i.e., has minimum makespan.

In Figure 2.1 operations are represented by circles, with processing times inside. Between operations of the same job, set-up times enforce the operation order (shown with black, vertical arrows). Between the same operation of subsequent jobs, set-up times enforce the job order (shown with black, horizontal arrows). The order of operations on machine μ_2 is enforced through sequence-dependent set-up times (shown with blue dashed edges).

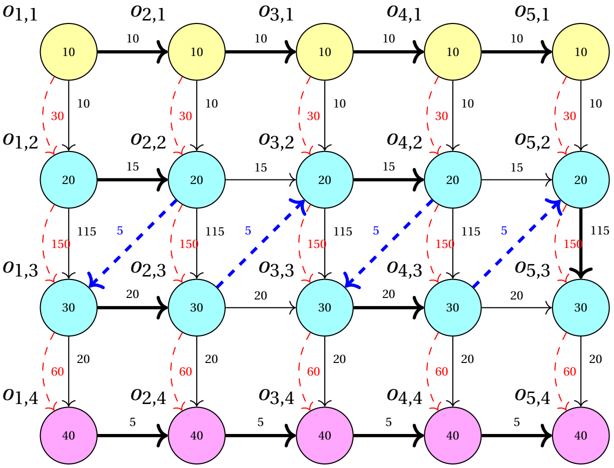


Figure 2.1: Example re-entrant flow shop with ordering sequences; the operations are represented by circles, with their processing times inside. Operations with the same colour are mapped to the same machine. Set-up times are shown by black edges, due dates by dashed red edges. Additional set-up times due to the scheduled sequence of operations are shown in dashed blue edges. The effective order of operations on the machines is indicated with thicker edges.

Scheduling a re-entrant flow shop boils down to ordering the operations per machine followed by determining the operation begin times. Figure 2.1 shows orders per machine in thick edges. The ordering sequences for machines μ_1 and μ_3 are simply the order in which the jobs need to be processed, as overtaking of products is not allowed. For machine μ_2 a scheduler needs to enforce sequence dependent set-up times between re-entrant operations of the machine to ensure that only one operation can be using the machine at any given point in time. The selected order in Figure 1 is $o_{1,2}$, $o_{2,2}$, $o_{1,3}$, $o_{2,3}, \ldots o_{5,3}$ and is denoted by thick black and dashed blue edges. The chosen sequence of operations leads to sequence-dependent set-up times, indicated by the annotations of the blue dashed edges. When such a sequence has been determined the earliest begin times of the operations can be computed efficiently by applying a longest-path algorithm between the first operation and the other operations (see Section 2.3.2 and 2.3.4).

2.3 Scheduling approach

We present the outline and describe components of the scheduling approach before investigating them in detail.

2.3.1 Scheduling outline

The scheduling outline of HCS [122] is the basis for the Bounded HCS (BHCS) algorithm. BHCS is defined by Algorithms 1 to 4. (B)HCS is a list scheduling approach that per iteration adds one operation into a sequence until a complete feasible schedule is obtained. It has two stages per iteration; (1) finding (potentially) productive sequencing options for operations on the re-entrant machine, followed by (2) finding the earliest begin times that satisfy all requirements for each of the operations. It then selects one of the feasible options. BHCS follows the structure of HCS but deviates most notably in the implementation of generating the sequences (Algorithm 4) and finding operation times for the generated sequences (Algorithm 19). Finding a maximally productive sequence is a difficult combinatorial optimization problem, but finding the earliest possible begin times once an operation sequence has been determined is rather efficient (see Section 2.3.2).

(B)HCS constructs sequences of operations. These sequences of operations can then be translated to their corresponding sequence-dependent set-up times between operations to find operation begin times and check their feasibility. (B)HCS starts with creating an initial sequence (the function CREATE_INITIAL_SEQUENCE in Algorithm 2) of the first pass operations of each job, followed by the higher pass operations of the last job for each re-entrant machine. For example, Figure 2.2 shows a flow shop f for which CREATE_INITIAL_SEQUENCE(f, μ) generates the sequence $o_{1,1}$, $o_{2,1}$, $o_{3,1}$, $o_{4,1}$, $o_{5,1}$, $o_{5,2}$ for the only machine in the example, μ. These operations must follow a pre-defined order defined by the scheduling constraints. In Algorithms 1, 2, and 3 each sequence s has a schedule $\underline{t}$ associated with it. The schedule $\underline{t}$ contains the earliest possible begin times of the operations that respect the sequence s.

(B)HCS then iteratively inserts higher pass operations, such as eligible operation $o_{1,2}$ in Figure 2.2, into the sequence s, adding the corresponding sequence-dependent set-up times (e.g., add the edges from $o_{3,1}$ to $o_{1,2}$ and $o_{1,2}$ to $o_{4,1}$) before checking feasibility of the timing requirements on $\underline{t}$. NEXT_ELIGIBLE_OPERATION is a function that returns the first re-entrant operation that does not yet occur in the ordering sequence in a job-first operation-first order.

The scheduler extends the sequence of operations by inserting a higher pass operation of jobs downstream in the input into the current sequence. At each iteration, the partial sequences have updated associated schedules. The schedules contain a lower bound on the begin times of operations. GENERATE_OPTIONS generates options by inserting an operation in multiple places in the given sequence s to produce new partial sequences s' and updates copies of $\underline{t}$ to new schedules $\underline{t}'$. When all operations of a job have been included in a sequence, no first or second pass can be inserted before the second pass of this job, and lead to a feasible schedule. The begin times of the operations of the current job can therefore be used to instruct the machine to start processing a job. The scheduler finally returns all the operation begin times associated with the chosen sequence.

The RANK function computes an absolute assessment for each partial solution based on a number of metrics from which (B)HCS greedily selects the option with the best assessment. The metrics we used are introduced in Section 2.4.3.

For each eligible operation, the sequencing options are created and evaluated in Algorithm 3. It creates a set l of feasible sequences and associated schedules by inserting the eligible operation before any of the operations returned by Algorithm 4. The sequences for each option are evaluated, and added to the list only when they are feasible.

The begin times are calculated using the BFM [12, 41] longest-path algorithm on a directed graph (Section 2.3.2) created from the combination of the processing

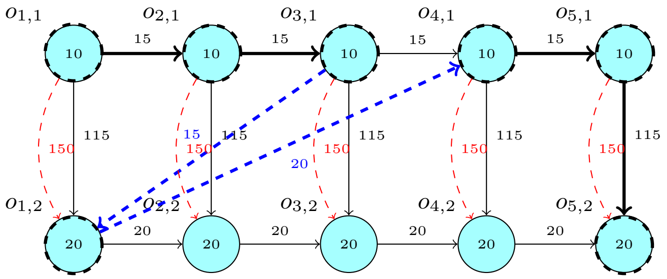


Figure 2.2: A sequencing option (dashed arrows), the sequenced operations (dashed outlined vertices) and the corresponding sequence-dependent set-up times (thick edges).

Algorithm 1 Scheduling outline of BHCS

```
function SCHEDULE(flow shop $f$, re-entrant machine $\mu$, ranking RANK)
    $(s, \underline{t})$ = CREATE_INITIAL_SEQUENCE($f$, $\mu$)
    repeat
        $o_e$ = NEXT_ELIGIBLE_OPERATION($f$, $s$)
        // Generate feasible options and update operation times
        $l$ = GENERATE_OPTIONS($f$, $o_e$, $(s, \underline{t})$)
        // Select the best result from the set
        $(s, \underline{t})$ = SELECT_BEST(RANK, $f$, $l$, $o_e$)
    until all re-entrant operations of $\mu$ included in $s$
    return $\underline{t}$
```

Algorithm 2 Create initial sequence and initialize begin times

```
function CREATE_INITIAL_SEQUENCE(flow shop $f$, re-entrant machine $\mu$)
    $s = \langle\rangle$
    // Add first passes of all jobs
    for each $i = 1$ to $|J|$ do append $o_{i,\mu,1}$ to $s$
    // Followed by re-entrant operations of last job
    for each $i = 2$ to $|r|$ do append $o_{|J|,\mu,i}$ to $s$
    // Initialize all begin times associated with s
    for each $o \in O(f)$ do $\underline{t}[o] = 0$
    return $(s, \underline{t})$
```

Algorithm 3 Find and apply all sequencing alternatives, and update begin times accordingly

```
function GENERATE_OPTIONS(flow shop $f$, operation $o_e$, ordering sequence
        with associated begin times $(s, \underline{t})$)
    $l = \emptyset$
    for each $o_p$ in FIND_INSERTION_POINTS($f, s, o_e$) do
        $s'$ = copy of $s$
        insert $o_e$ into $s'$, before $o_p$
        $(\underline{t}', v)$ = UPDATE_BEGIN_TIMES($f$, $s'$, $\underline{t}$)
        // If $v$ is true, the schedule is feasible so far
        if $v$ then
            $l = l \cup \{(s', \underline{t}')\}$
    // Return the set of feasible sequences and associated times
    return $l$
```

Algorithm 4 Find the set of sequencing options

```
function FIND_INSERTION_POINTS(flow shop $f$, sequence $s$, operation $o_{i,k}$)
    $r = \emptyset$
    $o_p = o_{i,k-1}$
    slack = smallest transitive due date in $D$ on $o_p$
    // Add all successors until slack is non-positive
    while slack $> 0 \wedge$ ($s$ has operation after $o_p$) do
        $o_n$ = NEXT($o_p$, $s$)
        $r = r \cup \{o_n\}$
        $d$ = smallest transitive due date in $D$ on $o_n$
        slack = min(slack - $P(o_p) - SS(o_p, o_n)$ , $d$)
        $o_p = o_n$
    return $r$
```

times, set-up times, and the relative due dates. When BFM detects positive cycles in this graph, then the sequence has no feasible schedule.

In the following sections, we introduce these ingredients in more detail. In particular, we identify two properties of the problem that allow us to bound the execution time of the online algorithm. These two properties allow us to define BHCS, which significantly improves the worst-case time complexity of HCS. We also introduce new metrics that improve the quality of the schedules. The first property allows us to purge many infeasible sequencing options (Section 2.3.3). The second property allows us to calculate begin times for only a bounded horizon and to still guarantee detection of infeasible sequencing options (Section 2.3.4). This results in an online adaptation of the Bellman-Ford-Moore longest-path algorithm.

2.3.2 Computing operation begin times from a sequence

The decisions made by the scheduling algorithm (Algorithm 1) ensure that all re-entrant operations are eventually scheduled into a single sequence per machine. Such a sequence leads to sequence-dependent set-up times, shown as dashed blue edges in the example in Figure 2.2. A schedule of begin times can be computed from the sequence of operations by a longest-path algorithm. The earliest possible begin times $\underline{t}$ for a given operation sequence are the longest-path time distances of all operations starting from the initial operation [103]. The earliest possible operation times $\underline{t}$ are the most productive operation times for a given operation sequence, and are therefore used as the schedule B.

The graph for longest-path computation consists of vertices representing the begin times of operations, and edges that impose timing constraints between operations. An edge from o_x to o_y with weight δ in the converted graph means that $B(o_y) \geq B(o_x) + \delta$. The constraints of Definition 1 are translated from the processing, (sequence-dependent) set-up times, and relative due dates as follows:

$$B(o_y) \geq B(o_x) + P(o_x) + S(o_x, o_y)$$

$$B(o_y) \geq B(o_x) + P(o_x) + SS(o_x, o_y)$$

$$B(o_y) \leq B(o_x) + D(o_x, o_y) \Leftrightarrow B(o_x) \geq B(o_y) - D(o_x, o_y)$$

The resulting graph looks similar to Figure 2.1, except that the red dashed edges corresponding to due dates are reversed, and their values are negated. Processing times are propagated into all of the vertex's corresponding set-up times [122].

The longest-path computation is equivalent to finding the solution to the system of inequalities that represents the scheduling constraints for the chosen sequence. Appendix A explains how the BFM algorithm can be used to compute such begin times in an on-line manner. If a solution exists (in the form of begin times for each operation), then the schedule is feasible. If no such solution exists, then the timing constraints are inconsistent, and such a sequence can never lead to a feasible schedule.

If a feasible schedule is found for a sequence that contains only the re-entrant operations for the first i jobs, then the (B)HCS scheduler will always be able to finish inserting operations and produce a feasible schedule for the full problem. As the deadlines for the jobs are contained within the jobs, there are no infeasibility issues between jobs, if there are no sequence-dependent set-up times that interleave products. In other words, no positive cycle can be created if a constraint from an operation of job i to an operation of job $k > i$ is made. It is necessary, however, to include the operation and job ordering constraints: the products that are already (partially) interleaved, need corresponding sequence-dependent set-up times for the second passes to exit the re-entrant loop. Despite the restriction to intra-job due dates, this statement holds for 2-re-entrant systems, but not for higher re-entrancies.

2.3.3 Purging infeasible sequencing options

We can determine the sequencing options for each of the re-entrant operations (Algorithm 4) by iteratively generating options until sequencing options become definitely infeasible. We use the property that an eligible operation $o_{j,x}$ must obey the operation sequence for job j and also obey the job sequence for the operation x. So $o_{j,x}$ cannot begin *before* any of its predecessors $o_{i,y}$ such that $i \leq j \wedge y \leq x$, nor *after* any of its successors $i \geq j \wedge y \geq x$. Consider an FMS with one re-entrant machine where three passes of a job need to be scheduled on a single machine. In the example of Figure 2.3 all hatched operations cannot be sequenced immediately before operation $o_{3,2}$ due to job and operation sequence constraints.

Additionally, we use the invariant that the scheduling algorithm does not reorder operations in the scheduling sequence. The algorithm will only insert more second pass operations into this sequence. This property and invariant allow us to determine which scheduling option is the last possible insertion point in the given sequence using static information about set-up times and due dates. This allows us to conclude infeasibility without calculating earliest begin times. The minimum time approximation in Algorithm 4 is used to determine which due date is the first to be certainly violated. The algorithm traverses the sequence starting from the source of the due date $o_{i,k-1}$ by iteratively investigating the NEXT operation in the sequence. NEXT returns the immediate successor of o_p in the sequence s. Summing the processing and set-up times for the given sequence provides a lower bound on the time difference between the given operation and its source in any valid schedule.

An example of five jobs executed in a 3-re-entrant flow shop is shown in Figure 2.3. In this example, a partial sequence has already been applied by previous scheduling decisions, as indicated by the dotted blue edges. All due dates are 150 time units. The lower bound on the longest path starting from $o_{3,1}$ starts at 0. This lower bound is increased by the sum of the processing time of that operation and the set-up time to the next operation, for example: at least $0+5+15 = 20$ is required before operation $o_{1,2}$ can start, at least $20+10+15 = 45$ is required before operation $o_{2,2}$, at least $45+10+25 = 80$ before $o_{1,3}$, at least $80+15+20 = 115$ before $o_{2,3}$, at least $115+15+20 = 150$ before $o_{4,1}$, and at least $150+5+15 = 170$ before $o_{5,1}$. The operation $o_{3,2}$ therefore cannot begin after operation $o_{5,1}$, $o_{5,2}$ or $o_{5,3}$, as this would violate the due date from the start of $o_{3,2}$ to the start of $o_{3,1}$; the lower-bound time differences following the given sequence between $o_{3,1}$ and these operations are re-

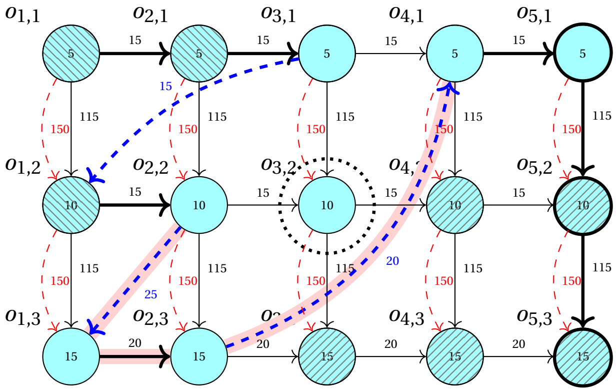


Figure 2.3: Operation $o_{3,2}$ might only be feasibly inserted between operations $o_{2,2}$ and $o_{1,3}$, $o_{1,3}$ and $o_{2,3}$, or $o_{2,3}$ and $o_{4,1}$ which are marked by a red highlight. Inserting operation $o_{3,2}$ into the sequence immediately after hatched operations (marked with diagonal lines) is infeasible due to sequence constraints on jobs and operations. Sequencing $o_{3,2}$ anywhere after $o_{4,1}$ (operations with thick border) is infeasible due to violation of the due date of $o_{3,1}$ to $o_{3,2}$.

spectively 170, 290, and 415, which are all more than the due date constraint of 150. When this lower bound on the minimum time between two operations exceeds the tightest due date encountered, then any schedule containing this ordering will be infeasible. Scheduling after the last possible operation is definitely infeasible as we found that there is at least one path which has a total time that is already larger than the due date imposed on the scheduled operation. As all the processing and set-up times are non-negative, we cannot insert more lower pass operations in the re-entrant buffer when the due date would already be violated at some point. The due date with the least slack typically corresponds to the smallest re-entrant buffer time in the FMS. In other words, the number of sequencing options to be considered is bounded by the ratio L which is the largest buffer time of the FMS divided by the smallest processing time.

If we would not take into account this property, we would end up with $\mathcal{O}(|J|)$ options per iteration. The bound described in this subsection allows us to stop generating options which would never be chosen, as they can never lead to feasible schedules. Their infeasibility would be detected by checking for positive cycles in a graph, which typically incurs the worst-case runtime of the BFM algorithm (see Appendix A). It is, therefore, much more efficient to avoid evaluating them. The difference in worst-case complexity with and without purging is studied in more detail in Section 2.3.5.

2.3.4 Bounded horizon

For online scheduling, it is only necessary to find feasible (and hopefully productive) begin times for the operations that are to be executed next. BHCS can defer computing the begin times of operations that do not influence the current decision. Such operations can still be indefinitely postponed and do not influence the feasibility of the current sequencing option.

To detect infeasibility of a sequencing option, BHCS needs to include at least those edges that are potentially involved in a positive cycle. Only the first of the two edges introduced by a sequencing option (such as $o_{3,1}$ to $o_{1,2}$ in Figure 2.2) creates additional cycles. To check for positive cycles, BFM needs to check the sub-graph induced by all operations in jobs between the job of the sequenced operation and the job of its predecessor in the sequence. All operations of these jobs are included, as they may have intra-job due dates associated with them.

The sources V_s are the operations which have been fully scheduled before. They indicate when the machines become available to process the next operations. They are all operations that have set-up times into that smallest sub-graph, but are not included in that smallest sub-graph. For example, the smallest sub-graph for scheduling and checking feasibility of inserting $o_{2,2}$ between $o_{4,1}$ and $o_{5,1}$ in Figure 2.2 consists of the operations of jobs 2 to 4. Its sources are the operations of job 1. The begin times for the sources are not allowed to be changed, as these operations may have been executed already.

Although the longest-path computation is computationally efficient, it is still the most time-consuming element of the scheduling algorithm when the job set becomes larger. Typical speed-up approaches such as GPU-acceleration can improve the worst-case execution time by a factor 50 to 100 [18] for some cases. However, this gain is not enough as the number of jobs can increase arbitrarily. Such accelerations can be implemented orthogonally, but without bounding the subgraph to be analysed, the longest-path computation will always become a limiting factor to the online applicability of the algorithm. We use the bounded subset of operations described in the previous paragraph as input for the modified BFM algorithm, defined in Algorithm 19 in Appendix A, to reduce the computational effort to evaluate a scheduling option.

We assume that the initial graph without sequence-dependent set-up times is feasible, and that we schedule one operation at a time. Inserting an operation in the sequence of scheduled operations introduces additional set-up times. An additional positive cycle must include at least one of the two edges introduced by the scheduling option, the processing and set-up times, and at least one due date. As we consider only intra-job due dates, operations of jobs later than the insertion point cannot create additional positive cycles.

If a sequence is considered feasible, then for an FMS with one 2-re-entrant machine this is sufficient to guarantee that this sequence can be extended to at least one feasible solution to the overall problem. The sequence-dependent set-up times that are necessary to empty the loop from higher pass operations returning from the buffer are already included as constraints, and have already been verified to not violate due dates. The set-up time after emptying the buffer delays operations which can still be delayed indefinitely and, therefore, such a decision will not violate any due date. Operations of a job can still be delayed whenever there is no path of constraints that can lead to a previous job. When feasible begin times are returned for a given sequence, it is guaranteed that this partial schedule is feasible for the overall problem with this sequence. For higher-re-entrancy FMSs or multiple re-entrant machines however, the heuristic might select a feasible sequence that does not have any feasible follow-up option.

Algorithm 19 runs in $\mathcal{O}(\max(|E_s|, |V_a| \cdot |E_a|))$ where E_s is the set of edges from source vertices, V_a is the set of active vertices, and E_a its corresponding set of edges. This helps us to greatly decrease the run-time complexity of the scheduling algorithm.

2.3.5 Worst-case time complexity of HCS and BHCS

We assess the impact of the improvements by comparing the worst-case time complexity of HCS to BHCS for flow shop instances that are derived from the structure of an FMS. Recall that $|J|$ denotes the number of jobs and r the number of operations per job. The number of vertices in the converted graph (Section 2.3.2) is $\mathcal{O}(|J| \cdot r)$ and the number of edges is also $\mathcal{O}(|J| \cdot r)$ as the number of edges per vertex is bounded. We assume there are no inter-job due dates, and set-up times occur only between operations of the same job, between same pass operations of subsequent jobs and in sequencing options.

The worst-case time complexity of HCS occurs when the last evaluated option is the only feasible option, as it then encounters the worst-case execution time for BFM, which is $\mathcal{O}(|V| \cdot |E|) = \mathcal{O}(|J|^2 \cdot r^2)$, for each scheduling option. The number of options to be evaluated may grow in the worst case with the size of the partial sequence for each re-entrant operation, $\mathcal{O}(\sum_{i=1}^{|J| \cdot r} i) = \mathcal{O}\left(|J|^2 \cdot r^2\right)$, the worst-case complexity for HCS is $\mathcal{O}\left(|J|^4 \cdot r^4\right)$.

BHCS creates at most L sequencing options for each of the $\mathcal{O}(|J| \cdot r)$ re-entrant operations (see Section 2.3.3). The largest subset of edges and vertices that needs to be used to detect infeasibility is then of size $\mathcal{O}(L \cdot r)$, and we only need to use a limited set of operations as sources, the worst-case complexity of Algorithm 19 is $\mathcal{O}(|V_a| \cdot |E_a|) = \mathcal{O}\left(L^2 \cdot r^2\right)$.

Therefore, the worst-case complexity of BHCS is the number of re-entrant operations to be scheduled ($|J| \cdot r$) multiplied by the number of sequencing options to be evaluated (L) multiplied by the evaluation cost of one option ($L^2 \cdot r^2$); totalling $\mathcal{O}\left(|J| \cdot L^3 \cdot r^3\right)$, where $r \ll |J|$ and $L \ll |J|$ in typical cases. The reduction in evaluation of options and the bounded horizon lead to a worst-case complexity of BHCS which is linear in the number of jobs and is therefore preferred over the super-linear complexity of HCS.

2.4 Exploring trade-offs in scheduling decisions

We first introduce a multi-dimensional Pareto meta-heuristic in Section 2.4.1. This algorithm is based on CPH [106]. We then apply this meta-heuristics to introduce the MD-BHCS (Section 2.4.2) and define the metrics to rank schedules (Section 2.4.3). In the scheduling outline (Algorithm 1, Line 8) we implicitly used the idea of good schedules. These metrics are used in a linear combination in (B)HCS to select a single option greedily. They are used as separate metrics in MD-BHCS to simultaneously explore multiple Pareto-optimal options.

2.4.1 Constructive Pareto Meta-Heuristic

In this section, we present a meta-heuristic template that can be used to simultaneously explore multiple options, without requiring backtracking. The Constructive Pareto Meta-Heuristic (CPMH) is inspired by the Algebra of Pareto points [45, 46], CPH [106], and Strength-Pareto Evolutionary Algorithm (SPEA2). A Pareto-optimal solution is a solution for which no other solution improves on one or more of the objectives without worsening any of the others. If one solution is better in one objective than another solution and not worse in any other objectives, the latter is *dominated* by the former [46]. The Pareto-optimal solutions are those solutions that are not dominated.

The goal of the meta-heuristic is to construct (all or a representative subset of) Pareto-optimal (partial) solutions. In other words, to eliminate those options for which there is some other option that is at least as good in any of the metrics and strictly better in at least one metric. Note that two or more partial solutions may have the same values for all metrics. If they are Pareto-optimal, then both options are kept.

The Pareto meta-heuristic incrementally extends a set of partial solutions with items to be scheduled to generate new partial solutions[2]. At most k representative solutions are kept at the beginning of every iteration. New partial solutions are created by extending each of the k partial solutions with a new item. Optionally, these partial solutions are improved by a fast online local optimization. The set of partial solutions is minimized through the Pareto-cull process which removes all dominated solutions from the set; only the Pareto-optimal solutions are kept for the next iteration.

Most multi-objective optimization algorithms create complete solutions, and assume that all information is known. Local search algorithms [109] construct an initial solution, which is iteratively improved until no further local improvement is found. Evolutionary algorithms [128] heuristically or randomly construct an initial generation of solutions for the full problem, which are then stochastically mutated and combined into new generations of solutions.

The Pareto meta-heuristic can easily be applied to algorithms that incrementally add the items to be scheduled. In CPH for the Multi-dimensional Multiple-choice Knapsack Problem (MMKP) partial solutions correspond one-to-one to Pareto Algebra configurations [46], and can therefore be evaluated in terms of optimization objectives. The algorithm improves the running time greatly by applying a number of reductions; at each iteration at most a given number k of partial solutions are taken into account, and Pareto-optimality is determined on a 2D projection of the configurations (i.e., solution quality versus aggregated resource usage). CPH for MMKP is an exact algorithm when these reductions are not used [106].

In our version of the Pareto meta-heuristic we use objectives in such a way that Pareto-optimal partial solutions present interesting trade-offs in the exploration of the schedule solution space. Partial schedules cannot be meaningfully compared in terms of the lower bound on the final makespan objective. Such a comparison would not make any trade-offs between earlier completion, buffer filling and remaining work explicit. We introduce auxiliary metrics to capture those aspects. Pareto-dominance is then used to allow simultaneous exploration of promising alternatives, without requiring backtracking. The meta-heuristic is a powerful tool for online optimization, due to the combination of constructive simultaneous exploration of alternatives, while reducing the number of representatives. The algorithm has been applied to MMKP, multi-objective bin-packing [87], and a multi-objective generalized travelling salesman problem (see Chapter 3).

---

[2] If partial solutions can be efficiently combined with other partial solutions, the approach is *compositional* [106]

Algorithm 5 Pareto meta-heuristic

```
function PARETO_METAHEURISTIC(problem instance $P$, size of partial solutions set $k$)
    // step 1
    Initialize the set of initial partial solutions $S_p$
    // step 2
    for each item $i$ to be scheduled do
        // 2a: Select at most k partial solutions
        $S_p = \text{REDUCE}(S_p, k)$
        // step 2b
        $S_p = \text{EXTEND}(S_p, i)$
        $S_p = \text{IMPROVE}(S_p)$ // Optional online improvement
        // step 2c
        $S_p = \min(S_p)$
    $S_p = \text{POSTPROCESS}(S_p)$ // Optional post improvement
    // step 3
    return $S_p$
```

When many of the partial solutions are Pareto-optimal then the set of partial solutions may grow very large. This can be prohibitive for online performance. However, as partial solutions with similar metric values are likely to produce similar results, we approximate the full set with a subset of bounded size which is as diverse as possible in the considered metrics. The number of partial solutions to consider is a parameter of CPMH. Keeping more partial solutions typically leads to a higher runtime but also to a higher schedule quality.

The template for a CPMH-based algorithm is shown in Algorithm 5. It abstracts from the particular problem definition, and shows how multi-objective optimization problems can be tackled.

Step 1 initializes the partial solutions S_p that will develop into the final solutions, i.e., Pareto-optimal solutions to the problem instance. Each of the partial solutions is extended with a new part of the problem instance in *Step 2*. This step is repeated once for every unscheduled item. *Step 2a* ensures that no more than k elements are kept in S_p. *Step 2b* extends the partial solutions to create a new set of partial solutions, which include all previously scheduled item, plus the item i. From the resulting solutions, we only keep those that are non-dominated by applying Pareto-minimization at the end of each iteration (*Step 2c*). Once all sets are combined, the results are returned in *Step 3*.

Steps 2b and 2c can be combined in an implementation to increase the computational efficiency of the heuristic. Pre- and post-processing steps can be introduced to tune the search direction of the algorithm. In addition, during the extension phase, a quick, typically greedy, improvement heuristic can be applied after *step 2c* to improve partial solutions.

2.4.2 Multidimensional scheduling outline

The Pareto meta-heuristic allows the scheduler to explore multiple sequencing options simultaneously, while still limiting the computation time. Where the combination of metrics in (B)HCS is used to select a greedily single (Pareto-)optimal trade-off, MD-BHCS explores multiple Pareo-optimal trade-offs.

Algorithm 6 MD-BHCS

```
function SCHEDULE(flow shop $f$, re-entrant machine $\mu$, size of partial solutions set $k$)
    $g = \{\text{CREATE\_INITIAL\_SEQUENCE}(f, \mu)\}$
    repeat
        REDUCE($g, k$)
        take an arbitrary $(s, t) \in g$
        $o_e = \text{NEXT\_ELIGIBLE\_OPERATION}(f, s)$
        $g' = \emptyset$
        for each $(s, t) \in g$ do
            // Combine - Generate feasible options, update operation times
            $l = \text{GENERATE\_OPTIONS}(f, o_e, (s, t))$
            $g' = g' \cup l$
        $g = \min(g')$
    until all re-entrant operations of $\mu$ included in $g$
    return $s \in g$ with the smallest makespan
```

MD-BHCS (Algorithm 6) is a heuristic that follows the template of the Pareto meta-heuristic to generate partial solutions as follows. Similar to BHCS, the initial sequence is initialized before any operation is scheduled and it is the starting point for all partial solutions that will be explored. A set of partial sequences is explored, starting with only the initial sequence. The outer loop ensures that each sequence eventually contains all re-entrant operations. The inner for-loop creates a new generation (i.e., a set of partial solutions) of sequences by scheduling one eligible operation, starting from each of the previous generation's sequences in g. A representative set of partial solutions at most size k survives from the current generation to the next generation. Algorithm 19 is again used to evaluate feasibility of sequencing options and to compute the begin times of operations.

Instead of selecting a single 'best' option, we make a multi-dimensional assessment of the partial solutions. The partial solutions generated by the EXTEND operator may not all be Pareto-optimal. We can remove all dominated solutions from the solution set by applying Pareto minimization. This minimization process removes all dominated solutions and is typically implemented by the *simple cull* algorithm as described extensively in [45, 106] and [125]. The reduction operator is explained in Section 2.4.4.

2.4.3 Assessing partial solutions

We divide a schedule into three parts for the assessment: (1) the part containing jobs for which all operations are sequenced and should be executed as fast as possible, (2) the part that reflects the impact of the sequencing options on the state of the re-entrancy buffer, and (3) the part which can still be indefinitely postponed.

MD-BHCS assesses these three parts with metrics that can be readily obtained from the begin times provided from Algorithm 19 and that together characterize partial solutions. For the last inserted operation o, and a sequence s with an associated partial schedule B:

- *past work* is assessed by measuring the earliest possible begin time B for o: $P(B,o) = B(o)$ (lower is better),
- *committed work* is assessed by the earliest possible begin time for the operation immediately following o in the scheduled sequence: $W(B,o) = B(\textsc{next}(o,s))$ (lower is better),
- *future work* is assessed by the productivity of the remaining part (higher is better). We approximate it by the number of operations $nr\_ops(s,o)$ that can be delayed indefinitely (lower is better).

The metrics influence the makespan of the final schedule as follows. Past work compares the influence of a scheduling option on the begin time of the eligible operation, taking into account how much buffer time is used by processing lower pass operations. The metric favours sequences for which the last inserted operation begins earlier. Delaying the last inserted operation is only interesting when we would benefit in the future from avoiding work or penalties for unsequenced operations. The committed work indicates the amount of work added to the re-entrant buffer, trying to minimize the work that needs to be done directly after this option. Future work measures how many units of work remain after all the committed work has been done, favouring less work. These metrics are heuristics to determine the different qualities of a sequencing option for the final makespan. We evaluate the metrics through the schedule quality in Section 2.5.3 by comparing the different variants of HCS to lower bounds.

2.4.4 Reducing the set of candidate solutions

We reduce the set of partial solutions such that it contains at most a pre-defined maximum number of partial solutions k. We use the archive truncation method from the multi-dimensional genetic algorithm SPEA2 [128]. It iteratively removes the solution which has the smallest distance to other solutions, until there are only k partial solutions left [128]. To reduce the impact of different magnitudes of the metrics on the distance, we normalize each metric between 0 and 1 by scaling them linearly to fit between their minimum and maximum observed value.

The normalized metrics are used in the ranking function of BHCS which defines the respective relevance of the metrics. In the Pareto-cull process of MD-BHCS the metrics are used without normalizing them, as the relative weight has no effect on Pareto optimality.

2.4.5 Worst-case time complexity of MD-BHCS

The main difference between BHCS and MD-BHCS is in the number of the partial solutions and how they select their next generation of sequences. Whereas BHCS starts each iteration with one solution and generates at most L solutions to evaluate, MD-BHCS starts with k solutions and generates at most $k \cdot L$ solutions for each of the $\mathcal{O}(|J| \cdot r)$ re-entrant operations. The Pareto minimization uses the Pareto-cull operation and takes at most $\mathcal{O}\left(k^2 \cdot L^2\right)$. The reduction process of SPEA2's archive truncation method for a solution set of size M has a worst-case time complexity of $\mathcal{O}\left(M^3\right)$ [128], where $M \leq k \cdot L$. The minimization, reduction, and evaluation of begin times are all in the inner loop and are executed for each iteration. The complexity of evaluating the begin times of all $k \cdot L$ options takes $\mathcal{O}(k \cdot L \cdot L^2 \cdot r^2)$. The worst-case time complexity of MD-BHCS is therefore $\mathcal{O}\left(|J| \cdot r \cdot k \cdot L^3 (k^2 + r^2)\right)$.

As k, r and L are constants, the algorithm runs in time linear to the number of operations $|J|$ in the flow shop problem. The worst-case time complexity of BHCS as explained in Section 2.3.5 is $\mathcal{O}\left(|J| \cdot L^3 \cdot r^3\right)$. The additional complexity of MD-BHCS over BHCS for evaluating more options is at most cubic in k. In practical applications however, the evaluation of feasibility remains the bottleneck, as its absolute runtime is much larger than the other components.

The number of items that fit in the loop increase as the re-entrant loop time increases, or as smaller products are supported. This complexity analysis hints when this scheduler may become the bottleneck in system productivity. The impact of the productivity optimization is then nulled by taking too much time to compute the instructions for the operations in the FMS.

2.5 Experimental evaluation

In this section we describe the experimental set-up, compare the makespan of the generated schedules, and evaluate the runtime of the schedulers.

2.5.1 Experimental set-up

We have implemented the two scheduling-algorithm variants, i.e., BHCS and MD-BHCS. As in the original HCS implementation, the processing times, set-up times and due dates are encoded as fixed precision values to avoid rounding errors that are typical for floating point implementations. All experiments have been executed on the same 8-core Intel i7 running at 3.0 GHz. All algorithms are implemented as single-thread programs.

The 3-machine, 2-re-entrant benchmarks representing print requests for the industrial printer as introduced in [122] are used to assess the quality of the schedules and the runtime of the algorithms. The benchmark consists of 85552 sheets in 701 print requests. For the details of the test set, we refer to [122]. Each print request is taken from one of the following categories:

H Homogeneous: repetition of one sheet type

RA Repeating A: repetition of one sheet type followed by another sheet type

RB Repeating B: repetition of one to three sheets of a type followed by another sheet type

BA Block A: 5 blocks of 5 different sheet types, each block contains 10 or 20 sheets of the same type

BB Block B: 5 blocks with two alternating sheet types, each block contains 5 to 25 sheets of the same type

We compare the makespans generated by our algorithms with the makespans of schedules generated by HCS and the results of a mixed integer programming (MIP) formulation of the scheduling problem. While searching for the optimal solution for the MIP, CPLEX [64] (a generic MIP solver) also computes the optimal objective value for linear relaxations of the MIP. Such relaxations give a lower bound for the makespan of the MIP. A schedule computed for a linear relaxation typically violates integer feasibility constraints, and as such does not necessarily represent a solution to the original problem. When our heuristic or CPLEX finds a schedule that has a makespan equal to the lower bound, then the bound is exact and that schedule is proven to be optimal. Otherwise, it is unknown how close the lower bound is to the actual optimum value. The lower bounds for the benchmarks have been calculated with CPLEX 12.6.0 with a time limit of 1000 seconds (pre-solve and solve).

We used a weighted combination of the normalized metrics for the ranking mechanism of BHCS. These weights were chosen empirically by selecting initial weights and tuned by iteratively applying small variations and re-scheduling the full benchmark. Tuning was repeated until no further improvements in makespan were found. The best results by BHCS, for the benchmark defined in our set-up, were found using the following weighted combination for a given schedule B, associated sequence s and last inserted operation o:

$$rank(B,o,s) = 0.3\cdot\textsc{norm}(P(B,o)) + 0.6\cdot\textsc{norm}(W(B,o)) + 0.1\cdot\textsc{norm}(nr\_ops(s,o))$$

NORM ensures that a dimension is normalized between 0 (best observed) and 1 (worst observed).

2.5.2 Sensitivity to number of partial solutions

The performance of the MD-BHCS algorithm, both in quality (Figure 2.4) and runtime (Figure 2.5), depends on the number of partial solutions k. The makespan of a schedule typically becomes shorter when parameter k becomes higher, while the average time per iteration increases. MD-BHCS with $k = 2$ is faster than BHCS and also produces worse schedules. With $k = 2$, the scheduler can only account for two extremes in a three-dimensional assessment and, therefore, cannot cover the trade-offs accurately. Additionally, BHCS' weighted sum does not focus on the extremes of the trade-offs. They are likely to be too aggressive in one aspect, for example, aggressively filling the buffer. The generated Pareto points for $k = 2$ are less likely to have many follow-up options, and typically lead to fewer evaluations of infeasible sequences.

The quality benefits of the multi-dimensional start to diminish when $k = 10$ or more partial solutions; the schedules improve only slightly, but at the cost of additional runtime. Figure 2.4 shows that when $k = 20$ partial solutions are taken into account, MD-BHCS produces the best makespans. The effect of exploring multiple options simultaneously with the meta-heuristic does not increase the observed runtime significantly, as the number of Pareto-optimal partial solutions is often less than k.

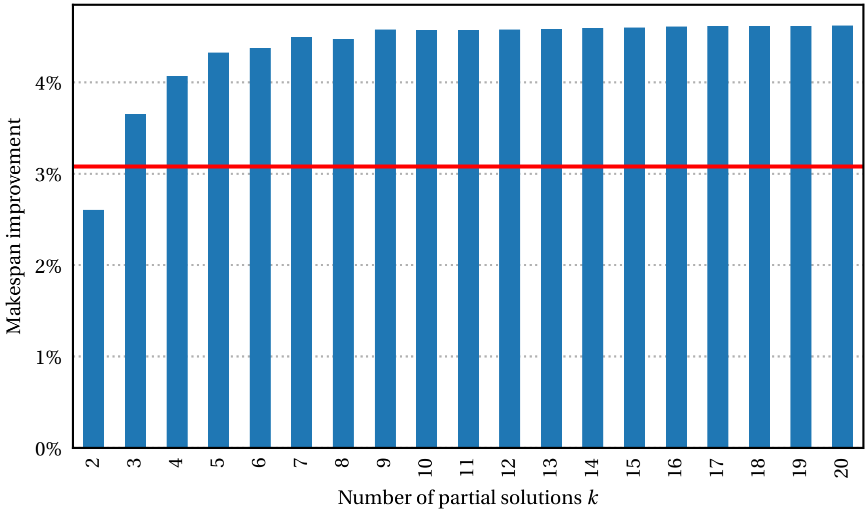


Figure 2.4: Average makespan improvement of MD-BHCS (vertical blue bars) and BHCS (horizontal red line) over HCS.

2.5.3 Makespan comparison

The box plot in Figure 2.6 shows the improvement over the makespans from HCS. The median improvement lies between 0% and 4%, with several outliers over 20%.

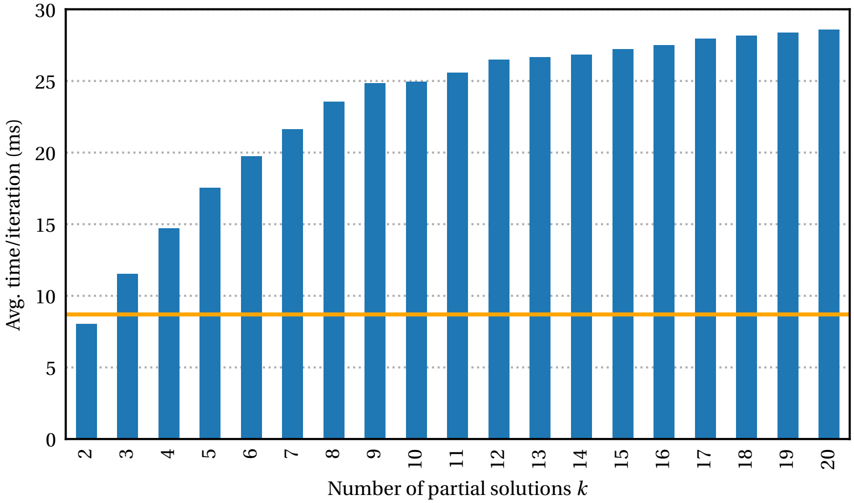


Figure 2.5: Average time per iteration for MD-BHCS for varying k. The horizontal orange line is the average time per iteration for BHCS.

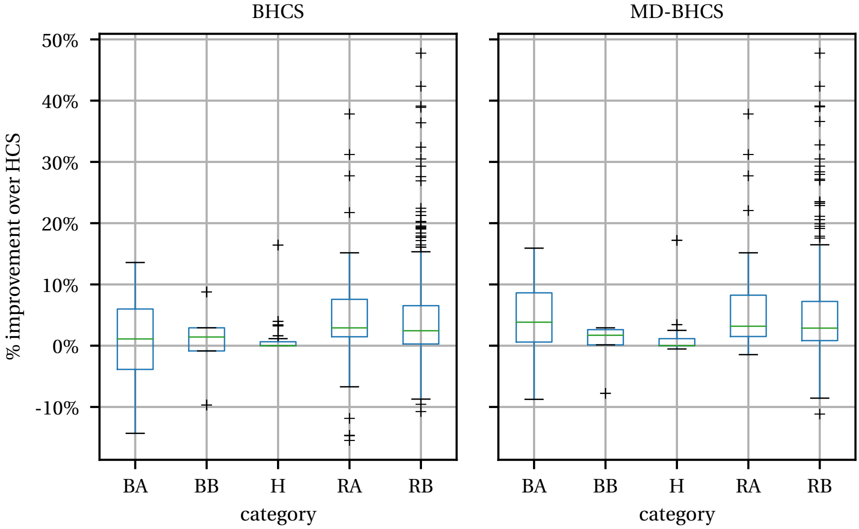


Figure 2.6: Makespan improvement of BHCS and MD-BHCS ($k = 20$ solutions) relative to HCS (higher is better).

Table 2.1: Makespans compared to CPLEX lower bounds; $CPLEX_{all}$ is average percentage above the lower bound (all 701 cases), $CPLEX_{opt}$ is average percentage above lower bound for 108 CPLEX optimal cases, OPT is number of cases where the makespan is known to be optimal.

		$CPLEX_{all}$	$CPLEX_{opt}$	OPT
HCS		21.90%	4.40%	39
BHCS		18.26%	1.48%	48
MD-BHCS	k=2	18.96%	2.33%	40
	k=6	16.87%	0.64%	74
	k=10	16.60%	0.62%	80
	k=20	16.54%	0.62%	80

The difference in the H and BB category is very small as the set-up times in all of these job sets are very small. Both schedulers manage to keep the buffer occupied for these cases which produces close-to-optimal results. For the other categories, MD-BHCS typically out-performs HCS.

Table 2.1 shows how close the results of different variants of HCS are to the CPLEX lower bounds. The first two columns show the average percentage above the lower bound; column $CPLEX_{all}$ includes all benchmark instances, and column $CPLEX_{opt}$ includes only those instances for which CPLEX found an optimal solution. In the OPT column, the number of solutions which have makespan equal to their lower bound are shown. These results show that the makespans are over 15% higher than the CPLEX lower bounds ($CPLEX_{all}$). However, when we compare only the cases where CPLEX found optimal schedules, ($CPLEX_{opt}$), we see that the results are within 1% of the optimal result. CPLEX found 108 optimal schedules, and the different schedulers found several optimal schedules. The HCS variants sometimes manage to find optimal schedules when CPLEX only determined a lower bound but could not determine whether it is indeed possible to achieve such a schedule.

2.5.4 Runtime evaluation

The runtime per category in Figure 2.7 for this benchmark with $k = 20$ is below 600 ms, with an average runtime of 28 ms. This shows that the proposed algorithm is indeed fast enough for online computation of schedules for the industrial printer, indicating that it can be applied to re-entrant flow shops originating from FMSs. Figure 2.7 also shows that the observed worst-case running time is 10 higher than the median, due to the infeasibility checks still incurring worst-case behaviour. Due to the large data set the many outliers have been included in the whiskers.

We have also compared the runtime of our implementations to the implementation of HCS [122] and noticed that the difference in runtime can become arbitrarily large. In the benchmark, the runtime is up to 100 times slower than BHCS. The observed runtime of HCS scales super-linearly with the number of jobs in the input. The average processing time per scheduling decision in the complete benchmark is 240ms for HCS and 8.6ms for BHCS, on average 28 times faster. For only 14% of the cases, HCS is up to twice faster than BHCS. MD-BHCS with $k = 20$ is 3 times slower than BHCS and, therefore, only 9 times faster than HCS on the whole benchmark.

2.6 Related work

Manufacturing-line scheduling has received a lot of attention in the operations research field. Manufacturing systems are traditionally modelled as job shops or flow shops. Many specialized exact algorithms and heuristics have been developed to deal with offline and online optimization of job shop and flow shop variants. A plethora of optimization objectives have been defined [50], among which are minimizing makespan [11, 20, 69, 93], total tardiness [65], maximum lateness [29] mean flow time [74], and total completion time [19, 66]. Some approaches can deal only with processing times, others include set-up times, or due dates. Other ap-

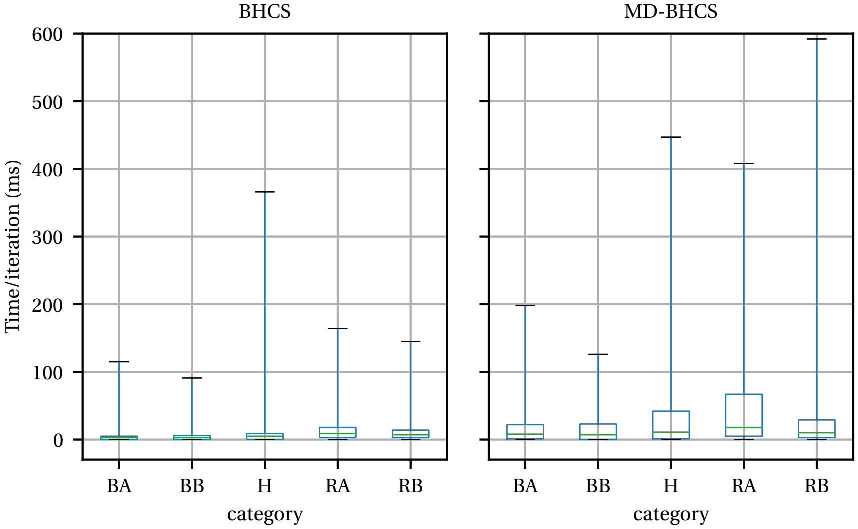


Figure 2.7: Runtime per scheduling decision for BHCS and MD-BHCS ($k = 20$) (outliers included in whiskers).

proaches deal with claiming (multiple) shared resources and simultaneously solve a mapping and a sequencing problem [11]. Use cases such as production lines for mirrors, lacquer [11] and textile [69] have been researched. For this chapter we assume that the mapping of operations to machines, the flow of operations, and the sequence of jobs is fixed, and that a product is processed twice by one of the machines. The scheduling freedom in our FMS consists of choosing the sequence of operations on the re-entrant machine and selecting efficient operation start times.

In our use case, the due dates are relative to the begin times of the operations, and are not allowed to be missed. Relative due dates are more general than absolute due dates. In most of the related work, due dates are absolute and do not change when the sequence of operations changes. When due dates are involved, most algorithms try to minimize the maximum lateness [29, 48]; i.e the maximum time that a job is completed later than its due date. In case any of the due dates is violated in the type of re-entrant FMS, the products collide. This will lead to damaged products at best, and a malfunctioning system at worst.

Most scheduling problems also allow for arbitrary ordering of the jobs, but in our case it is required to complete them in a specific order. The HCS [122] heuristic addresses the same problem as in this work but it approximates the impact of a scheduling decision, and explores a single option; is therefore more sensitive to the tuning of its internal ranking mechanism. We improve on the current state of the art, HCS. The effectivity of our MD-BHCS algorithm arises from the combination of leveraging necessary properties of feasible schedules for the flow shop problem with exploring multiple options simultaneously.

MIP Heuristics for job and flow shops such as the Rolling Horizon Procedure (RHP) [19] and decomposition methods [29] have been used to improve the efficiency of exact algorithms by applying them to smaller sub-problems and then implementing a portion of the solution to the sub-problem. This yields close-to-optimal solutions for problems with only sequence-dependent set-up times. Solving a sub-problem with relative due dates with CPLEX may yield optimal solutions for RHP, but does not typically terminate within less than 10 seconds. For some cases, it did not even return any feasible schedule within 10 seconds, making it ineffective for online scheduling.

Many online algorithms such as rate-monotonic and priority scheduling cannot be applied on FMSs, or they would significantly restrict the flexibility of the system. These online algorithms require cyclic behaviour in the input sequences, or the possibility to assign a priority to the operations beforehand. They moreover do not take into account the relative deadlines of the operations. Online algorithms such as Earliest-Deadline-First (EDF) and Least-Slack-Time (LST) will schedule the operations in a job-first operation-first order, effectively leading to a single product flowing through the re-entrant buffer. This is ineffective, as in the time between a product's first and second pass the FMS could have used the re-entrant machine to finish other operations. It is also not effective to greedily fill the buffer such that as many products as possible are in the loop, as the sequence-dependent set-up times of a future product possibly cannot be accommodated.

This leads to emptying the re-entrant buffer too often. The Nearest Edge Heuristic [65] modified to include relative due dates [122] has also been shown to yield much lower-quality schedules than the current state of the art, HCS.

The problem in this chapter concerns 2-re-entrant flow shops. This problem has been generalized to allow mixes of product flows with one and two re-entrances [116], where overtaking of products with different product flows is allowed at the re-entrant machine. This generalization can also be supported by the algorithms described in this chapter. To support variable re-entrant flows, the initial sequence generation needs to be adapted, and the operation sequence at the input machines (before the re-entrant machines) must be enforced in the same order as they enter the re-entrant machine [116].

Several specialized algorithms have been devised for re-entrant hubs [74], and multiple re-entrant stations [21, 65]. If the re-entrancies of different machines are to be interleaved with each other, then the selection of product sequencing for earlier machines directly influences the options for later machines. Our algorithm does not guarantee feasibility for such cases. If, however, the flow of the flow shop first finishes all re-entrant operations in a station, before processing in the following re-entrant station, then our proposed algorithm will work if there is sufficient product buffering capabilities between the re-entrant machines. The streams of products exit a re-entrant machine in the same order as they enter, and each machine can be emptied to continue processing. This means that the feasibility of the operation sequences at the re-entrant machines is decoupled, and our algorithm will always find a feasible schedule. Maintaining the same ordering of operations at each re-entrant station is another common way [21, 65] to deal with multiple re-entrant stations. Our heuristic algorithm is expected to significantly drop performance when the sequence-dependent set-up times are small for most re-entrant machines, and large for one re-entrant machine. The sequence-dependent set-up times are then close to the maximum of either set-up time.

2.7 Conclusion

We have shown that we can drastically reduce the worst-case time complexity and average running time of a heuristic scheduler for re-entrant flow shops that originate from FMSs compared to the scheduler of [122]. The resulting MD-BHCS scheduling algorithm investigates a smaller part of the timing of operations, and avoids the evaluation of infeasible sequencing options. We have used properties of the scheduling problem to quickly remove infeasible options by inspecting local information. We have also shown that the worst-case time complexity of the heuristic depends on the maximum number of products that fit simultaneously in the re-entrant buffer of an FMS.

The performance of the multi-dimensional Pareto meta-heuristic CPMH on this flow shop scheduling problem confirms it is a promising meta-heuristic for online scheduling. MD-BHCS produces on average 4.6% shorter makespans on a relevant industrial benchmark than the previous state of the art, and is 9 times faster. Trading off solution quality for runtime is useful to avoid the scheduler becoming a bottleneck for productivity.

Products within a product batch are required to be processed in a particular order. Once an order of batches has been determined, the online scheduler optimizes the interleaving of processed and unprocessed streams of sheets. In the next chapter, we provide an approach to find good orderings of batches.

3

Multi-objective Optimization of Product Batches

The previous chapter presented an algorithm to optimize the merging of re-entrant product streams. A key restriction in that problem is that the products must leave the FMS in a given order. In industrial printing, the sheets must be output in the sequence requested by the customer. Product batches are logical units of work that need to be processed together. Although the products within a batch (e.g., pages within a book) must not be shuffled, it is still allowed to determine the order in which different batches are manufactured. In this chapter, we therefore allow the output order of *product batches* to change, and use this to optimize for productivity. An FMS operator not only needs to select the product batch *order*, but also the settings that determine the production *mode*. The settings of FMSs typically lead to different productivity, quality, and/or running costs. E.g., if the highest printing quality is required, the productivity may be relatively low, at additional cost for using extra ink to achieve a particular quality. If subsequent batches need different settings, the machine needs to reconfigure, which leads to unproductive set-up times. In the printing case of the previous chapter, set-up times occur when interleaving sheets from different print jobs inside the printer. These set-up times affect productivity and potentially cost and quality.

In this chapter, we contribute a model and algorithms that sequences batches and selects appropriate system settings such that an operator can choose from Pareto-optimal trade-offs in aspects like, productivity, quality, and costs. As a first contribution, we provide a reference heuristic that focuses on solution quality but is not applicable to online optimization. Online optimization is needed when batches arrive on-the-fly. As a second contribution, we provide an online heuristic that provides good solution quality in acceptable time.

3.1 Introduction

Many scheduling problems that require sequence optimization can be defined as a Traveling Salesman Problem (TSP) [99], or a variant of the TSP [76, 89]. The TSP is a well-known combinatorial optimization problem that aims to optimize the cost

---

The content of this chapter is an extension of the work published in DATE 2016 [118].

of a tour, i.e., minimizing the cost to visit each of a set of cities exactly once. We explore online optimization techniques for scheduling FMSs that besides sequencing, involve selection of settings, while optimizing for multiple objectives. Such problems are then expressed as a Multi-Objective Generalized TSP (MO-GTSP).

The FMS can produce the product batches in one of several modes, resulting in different processing values (time, quality, etc.). Changing modes occurs only between batches, and typically has an associated processing penalty, for instance in time and/or cost. An online scheduling algorithm needs to determine in what order to execute the batches, and in which modes. Figure 3.1 gives an example of such trade-offs, encoded in an MO-GTSP.

Real-world optimization problems are often multi-objective in nature. The goal of (meta-)heuristics for multi-objective optimization problems is to find the efficient set, a set of solutions with Pareto-optimal values. The efficient set consists of all solutions that have values that are not dominated by any other solution. The Multi-Objective Traveling Salesman Problem (MO-TSP) is a TSP variant in which multiple objectives, i.e., tour metrics, need to be optimized simultaneously.

Many problems are also multiple-choice in nature; we have to select, e.g., exactly one system mode per product batch. Sequencing choices occur in reconfigurable FMSs that have sequence-dependent or even history-dependent reconfiguration times. Selection choices occur when the input sequences can be adapted, e.g., to increase performance. This freedom of choice is captured by the generalized aspect of the Generalized TSP (GTSP), where an optimal tour through clusters of cities needs to be found, visiting exactly one city from each cluster.

Not all information may be available before any decisions need to be taken. For a printer, batches can arrive at any point in time. An algorithm either needs to be fast enough to recompute all optimization decisions, or to have an online component, where information can be incorporated on-the-fly.

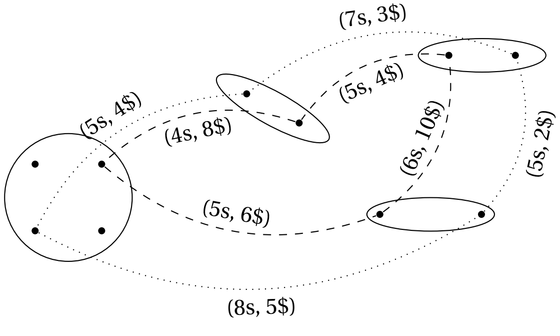


Figure 3.1: Example MO-GTSP, with time and cost objectives. The ellipses with dots are clusters of cities in the MO-GTSP, representing print jobs and printer modes, resp. The two tours, print sequences and mode selections, have objective values (25s, 14$) and (20s, 28$) resp., showing a time, cost trade-off.

We apply the Constructive Pareto Meta-Heuristic (CPMH), the constructive meta-heuristic that was introduced in Section 2.4.1, to MO-GTSP. The application of CPMH to MO-GTSP is incremental in nature, enabling online computation of solutions. CPMH is inspired by CPH, originally applied to the Multi-dimensional Multiple-choice Knapsack Problem (MMKP) [106]. Such problems are found in, e.g., online resource management and in routing of wires on chips. The incremental and parametrized nature of CPH has shown particularly flexible and powerful for use as a tunable, online algorithm. The main difference between MMKP and MO-GTSP is that MMKP is essentially a selection problem whereas MO-GTSP is a combined sequencing and selection problem. Whereas TSP does not impose bounds on any of the dimensions of the optimization problem, MMKP enforces hard bounds.

We are the first to develop dedicated heuristics for the MO-GTSP. Our MO-GTSP solutions build upon approaches for solving MO-TSP and GTSP, The state-of-the-art algorithms for solving MO-TSP are non-deterministic meta-heuristics with powerful local search [109]. Pareto Local Search (PLS) [5,96] has been used to find approximations of the efficient set of MO-TSP. PLS explores the neighbourhood of solutions to find neighbouring, hopefully non-dominated, solutions. Two-Phase Pareto Local Search (2PPLS) [83] is the current state-of-the-art for MO-TSP. In its first phase it uses a strong single-objective solver to generate an initial approximation of the efficient set. In the second phase, it uses PLS to find additional non-dominated solutions. The Memetic Algorithm for GTSP (MA-GTSP) [52] is currently the state-of-the-art algorithm for GTSP.

We combine the state-of-the-art approaches (2PPLS and MA-GTSP) to create a reference heuristic for MO-GTSP that focuses on solution quality. It is not suitable for online application. We use CPMH and PLS as a basis for our online MO-GTSP heuristic.

3.2 Multi-Objective Generalized TSP

We extend the single-objective GTSP model presented in [39] to a MO-GTSP. We are given a complete weighted directed graph $G(V,E)$ with vertices V and edges $E = V \times V$. In addition, a partitioning $C = C_1,\dots,C_n$ of $V = C_1 \cup C_2 \cup \cdots \cup C_m$, such that $C_i, C_j \in C : i \neq j \implies C_i \cap C_j = \emptyset$ defines the clusters to be visited. A cycle is *feasible* if and only if it visits each cluster *exactly* once. The objective function $z : E \to \mathbb{R}^n_{\geq 0}$ maps each edge to its corresponding n-dimensional objective. These objectives can be length, weight, and/or cost. Solutions to the problem are tours $T \subseteq E$, which are feasible cycles with a Pareto-optimal multi-objective tour cost $\sum_{e \in T} c(e)$. The sum of value-tuples is obtained through element-wise addition.

A tour T is *dominated* if and only if there exists another tour T' in a set such that $z(T')$ dominates $z(T)$, i.e., if and only if each objective is worse or equal to that of tour T' and the tour objectives are not the same; $z(T) \neq z(T')$. The Pareto-optimal tours are those that are not dominated by any other tour. The two tours in

Figure 3.1 are not dominated by each other.

The MO-GTSP corresponds to print batch optimization as follows. Each cluster of the MO-GTSP corresponds to a print batch. Each city in such a cluster corresponds to one of the print modes for that print batch. Exactly one city of each cluster must be visited; i.e., each print batch must be executed exactly once in one of the given print modes. The edges between cities correspond to the time or cost associated with executing subsequent print batches in certain print modes. The MO-GTSP therefore matches the structure of the print batch optimization problem directly, making it the appropriate modelling choice.

3.3 Related work

Although many algorithms have been developed for many TSP variants, like the GTSP and MO-TSP, no algorithms dedicated to solving the combined challenge of GTSP and MO-TSP were published prior to our own work. The preliminary work published in [118] is further developed in this chapter. We discuss work related to TSP, MO-TSP, GTSP, and to MO-GTSP.

3.3.1 Traveling Salesman Problem

The TSP is a classical problem [7,54,77], famous for its simple definition, and notorious for its difficult optimization landscape. Many algorithms have been defined to solve geometric/Euclidean versions of the TSP [6,7,77], either exactly or heuristically. It has been a proving ground for general optimization techniques such as branch-and-bound, and branch-and-cut algorithms [68, 92] and sophisticated approximation algorithms for the geometric TSP [22]. Such approximation algorithms for TSP typically rely on the computation of closely related problems for which optimal solutions can be computed in polynomial time, such as minimum spanning trees, and perfect matchings. Most algorithms focus on the instances where distances between cities are symmetric; i.e., symmetric TSP. Several shortcuts can be taken when the instances are symmetric, and therefore the algorithms for the asymmetric TSP are typically slower and yield lower quality solutions.

Exact algorithms require a significant amount of time to execute, as the TSP is an NP-hard optimization problem. The difficulty lies both in finding the best solution, and in proving that no other solution can be better. Heuristics have been found that typically find very good solutions in a small amount of time. Such heuristics typically start from some initial solution, and iteratively apply improvements. Several local modification algorithms exists for the TSP. By applying a local modification to a tour, a *neighbouring* solution can be found. The neighbourhood of a solution consists of all its neighbouring solutions for some given local modification approach.

Iterated Local Search (ILS) [109] iteratively accepts a neighbouring solution if it improves the objective. In case no further improvement can be found, the local search stops, as it has found a local optimum. The 2-exchange and 3-exchange neighbourhoods are well-known examples of effective local searches; they modify a tour by cutting it into two respectively three pieces, and assembling the pieces in a different order. ILS eventually finds a local optimum, but may miss the global optimum if this is not reachable by locally improving solutions. Stochastic Local Search (SLS) [62] adds a non-deterministic *perturbation* step that creates one arbitrary modification. Such a solution is likely worse, but the new neighbourhood may allow the ILS to escape from a local minimum.

3.3.2 Multi-Objective Traveling Salesman Problem

Many heuristics have been defined for MO-TSP, but only recently has an exact algorithm been applied successfully to find the efficient set [40]. The efficient set for MO-TSP instances has been calculated with the Augmented Epsilon Constraint algorithm (AUGMECON2) [40] using a branch-and-cut algorithm for the single-objective TSP. AUGMECON2 generates all the solutions in the efficient set using an augmented version of the ϵ-constraint method. The ϵ-constraint method generates single-objective mixed integer programming (MIP) instances from a multi-objective problem instance, which are solved by a generic solver for one objective, while the other objectives are constrained. As the objective values for TSP are integer, a grid of instances can then be explored for each combination of the objective values between the origin and the worst attainable objective values. The approach uses slack information of the previously calculated Pareto-optimal values to drastically increase the efficiency of searching the grid. Despite this improvement, it takes almost a day to compute the efficient set for instances with 100 cities. This is unsurprising because of the computational complexity of the single-objective TSP. The computational complexity of multi-objective combinatorial optimization problems in general is very high [33].

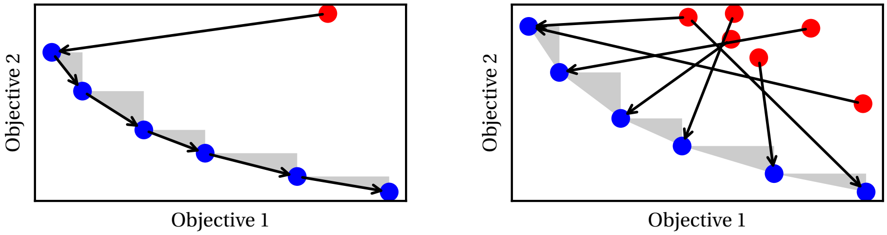
Paquete et al. have extended SLS for single-objective TSP to bi-objective TSP [96, 97]. Their algorithm, Pareto Double Two Phase Local Search (PD-TPLS) and its implementation are still among the state-of-the-art MO-TSP metaheuristics, and is visualized in Figure 3.2a. In a first phase, the algorithm generates a solution that is near-optimal for a single-objective TSP instance, and in the second phase the algorithm applies weighted aggregation to find neighbouring non-dominated solutions. The heuristic sweeps over the objective space by gradually modifying the local search direction in a pre-defined number of discrete steps. Algorithms that use weighted averages of multiple objectives can only find points on the convex hull of the Pareto-front. That is, they only find *supported* Pareto points.

Most Pareto-optimal solutions of a bi-objective TSP are reachable through the immediate 2-exchange neighbourhood [15, 95]. This observation has been leveraged to quickly search through the neighbourhood of a previously generated solution to find other solutions that are likely in the efficient set [97]. Several algorithms use Pareto Local Search [5,94], where a neighbourhood of a (set of) Pareto-point(s) is explored to iteratively find neighbouring non-dominated points. The 2PPLS [83] algorithm (visualized in Figure 3.2b) generates supported points for a bi-objective TSP through weighted aggregations in its first phase according to a dichotomic scheme originally proposed in [4]. In the second phase, it uses PLS to find nearby non-supported Pareto points. This relatively simple approach is currently among the state-of-the-art. The choice of neighbourhoods for SLS and PLS significantly impacts the effectiveness of these algorithms [96].

3.3.3 Generalized TSP

Memetic algorithms such as MA-GTSP [52,53] are regarded as the state-of-the-art heuristics for GTSP. Memetic algorithms combine the ideas of crossover and mutation from genetic algorithms with a strong local improvement element. These algorithms differ from SLS as they keep more than one solution at a time, and try to recombine the good parts of different tours into a new tour. Memetic algorithms are typically slower than their deterministic counter-parts, but have more opportunities to escape local minima.

A branch-and-cut algorithm [39] has been created for GTSP to find the exact optimal solution of a GTSP instance. The work presents several sophisticated heuristics to generate additional constraints (i.e., cuts) for a MIP formulation of a relaxed version of GTSP until a feasible solution to the GTSP is found. As this is an exact algorithm, very large instances require long computation times.



(a) Pareto Two-Phase Local Search (PT-PLS) [97] starts from a random solution, and optimizes gradually to the second objective..

(b) Two-Phase Pareto Local Search (2PPLS) [83] starts from several random solutions, which are optimized for different weighted aggregations.

Figure 3.2: Random initial solutions (red) are optimized through SLS or similar 1D optimization (arrow) to find supported non-dominated points (blue). PLS explores the neighbourhood of the supported points to find non-supported points in the grey triangles.

3.3.4 Multi-Objective Generalized TSP

Prior to the work in this thesis, no dedicated algorithm or heuristic for MO-GTSP had been presented in the literature. An evolutionary approach has been suggested for a generalized bi-objective TSP [75]. The generalization in this case does not consider clusters of cities. Instead, it consists of allowing multiple (non-dominated) edges between cities. For our printing case, we need to encode that selecting a particular print mode for one batch will lead to a limited number of options for the subsequent batch. This restriction cannot be encoded in the generalized version of [75]. The MO-GTSP as introduced in Section 3.2 can express the generalized version of [75] by expanding each city into a cluster containing one city for each incoming edge.

Because no prior solutions for MO-GTSP exist, we approximate the efficient set of an MO-GTSP by modifying the components of the 2PPLS. We combine the state-of-the-art memetic algorithm MA-GTSP [52] and the dichotomic scheme of [4] to generate supported Pareto-optimal points, as elaborated in Section 3.4. Alternatively, we could have combined AUGMECON2 [40] with the branch-and-cut heuristic for GTSP [39]. However, [39,53] suggest that this exact approach does not scale well for asymmetric instances, indicating that AUGMECON2 for GTSP does not scale well to larger instances.

We then apply the CPMH meta-heuristic to create an online heuristic for MO-GTSP. The incremental CPH approach presented for MMKP [106, 107] already deals with multiple choices and multiple objectives. We deal in addition with sequencing (of clusters), by combining the incremental approach of CPMH with PLS.

3.4 2PPLS for MO-GTSP

We first outline the two phases of the 2PPLS algorithm. We then show how MA-GTSP is used in the first phase, and how PLS is used in the second phase.

3.4.1 2PPLS

2PPLS [83] (Algorithm 7) consists of two phases. It takes an MO-GTSP instance consisting of graph G, clustering of cities C and objective function z as input. In the first phase an approximation P_x of the supported Pareto-optimal solutions is found through any existing single-objective algorithm. Supported Pareto-optimal solutions are those solutions that lie on the convex hull of the Pareto-optimal solutions. Such solutions are the subset of the efficient set that can be found by applying a good single-objective heuristic to weighted aggregations of the multi-objective instances (see Section 3.4.2). Figure 3.2b shows how some random initialization (i.e., the red points) is explored towards close to efficient solutions (i.e., the blue points).

In the second phase, a neighbourhood $\mathcal{N}$ is explored around each current non-dominated solution $p \in P_x$ (see Section 3.4.3). All non-dominated neighbouring solutions are added to the approximation of the efficient set. The procedure AddSolution adds the given solution to a set if and only if it is not dominated by any other solution in the set. In addition, it removes all solutions that are dominated by this solution from the set. It returns true when the solution was not dominated, and false otherwise. If a new non-dominated point is found, its neighbourhood is eventually explored, unless in the mean time it becomes dominated by a solution from a different neighbourhood. These solutions will typically lie in the grey triangles between the supported solutions. Similar to how ILS finds local optima, this approach finds a set of locally Pareto-optimal solutions [83, 94, 95].

Algorithm 7 2PPLS [83] for MO-GTSP

```
function 2PPLS(MO-GTSP $P = (G = (V, E), C, z)$)
    // First phase; generate initial approximation of the efficient set
    $P_x$ = Dichotomic_MAGTSP($P$)
    $X = P_x$
    // Second phase; Pareto Local Search
    while $P_x \neq \emptyset$ do
        Select some $p \in P_x$
        for each $p' \in \mathcal{N}(p)$ do
            if $z(p) \nprec z(p')$ then
                if AddSolution($X, p', P$) then
                    // $p'$ is not dominated by any other solution
                    // Explore its neighbourhood in a later iteration.
                    AddSolution($P_x, p', P$)
        // Finished exploring all points in the neighbourhood of p
        $P_x = P_x \setminus \{p\}$
    return $X$ // Return the approximation of the efficient set
```

3.4.2 Dichotomic scheme with MA-GTSP

For the first phase of 2PPLS, a single-objective heuristic is used to estimate the supported points of the efficient set for bi-objective problem instances. In general, multi-objective problem instances (with objectives z_1 to z_n) can be transformed into a single-objective instance by using the weighted-sum method [36]. This method minimizes a positively weighted sum of the objectives:

$$\min \sum_{i=1}^{n} (\lambda_i \cdot z_i(x)), \text{ with } \lambda_i > 0, \sum_{i=1}^{n} \lambda_i = 1$$

x represents some solution to the problem, and λ_i the weighting factors for each of the n objectives of the problem. We can find all Pareto points close to the convex hull of the efficient set by choosing different objective weights with $\lambda_i \in (0,1)$, and solving a single-objective problem.

The dichotomic scheme [4,83] provides a structured way of selecting appropriate weights for bi-objective instances, as illustrated in Figure 3.3. It first determines the range of the objectives by finding the best (or very good) solutions for either of the objectives. These solutions are the initial x_r and x_s. Based on two adjacent non-dominated solutions, the weights λ_i for each objective z_i are determined according to the normal of the line segment joining two supported points. Using these weights, a single-objective instance is created, which is subsequently solved by MA-GTSP [52] and its implementation[1]. The resulting solutions (i.e., tours) are evaluated to obtain the multi-objective values. If optimizing for the λ values for two points x_r and x_s (see Figure 3.3) has led to a supported solution x_t, then the algorithm branches to x_r, x_t and x_t, x_s. In theory, a supported point x_t could dominate one or both of the points x_r or x_s, if the heuristic found sub-optimal solutions for x_r and x_s. This is however unlikely to occur when x_r and x_s are near-optimal. If it does occur, however, the exploration terminates. It also terminates when no new supported point x_t is found.

This convex hull approximation technique for bi-objective problems can be generalized to multiple objectives [27]. It is noted that the normal vector to a 3D surface may contain negative components, and can therefore not always be used as the weights. This work provides a technique to determine the facets that require additional exploration, and how to find the associated weights. Where determining the bounding hyper-surfaces in 2D is trivial, it requires more complicated techniques for three or more objectives. In addition, determining the initial range

---

[1] http://www.cs.nott.ac.uk/~dxk/gtsp_ma_source_codes.zip

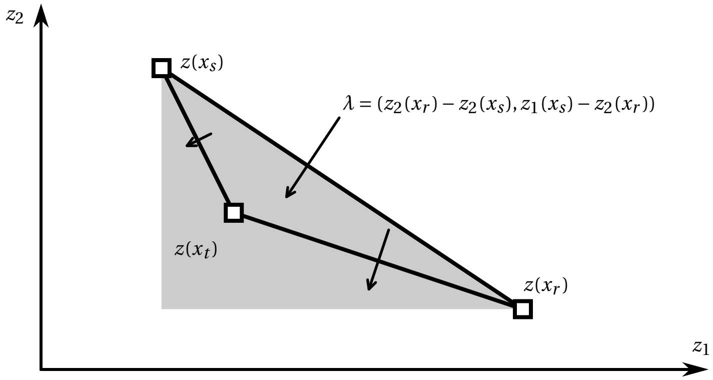


Figure 3.3: Dichotomic scheme [4,83]; use λ between x_r and x_s to optimize in a particular direction, and to find a new supported solution x_t. If x_t falls in the interior of the grey triangle, then the algorithm branches to the regions (x_s, x_t) and (x_t, x_r).

for the objective values (i.e., the Nadir point) is more complicated in multiple dimensions. As such, we focus on the bi-objective benchmarks and the bi-objective reference algorithm.

We will refer to the combination of the dichotomic scheme with MA-GTSP as dichotomic MA-GTSP. The combination of dichotomic MA-GTSP with 2PPLS is referred to as 2PPLS-MA-GTSP.

3.4.3 Neighbourhood for PLS

Local search techniques use local modifications to search a neighbourhood of a solution for improvements. For single-objective problem instances, the global optimum should preferably be reachable by several successive applications of local modifications, in not-too-large neighbourhoods. If no local modification can be made to improve a solution, then it is called a local optimum. PLS generalizes this idea to multiple dimensions; any non-dominated solution is an improvement, and is potentially locally Pareto-optimal. Keeping track of the non-dominated solutions leads to a Pareto local optimum set [94].

The solutions in the efficient set should preferably be reachable by local modifications from other non-dominated solutions. PLS heuristics for the MO-TSP typically use the 2-exchange or 3-exchange neighbourhoods [83, 94]. The standard 2-exchange respectively 3-exchange neighbourhood consists of cutting the tour into two respectively three parts, and assembling the tour in such a way that another tour is created.

We use the *generalized* 2-exchange neighbourhood for the MO-GTSP ($\mathcal{N}$ in Algorithm 7, Line 8), as introduced in [39]. The generalized 2-exchange neighbourhood changes the order in which *clusters* are visited, see Figure 3.4, while also considering the selected cities in the clusters. Consider cities x, u, v, w, where v follows w and x follows u in the tour. Let these cities be from the clusters $C_\alpha, C_\beta, C_\gamma, C_\delta$ respectively. Then we want to find cities x^*, v^*, u^*, w^* such that the cost of reversing a part of the tour leads to a non-dominated solution. In the single-objective version, this amounts to selecting the shortest path through the changed cluster ordering. For the symmetric TSP, reversing a tour only changes the values for two edges. For the symmetric GTSP, reversing a tour changes the values for at most six edges (the dashed edges in Figure 3.4). The reversed part of the tour maintains the same value. For the asymmetric (G)TSP more edges need to be re-evaluated once a part of a tour is reversed.

For the MO-GTSP we can enumerate all (possibly non-dominated) paths to find non-dominated city selections. Note that in general the bi-objective shortest-path problem is NP-Hard and intractable in the number of vertices [35]. However, the number of involved vertices (for symmetric MO-GTSP instances) is at most four. In a symmetric MO-GTSP enumerating all shortest paths for the four vertices x^*, v^*, u^*, w^* is limited to $\max_{c \in C}(|c|)^4$ paths. For asymmetric MO-GTSP instances, however, the number of non-dominated paths can scale exponentially with the number of clusters [39].

3.5 CPMH applied to MO-GTSP

The general outline of CPMH for MO-GTSP is shown in Algorithm 8. The partial solutions of CPMH for MO-GTSP are represented by tours, and their associated multi-dimensional costs. The clusters are disclosed one by one, and extend the partial solutions in each iteration. The following subsections provide the details of how we define the operators for the CPMH for MO-GTSP.

3.5.1 Initialization of partial solutions

We initialize the set S_p of partial solutions by creating a set of two-vertex tours. The Cartesian product of cities from the first two clusters is translated to a set of tours (i.e., partial solutions) in S_p by simply taking the Pareto-optimal tours for these two clusters. Operator 'min' in Algorithm 8 returns the non-dominated solutions of its argument.

3.5.2 Reduction operator

The REDUCE operator makes sure that the set of partial solutions is reduced to size $k \geq 2$ at the beginning of each iteration. There are several alternatives to compute such a reduction. For bi-objective problems, CPH [106] uses a selection mecha-

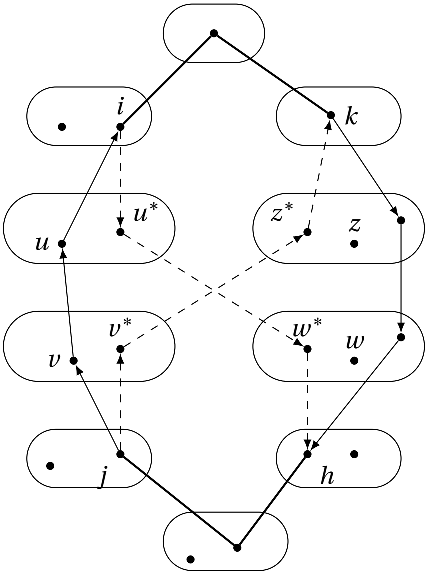


Figure 3.4: Generalized 2-exchange move [39]. While changing the cluster ordering, simultaneously change the visited cities in those clusters.

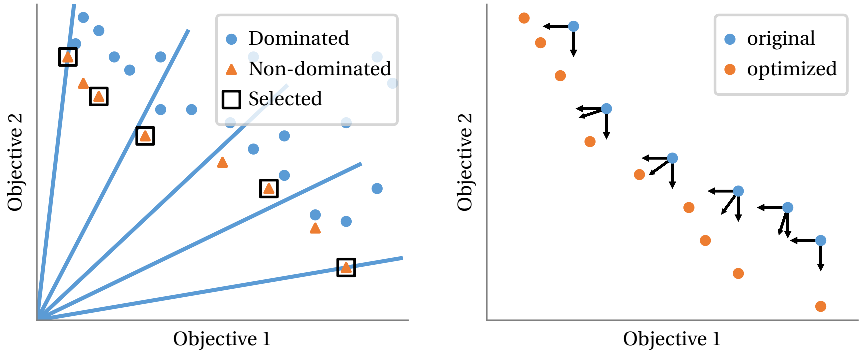
Algorithm 8 CPMH for MO-GTSP

```
function CPMH_GTSP(MO-GTSP $P = (G = (V,E), C, z)$, size of partial solutions set $k$)
    $S_p = \min(\text{INITIAL\_TOURS}(P, C_1, C_2))$
    mark $C_1$ and $C_2$ as visited
    for each unvisited cluster $C_i \in C$ do
        $S_p = \text{REDUCE}(S_p, k)$
        $S_p = \min(\text{EXTEND}(P, S_p, C_i))$
        $S'_p = \text{IMPROVE}(\text{COPY}(S_p))$
        $S_p = \min(S_p \cup S'_p)$
        mark $C_i$ as visited
    return $S_p$
```

nism that slices the 2D space into equal parts and selects one solution per slice. Figure 3.5a shows how the space between the two extreme non-dominated values is subdivided by $k-2$ lines with equal angles between subsequent lines. The operator returns the extreme points and one of the solutions per subdivision.

The archive truncation method from SPEA2 [128] can be applied to a data set of arbitrary dimensions. Given a data set, it iteratively removes the solution which has the smallest distance to other solutions, until there are only k solutions left.

The reduction mechanisms from CPH [106] and SPEA2 [128] both aim to find a set of k representatives that are evenly spaced over the objective space. We have not found significant differences in our experiments, and therefore do not distinguish between these two operators in the presented experiments.



(a) Reduction mechanism. (b) Improvement vectors.

Figure 3.5: Visualization of the effect of the reduction and improvement operators for CPMH on the objective space.

3.5.3 Extension operator

Each iteration, a cluster is disclosed and is added to the existing partial solutions, constructing new partial solutions. The extension operator inserts each city of vertex C_i into each position of a sub-tour $T \in S_p$, creating $|S_p| \cdot |T| \cdot |C_i|$ new solutions. Even though each position is used, there is no guarantee that a Pareto-optimal tour of size $|T|$ generates new Pareto-optimal tours for size $|T|+1$. Consider the example in Figure 3.6 where the goal is to minimize the tour length. The figure shows an example where the best possible insertion into an optimal tour is not an optimal tour itself.

As indicated on Line 6 of Algorithm 8, we can opt to apply Pareto-minimization to the newly extended tours, to reduce the number of tours which are optimized, in the subsequent step. This leads to a faster execution, but worse quality, compared to when minimization is omitted in this step.

3.5.4 Improvement operators

A sub-optimal tour can be improved by searching its neighbouring solutions. Another tour can be generated by swapping the position of two nodes in the tour, thereby changing four edges. Although such a swapping neighbourhood works well for certain scheduling techniques, it is famously irregular for the TSP [77], and is therefore ineffective for ILS. More effective neighbourhoods such as the 2-exchange and 3-exchange neighbourhood cut a tour into two or three parts respectively and join the parts into a different arrangement forming a tour.

The core idea of improvement operators in CPMH is that they can reconsider previous choices which may have led to sub-optimal results. Such sub-optimality may be introduced by a heuristic combination operator, by the reduction operator, or by previous optimizations. Single-objective optimization techniques can be used to optimize solutions towards a particular direction by using the weighted sum schema [36, 109] (i.e., a linear combination of all objectives), as illustrated in Figure 3.5b. This is similar to the dichotomy scheme, except that we use the fol-

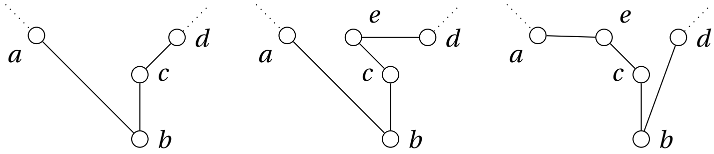


Figure 3.6: (left) Optimal partial tour a, b, c, d, (middle) best-insertion of new vertex e (creating a, b, c, e, d), (right) optimal tour a, e, c, b, d including vertex e (by a 2-exchange move to reverse sub-tour b, c, e).

lowing λ values as optimization directions for a solution x:

$$\lambda = \left(\frac{z_{1,\max} - z_1(x)}{z_{1,\max} - z_{1,\min}}, \frac{z_{2,\max} - z_2(x)}{z_{2,\max} - z_{2,\min}}\right) \text{ and } \lambda = (1,0) \text{ and } \lambda = (0,1)$$

These values are chosen such that the solutions are optimized in a direction to a value ratio that they already represent in the solution set. The extreme objectives $(z_{1,\max}, z_{2,\min})$ and $(z_{1,\min}, z_{2,\max})$ are optimized further to their extremes, to maintain a good spread over the whole range. For more than two objectives, these weights can be determined in a way similar to the generalization of the dichotomic approach given in [27].

As indicated by the current state-of-the-art [39, 52, 96], the 2-exchange and 3-exchange neighbourhoods can be applied relatively efficiently and reorder the solutions effectively to improve the results. We use the 2-exchange and reduced 3-exchange neighbourhood as described in ILS [109]. Iteratively applying these exchange neighbourhoods until no further improvement is found, leads to 2-opt and 3-opt tours. I.e., a tour is 2-opt or 3-opt when no 2-exchange respectively 3-exchange change can be made to improve the tour.

Several speed-up techniques can be applied to these improvement operators for the (MO-)TSP, such as limiting the search to a limited number of nearest neighbours [67], don't look bits [13] to avoid evaluating options that were recently considered and fixed radius search [13]. These techniques can significantly decrease the absolute running time, but do not significantly change the way these algorithms scale. The generalization of the problem to MO-GTSP does not allow trivial application of these speed-up techniques. We have therefore opted to not include these speed-up techniques in any of our implementations.

In addition to the 2-exchange and 3-exchange neighbourhoods, we have adopted the single-objective *Cluster-Optimization* heuristic introduced in the branch-and-cut approach for GTSP [39]. Given a fixed cluster order, it optimizes the selection of cities per cluster through a modified shortest-path algorithm. This heuristic has been shown to explore a very large neighbourhood [70] in polynomial time and is typically very fast. We always apply cluster optimization as the last improvement operator. The 2- and 3-exchange operators improve the ordering of the clusters, while the cluster optimization improves the selection of the nodes inside the cluster. Together, they effectively solve the simultaneous sequencing and selection problem of GTSP.

We will show two variants of our approach; one using only weighted improvement operators, and another that uses a PLS as a post-improvement step. We refer to these versions as CPMH and CPMH-PLS respectively. As performing PLS on relatively poor approximations of the Pareto fronts can lead to significant running time if a Pareto local minimum set is to be found, we instead search the generalized 2-exchange neighbourhoods of only the solutions obtained after our normal improvement round, and add the non-dominated solutions to the solution set. CPMH-PLS does not continue searching in the newly found non-dominated solutions, whereas the PLS introduced in Algorithm 7 does. This version of PLS is similar to the Pareto local search element of PD-TPLS [97].

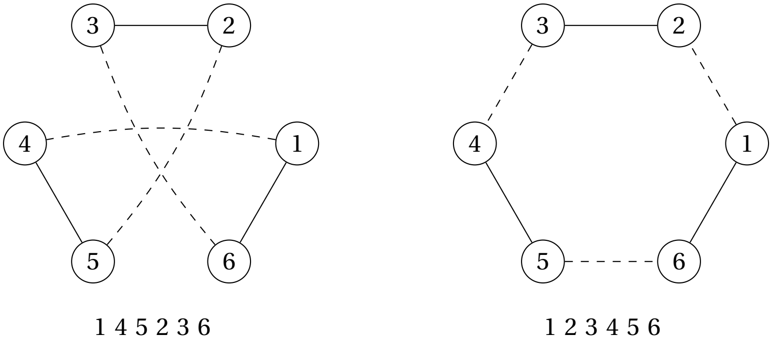
3.6 Benchmarks and experimental evaluation

We assess how CPMH performs as an offline and online MO-GTSP heuristic by comparing the runtime and solution quality to our reference algorithm integrating the state-of-the-art algorithms 2PPLS and MA-GTSP. The benchmark is described in the following subsection. We then present our method to compare the quality of the calculated Pareto fronts and finally perform the experiments that show the offline and online performance of CPMH compared to the reference. We have run all benchmark instances on an Intel Core i5-4300M notebook running Windows 10.

3.6.1 MO-GTSP benchmark

The GTSP Instances Library[2] (GTSPLIB) provides a subset of instances from the TSPLIB[3] [99] with cluster information [39]. The Krolak instances in this set represent distance problems with geographical clustering.

A single-objective GTSP definition in the GTSPLIB consists of two parts; the distances between nodes, and the clustering data. Multi-objective instances are obtained by using the distance data of two or more instances as the objectives, and the clustering data of one of the instances. The procedure that created the clustering [39] used geometrical information of the instances, and therefore the



(a) Sub-optimal tour.

(b) 3-exchange to a shorter tour.

Figure 3.7: One of the 3-exchange moves; the tour is cut in three places joined together in a different way.

---

[2] http://www.cs.nott.ac.uk/~dxk/gtsp.html

[3] https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

first objective of the MO-GTSP in our benchmark have a stronger correlation to the clustering than the second one.

We combine five instances (20KroA100 to 20KroE100), each containing 20 clusters, and two instances with 40 clusters (40KroA200 and 40KroB200) to form 22 bi-objective MO-GTSP instances. We adopt the notation used for both MO-TSP and GTSP as follows: 20KroAB100 denotes the instance with 20 clusters and 100 cities, where the objectives are defined by the 20KroA100 and 20KroB100 instances, and the clusters are defined by the 20KroA100 instance.

3.6.2 Pareto front comparison

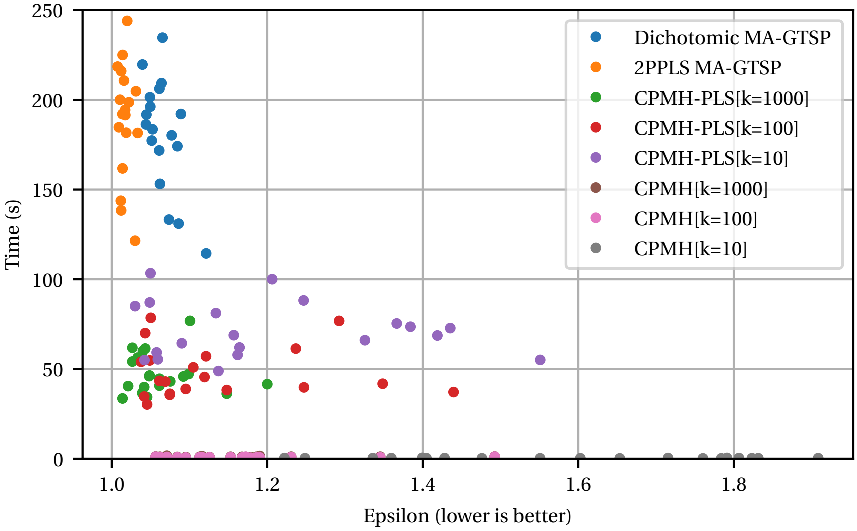
The solutions generated by the MO-GTSP algorithms should cover as much of the trade-off space as possible, while having solutions that are as close as possible to the efficient set. We approximate the efficient set by finding the non-dominated solutions in the union of all reported solutions per benchmark over all the experiments we performed.

Two Pareto fronts A,B can be compared in many different ways for an indication of the relative quality of the generated solutions [127]. We use the Epsilon $I_\epsilon(A,B)$ and Hyper-volume ratio $I_{HVR} = \frac{I_h(A)}{I_h(B)}$ indicators [126, 127] to gain insight into the relative quality of solutions. The Epsilon indicator between Pareto front A and Pareto front B is the largest scalar multiplication factor of the solution values of A for which A dominates B. The Hyper-volume ratio considers the relative surface (or hyper-volume) of the area bounded by the Pareto front and some upper bound point; we take the worst observed values in the two sets A and B per dimension. We compare each Pareto front A to the approximated efficient set s, using $I_\epsilon(A,s)$, and Hyper-volume ratio $I_{HVR} = \frac{I_h(A)}{I_h(s)}$. If $I_\epsilon(A,s)$ and I_{HVR} are close to 1, then the Pareto front A approaches the efficient set s. The Epsilon and hyper-volume indicators combined gives a better assessment of the relation between the results of two algorithms than the individual indicators [127].

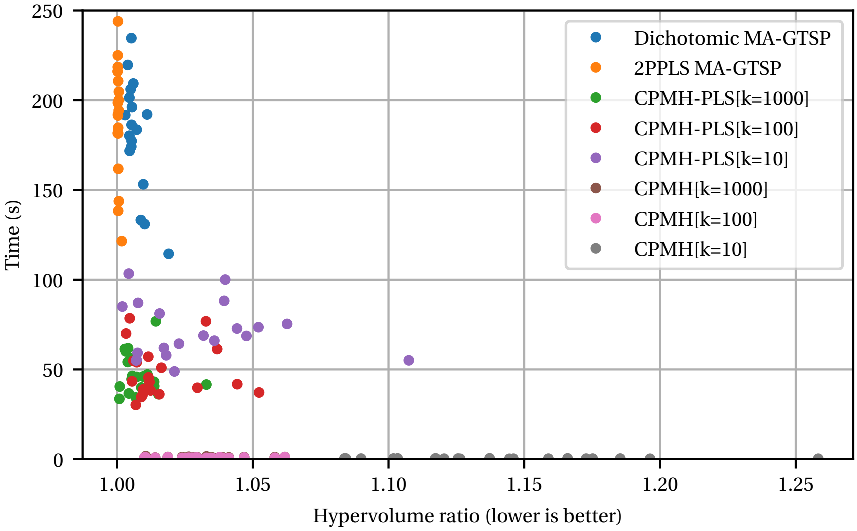
3.6.3 Offline optimization

In the offline experiment, all information is available at the start of the algorithm. The details of the experiment, including the benchmark files, and the results, can be found at www.es.ele.tue.nl/pareto/mogtsp. The trade-off between time and quality is visible in Figure 3.8; the reference algorithms have a very good solution quality (i.e., the indicators are both very close to one), but also have a very high running time. The variants of CPMH without PLS run very fast but do not yield high quality results, despite the iterative improvement operators that are employed. This figure also shows that the I_ϵ and I_{HVR} show roughly the same trends.

Interestingly, when k is smaller for CPMH-PLS, the running time tends to be longer than for larger k. We measured the time spent on the components of the algorithm, and our PLS implementation was the dominating factor. Apparently



(a) Comparison of trade-off in time and I_ϵ quality indicator.



(b) Comparison of trade-off in time and I_{HVR} quality indicator.

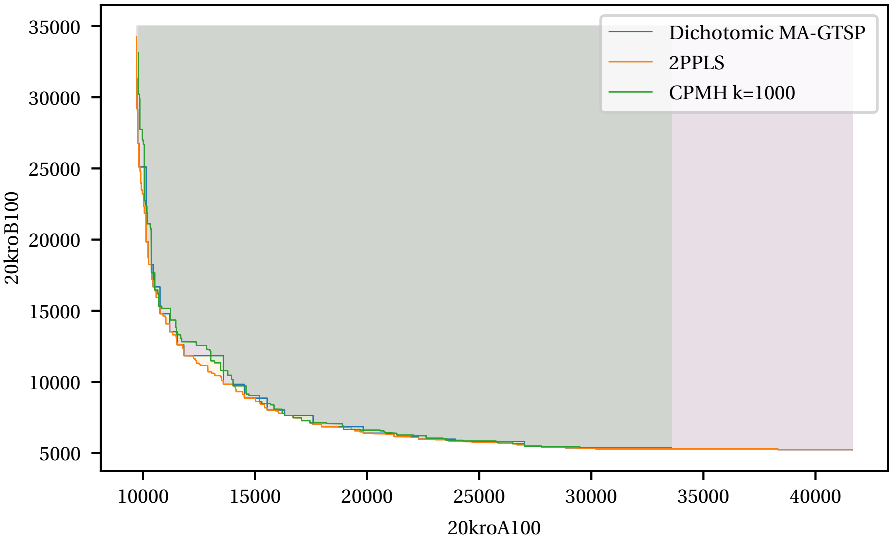
Figure 3.8: Trade-offs in quality of Pareto fronts (I_ϵ,I_{HVR}) versus time t for the benchmark instances with 20 clusters.

achieving a local Pareto-optimal set takes significantly longer when the approximation is not of high enough quality. The motivation for using a very strong single-objective solver in the first phase in [82] is based on the same observation.

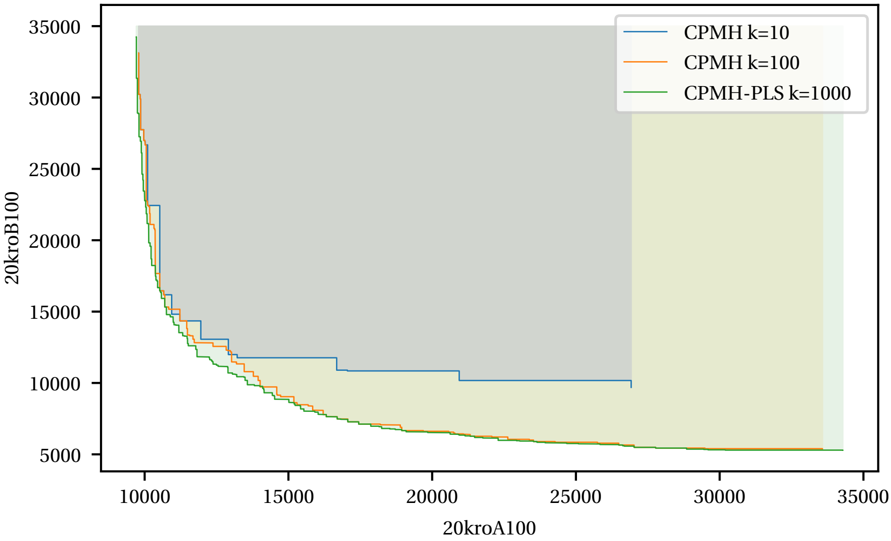
Figure 3.9 compares the results of the reference algorithm (Dichotomic MA-GTSP and 2PPLS-MA-GTSP) with several variants of the CPMH algorithm for the instance 20kroAB100. Figure 3.9a shows that CPMH with large k finds decent approximations, but fails to find a Pareto-optimal point with the optimal value for the second objective. In addition, it fails to get close enough to the reference at around objective values of 13000.

Figure 3.9b shows that for increasing parameter values of k the quality of the approximations also increases. When k is too small, the algorithm does not escape local minima often enough. The second objective seems to suffer more with regards to the decrease in k. We suspect that this is due to the clustering; the first objective has a stronger correlation to the clustering, which may make it easier to escape local minima for the first objective than for the second objective.

The results of the offline evaluation for the 22 benchmark instances are listed in Table 4.3. CPMH-PLS is three to five times faster, and approaches the reference set much better than CPMH without PLS for instances with 20 clusters. The running time of the algorithm does become a significant amount longer. This is partly due to PLS itself, but also due to the larger number of non-dominated partial solutions that are found in each iteration. The Pareto-minimization and improvement operators simply take more time in total because of the larger number of partial solutions to process.



(a) Pareto fronts generated by the references and CPMH without PLS.



(b) Pareto fronts generated by different versions of CPMH.

Figure 3.9: Pareto fronts generated for the benchmark of 20kroAB100.

Table 3.1: The bi-objective MO-GTSP benchmark results; t_{wall} is the wall clock time in seconds, I_ϵ and I_{HVR} are respectively the Epsilon Indicator and the Hyper-volume Ratio, the time t is in seconds CPU-time.

	Dichotomic MA-GTSP			2PPLS-MA-GTSP			CPMH-PLS (k=1000)		
Instance	I_ϵ	I_{HVR}	t	I_ϵ	I_{HVR}	t	I_ϵ	I_{HVR}	t
20kroAB100	1.0610	1.0046	137.872	1.0186	1.0003	147.758	1.0211	1.0010	40.475
20kroAC100	1.0861	1.0102	104.808	1.0121	1.0004	112.165	1.0139	1.0009	33.603
20kroAD100	1.0492	1.0046	161.136	1.0158	1.0004	170.428	1.0393	1.0044	36.623
20kroAE100	1.0515	1.0054	141.816	1.0093	1.0004	149.231	1.0334	1.0051	56.322
20kroBA100	1.0737	1.0088	106.600	1.0117	1.0007	117.097	1.0614	1.0054	44.556
20kroBC100	1.1214	1.0190	91.528	1.0300	1.0017	98.599	1.0753	1.0137	43.109
20kroBD100	1.0620	1.0097	122.536	1.0140	1.0005	131.144	1.0455	1.0069	34.374
20kroBE100	1.0446	1.0030	153.392	1.0221	1.0003	160.278	1.0989	1.0112	47.200
20kroCA100	1.0890	1.0111	153.672	1.0107	1.0006	161.625	1.0613	1.0137	40.712
20kroCB100	1.0771	1.0046	144.168	1.0138	1.0003	156.020	1.0431	1.0029	61.404
20kroCD100	1.0440	1.0054	149.032	1.0173	1.0006	157.060	1.0266	1.0041	61.822
20kroCE100	1.0613	1.0050	164.928	1.0122	1.0002	174.830	1.0402	1.0032	60.081
20kroDA100	1.0525	1.0072	146.864	1.0175	1.0004	154.793	1.1480	1.0152	36.240
20kroDB100	1.0496	1.0055	156.936	1.0312	1.0007	165.534	1.2001	1.0329	41.568
20kroDC100	1.0653	1.0053	187.688	1.0200	1.0003	197.037	1.0921	1.0072	45.833
20kroDE100	1.0597	1.0053	235.040	1.0143	1.0003	243.940	1.0265	1.0039	54.139
20kroEA100	1.0845	1.0052	139.320	1.0334	1.0004	146.690	1.0487	1.0056	46.370
20kroEB100	1.0641	1.0060	167.448	1.0076	1.0003	176.648	1.1007	1.0143	76.823
20kroEC100	1.0573	1.0063	215.056	1.0134	1.0007	220.778	1.0482	1.0096	46.044
20kroED100	1.0395	1.0039	175.728	1.0140	1.0003	181.058	1.0418	1.0089	39.923
40kroAB200	1.0316	1.0025	1316.55	1.0002	1.0000	1546.395	1.1908	1.0252	508.276
40kroBA200	1.0481	1.0030	1364.75	1.0000	1.0000	1661.658	1.1824	1.0300	390.341

Table 3.1 shows that the effect of the second phase of 2PPLS-MA-GTSP is small on the running time, but that it significantly increases the solution quality. The best instance of our algorithm (CPMH-PLS with $k = 1000$) finds solutions in a shorter time than the dichotomic scheme, and typically with a higher quality. The solution quality, however, is not quite as good as that found in 2PPLS-MA-GTSP. Our approach sometimes finds non-dominated solutions that are not found by 2PPLS-MA-GTSP, otherwise both quality indicators would have been equal to one for the solutions generated by 2PPLS-MA-GTSP.

All algorithms take a significant amount of time longer to compute a Pareto front for the instances with 40 clusters, 40kroAB200 and 40kroBA200. The size of the efficient set and the size of each solution increases with the number of clusters. Both of these properties lead to a longer running time. Doubling the number of clusters and cities leads to almost a ten-fold increase in time for the dichotomic scheme. The running time for CPMH-PLS is also significantly higher for larger clusters, and the Pareto fronts generated are of less quality. This is likely due to the early optimizations changing from global to local minima. The drop in quality for larger instances is possibly due to the maximum number of partial solutions becoming larger than 1000, which means that the intermediate set is reduced to a smaller size. As Figure 3.9b hints, reducing the number of intermediate solutions leads to a lower quality Pareto front, and to a higher running time due to the longer PLS phase.

3.6.4 Online optimization

We also assess how well CPMH for MO-GTSP performs as an online algorithm. To do so, we measure the time the algorithms need to (re-)compute a problem instance extended with a new cluster. The runtime for the instance 40kroAB200 is presented in Figure 3.10. All algorithms require increased runtime when the instance size grows; there are both more non-dominated solutions, the solutions are larger, and the neighbourhoods of the solutions are significantly larger.

When PLS is not used in CPMH, the running time is much smaller, yet we have already observed that this also leads to significantly worse quality. The difference in runtime between CPMH and CPMH-PLS, and between CPMH-PLS and 2PPLS-MA-GTSP is one and two orders of magnitude respectively. The larger instances suffer from more quality degradation, as indicated by the worse quality indicators in Table 3.1 for the instances with 40 clusters.

The running time for CPMH-PLS remains under a minute for adding the 40th cluster to the instance. The dichotomic scheme however, requires the full execution time, which is over 20 minutes, as it does not re-use information from previous computations. Note that the number of runs of MA-GTSP in the dichotomic scheme can be reduced to decrease the runtime by stopping the exploration, for example when a time limit is reached. The algorithm can then stop generating and evaluating new single-objective instances before its natural stopping criterion is reached. The supported Pareto-optimal points that have been found so far can then be used as the final result. However, this will decrease both the runtime as well as the approximation quality of the Pareto fronts. In addition, PLS will then take more time to ‘repair’ the quality of the Pareto front.

3.7 Conclusion

We have shown that product batch optimization corresponds to optimizing tours in the MO-GTSP. Optimizing the sequences and system modes for an FMS then corresponds to optimizing the sequencing and selection of cities in the MO-GTSP.

The state-of-the-art approaches for GTSP and MO-TSP have been combined into a new reference algorithm for the MO-GTSP. We have shown that the Constructive Pareto Meta-Heuristic combined with suitable improvement operators yields good results for MO-GTSP in a shorter running time than the reference algorithm. CPMH for MO-GTSP is suitable for online computation due to its runtime and incremental nature.

CPMH is a promising meta-heuristic to solve multi-objective, multiple-choice scheduling and packing problems online. We have shown that by incorporating local search heuristics, the runtime is much faster and the quality of solutions can approach that of the state-of-the-art non-deterministic algorithms. Depending on the time budget available, and the size of the instances, one can select one of the approaches presented in this chapter to optimize batches in FMSs.

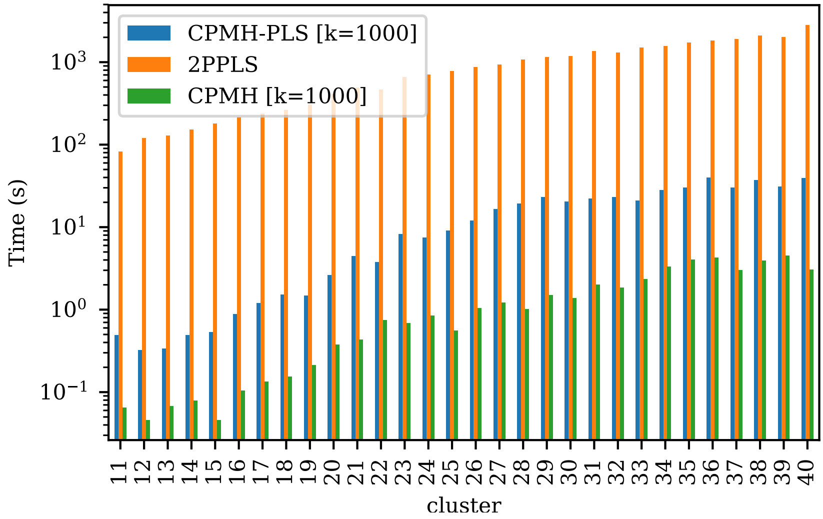


Figure 3.10: Running time when an MO-GTSP instance is extended one cluster at a time.

4

Parametric Critical Path Analysis

In the previous chapters, we have focused on online optimization of FMSs. In this and the next chapter, we contribute two design-time analysis tools that connect the performance of an FMS to multi-domain design parameters.

4.1 Introduction

High-end manufacturing systems are cyber-physical systems (CPSs) composed of several cooperating machines, which have strict timing requirements between their operations. These requirements can be modelled as minimal and maximal timing constraints between events, resulting in an event network with events and constraint relationships. Such constraints often occur due to some physical or computational process, and influence the productivity of the system. Groups of constraints typically share an underlying cause, such as a motor that actuates multiple components simultaneously. The interrelationship of the parameters influencing the constraints are explored during the design phase and one or several trade-offs are selected. It is of interest to quantify the relationship of (component) parameters to the system performance.

Scheduling activities according to a set of constraints is common in engineering [90], research and design [84] projects, and project management [71]. Identifying and alleviating performance bottlenecks is a core activity for improving the performance of schedules. The identified bottlenecks, which are critical paths in the event network, are important hints for reducing the earliest completion time of the schedules. We show that such bottleneck analysis techniques can also be extended to manufacturing CPSs.

In this chapter, we show that it is possible and useful to extend the critical path analysis [71] technique with parametric analysis, so that the interdependence between parameters is taken into account. In parametric analysis, the problem is to find solutions, e.g. critical paths, for all possible values of the parameters. Our approach first finds constraints that characterize all feasible combinations of parameters and then, for all feasible combinations of parameters, a symbolic critical path in the form of an expression in terms of the parameters of the event network.

The content of this chapter is an adaptation of the work published in CODES-ISSS 2018 [119].

This approach is related to parametric linear programming and Max-Plus sensitivity analysis, which is discussed further with the related work (Section 4.5). We show the effectiveness of our approach and observe that the scalability is primarily determined by the time to find the extremes of a polyhedron that captures a parameter value region in which the critical path does not change. Our method provides system designers with a quantitative approach to evaluate interaction between design parameters. We also show a method to efficiently determine the parameter combinations that yield Pareto-optimal performance-cost trade-offs. System designers can then take informed decisions selecting trade-offs between parameters.

Section 4.2 introduces terminology and notation of event networks. Section 4.3 first shows how to find critical paths and extends the terminology with parameters that can describe physical relations; it then shows how to find expressions for critical paths in parametrized event networks. In Section 4.4 we show the applicability of the method on two different manufacturing systems: the Twilight system [112], and a Large Scale Printer [111, 122], which was also used as a case study in Chapters 2 and 3. These two examples show that we can quantitatively relate relaxation of parameters to productivity gains of the machine. Section 4.5 positions our work in the body of existing work and Section 4.6 concludes the chapter.

4.2 Event networks

We adopt the following notation from Elmaghraby and Kamburowski [37]: an *event* is identified by $k \in \mathbb{E} = \{0, \ldots, N+1\} \subset \mathbb{N}$, and is represented by a node in a network graph. The source node 0 represents the start event and the sink node $N+1$ represents the finish event. An example network is shown in Figure 4.1. The realization time of event k is denoted t_k. The realization time of the source is fixed to 0.

A *minimal time lag* relation from event i to event j is captured in the standard form: $t_j \geq t_i + D(i,j)$. Such a relation is represented by an edge $(i,j) \in \mathbb{E}^2$ from event i to event j with weight $D(i,j) \in \mathbb{R}$. The interpretation of the minimal time lag $D(i,j)$ depends on its sign. I.e., we allow maximal time lags $L(i,j)$ from event i to event j, by transforming them into standard form [72]:

$$\begin{aligned} t_j \leq t_i + L(i,j) &\iff t_i \geq t_j - L(i,j) \\ &\iff t_i \geq t_j + D(j,i) \end{aligned}$$

That is, a *positive maximal* time lag $L(i,j)$ from i to j is equal to a *negative minimal* time lag $D(j,i) = -L(i,j)$ from j to i. Task graphs with minimal and maximal time lags may introduce cyclic time constraints. To ensure that all events are related to the source, we assume that the source has minimal time relations with zero lag to all other events. Each event, analogously, must be related to the sink, and has a minimal time relation with zero lag to the sink node.

A network graph is equivalent to a system of inequalities. A graph is feasible if and only if there exists a solution to its corresponding system of inequalities.

Equivalently, a network is infeasible if and only if it has a cycle with positive cumulative weight. The earliest *feasible* realization time $\underline{t}_k$ of an event k is the smallest number for which the system of inequalities is feasible. $\underline{t}_k$ is equal to the weight of the longest path from the source node to node k in network G:

$$\underline{t}_k = \max_{a \in paths(0 \to k)} \sum_{(i,j)\in a} D(i,j)$$

Definition 4 (makespan)**.** *The makespan M of a graph is the earliest possible realization time of the sink node, $M = \underline{t}_{N+1}$.*

$\overline{t}_k$ is the latest possible realization time of node k. The latest possible realization $\overline{t}_k$ of event k is found by subtracting the longest path from k to the sink from

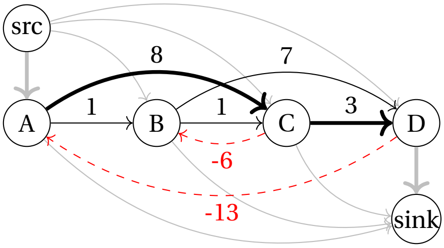


Figure 4.1: Example event graph with given time lags between events. Positive minimal time lags are shown with black edges. Negative minimal time lags are shown in dashed red edges and represent maximal time lags or, in other words, relative deadlines. The grey edges have zero time lag. The only critical path src-ACD-sink is shown with thick edges.

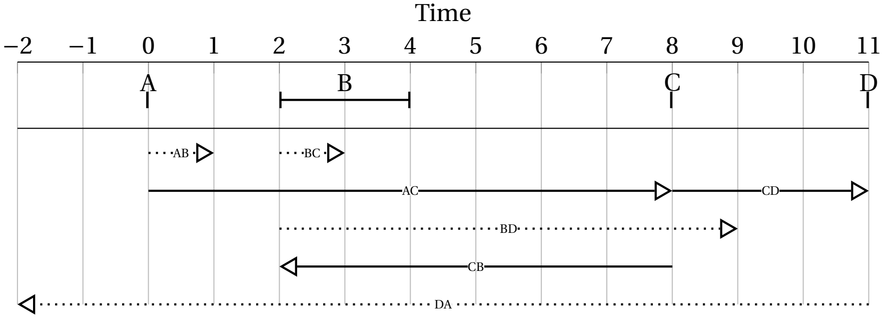


Figure 4.2: Possible realization intervals for event graph shown in Figure 4.1, shown in the top lane. The constraints are shown in the bottom five lanes as arrows, originating from the earliest realization time of its source event having length $D(i,j)$. Critical and non-critical constraints are represented with solid and dashed lines respectively.

the makespan:

$$\overline{t}_k = M - \max_{a \in paths(k \to N+1)} \sum_{(i,j) \in a} D(i,j)$$

For any graph, the latest possible realization time of the sink node is equal to the makespan: $\overline{t}_{N+1} = M$. The possible realization intervals $[\underline{t}_i, \overline{t}_i]$ of the example in Figure 4.1 are shown in Figure 4.2. Notably, the realization of event B can start at earliest 2 time units after A, even though the relation AB allows B to start one time unit after A. Path AB has length 1, while path ACB has length $8-6=2$, i.e., the earliest realization of B is bounded by the maximal time difference between B and C. The makespan of the graph in Figure 4.1 is 11, corresponding to the longest path src-ACD-sink. The latest realization of B is 4; the longest path from B to the sink is 7, and it can therefore occur in the range $[2,4]$ without affecting the makespan.

A relation from i to j has slack $S(i,j) = \overline{t}_j - (\underline{t}_i + D(i,j))$: the difference between the latest realization of the target event j and the earliest realization of the originating event i, taking into account the minimal time between i and j. The relation slack determines how much a relation can be increased without affecting the realization time of the sink node. In the example network, AB has a slack of 3. The relation $D(A,B)$ is allowed to increase by 3 time units before it starts affecting the makespan of the network. AC has slack 0, and it cannot be increased by any amount without affecting the realization time of the sink node.

Definition 5 (critical relation)**.** *A relation between i and j is critical if and only if there is no relation slack: $S(i,j) = 0$, or equivalently: $\underline{t}_i + D(i,j) = \overline{t}_j$.*

Definition 6 (critical path)**.** *A critical path is a simple sequence of connected critical relations (i.e., no event is traversed more than once), originating from the source and leading to the sink of the network.*

A longest path in a network defines the makespan of the network, and as such is equal to a critical path in the network. At least one critical path exists in the network as the latest allowed realization of the sink node is equal to its earliest possible realization. The makespan of the example network is 11 time units, and the critical path in the example network is src-ACD-sink. In this example, this is the only path which originates in the source, ends in the sink, and has zero slack for each edge in the path.

4.3 Parametric critical path analysis

In this section, we generalize the event model such that the time lags can be linear functions of parameters, and then show how to perform critical path analysis on such models. Parameters can correspond to the speed of operations, reconfiguration times, transport times, etc. An example parametrized event network is shown

Table 4.1: Path lengths of the example network in Figure 4.3.

Path	length	critical if and only if
ABCD	$2q+p$	$p \geq q+5 \wedge 2q \geq p+5$
ABD	$3q+5$	$p \leq q+5 \wedge p+q \geq 5 \wedge 3q \geq 2p$
ACD	$2p+5$	$2q \leq p+5 \wedge 3q \leq 2p \wedge 3p \geq 2q+5$
ACBD	$-p+2q+10$	$p+q \leq 5 \wedge 3p \leq 2q+5$

in Figure 4.3, where relations between events are denoted as functions of parameters p and q, such as $D(A,C) = 2p+5$. The example in Figure 4.1 is an instance of Figure 4.3 with $(p,q) = (3,1)$. There are four simple paths in Figure 4.3 from the source to the sink for non-negative values of p and q. The path lengths are listed in Table 4.1, where the last column indicates for which part of the parameter space the path length is maximal. The expression in the last column is constructed as follows. Let $\mathscr{P} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3, \boldsymbol{e}_4\}$ be the path expressions of Table 4.1. Then $\boldsymbol{e}_i \in \mathscr{P}$ is critical at point $\boldsymbol{p} = (p,q)$ if and only if $\boldsymbol{e}_i(\boldsymbol{p}) = \max_{\boldsymbol{e} \in \mathscr{P}}(\boldsymbol{e}(\boldsymbol{p}))$.

4.3.1 Overview of the approach

Our approach finds the critical paths for all feasible parameter combinations after it finds which combinations of parameter values have feasible results. The results of our approach for the example event network are illustrated in Figure 4.4. Figure 4.4 shows four regions in the 2D parameter space. In one region (blue), the network is infeasible. In each of the three other regions, different paths in the event network are critical. Each path, and hence each region, is associated with its own critical path expression. The makespan of the example in Figure 4.1 is 11 time units, which corresponds to the expression $2p+5$ at $(p,q) = (3,1)$, which is part of the yellow region in the figure. Parameters often relate to costs, e.g., faster transport is more expensive. Thus the identified regions enable trading off cost and

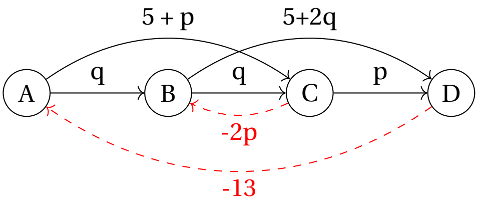


Figure 4.3: Example parametrized event network with minimal and maximal time lags. Figure 4.1 is an instance of this parametrized network with $(p,q) = (3,1)$. The source, sink, and their relations have been omitted.

performance. We provide further insight into cost/performance trade-offs in Section 4.3.6. The rest of this section shows how such expressions can automatically be found for all parameter combinations in a specified range.

We start from an existing algorithm that can be applied to non-parametrized event networks with maximal time lags (Section 4.3.2). As a first contribution, we use a symbolic version of this method to find a critical path expression for one combination of the parameters (Section 4.3.3). We prove that such critical-path expressions, as well as the conditions for which the network is feasible, are convex. Our second contribution is explained in detail in Section 4.3.4, where we introduce an algorithm that removes all infeasible parameter combinations from a parameter space for a given event network. This algorithm provides the conditions under which feasible solutions exist. Our third contribution is that we introduce parametric analysis for event networks to find critical-path expressions for each of the feasible parameter combinations (Section 4.3.5). We show that the divide-and-conquer method of [47] can be extended to find all such expressions relatively efficiently. Finally, as a fourth contribution, we show that we can find the Pareto-optimal trade-offs for linear cost functions, and provide some interpretation of the expressions found as a basis for optimization in Section 4.3.6.

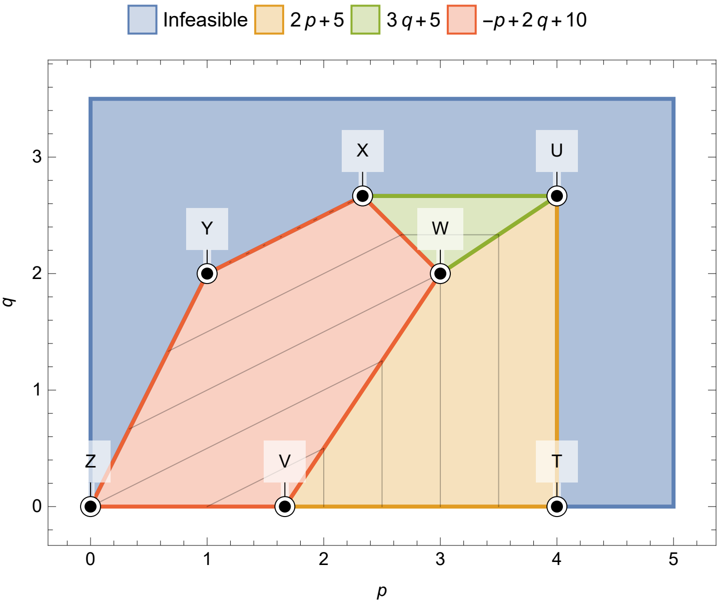


Figure 4.4: Critical-path expressions and infeasible parameter combinations for the example in Figure 4.3. The grey contour lines inside a polyhedron are iso-makespan lines. The extremes of the polyhedra are labelled with a letter.

4.3.2 Critical Path Analysis for event networks with minimal and maximal time lags

We first show how to calculate earliest and latest realizations for networks with fixed minimal and maximal time lags for a particular parameter point. Event realization times are calculated efficiently using a longest-path algorithm such as the BFM algorithm [12, 41]. BFM has been used in the (E)MPM algorithm [72, 102], as well as activity networks with generalized precedence relations (GPRs) [37]. Their algorithms also detect infeasibility in event networks.

The BFM algorithm can provide both critical paths and infeasible cycles. A network is infeasible when a cycle with positive weight exists in the network. Kerbosch and Schell [72] and Sedgewick and Wayne [103] showed that it is possible to find an arbitrary critical path by doing an as-soon-as-possible (ASAP) analysis and keeping track of the relaxations per node. In case the network is feasible, the BFM algorithm keeps track of a *parent* tree [103] that defines which of the incoming nodes was last used to relax a node. This data structure can be efficiently updated while finding the ASAP times of the events in the network. A critical path can be found by following the parent relationships from the sink node up the parent tree until the source node has been reached. Or, in case the network is infeasible, a positive cycle is found by tracking back the parent graph from the sink node until a node is encountered that has been visited already. These algorithms are detailed in Appendix A.

4.3.3 Parametrized event networks

In networks resulting from CPSs, the relations between events are typically derived from physical or control constraints that the CPS needs to adhere to [11, 112, 122]. It is often possible to parametrize the relations in the event network such that they are linear combinations of some (physical) parameters. The travelling time t of a product at a velocity v, for example, can be modelled as the linear combination of the required displacement x and displacement rate $\delta_x = \frac{1}{v}$, resulting in the linear relation $t = \frac{x}{v} = x \cdot \delta_x$. In the example parametrized network (Figure 4.3) p and q are two such displacement rates, for example of two robotic arms moving at different speeds.

We first show how to assess the feasibility and the makespan of an event network for a particular parameter through the critical path analysis detailed in Section 4.3.3. We then prove that such critical path and infeasibility expressions relate to geometrical half-spaces and form convex polyhedra in Section 4.3.3. We use the half-space representation in a feasibility detection algorithm (Section 4.3.4), after which we apply a divide-and-conquer to find all expressions in the feasible space (Section 4.3.5).

Relating critical paths and positive cycles to parameters

The parametric weight $D(i,j)(\boldsymbol{p})$ of a relation $r=(i,j)$ is a parametric affine expression $\boldsymbol{e}(\boldsymbol{p})=\boldsymbol{b}_r\cdot\boldsymbol{p}+c_r$, where $\boldsymbol{p}$ is a vector of parameters, $\boldsymbol{b}_r$ is a vector of weights, c_r is a constant and $\cdot$ denotes a vector inner product. For d parameters, the affine function $\boldsymbol{e}$ can be represented by a vector consisting of coefficients $\boldsymbol{b}_r$ and the constant c_r in the $\mathbb{R}^{d+1}$ space, and the function evaluation at a parameter point $\boldsymbol{p}\in\mathbb{R}^d$ of $\boldsymbol{e}$ becomes $\boldsymbol{e}(\boldsymbol{p})=[\boldsymbol{b}_r\ c_r]\cdot[\boldsymbol{p}\ 1]$. Consequently, the makespan for a parameter point $\boldsymbol{p}$ becomes:

$$M(\boldsymbol{p})=\max_{a\in paths(0\to N+1)}\left(\sum_{(i,j)\in a}D(i,j)(\boldsymbol{p})\right)$$

A path in the parameter space is critical if the path has the maximal weight of all paths for some combination of parameters. The makespan M can be expressed as a linear combination $\boldsymbol{b}\cdot\boldsymbol{p}+c$ of the occurrences of parameters in some critical path P for all $\boldsymbol{p}$ for which the expression of P is critical:

$$M(\boldsymbol{p})=\sum_{(i,j)\in P}D(i,j)(\boldsymbol{p})=\sum_{r=(i,j)\in P}\boldsymbol{b}_r\cdot\boldsymbol{p}+c_r=\boldsymbol{b}\cdot\boldsymbol{p}+c,$$

where $\boldsymbol{b}=\sum_{r\in P}\boldsymbol{b}_r$ and $c=\sum_{r\in P}c_r$.

A critical path expression can only be found in case the network has feasible solutions. Otherwise, instead of a critical path expression, we extract a positive cycle C causing infeasibility at a parameter point $\boldsymbol{p}$ by retrieving the expression V of that cycle C:

$$V(\boldsymbol{p})=\sum_{r\in C}\boldsymbol{b}_r\cdot\boldsymbol{p}+c_r=\boldsymbol{b}\cdot\boldsymbol{p}+c>0,$$

where $\boldsymbol{b}=\sum_{r\in C}\boldsymbol{b}_r$ and $c=\sum_{r\in C}c_r$.

Any point $\boldsymbol{p}$ for which a positive cycle exists (i.e., $V(\boldsymbol{p})>0$) is infeasible. This inequality therefore defines a half-space for which no feasible solutions exist for the network. When the cycle BCB in Figure 4.3 of length $q-2p$ is greater than zero, the network is infeasible; we therefore recognize that all points (p,q) from the parameter space for which $q-2p>0$ are infeasible.

Convex polyhedra of critical-path expressions

We adapt Proposition 5 from [47] to show that the critical-path expressions form convex polyhedra in the parameter space. The proof of that proposition also applies to Proposition 3.

Proposition 1. $\left\{\boldsymbol{p}\in\mathbb{R}^d \mid M(\boldsymbol{p})=\boldsymbol{e}(\boldsymbol{p})\right\}$ *is a convex polyhedron for any (critical path) expression* $\boldsymbol{e}$.

Any half-space s can be described by an affine expression $\boldsymbol{e}$ such that $s(\boldsymbol{e}) = \{\boldsymbol{p} \in \mathbb{R}^d \mid \boldsymbol{e}(\boldsymbol{p}) \geq 0\}$. A convex polyhedron can also be represented as the intersection of a finite set of half-spaces, each of which is represented by an expression or vector. i.e., a polygon can be represented by a subset $h \subset \mathbb{R}^{d+1}$. We lift the function s to sets of half-spaces: $s(h) = \bigcap_{\boldsymbol{e} \in h} s(\boldsymbol{e})$. A convex polyhedron can therefore be described by a set of expressions that correspond to half-spaces. Proposition 4 helps to determine the feasible parameter combinations.

Proposition 2. *If all corners of a convex polyhedron with half-space representation h are feasible, then all points in $s(h)$ are feasible.*

Proof. If there would be a parameter point $\boldsymbol{p}$ in the polyhedron (with half-space representation h) that is infeasible, then at that point a positive cycle with cumulative weight $V(\boldsymbol{x}) = \boldsymbol{b} \cdot \boldsymbol{x} + c$ must exist such that $V(\boldsymbol{p}) > 0$. The inequality $\boldsymbol{b} \cdot \boldsymbol{x} + c$ corresponds to a half-space that contains $\boldsymbol{p}$. The half-space containing $\boldsymbol{p}$ includes at least one corner point of h. Therefore, that corner point must be infeasible too. This is a contradiction and therefore proves the proposition. ☐

4.3.4 Determining feasible parameter combinations

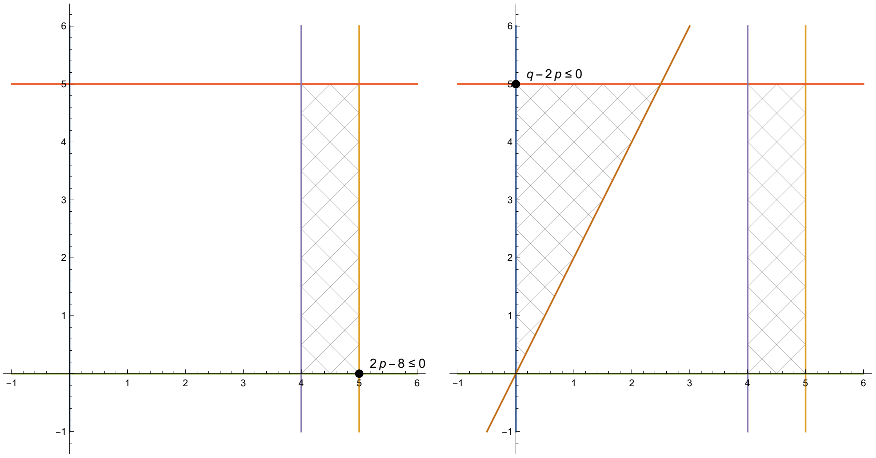
Consider a situation where a convex polyhedron (possibly the entire parameter space) is explored for infeasible points, with the goal to prune these points. Algorithm 9 removes infeasible half-spaces using the information from a positive cycle found in the graph. If BFM (in EVALUATE) finds an infeasible corner point of the polyhedron, it removes the half-space described by the positive cycle expression $V(\boldsymbol{p}) > 0$, as defined in Section 4.3.3. It does so by adding the restriction $-V(\boldsymbol{p}) \geq 0$ to the polyhedron. The algorithm continues in a recursive fashion with the remaining space until all corner points are feasible. This process is illustrated in Figure 4.5a-4.5d. The result is returned through the recursive calls. From Proposition 4 it follows that all parameter combinations in that (possibly empty) polyhedron are feasible.

Algorithm 9 Determine feasible parameter combinations

```
function DETERMINEFEASIBLEPOINTS(event network $G$, convex polyhedron $CP$)
    for each corner $c_i$ of $s(CP)$ do
        feasible, cycle_expression = EVALUATE($G, c_i$)
        if ¬ feasible then
            $CP' = CP \cup \{-\text{cycle\_expression}\}$
            return DETERMINEFEASIBLEPOINTS($G$, $CP'$)
    return $CP$ // All points in CP are feasible
```

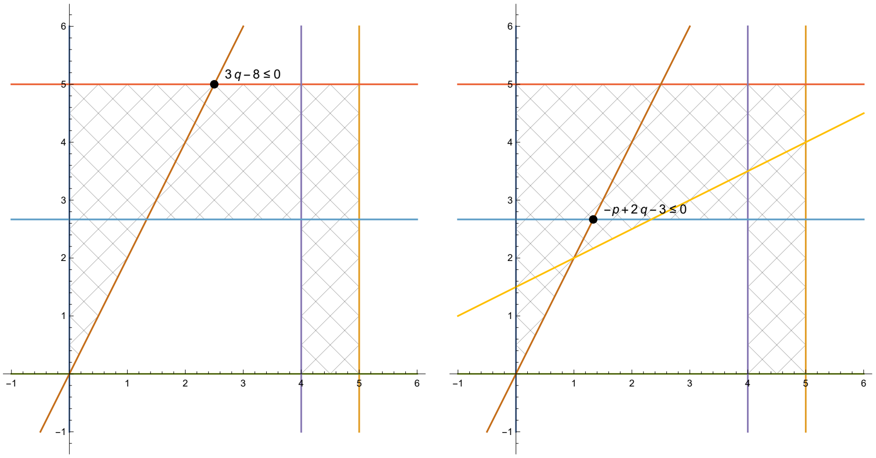
4.3.5 Divide-and-conquer approach

The polyhedron obtained from Algorithm 9 contains only feasible points, and allows the use of the divide-and-conquer approach presented in [47] (Algorithm 10). That approach finds all critical-path expressions for all parameter combinations by partitioning the parameter space into smaller and smaller pieces until an ex-



(a) Alg. 9: remove infeasible region (ACDA) $2p-8>0$

(b) Alg. 9: remove infeasible region (BCB) $q-2p>0$

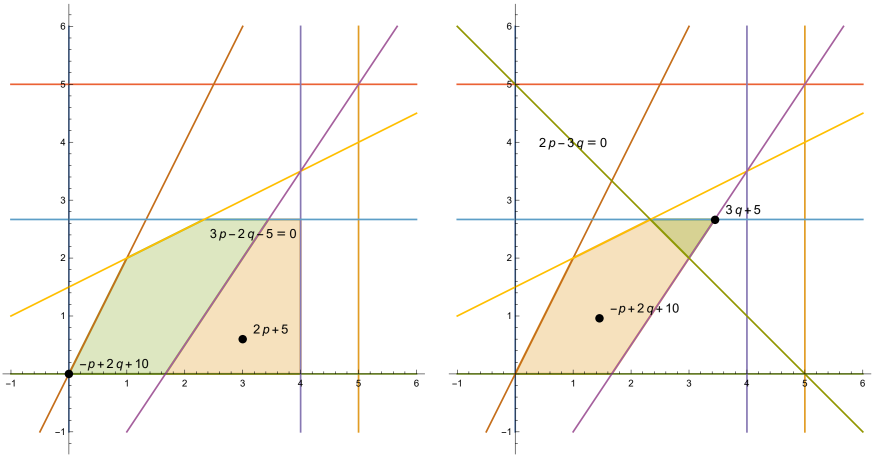


(c) Alg. 9: remove infeasible region (ABDA) $3q-8>0$

(d) Alg. 9: remove infeasible region (ACBDA) $2q-p-3>0$

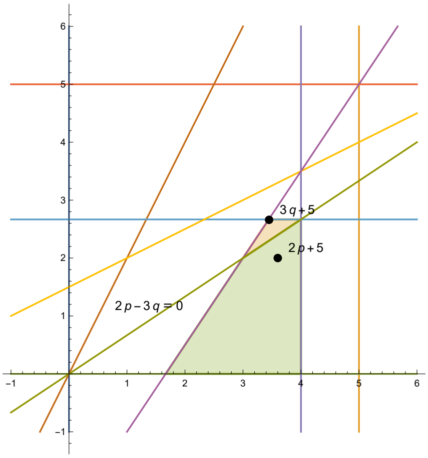
Figure 4.5: Illustration of our method for the example in Figure 4.3. p and q both range from 0 to 5.

pression is found that holds for all corner points of the polyhedron. When an expression e_i found in the interior of a polyhedron does not hold for one of its corner points c with expression e_c, the polyhedron is split across the hyperplane $e_i = e_c$; i.e., to one polyhedron the equation $e_i - e_c \geq 0$ is added and to the other $e_i - e_c \leq 0$. For the example in Figure 4.3, the algorithm starts from the feasible parameter combinations found in Figure 4.5d. It finds expression $M(p,q) = 2p+5$ for



(e) Alg. 10: expression $-p+2q+10 \neq 2p+5$ at $(3, 3/5)$, split by $3p-2q-5=0$

(f) Alg. 10: expression $-p+2q+10 \neq 3q+5$ at $(16/11, 24/25)$, split by $-p-q+5=0$



(g) Alg. 10: expression $2p+5 \neq 3q+5$ at $(36/10, 2)$, split by $2p-3q=0$

Figure 4.5: Illustration of our method for the example in Figure 4.3. p and q both range from 0 to 5. (Cont.)

the point $(p, q) = (3, 3/5)$, which does not hold at corner $(0,0)$ as $M(0,0) = 10$ (Figure 4.5e). The expression $M(p,q) = -p+2q+10$ is found at $(0,0)$, and we therefore split the polyhedron into two pieces, by the equation $-p+2q+10 = 2p+5$. This process is repeated two more times as illustrated in Figure 4.5f and 4.5g.

Once an expression is found that holds for all corner points of a polyhedron, the algorithm stops exploring that part of the parameter space and continues with another polyhedron. For the two polyhedra found in Figure 4.5f the algorithm finds two expressions that hold in these polyhedra respectively. It continues with a polyhedron for which no expression has been found yet in 4.5g. The results of Algorithm 9 and 10 are combined in Figure 4.4.

The divide-and-conquer approach of [47] is reproduced in Algorithm 10. This algorithm requires a feasible solution to be found for an arbitrary point inside the polyhedron, along with an expression that holds for that point.

Algorithm 10 Divide-and-conquer

```
function DIVIDECONQUER(event network $G$, convex polyhedron $CP$)
    $\boldsymbol{p}$ = random point in $s(CP)$
    $\boldsymbol{e_c}$ =FIND_EXPRESSION($G$, $\boldsymbol{p}$)
    for each corner $\boldsymbol{c}_i$ of $s(CP)$ do
        $\boldsymbol{e}_i$ = FIND_EXPRESSION($G$, $\boldsymbol{c}_i$)
        if $e_c(\boldsymbol{c}_i) \neq \boldsymbol{e}_i(\boldsymbol{c}_i)$ then
            // Divide CP into two polyhedra
            $CP_1 = CP \cup \{\boldsymbol{e}_i - \boldsymbol{e}_c\}$
            $CP_2 = CP \cup \{\boldsymbol{e}_c - \boldsymbol{e}_i\}$
            $S_1$ = DIVIDECONQUER($G, CP_1$)
            $S_2$ = DIVIDECONQUER($G, CP_2$)
            // Return all expressions found in polyhedra
            return $S_1 \cup S_2$
    return $\{\boldsymbol{e}_c\}$ // Expression holds for each corner
```

Unfortunately, no efficient algorithm is possible for generating extreme points (i.e., the corners) of a convex polyhedron [42]. When all parameter combinations are feasible, Algorithm 10 initially needs to transform a non-degenerate set of half-spaces in d dimensions into 2^d corner points. Enumerating an exponential number of corner points is expensive, but it turns out to be feasible for a moderate number of parameters (e.g. < 15). Furthermore, the number of calls to DIVIDECONQUER is at least the number of expressions to be identified, and possibly more. The polyhedron for $3q+5$ for the example of Figure 4.3 is found by combining the results of two different sub-problems (Figure 4.5f, 4.5g). Each call to FIND_EXPRESSION costs at most $\mathcal{O}(|E||R|)$ where $|E|$ is the number of events (nodes), and $|R|$ the number of relations (edges).

In comparison with the modified BFM algorithm of Levner et al. [79], we see that their approach is more efficient for a single parameter, but it cannot take into account multiple parameters simultaneously. Their modified algorithm finds the critical paths in $\mathcal{O}(|R|^2 \cdot |E|)$ time when the parameter coefficient on the relations is chosen from $\{-1,0,1\}$. For arbitrary integer values, the approach takes $\mathcal{O}((b \cdot |R|)^2 \cdot |E|)$, where b is the largest parameter coefficient occurring on any edge. Our method solves a generalized version of the problem of [79], where parameters can be rational numbers, and multiple parameters are taken into account. For a given number of parameters, the running time of our approach depends on the number of regions to be found, the time to evaluate a particular parameter point, and the time to convert the half-space representation into corner points. The number of regions that will be found depends on the parameter range, which is selected by the designer.

4.3.6 Assessing the parameter regions

The results of our approach can be used to determine the quantitative impact of decreasing/increasing parameters. The rate of change per unit of reduction is found in the gradient ∇ of the expression in the region around the current set-point or working point. The gradient component for a single parameter i at a point with expression $\boldsymbol{e} = (b_1, \ldots, b_d, c)$ is equal to the aggregate contribution b_i of parameter i to the critical path, i.e., $\nabla_i \boldsymbol{e} = b_i$. In Figure 4.4, $\nabla M(3,1) = (2,0)$ and $\nabla M(1,1) = (-1,2)$. Changing the parameter by Δ_i will impact the makespan by $\boldsymbol{b}_i \cdot \Delta_i$, as long as the critical path expression still holds for the new parameter point, i.e., it lies in the same region.

We assume that the makespan and a cost function, say $C(p,q) = p - 2q$ in the example of Fig 4.4, are both to be minimized. Typically, the optimal cost is found at a different parameter combination than the optimal makespan, and therefore trade-offs exist between makespan and cost. For example, starting from point $X = (7/3, 8/3)$, any point on the line segment XW in Figure 4.4 closer to W has shorter makespan $M(p,q) = -p + 2q + 10$ and higher cost $C(p,q) = p - 2q$. On the other hand, the cost-makespan trade-off at X is preferred over any of the cost-makespan points found on the line XU. The parameter selection can be improved by following the gradient such that the cost decreases or stays the same, and the makespan decreases, or vice versa. A parameter combination is called dominated iff it is worse in at least one of the two aspects and not better in the other than some other parameter combination.

In general, we want to find all Pareto-optimal cost-performance trade-offs in the parameter space. The Pareto-optimal parameter combinations are those for which no other parameter combinations exist that dominate it. For an arbitrary linear cost function $C(\boldsymbol{x}) = \boldsymbol{\alpha} \cdot \boldsymbol{x}$, the cost and makespan can be computed for each point in the space. All Pareto-optimal parameter combinations can be found by projecting the parameter-space to the cost-performance trade-off space, finding the trade-offs in that space and translating them back to the parameter combinations. Transforming a polyhedron from the parameter space to the cost-makespan space, where $M(\boldsymbol{x}) = \boldsymbol{b} \cdot \boldsymbol{x} + c$ is the associated expression, is defined by:

$$T(\boldsymbol{x}) = \begin{bmatrix} M(\boldsymbol{x}) \\ C(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} & c \\ \boldsymbol{\alpha} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{\alpha} \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} c \\ 0 \end{bmatrix} = \boldsymbol{P}_1 \boldsymbol{x} + \boldsymbol{p}_2$$

where $\boldsymbol{P}_1$ is the matrix $\begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{\alpha} \end{bmatrix}$ and $\boldsymbol{p}_2$ is the vector $\begin{bmatrix} c \\ 0 \end{bmatrix}$.

Applying the affine transformation T to all points of a convex polyhedron in the parameter space yields a new convex polyhedron in the cost-makespan space. It is sufficient to transform the extreme points of the polyhedron to obtain the polyhedron in the cost-makespan space. An example of such a projection is shown in Figure 4.6. The transformed corner points of Figure 4.4 are labelled with the same letters, and the regions have the same colours. The Pareto-optimal corner points are efficiently identified in the cost-makespan space by applying algorithms such as Simple Cull [125] to the finite set of corner points. Point U in Figure 4.6 is dominated by point Y and X, and point T is dominated by the points V, Z and W. The set of Pareto-optimal corner points is the set of remaining, non-dominated points V, Z, W, Y, X.

The Pareto-optimal line segments can be found as follows. First determine the centroid (i.e., the mean of the corner points) and assign a counter-clockwise direction to the Pareto-optimal corner points [49], such as V, Z, W, Y, X. Now each line segment between Pareto-optimal corners has a direction, it can be interpreted as a vector. If this vector between two subsequent Pareto-optimal corner points $\boldsymbol{c}_i$, $\boldsymbol{c}_{i+1}$ has a direction such that $M(\boldsymbol{c}_{i+1}) - M(\boldsymbol{c}_i) > 0$ and $C(\boldsymbol{c}_{i+1}) - C(\boldsymbol{c}_i) < 0$ then each point that lies on the line segment between these points is Pareto-optimal as well [32]. In the example, all points on the line segments between corner points V and X (Y) are Pareto optimal.

Transformation T is not necessarily bijective. That is, multiple parameter combinations $\boldsymbol{x}_i, \boldsymbol{x}_k$ may map to the same makespan-cost trade-off $(M(\boldsymbol{x}_i), C(\boldsymbol{x}_i)) = (M(\boldsymbol{x}_k), C(\boldsymbol{x}_k))$. Each Pareto-optimal point $(M(\boldsymbol{x}), C(\boldsymbol{x}))$ in a convex region in the trade-off space can be translated back to the parameter space through the pseudo-inverse of the transformation associated with that convex region:

$$\boldsymbol{x} = T^{\dagger}(M, C) = \boldsymbol{P}_{\boldsymbol{1}}^{\dagger} \left(\begin{bmatrix} M \\ C \end{bmatrix} - \boldsymbol{p}_{\boldsymbol{2}} \right)$$

Each Pareto-optimal point $(M(\boldsymbol{x}), C(\boldsymbol{x}))$ on the Pareto-optimal line segments maps to the space $(T^{\dagger}(M(\boldsymbol{x}), C(\boldsymbol{x})) \oplus K(\boldsymbol{P}_{\boldsymbol{1}})) \cap E$ where E is the polyhedron corresponding to the transformation T, and $K(P1)$ is the kernel of $P1$. For two subspaces A and B, $\oplus$ is their extension: $A \oplus B = \{\boldsymbol{a} + \boldsymbol{b} \mid \boldsymbol{a} \in A, \boldsymbol{b} \in B\}$. Each extension of a kernel and a point in the parameter space is a subspace of the parameter space.

In the example of Figure 4.6, the line segment XW of region $3q+5$ has a transformation T with full rank $\boldsymbol{P}_{\boldsymbol{1}}$, and each point on the line segment therefore corresponds uniquely to a point on the line segment XW in the parameter space of Figure 4.4. Similarly, the line segment VW for region $2p+5$ maps uniquely to the line segment VW in the parameter space. However, all points of the 2D polyhedron for the expression $-p+2q+10$ in Figure 4.4 have been mapped to points on the line VX in the trade-off space. Its corresponding $\boldsymbol{P}_1$ has less than full rank. The kernel of transformation $\boldsymbol{P}_1$ is the same for all of the Pareto-optimal line-segments, and these line segments map to the intersection of two half-spaces. Each point in the region $-p+2q+10$ is therefore a Pareto-optimal parameter combination: for example, the point Y in the cost-makespan space maps to a line $2p+q+d=0$ such that the line passes through $Z=T^{\dagger}(13,-3)=(8/5,4/5)$ in the parameter space, i.e., $d=-4$. On this line, the combinations of p and q are such that the makespan remains the same, i.e., they coincide with an iso-makespan line, and also such that the cost does not change. As one point on the line has been shown to be Pareto-optimal, each point in the region that falls onto that line is also Pareto-optimal. The resulting Pareto-optimal parameter combinations are thus the polyhedron for $-p+2q+10$.

4.4 Case studies

We describe two case studies and perform parametric temporal critical path analysis on them. We investigate the relative speeds of two robot arms for the Twilight system [112], and a specific reconfiguration aspect of the print head of a industrial printer [111]. Algorithms 9 and 10 have been implemented in C++ and run on a 64-bit Ubuntu machine with a 3.0Ghz Intel Core i7 950 processor. We have used the C-library of the Double Description method [42] in combination with the GMP library [51].

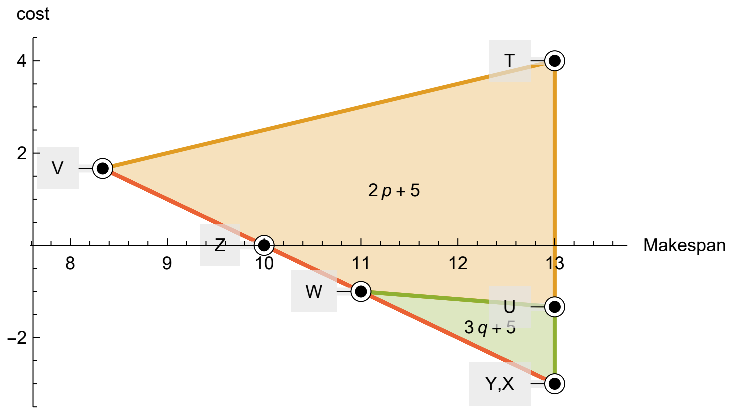


Figure 4.6: Cost-makespan trade-off space for $c(p,q)=p-2q$ for Figure 4.3. The labelled corners correspond to those in Figure 4.4.

4.4.1 Twilight System

The Twilight System [112] (see Figure 4.7) is an example created for the study of controller synthesis and performance analysis and optimization of manufacturing systems. The manufacturing system processes balls that need to be drilled. Before drilling is allowed, the ball needs to be conditioned to the right temperature. First, a ball is picked up at the input buffer by the load robot (LR). Once it is brought to the conditioner (COND) it is processed immediately. Once the conditioning of the ball has finished, it immediately needs to be transported by either one of the robots to the drill (DRILL), where it is drilled before the conditioning of the ball expires. Finally, the drilled ball is transported to the output by the unload robot (UR). Figure 4.8 depicts a simple schedule where the unload robot moves the product from COND to DRILL. Consider that the time it takes for these robots to travel one unit of distance at movement speeds v_{LR} and v_{UR} is $LR = 1/v_{LR}$ and $UR = 1/v_{UR}$ respectively. Increasing LR and UR corresponds to reducing the movement speeds.

The robots can travel either horizontally or vertically, but not diagonally, and always need to move to the highest vertical position before moving horizontally. The horizontal distance between input and conditioning is 10 units, from conditioning to drilling and drilling to output is 5 distance units each. The vertical distance to the input box is 3 units, to the conditioning and drilling platforms 1 unit, and the ball is allowed to be released at any height above the output buffer. Handing over a ball is synchronized and immediate. Conditioning takes exactly 9 time units and expires 8 time units after conditioning has finished. Drilling takes exactly 3 time units. The processing rate of the conditioning, C, and the drilling D are fixed to 1. UR and LR range between 0 and 1.5. Even though they are fixed, C and D are still annotated as parameters in the model to distinguish their contribution to the critical paths.

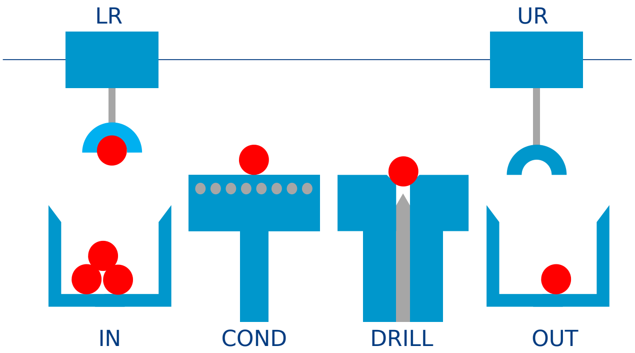


Figure 4.7: Twilight manufacturing system, from [112].

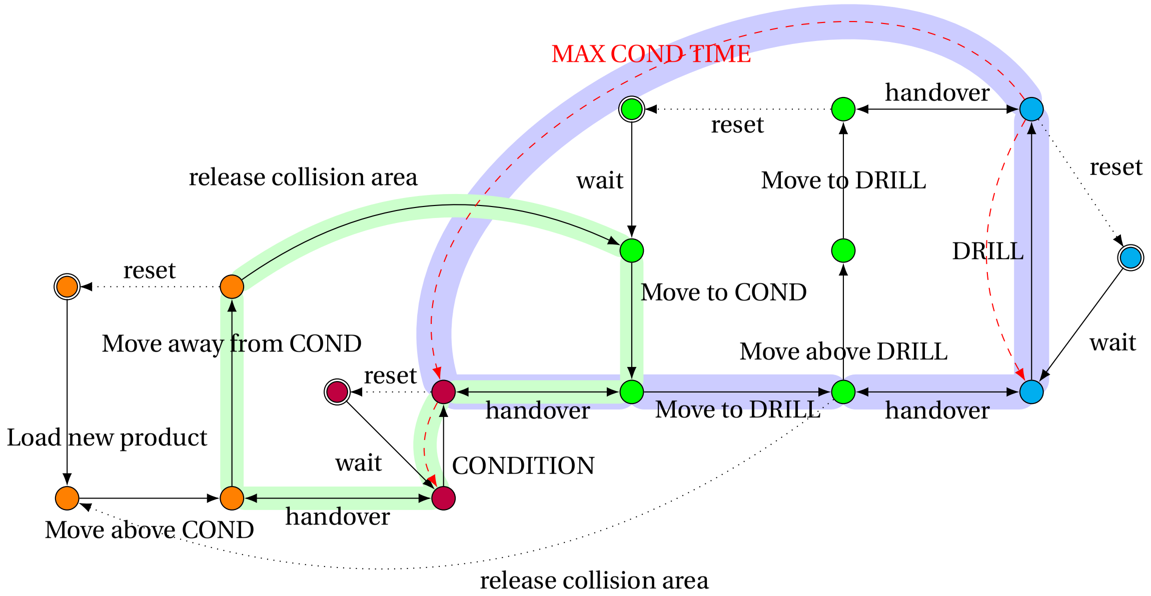


Figure 4.8: Example life-cycles and scheduling dependencies for the Twilight System (Figure 4.7); the Load Robot (orange) loads a product and delivers it to the conditioning stage (purple). The Unload Robot (green) picks it up and needs to deliver it to the Drill (cyan) before the conditioning expires. Dotted edges are dependencies from the current product to the next product. The two positive cycles are shown with green and blue highlighted edges.

Figure 4.9 shows the critical-path expressions as function of UR, LR, C, and D. For the given ranges, two positive cycles are detected (blue region). UR and LR become too large to meet the required maximum time between conditioning and finishing the drilling, leading to a positive cycle. In the other cycle, the time it takes for one robot to move away and the other to pick the ball from the conditioner is too large, and the conditioning deadline is violated. Three critical path regions denote the behaviour of the schedule for particular combinations of the robot travelling rates. As LR and UR tend towards zero, the movement speed of the robots becomes infinitely fast, and the makespan becomes $M(0,0) = 90C + 3D + 26LR + 70UR = 90C + 3D$. This is a lower bound on the performance imposed by the time needed for the drilling and conditioning process.

In the green region, changes in UR have the highest impact on performance. The unload robot performs some movements in parallel with the drilling process. Before the unload robot becomes too slow to pick up the ball immediately after drilling ends, there is another cycle that becomes positive. When the unload robot is faster than drilling, the drilling time is always in the critical path, as shown by component $30D$. In the yellow region, the load robot is most often in the critical path. Even though the unload robot performs most actions, our result shows that improving the load robot's speed will give the highest gain. The load robot's speed is in the critical path more often due to the scheduled dependencies. In the red region, the conditioning process becomes the most important bottleneck, especially when LR and UR tend towards zero; the smallest possible makespan for 10 products is $M(0,0) = 93$ time units. This is significantly less than the largest discovered makespan $M(1.5,0) = 483$ time units.

These results show that optimization efforts can be based on the relative and

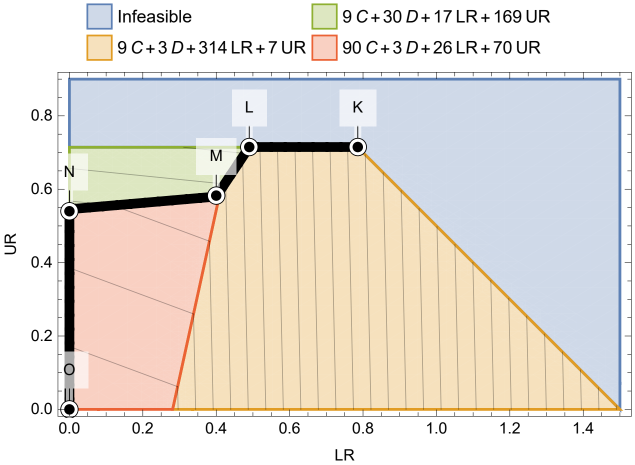


Figure 4.9: Expressions for the Twilight system.

absolute travelling rates of the system. This makes it possible to investigate the interaction of system parameters and performance. Discussing such relations can be valuable when deciding with different stake-holders about the performance of the systems components, such as the robot travelling speeds, conditioning times and interdependencies. Also cost-performance trade-offs are possible. Figure 4.9 shows the Pareto-optimal combinations for a cost function $C(LR, UR) = -LR - 3UR$ with a thick black line.

4.4.2 Large-Scale Printer

The paper path of an industrial printer [111] is defined as a path that sheets follow in the printer. The paper path consists of several motors, switches, and functions that perform actions on the sheets. The sheets are guided on a metal track and their speed and acceleration are controlled by pinches. Figure 4.10 shows the topology of a paper path. The sheets need to move twice through the image transfer station (ITS) before going to the output. Duplex sheets enter the duplex loop (DL), and a turn track (TT) reverses the sheet's direction, for printing on the opposite side. The sheets return from the duplex loop to the merge point (MP) within a pre-defined interval from their first print. The sheets are not allowed to overlap or collide with the sheets coming from the paper input module (PIM). When the sheet has been processed fully, it leaves the printer through the finisher (FIN).

The acceleration profiles of sheets are determined almost completely beforehand; a pre-determined buffer region is used to somewhat slow down the sheets. The range of this buffer is encoded as a minimum and maximum travelling time by the vertical edges in the example event network (Figure 4.11). Even though the relation between acceleration profiles and minimum and maximum loop times is nonlinear, separating these two variables still allows them to be modelled as linear constraints. The horizontal and diagonal edges in the example encode the non-overlapping constraints.

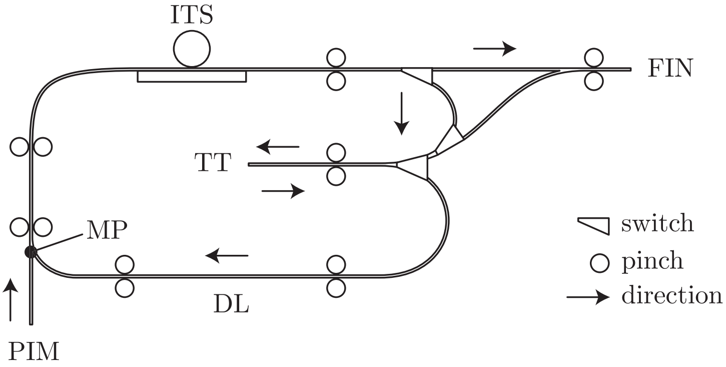


Figure 4.10: Industrial printer schematic overview, from [111].

Table 4.2: Sheet specifications.

(a) Sheet details

	L	H
A: A4	210	0.25
B: A3	420	0.1
C: A3+	483	0.3

(b) ITS reconfiguration times

		Current		
		A4	A3	A3+
Prev.	A4	0.26	4.51	2.01
	A3	4.78	0.53	6.03
	A3+	2.35	6.10	0.60

The sheets can have highly varying specifications, and the modules therefore may need to reconfigure themselves to another operating point to achieve the required quality. One such reconfiguration occurs at the ITS; the print head may need to be raised or lowered between sheets to achieve the proper print gap distance for image quality. The print head height H, for example, can be modelled as a linear movement, which can be started after the previous sheet has been fully printed (i.e., has left the ITS). The ITS is ready for the next sheet when it has moved for its full length L, and the print head moved and the stabilization time γ has passed. That is, the time that the ITS can start processing the current product ($t_{ITS,cur}$) is limited by the start of the previous process ($t_{ITS,prev}$), and the time it takes to process the previous product and prepare for the current product:

$$\begin{aligned} t_{ITS,cur} &\geq t_{ITS,prev} + \Delta_t \\ &= t_{ITS,prev} + \frac{L_{prev}}{v_{ITS}} + \frac{\left|H_{prev} - H_{cur}\right|}{v_{vert}} + \gamma \\ &= t_{ITS,prev} + \alpha L_{prev} + \beta \left|H_{prev} - H_{cur}\right| + \gamma \end{aligned} \tag{4.1}$$

As the speeds v_{ITS} and v_{vert} are constant, we can use parameters $\alpha = 1/v_{ITS}$, $\beta = 1/v_{vert}$. The equation then becomes a linear expression, as H_{prev}, H_{cur}, L_{prev} are all sheet-dependent and assumed constant. The constraint simplifies to the following when no head movement is needed $t_{ITS,cur} = t_{ITS,prev} + \alpha L_{prev}$. Decreasing the value of α means increasing the speed at the ITS. Decreasing β means increasing the speed of the vertical movement. Decreasing γ means that oscillations dissipate faster, perhaps due to higher damping in the system.

A repeated pattern of duplex sheets is fed into the printer, i.e., $(ABC)^{60}$. The symbols A, B, C refer to one of the types of sheets denoted in Table 4.2a and move at the ITS at $v_{ITS} = 800 mm/s$. The nominal speed of the head movement is 0.04 units per second.

Let's assume that the ITS is the bottleneck; the PIM and FIN are always ready to provide/receive a sheet. In the event network for the schedule for this print job (Figure 4.11), each sheet returns to the merge point in the interval $t_r \in (10, 15)$ from the first time at the merge point. The sequence of first prints and second prints that the scheduler has chosen leads to a requirement on the reconfiguration times between passes in the sequence (diagonal edges).

The critical-path expressions associated with different combinations of β and γ are shown in Figure 4.12. Depending on β and γ, the majority of the time in the critical path is spent on either: (α) moving the sheet under the ITS, (β) head movements, (γ) oscillations, (l_{min}) travelling through the loop. As β and γ decrease, they also occur less often in the critical path. The performance of the system is lower bounded by the loop time l_{min}, which occurs 26 times in the region with the lowest makespan. The bottleneck changes from the loop time to the head movement parameters as β and γ increase. The expressions show that α becomes relevant for performance, even though only β and γ are varied in the experiment. Eventually, β and γ become so large that l_{max} is violated.

For $\beta \geq 0.9$, the iso-makespan lines are much closer together, showing that from this point onward, the influence of β and γ are much higher. In these pa-

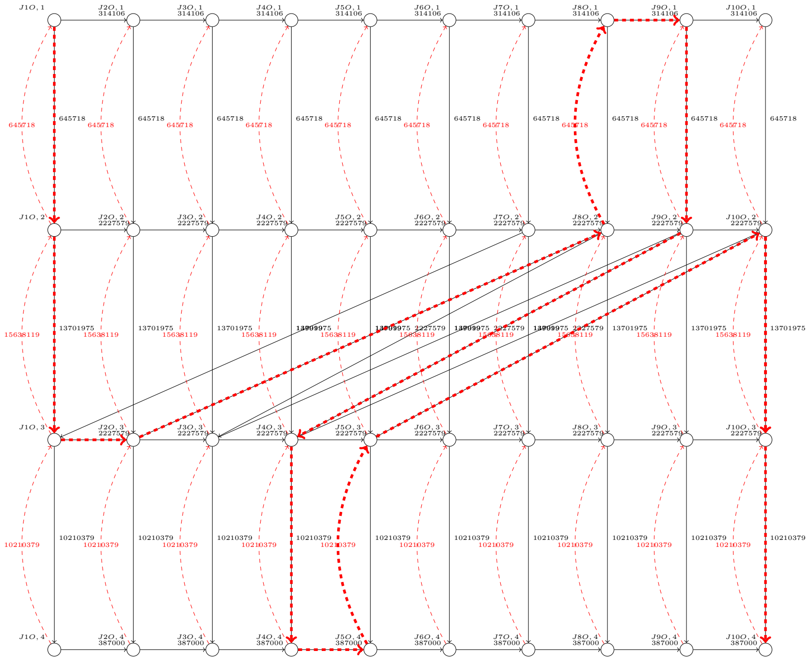


Figure 4.11: Example event network and critical path for an industrial printer flow shop instance with 10 products. Each column describes a product travelling through the flow shop, and each row shows the operations on each product. The second and third operation (rows) are mapped onto the same machine. Sequencing edges (diagonal) ensure that at most one product occupies the machine at any time.

rameter ranges, the makespan is more sensitive to the head movement rate β than to the settling time γ, as can be seen from the contribution of these components in the critical-path expressions. One can conclude from this analysis that it is more meaningful to spend development effort on increasing the head movement speed, rather than reducing the settling time.

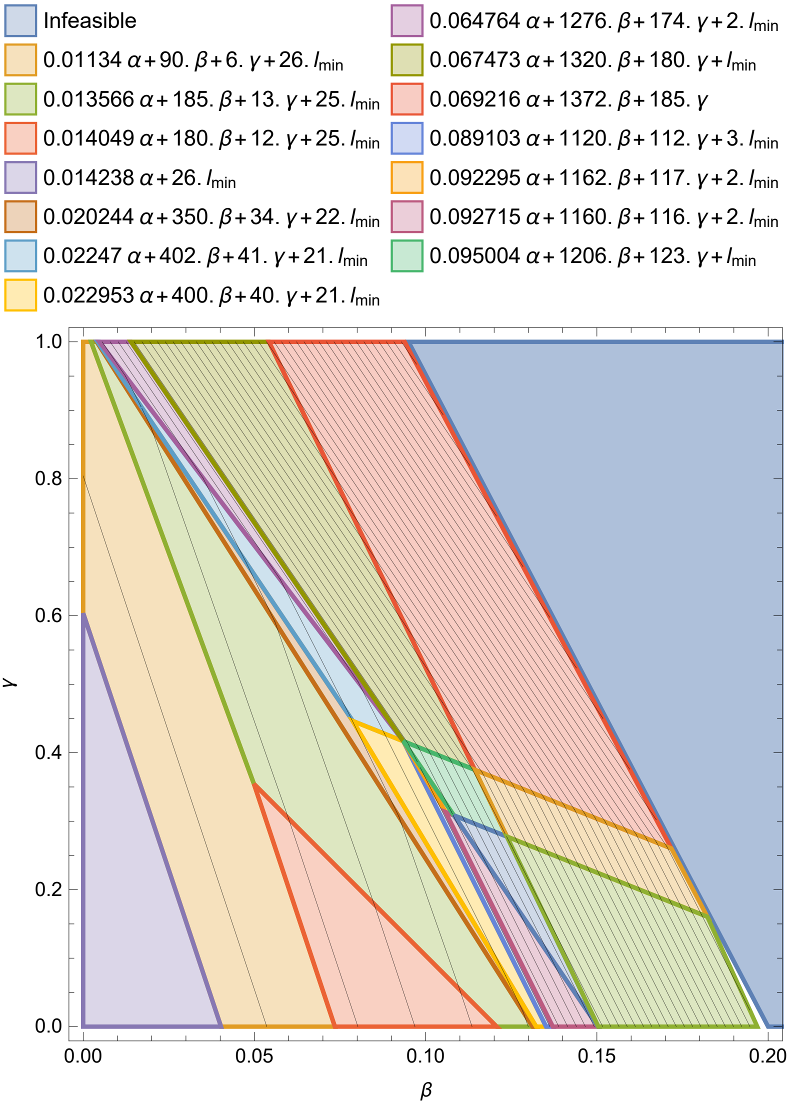


Figure 4.12: Example critical-path expressions for an industrial printer.

Table 4.3: Experimental results of parametric critical path analysis

	Event network		Parametric Critical Path Analysis			
Instance	$\|E\|$	$\|R\|$	t (s)	paths	eval.	splits
Example (Figure 4.3)	6	15				
$p \in (0,1), q \in (0,1)$			0.03	3	19	3
Twilight (Figure 4.9)	162	605				
$UR \in (0,100), LR \in (0,100)$			0.355	3	32	7
$UR \in (0,100), LR \in (0,100), C \in (0,1)$			0.631	3	61	12
$UR \in (0,100), LR \in (0,100), C \in (0,1), D \in (0,1)$			1.502	3	167	27
Industrial printer	362	1785				
$\beta \in (0,0.4), \gamma \in (0,1)$			9.315	17	210	60
$\alpha \in (0,1.25), \beta \in (0,0.4), \gamma \in (0,1)$			26.7	23	574	152
$\alpha \in (0,1.25), \beta \in (0,0.4), \gamma \in (0,1), l_{min} \in (0,10)$			71.9	40	1709	430
Packing 200 products	6614	21265				
$UR \in (0,1)$			59	55	279	92
$UR \in (0.9,1), LR \in (0.9,1)$			231	99	1078	99
Packing 500 products	16514	53114				
$UR \in (0,1)$			373	136	714	237
$UR \in (0.9,1), LR \in (0.9,1)$			5484	665	9387	2988
Packing 1000 products	33014	106243				
$UR \in (0,1)$			1492	262	1404	467
$UR \in (0.9,1), LR \in (0.9,1)$			49236	2639	39378	12634
Packing 2000 products	66014	212430				
$UR \in (0,1)$			6484	551	2976	991
Packing 3000 products	99014	318561				
$UR \in (0,1)$			14995	852	4569	1522

4.4.3 Evaluation

We have evaluated the DETERMINEFEASIBLEPOINTS and DIVIDECONQUER algorithms on several event networks using different parameter ranges. The examples that have been presented so far are summarized in Table 4.3. The illustrated examples have been extended by including more parameters. All experiments are run on an Intel Core i7 950 at 3GHz.

The Packing instances are variations on the Twilight example which only contain minimal time lags. Each product is modelled with 30 events, and dynamic relations exist between events of subsequent products. The time constraints of the relations have been determined stochastically by drawing from several PERT distributions, and are associated with their respective machines, UR or LR. The resulting network is a rather large directed acyclic network without negative constraints, and as such the topological sorting as used in Critical Path Method (CPM) [71] has been used in our experiment, instead of the slower, but more generic BFM.

We observe that the number of critical paths found in the packing cases grows with $\mathcal{O}(|E|)$ when only the UR is parametrized, and with $\mathcal{O}(|E|^2)$ when both the LR and UR are parametrized. We conjecture that the number of critical paths grows in general with $\mathcal{O}(|E|^d)$ where d is the number of parameters.

Due to the growing number of critical paths and corner points of regions for problems with multiple parameters, both the number of splits and number of evaluations go up. The time taken for each evaluated point remains roughly the same for each network instance, and does not depend on the number of parameters.

4.5 Related work

Event and activity networks have been used in project modelling and other kinds of scheduling problems to model minimal and maximal time lags between events or activities. We use the project modelling approach started by Roy [102], and Kerbosch and Schell [72], and further developed by Elmaghraby [37]. The time lags make it possible to describe constraints on the feasible time ranges between events in a CPS. In this chapter, we use minimal and maximal time lags to express constraints between events in terms of parameters.

The CPM [71] and GPRs [37] are used to model event networks, and also to find critical paths in such networks. The CPM and GPRs use the difference between the occurrence of an event in an ASAP and as-late-as-possible (ALAP) schedule to determine the slack of events/activities. Those with zero slack are critical, and any delay in any of these critical events leads to an according delay in the completion of the network. It does not, however, give the designer any insight into how much system productivity is gained when critical constraints are relaxed. Some critical path methods assume stochastic time lags and are applied to observations of systems, such as the shifting bottleneck detection [101]. In this method, the machines that are most often active are regarded as the bottleneck. This method gives an indication of the bottlenecks over time. However, this approach does not show the impact of deadlines, as imposed by maximal time lags, and can therefore hide the true interaction of different machines, especially when deep pipelining behaviour occurs. Stochastic time lags have also led to investigations of the stochastic criticality index, such as shown in [9]. In this chapter, we assume that time lags are deterministic, and focus on the interaction between parameters.

As described in [55], when cost slopes are known for shortening activities (also known as *crashing*), CPM can be used with Linear Programming to give the efficient trade-offs between project duration and project (direct) cost. There are several methods available that can be applied to networks that do not have maximal time lags (Fondahl's method, Siemens method) [55]. The work of [37] generalizes these crashing methods such that maximal time lags can be taken into account. These works, however, do not take into account that the underlying cause for each constraint can be shared. That is, each constraint is independently relaxed, and it is assumed that other relationships remain the same. Ignoring interdependencies between parameters in CPSs and FMSs leads to incorrect conclusions from a system perspective.

Critical path *drag* has been introduced in [30] as the amount of time a particular relation can be decreased before another path becomes critical. Drag is defined as the minimum slack on parallel paths. Parallel paths, however, are not well-defined for graphs that contain cycles. Maximal time lags are not taken into account by *drag*. Each time lag must have a value that is independent of the others.

Approaches using parametric analysis for single or multiple parameters have been developed for many kinds of problems. Generic techniques such as parametric linear programming [43] [38] use a symbolic simplex algorithm and symbolic Gomory cuts to find the lexicographical minimum candidates to both continuous and (mixed-)integer problems. It is intended to solve small parametric problems to find integer values as a sub-procedure for automatically rewriting (i.e., optimizing) computer programs. Instead of the lexicographical minimum we are interested in finding the more typical economical minimum (i.e. a linear preference function as objective). It is possible to use the parametric linear programming approach to find the economic minimum. The approach can replace the divide-and-conquer algorithm for finding makespan expressions. The approach also has a worst case complexity that is exponential in the number of parameters.

Max-Plus arithmetic has been used to determine the sensitivity of assembly line parameters in the context of manual workstations [104]. Kobayashi has presented some theoretical results on the adjacency of critical paths using Max-Plus tropical geometry [73]. The algorithm of Kobayashi assumes that each relation is independent of all other relations, while we are interested in finding the interrelation between parameters.

The modified Bellman-Ford-Moore (BFM) algorithm of Levner [79] efficiently finds all critical paths for the feasible values of a *single parameter* for integer edge weights. Our approach allows an arbitrary number of parameters, with rational numbers as values and weights. Graphs with rational values can be used in the approach of Levner [79] after multiplying all expressions with an integer factor such that all rational numbers become integer values.

Most methods for multiple parameters are based on the observation that parameter combinations occur in convex regions [47, 58, 63]. Parametric analysis of synchronous data-flow graphs has been used to determine, for example, the parameter combinations for which the same maximum cycle mean expression holds [28, 47]. The divide-and-conquer approach from [47] shows that performance expressions can be found even when the parameters do not necessarily take integer values. In this chapter, we prove that this parametric analysis can be applied to event networks with maximal time lags.

Some work has been carried out to identify for which parameter combinations parametrized models have feasible solutions. The work of Elmaghraby [37] introduces *flexibility* of an activity/constraint in an activity network as the amount of time it can be increased/decreased such that all constraints can still be met in a schedule; i.e. it indicates when networks become infeasible. The convexity of parameter regions has also been used by [14] to determine for which parameter combinations a periodic fixed priority system is schedulable. Interaction between tasks and their relation to system utilization are modelled with fixed priority scheduling rules, and precedence relations between tasks are explicitly not allowed.

Several parametric methods have also been developed for timed automata [23, 63]. In [63] a subclass of parametric timed automata has been identified for which it can decide on the emptiness problem when automaton variables are upper and lower bounded by parameters. A parameter may either occur in lower bounds, or in upper bounds, but never in both. The approach of [63] can detect parameter combinations for which the system does not have conflicting requirements, i.e., is feasible. Our extension of the parametric method allows parameters both in lower and upper bounds (minimum and maximum time lags). In [23], the feasibility of activating a set of real time tasks is checked through parametric timed automata. Similar to the work of [14] the approach does not allow precedence constraints, as it models interaction between tasks through periodic activation patterns.

Finding Pareto-optimal performance-cost trade-offs for a single convex region is equivalent to finding Pareto-optimal facets in a multi-objective linear program. Such problems have received attention in their general form since the 1980s [32]. Current research contributes on relations between problems [81] and improvements of search algorithms [34] are still relevant research areas. These algorithms are limited to find Pareto-optimal/efficient solutions in a single convex region. We provide a method that can find Pareto-optimal faces for sets of convex regions in two-dimensional spaces.

We extend this existing body of work by finding all feasible parameter combinations and their associated performance characterization through parametric temporal analysis of event networks with minimal and maximal time lags. The quantitative analysis allows finding Pareto-optimal performance-cost trade-offs.

4.6 Conclusion

We have shown that it is possible and useful to identify quantitative relationships between system parameters and system performance when the timing constraints in schedules are annotated in linear combinations of design parameters. If the parameters have non-linear relationships, the non-linear model can be split into several models, for which linearised expressions hold around a working point. We have shown that parametric analysis can be applied to the classical Critical Path Method such that regions of critical-path expressions and infeasibility expressions are found. These regions and expressions are used to quantify the impact of a parameter change, such as a settling time or a robot travelling rate, on the makespan of the generated schedules. The results of our approach give insight into the interrelationships between design parameters.

We use a divide-and-conquer approach to find the critical-path expressions in the parameter space. This chapter shows for two manufacturing CPSs how to interpret such relations by combining this information with the original parametrized graph. We also show that Pareto-optimal parameter combinations can be found by transforming the found polyhedra to the cost-makespan space, finding the Pareto-optimal points in that space, and translating the results back to the parameter space. These results can form a sound basis for discussing the trade-offs between cost and performance.

The analysis in this chapter focuses on the relation between parameters and the productivity for a particular schedule. In the next chapter, we extend the parametric analysis to include the discrete decisions that schedulers make.

5

Parametric Scheduler Characterization

In the previous chapter, we have shown a technique to find expressions for the performance of an event network (representing a schedule of, e.g., flow shop operations) in terms of design parameters. In this chapter, we show that it is possible to find which schedule is used for which parameter combinations. It is useful to understand how the parameters of a system impact the scheduling decisions that are made. The parameters can again be used to relate physical quantities, i.e., timing constraints, to their underlying cause. In Chapter 4, we have shown that if the event network is known, the bottlenecks in the system can be quantified. It is therefore of interest to find which event network is used to compute the earliest possible realization times of events for each parameter combination. We show that it is also possible to model some of the decision mechanisms that are used in deterministic schedulers.

We consider deterministic schedulers that need to enforce orderings, while meeting certain requirements on the timing of events in the system. The technique requires a scheduler to (re-)construct symbolic conditions for its decisions. We use these symbolic conditions in a divide-and-conquer algorithm to find regions for which the same parametric result is produced. Expressions for the makespan of the result can then be found. A key challenge is to determine the parameter combinations to evaluate such that the parametric decision regions are characterized including their borders, where a scheduler changes its decision because of some critical constraint or requirement.

We first define symbolic scheduling (Section 5.1) and illustrate how to integrate it into a classical scheduling heuristic (Section 5.2). We show how such a symbolic scheduler is used to evaluate a concrete parameter combination (Section 5.2) and how to interpret the information it returns (Section 5.3). In Section 5.4, we introduce an exact scheduler characterization algorithm that covers all parameter combinations in a given range. In Section 5.5, we evaluate the algorithm for the classical Shortest Processing Time First (SPTF) scheduler and schedulers for a manufacturing system. Related work is discussed in Section 5.6. In Section 5.7, we conclude this chapter.

5.1 Symbolic scheduling

Schedulers decide in what order and at what time events should take place, to optimize some objective. A scheduling problem often contains a set of static timing constraints that are enforced regardless of the scheduler's decisions. For example, there may be a minimum and maximum time constraint between the start and the end of a task. Conditional timing constraints, such as reconfiguration times or sequence-dependent set-up times, are active only when certain conditions are met. The scheduler needs to take into account that the set of active constraints can change when the order of events or the resource allocation changes. Many schedulers first decide on which resource and/or in what order the required events should take place, before start times are determined. Such decisions on resource allocation and ordering are typically encoded in an event network by adding time constraints between events. The relations in the event network can express minimal or maximal time lags between events, similar to the previous chapter. The scheduler must ensure that all timing constraints of the scheduling problem are satisfied. A sequence of decisions is taken, based on the evaluation of certain alternatives. The scheduler then computes the earliest possible event realization times from the event network, so that they can be used to instruct the manufacturing system. We can view schedulers as algorithms that enforce relations between events in such a way that a feasible schedule is generated for the scheduling problem, while optimizing some objective.

Schedulers typically use concrete values to choose between options. In symbolic scheduling, these concrete values are replaced by symbolic expressions, that relate the values to system parameters. The scheduler can then evaluate the symbolic expressions by substituting the parameters with their concrete values. We show that such a symbolic scheduler can determine to what extent the parameters may change before any decision in the decision sequence changes. In this chapter, we show that if the conditions used in the scheduler are restricted to conjunctions of affine inequalities of parameters, then we can relatively efficiently determine for all parameter combinations which decision sequence will be taken. We assume that a symbolic scheduler returns the following information:

- the schedule produced at a particular parameter point,
- a set of necessary symbolic conditions, i.e., affine inequalities on the parameters, defining a region that includes all points that lead to the same decision (i.e., outside of which, the scheduler will definitely take a different sequence of decisions),
- the result of a concrete decision sequence, e.g., the parametrized timing relations between events (e.g., in the form of a parametrized event network).

All symbolic conditions in the set should hold at the parameter point for which the scheduler returns them. The set of symbolic conditions contains conditions that are necessary for the decisions made by the scheduler.

5.2 Running example

We first introduce a basic scheduling model: an m-machine n-task allocation problem. We then explain the SPTF heuristic. We use this particular scheduler to show that our approach is applicable to commonly used types of schedulers and as running example.

5.2.1 m-machine n-task allocation problem

A queue[1] $T = \langle (t_1, d_1), \ldots, (t_n, d_n) \rangle$ of tasks t_i with duration $T(t_i) = d_i$ needs to be allocated on a set of m identical machines. The tasks have no dependencies among each other. The objective is to complete the tasks and return both machines to idle state as quickly as possible, i.e., minimize the maximum completion time $C_{\max}$ of all machines. Once a task $t_i \in \operatorname{dom} T$ has started on a machine, it is executed without interruptions for the specified duration $d_i = T(t_i)$. This means that if task t_i starts at time $\sigma(t_i)$, then no other task can start on that machine in the interval $(\sigma(t_i), \sigma(t_i) + d_i)$. The problem is to assign each task to a machine and determine a feasible schedule σ of realization times for the start of each task such that $C_{\max}$ is minimized. This optimization problem has been shown to be NP-hard for $m \geq 2$ [78], but this problem has been selected due to its easy description and many relevant simple heuristic scheduling strategies.

We use a simple list scheduler heuristic as an example for our approach. List schedulers iteratively schedule the *highest ranked task* onto the *highest ranked resource*. It suffices to define task and resource ranks to complete the algorithm. I.e., a list scheduler is typically used together with ranking methods, such as outlined in Algorithm 11.

Algorithm 11 List scheduling outline

```
function LIST_SCHEDULING(task queue $T$, resources $R$)
    // initialize empty queues Q for each resource
    for each $r \in R$ do
        $Q(r) = \langle \rangle$
    repeat
        take a task $t$ from $T$ with highest task rank and remove $t$ from $T$
        $r$ is the resource from $R$ with highest resource rank
        append $t$ to $Q(r)$
        $\sigma(t)$ = earliest start time for $t$
    until $T$ is empty
    return $Q, \sigma$
```

---

[1] A queue is a set for which the index of the elements denote a total ordering. We implicitly use this notation to indicate that the elements of a set are ordered.

5.2.2 Shortest Processing Time First Scheduler

An SPTF scheduler is a list scheduler that iteratively schedules a remaining task with the shortest processing time on the earliest available resource (Algorithms 12 and 13). The result is a partitioning of the task queue over resource queues. The resource queues represent dependencies between tasks due to the shared resources, for example $\langle (t_1, a), (t_2, 2a) \rangle, \langle (t_3, b) \rangle$ expresses two resource queues of tasks with parametric execution times with parameters a and b in which t_2 depends on the completion of t_1, both allocated to the first resource.

At each step, the scheduler must select exactly one option; a tie-breaking condition is required if several options are equally preferable. If several tasks or resources have the same duration or availability time, then Algorithm 12 always selects the first occurrence, which makes it deterministic. In literature, such a tie is typically broken arbitrarily.

Algorithm 12 Shortest processing time first scheduler

```
function SPTF_SCHEDULE(task queue $T = \langle (t_1, d_1), \ldots, (t_n, d_n) \rangle$, resources $R = \langle r_1, \ldots, r_m \rangle$)
    for each $r \in R$ do
        $Q(r) = \langle\,\rangle$ // empty queues Q for each resource
    repeat
        remove first task $x = (t_x, d) \in T$ with duration $d = \min_{(t_i, d_i) \in T} d_i$
        select first resource $r \in R$ with the smallest COMPLETION($Q(r)$)
        $s$ = COMPLETION($Q(r)$)
        append $x$ to $Q(r)$
        $\sigma(t_x) = s$
    until $T$ is empty
    return $Q, \sigma$
```

Algorithm 13 Calculate completion time of a task queue on a processor

```
procedure COMPLETION(queue $q$)
    return $\sum_{(t_i, d_i) \in q} d_i$
```

5.3 Parametrized schedulers

We introduce the concept of decision regions, taking the shape of polyhedra in the design parameter space, in Section 5.3.1, before extending the SPTF scheduler with symbolic scheduling in Section 5.3.2. In Section 5.3.3, we show how to extend the evaluation of conditions in the symbolic scheduler such that it can evaluate points that are arbitrarily close to, but not on, the edge of a polyhedron. We use this concept to deal with tie-breaking conditions, characterizing the parameter combinations where a scheduler changes its decision. In Section 5.3.4, we visualize the branching conditions used in the symbolic scheduler as a decision tree.

5.3.1 Decision regions

We consider deterministic schedulers for which the conditions of scheduler decisions can be denoted as the conjunction $\bigwedge S$ of a set S of symbolic affine inequalities. The universe of symbolic affine inequalities with d parameters is represented by $U = \mathbb{R}^{d+1} \times \{<, \leq\}$. We denote a symbolic affine inequality by a tuple $(\boldsymbol{e}, \lhd)$ with $\boldsymbol{e} \in \mathbb{R}^{d+1}$ and $\lhd \in \{<, \leq\}$. A symbolic scheduler evaluates parametric affine expressions $\boldsymbol{e}(\boldsymbol{p}) = \boldsymbol{b} \cdot \boldsymbol{p} + c$, where $\boldsymbol{p}$ is a vector of parameters, $\boldsymbol{b}$ is a vector of weights, c is a constant and $\cdot$ denotes a vector inner product. A symbolic affine inequality can be evaluated to a boolean at a parameter point p as $\boldsymbol{e}(\boldsymbol{p}) \lhd 0$, where $\lhd$ is either $<$ or $\leq$. Note that in Chapter 4, inequalities were always inclusive inequalities $\leq$. In this chapter, we need their strict counterparts. The negation of an affine inequality $\neg(\boldsymbol{e} \lhd 0)$ is defined as follows: $\neg(\boldsymbol{e}, <)$ is equivalent to $(-\boldsymbol{e}, \leq)$, and $\neg(\boldsymbol{e}, \leq)$ is equivalent to $(-\boldsymbol{e}, <)$.

A conjunction of symbolic affine inequalities can be interpreted as the intersection of half-spaces in the parameter space, as shown in Proposition 3 in Chapter 4. The resulting polyhedron is convex. For strict inequalities, the bounding hyperplane is excluded. We want to find a polyhedron for every point in which, the scheduling algorithm makes the *same sequence of decisions*. We use the term *decision region* to refer to such a convex polyhedron.

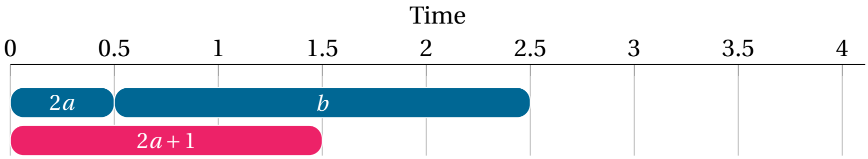
Algorithm 12 returns the task ordering $Q(r)$ for each resource r, which implicitly represents an event network, and for each task $t \in \operatorname{dom} T$ the earliest possible starting time $\sigma(t)$. In addition, the scheduler can keep track of the reasons why it made these decisions for the given parameter values. For example, let the queue of symbolic tasks and their duration expressions be $\langle (t_1, 2a), (t_2, b), (t_3, 2a+1) \rangle$. The selection condition $d = \min_{(t_i, d_i) \in T} d_i$ (used in Algorithm 12 Line 5) can be written as the conjunction of inequalities such that the duration d_x of task t_x is at most the duration of any task in T: $\bigwedge_{(t_i, d_i) \in T} d_x \leq d_i$. When $a = 1/4$, $b = 2$, the scheduler picks task t_1 with duration a first, and will do so for other parameter combinations, as long as $2a \leq b$.

Similarly, the minimum completion time condition can also be expressed as a symbolic inequality, as the completion time of a resource is the sum of a set of symbolic duration expressions. In the example, after selecting task t_1, the first, least busy resource is selected to execute for an additional $2a$ time units. Both resources are available at time 0, and therefore task t_1 is scheduled on r_1. The completion times of r_1 and r_2 are then $2a$ and 0 respectively. Task t_3 with time $2a + 1 = 3/2$ then remains the smallest task as long as $2a + 1 \leq b$. Resource r_2 will be selected for task t_3 under the condition that there is no other resource than r_2 that is idle earlier, i.e., the scheduler requires that $0 \leq 2a$. Moreover, as resource r_1 occurs earlier in the resource list than r_2, the scheduler also requires that r_2 is the *first* resource *in the list* that is idle at the given time, i.e., the scheduler requires that $\neg(2a \leq 0)$. The two conditions combined simplify to $0 < 2a$. After scheduling t_3, the resource completion times are $2a$ and $2a+1$ for r_1 and r_2 respectively. Finally, the only remaining task t_2 with duration b is added to the resource with minimum completion time. The resource selection condition $2a \leq 2a+1$ is true regardless of the value of a, and t_2 is therefore scheduled on resource r_1. This schedule is shown in Figure 5.1a. For this example, $C_{\max} = \max(2a+b, 2a+1)$, for the parameter values$(a,b) = (1/4, 2)$: $C_{\max} = \max(2a+b = 5/2, 2a+1 = 3/2) = 5/2$ time units.

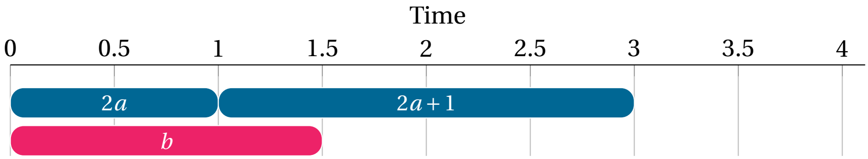
5.3.2 Symbolic SPTF scheduler

Given a particular parametrized set of tasks and resources, a deterministic scheduler makes its decisions sequentially. Algorithm 14 extends Algorithm 12 with symbolic scheduling. It captures the conditions that distinguish between the selection of scheduling options. In Algorithm 14, the condition for selecting task t on Line 6 is captured on Line 7. Similarly, the resource selection condition for r is evaluated on Line 8 and is captured on Line 9.

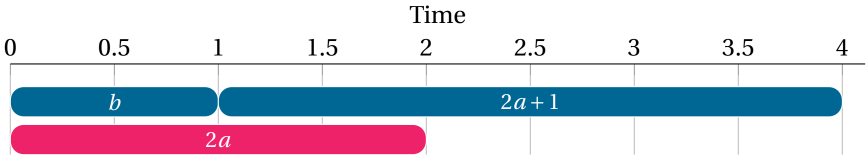
Note that the symbolic scheduler captures only closed intervals, even though on the interface between two decisions, only one decision is taken. We show in Section 5.4 that it is not necessary to explicitly return the tie-breaking condition



(a) Schedule for parameter combination $(a,b) = (1/4, 2)$.



(b) Schedule for parameter combination $(a,b) = (1/2, 3/2)$.



(c) Schedule for parameter combination $(a,b) = (1,1)$.

Figure 5.1: Shortest Processing Time First (SPTF) example schedules for task queue $\langle (t_1, 2a), (t_2, b), (t_3, 2a+1) \rangle$.

Table 5.1: Decision region details for task queue $T = \langle (t_1, 2a), (t_2, b), (t_3, 2a+1) \rangle$.

Colour	$\boldsymbol{p} = (a, b)$	Task cond.	Resource cond.	Queues
Red	$(1/4, 2)$	$2a + 1 < b$	$0 < 2a$	$\langle 2a, b \rangle, \langle 2a+1 \rangle$
Green	$(1/2, 3/2)$	$2a \leq b \wedge b \leq 2a+1$	$0 < 2a$	$\langle 2a, 2a+1 \rangle, \langle b \rangle$
Blue	$(1, 1)$	$b < 2a$	$0 < b$	$\langle b, 2a+1 \rangle, \langle 2a \rangle$

for our approach.

Algorithm 14 Symbolic shortest processing time first scheduler

```
function SYMBOLIC_SPTF_SCHEDULE(task queue $T = \langle (t_1, \boldsymbol{d}_1), \ldots, (t_n, \boldsymbol{d}_n) \rangle$, resources $R = \langle r_1, \ldots, r_m \rangle$, parameter values $\boldsymbol{p}$)
    $c = \emptyset$ // initialize an empty set of conditions
    for each $r \in R$ do
        $Q(r) = \langle\,\rangle$ // empty queues Q for each resource
    repeat
        remove first task $x = (t, \boldsymbol{d}) \in T$ with duration $\boldsymbol{d}(\boldsymbol{p}) = \min_{(t_i, d_i) \in T} \boldsymbol{d}_i(\boldsymbol{p})$
        $c = c \cup \{ (\boldsymbol{d} - \boldsymbol{d}_i, \leq) \mid (t_i, \boldsymbol{d}_i) \in T \}$
        select first resource $r \in R$ with the smallest $\text{COMPLETION}(Q(r))(\boldsymbol{p})$
        $c = c \cup \left\{ (\text{COMPLETION}(Q(r)) - \text{COMPLETION}(Q(r_i)), \leq) \mid r_i \in R \right\}$
        $\boldsymbol{s} = \text{COMPLETION}(Q(r))$
        append $x$ to $Q(r)$
        $\sigma(t) = \boldsymbol{s}$
    until $T$ is empty
    return $c, Q, \sigma$
```

Let the queue of symbolic tasks and their duration expressions again be $\langle (t_1, 2a), (t_2, b), (t_3, 2a+1) \rangle$. Figure 5.1 shows three symbolic schedules that are produced for this task set for three different parameter combinations ($(a = 1/4, b = 2)$, $(a = 1/2, b = 3/2)$ and $(a = 1, b = 1)$). At these parameter combinations the scheduler returns several necessary conditions for the same decisions to be taken as shown in Figure 5.2. The relevant task and resource selection conditions are shown in Table 5.1[2]. These selection conditions result in polyhedra that overlap, yet the scheduler only returns one specific result and schedule for each parameter combination; each point in the parameter space is associated to exactly one schedule. In Figure 5.2, the points on the line $b = 2a$ belong to the blue region, as the task with duration $2a$ (as the first task in the task queue) would be picked first. Similarly, the points on the line $b = 2a+1$ belong to the green region, as task t_2 with duration b would be picked before task t_3.

---

[2] In figures and tables, the task indices have been omitted.

5.3.3 Evaluating points strictly inside a decision region

For our exploration algorithm, we need to distinguish between parameter combinations that lie on the border of a polyhedron, and those arbitrarily close to, *but not on*, the border to detect in which decision region they fall. If at parameter combination p multiple tasks tie for the smallest duration, Algorithm 14 chooses the task that occurs first in the queue. I.e., on the 'border' between two decision regions, exactly one of several mutually exclusive branching decisions is taken. This is in contrast to the problem in Chapter 4, where the critical-path expressions of either region were valid on the border between regions.

The decision regions can be open or closed such as visualized for task queue $\langle(t_1,a),(t_2,2a),(t_3,2),(t_4,5)\rangle$ in 1-dimensional intervals in Figure 5.3a. This example is synthetic as it allows negative task execution times. However, it illustrates aspects that may occur in realistic, more complex examples, for more complex schedulers. The figure shows that any side of a scheduling region can be open or closed, depending on the conditions used in the scheduler. Moreover, it shows that schedules may occur for only a single parameter value. Point $a=0$ is the only point in the example that leads to schedule $\langle(t_1,a),(t_2,2a),(t_3,2)\rangle,\langle(t_4,5)\rangle$. The maximum completion times associated with the schedules have different slopes for different decision regions, as shown in Figure 5.3b. In more complex examples, the makespan may even make discrete, discontinuous steps at borders of regions. The slopes of the regions can be determined by Algorithm 10 of Chapter 4.

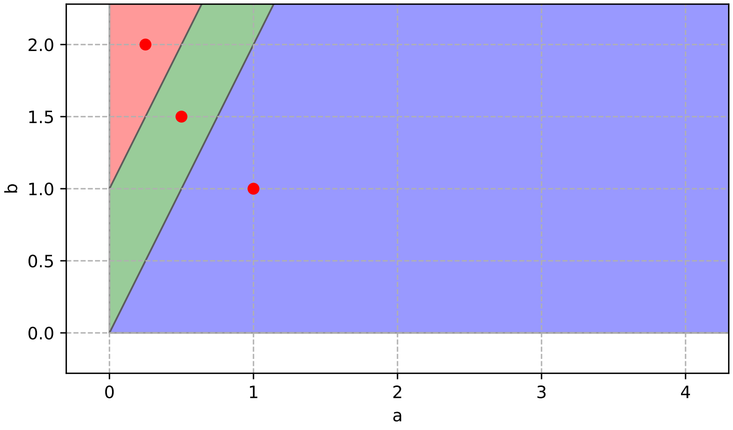
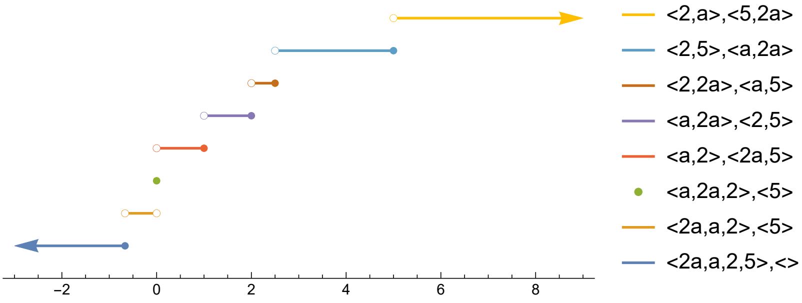


Figure 5.2: Example polyhedra returned by the symbolic scheduler for task queue $T=\langle(t_1,2a),(t_2,b),(t_3,2a+1)\rangle$, at parameter points $(a,b)=(1/4,2)$, $(1/2,3/2)$ and $(1,1)$ (the red dots). The regions denote for which parameter combinations the same sequence of decisions are taken.

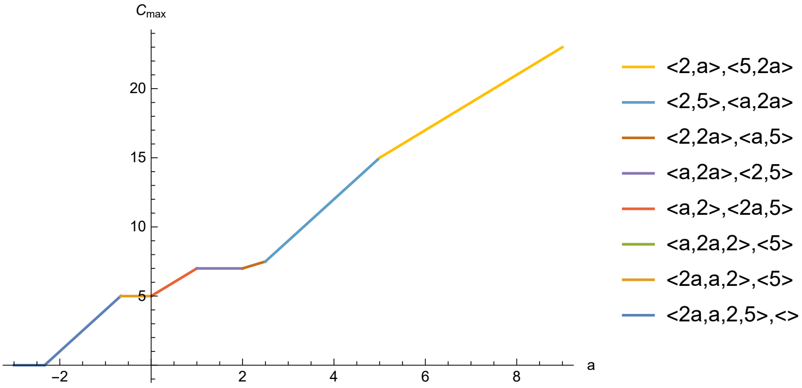
For our approach, we need to evaluate parameter combinations that are *just off* a region border, to find to which decision region a border belongs. We add an infinitesimally small ϵ to some of the parameters such that the scheduler evaluates conditions as if the parameter point is strictly inside a polyhedron to influence the tie-breaking decisions that the scheduler makes. This procedure is explained in Section 5.4.2.

5.3.4 Interpreting a symbolic scheduler as a decision tree

The decision sequences that can be taken by the SPTF scheduler for the task queue $\langle (t_1, 2a), (t_2, b), (t_3, 2a+1) \rangle$ are visualized in Figure 5.4 as a decision tree. For each parameter combination $\boldsymbol{p} = (a, b)$ a particular sequence of decisions (i.e., a path in the decision tree) is followed. Starting from the left, the branches of the tree contain the minimum task and resource selection, and the leaves show the result-



(a) Open and closed intervals for different scheduling regions for varying values of a.



(b) Maximum completion time of the resources for varying values of a.

Figure 5.3: Scheduling regions and maximum completion times for scheduling the task queue $\langle (t_1, a), (t_2, 2a), (t_3, 2), (t_4, 5) \rangle$ on two resources.

ing resource queues. In Section 5.4 we present a structured way to find for all parameter combinations which branch of the decision tree is taken. This allows the behaviour of the scheduler to be characterized with respect to the parameters.

There are branches that cannot be taken for any $\boldsymbol{p}$. The task t_1, with duration $2a$, for example, is always smaller than t_3 with duration $2a+1$, so task t_3 cannot be scheduled before scheduling t_1. Similarly, if a task selection requires that the parameters are related in a particular way, then certain resource allocations can never be selected. Figure 5.2 shows the decision regions for strictly positive parameters. Figure 5.5 shows the geometric decision regions for each combination $\{(a,b) \mid -1 \le a \le 8/5 \wedge -1 \le b \le 8/5\}$. Only three decision regions fall in the positive quadrant. For the remaining decision regions, at least one parameter is negative.

As a or b becomes negative, at least one task duration becomes negative. As before, we allow negative tasks durations for illustration purposes. Behaviour that we see in this example may also occur in more complex schedulers. For some negative task durations the SPTF scheduler delivers the same schedule as for certain positive parameters, although following a different decision sequence. An example occurs in Figure 5.8, the two decision regions that are labelled with C correspond

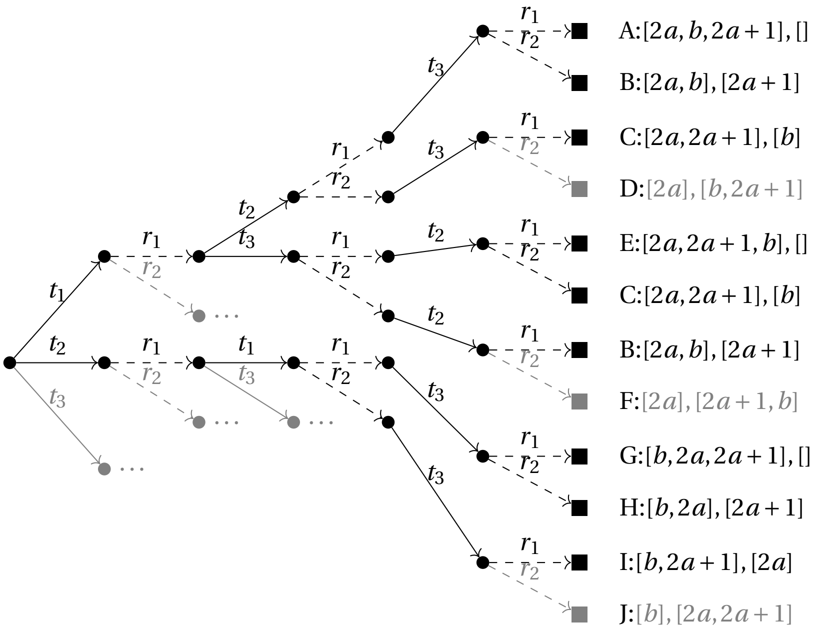


Figure 5.4: Example decision tree for SPTF scheduling tasks $\langle (t_1, 2a), (t_2, b), (t_3, 2a+1) \rangle$. A solid edge is a decision on a task, and a dashed edge is a decision on a resource. Greyed out decisions cannot be taken due to conflicting requirements on the parameter values.

to different paths in the decision tree visualized in Figure 5.4. Different sequences of decisions can lead to the same internal state of the scheduler, which may lead to the same scheduling result.

5.4 Exploring parameter combinations

In Section 5.4.1, we present a divide-and-conquer approach that identifies to which decision regions each parameter combination belongs. To evaluate all parameter combinations for polyhedra with strict inequalities, we describe a procedure to generate ϵ corners, i.e., corner points that are an infinitesimal distance away from an open corner of a polyhedron, in Section 5.4.2.

5.4.1 Divide-and-conquer approach

Algorithm 15 shows a divide-and-conquer approach that covers all parameter combinations in a given bounded convex parameter region CP. It returns a set of pairs consisting of a set of constraints defining a convex polyhedral parameter region and the parametric result that the scheduler produces in that region.

Lines 5 to 7 collect all the constraints and all the scheduler results for all corner points of the parameter region CP. A convex polyhedron can be transformed

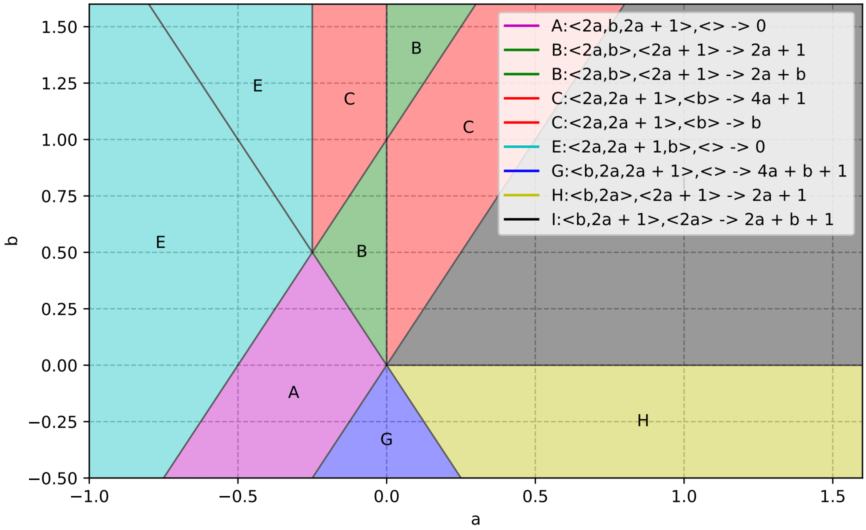


Figure 5.5: Geometric visualization of which parameter combinations lead to which resource queues for the example in Figure 5.4. The legend shows resource queues and symbolic maximum completion times for each explored decision region. The letters relate to the leaves in Figure 5.4.

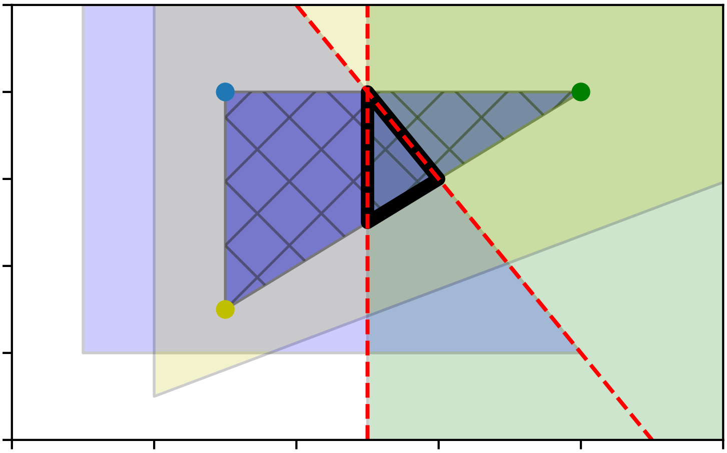
into a set of corner points using the double description library [42]. Lines 10 to 12 check if all constraints found (Line 11) hold in all corner points of CP (Line 10). If so, then the recursion ends and we arrive at Line 19. All corner points returned the same scheduling result (g_{tot} contains only one element); therefore we can return a pair with CP and the schedule result, in Line 19. In the case that one of the constraints was found not to hold for one of the corner points (Line 12), the region CP contains points from more than one decision region. This situation is visualized in Figure 5.6a. In that case, we use that constraint to split the region CP in two new (smaller) regions in Lines 13, 14; Algorithm 15 then recursively computes the results for the two new parameter regions (Lines 15, 16) and returns the union of their separate results (Line 17). The branching stops when a region is found for which all the corner points have conditions that hold for all the corner points, as visualized in Figure 5.6b.

Algorithm 15 Divide-and-conquer for symbolic scheduling

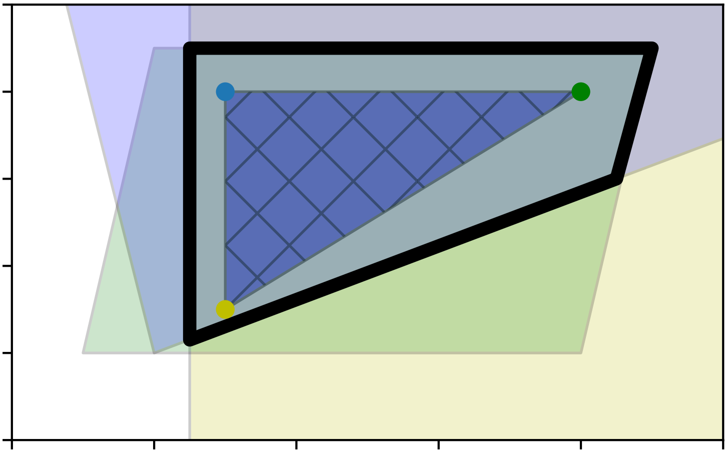
```
 1: function DIVIDECONQUERSCHEDULE(scheduling problem $P$, bounded convex
    parameter polyhedron $CP$)
 2:     // Empty sets to record conditions and scheduler results
 3:     $z_{tot} = \emptyset, g_{tot} = \emptyset$
 4:     // Combine the conditions of all corners
 5:     for each corner $p_c^\epsilon$ of EPSILON_CORNERS($CP$) do
 6:         $z_i, g_i$ = EVALUATE_PROBLEM($P$, $p_c^\epsilon$)
 7:         $z_{tot} = z_{tot} \cup z_i$
 8:         $g_{tot} = g_{tot} \cup g_i$
 9:     // Check whether all corners admit to all conditions
10:     for each corner $p_c^\epsilon$ of EPSILON_CORNERS($CP$) do
11:         for each $z \in z_{tot}$ do
12:             if $\neg z(p_c^\epsilon)$ then // Divide $CP$ into two disjoint polyhedra
13:                 $CP_1 = CP \cup \{z\}$
14:                 $CP_2 = CP \cup \{\neg z\}$
15:                 $R_1$ = DIVIDECONQUERSCHEDULE($P, CP_1$)
16:                 $R_2$ = DIVIDECONQUERSCHEDULE($P, CP_2$)
17:                 return $R_1 \cup R_2$  // Return all regions and schedule results
18:     Let $g_m$ be the only element of $g_{tot}$.
19:     return $\{\langle CP, g_m \rangle\}$
```

5.4.2 Evaluating polyhedra with strict inequalities

The conditions leading to decision regions may include strict inequalities and non-strict inequalities. We represent a polyhedron with strict inequalities that exclude bounding faces using corner points that are at an infinitesimally small distance from the concrete corner (ϵ corners), inside the polyhedron. To determine such



(a) The intersection does not cover all points of the hatched convex polyhedron. The algorithm can split the polyhedron along one of the two red dashed lines.



(b) The intersection covers all points in the hatched convex polyhedron.

Figure 5.6: Two situations for the intersection (outlined in thick black) of the green, yellow and blue conditions returned by their respectively coloured corners.

ϵ corners for strict inequalities (i.e., an open interface of a polyhedron), we use the technique visualized in Figure 5.7. This technique creates a smaller polyhedron, by tightening the conditions (i.e., translation in the figure) such that they cut off the points that lie on the interface of the strict inequality $S_i = (\boldsymbol{e}, <)$. The points $\left\{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{e}(\boldsymbol{x} = 0)\right\}$ are those that lie on the hyper-surface defined by $\boldsymbol{e}$; we call such points *adjacent* to S_i. We find a distance δ, to be added in the normal direction of the inequality S_i, such that concrete new corner points for a new polyhedron CP' (spanned by c_1 and the red corners $c'_{2,1}$, $c'_{2,2}$, $c'_{3,1}$, and $c'_{4,1}$) are created. δ is chosen small enough to retain all corners that are not adjacent to this strict inequality. The distance at which a corner c_k not adjacent to S_i would be cut off is the perpendicular distance $\delta(S_i, c_k)$. Therefore, any concrete translation distance δ of S_i less than the minimum perpendicular distance of S_i to any non-adjacent corner point c_k does not cut off corner points that are not adjacent to S_i.

Translating the inequality S_i with a concrete value leads to a smaller concrete polyhedron CP'. For example, when S_2 is translated by $\delta(S_2, c_3)/3$, a new polyhedron is created that is spanned by c_1, $c'_{2,1}$, $c'_{2,2}$, c_3, and c_4. The corner points of CP' that do not coincide with any corner of CP (in the previous example, $c'_{2,1}$, $c'_{2,2}$)

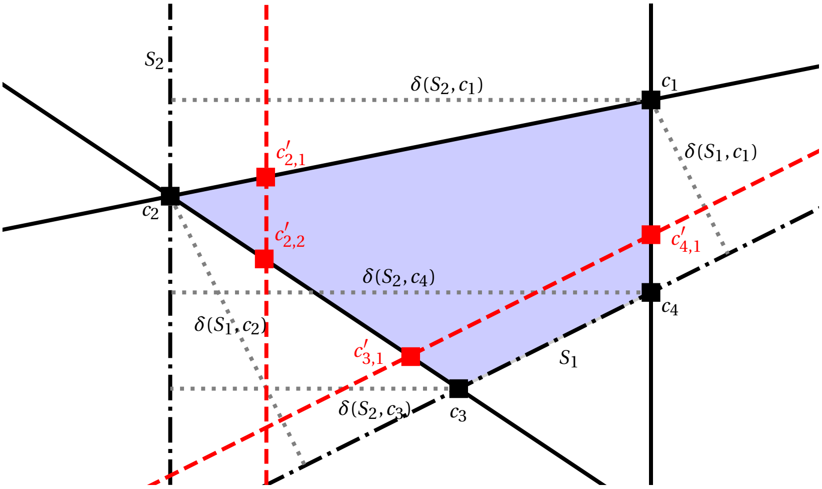


Figure 5.7: Creating ϵ corners. c_1, c_2, c_3, c_4 are corners of a polyhedron, and S_1 and S_2 are strict inequalities (denoted by dash-dot lines). Calculate new concrete corners ($c'_{2,1}$, $c'_{2,2}$,$c'_{3,1}$, and $c'_{4,1}$) by translating S_i a concrete distance, using one third of the minimum (perpendicular) distance $\delta(S_i, c_k)$ of S_i to each other corner c_k. This procedure generates new corner points (the red dots) without removing corners (such as c_1) on non-strict inequalities (denoted by solid lines).

are used to determine the ϵ corners. The ϵ corner is determined by moving the original corner in the direction of the corresponding concrete corner(s) of CP' by an infinitesimal distance. This creates a corner point for which no concrete point lies between that corner and the original corner, for example $c_2 + \epsilon(c'_{2,1} - c_2)$. As multiple corners may be cut and generated (as for S_1 and corners c_3 and c_4) we need a way to distinguish which new corners relate to the old corners. To do so, we take the concrete δ small enough so that any new corner is generated closer to its original corner than to any other corner. This distance should be less than half the minimum of the distance to adjacent corners and the perpendicular distances to non-adjacent corners.

In Figure 5.7, a simpler method seems possible; namely, for each corner point adjacent to a strict inequality, generate new corners that are in the direction of the adjacent corner points. When more than one strict inequality is supplied for a corner point, this method would generate multiple corner points, each of which is still adjacent to at least one other strict inequality. Therefore, this simpler procedure will not always lead to the correct corner points to describe the polyhedron with strict inequalities.

For the procedure sketched here to work, the symbolic scheduler that is used by the exploration algorithm should be capable to evaluate conditions for parameter values with infinitesimal parts.

5.5 Experimental evaluation

We apply our approach to several schedulers to show that the technique is applicable to both classical processor scheduling and scheduling techniques relevant for manufacturing systems. The result of our parametric scheduler characterization allows us to investigate scheduling productivity with respect to design parameters for re-entrant manufacturing systems. We do not compare our approach to other techniques as no other technique exists that can determine the scheduling regions.

The experiments described in this chapter have been implemented and executed with Python3, using the Python wrapper for the CDD double description library [42], on an Intel Core i5-4300M notebook running 64-bit Windows 10.

5.5.1 Shortest Processing Time First scheduler

The characterisation of the SPTF scheduler has already been shown for the task queue $\langle (t_1, 2a), (t_2, b), (t_3, 2a+1) \rangle$ in Figure 5.4. For $\boldsymbol{p}_+ = (a, b)$ with strictly positive parameters ($a > 0, b > 0$) three decision regions are found. As task durations $2a$ and $2a + 1$ are already ordered, these decision regions correspond to the relative ordering of b to $2a$ and $2a + 1$. The decision regions have maximum completion times that depend both on a and b. The static part of the task duration, 1, in $2a+1$ is not critical as long as $b > 2a + 1$.

Figure 5.8 shows the performance characterization for the task queue $\langle (t_1, a), (t_2, 2a), (t_3, b), (t_4, 1) \rangle$, for $-1/2 \leq a \leq 1, -1/2 \leq b \leq 2$; 22 different scheduler results are found in 26 decision regions. The maximum completion time in the region $R_1 = \{(a, b) \mid 0 \leq a \leq 1/2, 0 \leq b \leq 1\}$ depends either on a or on b, but not on both. For all parameter combinations in R_1, the task with duration 1 is the last to be scheduled as none of the other tasks are bigger in this region, and therefore always ends up in the maximum completion time expression.

The two examples of Figures 5.4 and 5.8 illustrate the type of analysis that our divide-and-conquer approach supports.

5.5.2 Schedulers for re-entrant FMSs

Re-entrant FMSs (see also Chapter 2) need to interleave product flows at different stages (passes) to optimize the productivity of the machines. We assume here that (i) the minimum re-entrant loop time l_{min} and maximum re-entrant loop time l_{max} are design-time parameters that are constant during scheduling, (ii) that the set-up times $SS : Z \times Z \to \mathbb{R}$ between two products $(z_1, z_2) \in Z^2$ at the re-entrant processing station are determined fully by their product types, (iii) that the set-up times between products with the same type are the smallest possible, and (iv) that the products all need to be processed twice by some processing station (i.e., products are processed in two passes). For sake of simplicity of presentation, the set-up times of the machines include the processing time, and are the same between first pass processing and the second pass processing.

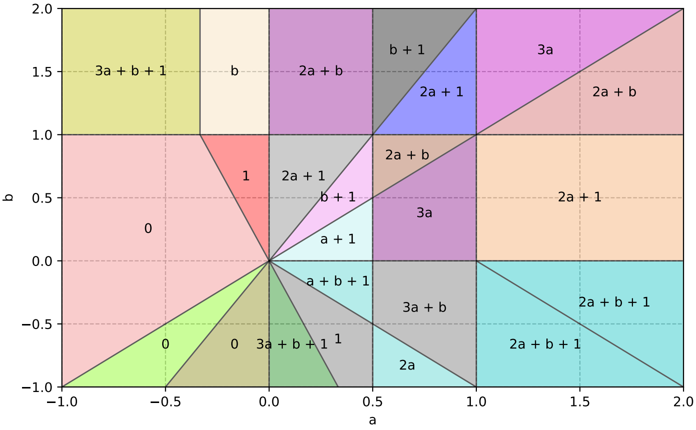


Figure 5.8: SPTF performance characterization for task queue $\langle (t_1, a), (t_2, 2a), (t_3, b), (t_4, 1) \rangle$. The maximum completion times are shown per decision region.

Eager Scheduler

The Eager Scheduler [122] selects the ‘first fitting slot’ in which an operation can be executed. It does so by trying to insert a re-entrant operation into an operation sequence in such a way that it does not influence the chosen timing of previously scheduled operations. The Eager Scheduler iteratively inserts the second pass operation of the next product into a sequence of (initially only first pass) operations. The scheduler fixes the re-entering time for a product after inserting its second pass to the earliest time allowed by the constraints. Evaluating whether an operation can be inserted while satisfying all constraints is possible through a modified longest-path algorithm that detects the presence of positive cycles and does not allow changing the operation times of previously scheduled operations (see Appendix A). The presence of a positive cycle indicates that the constraints cannot be met. The symbolic scheduler needs to determine a constraint on parameters for which a given cycle is positive.

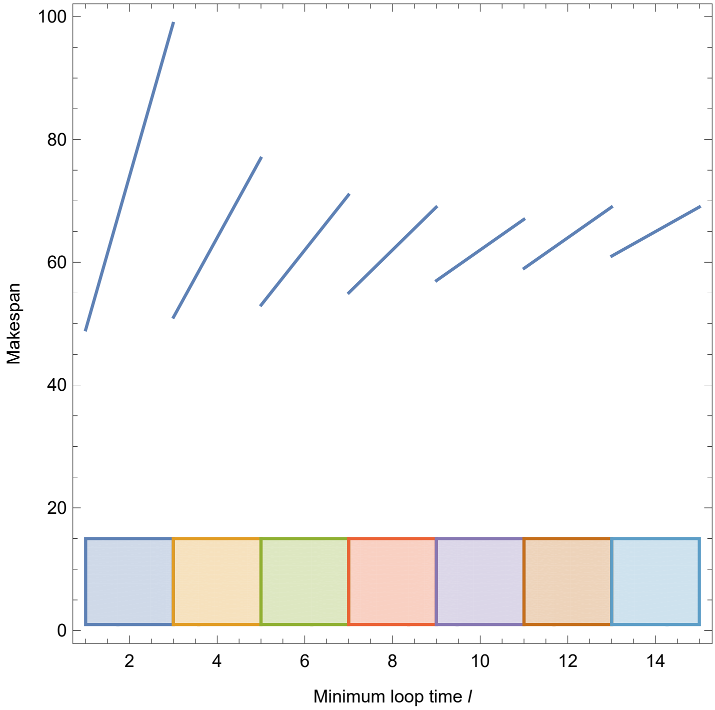


Figure 5.9: Eager Scheduler applied to 25 identical sheets, $l_{-} = l$, $l_{+} = l+3$. The different decision regions are colour coded in the bottom. The top part of the graph shows the makespan as a function of l_{-}. As the minimum loop time increases, the slope of the makespan becomes smaller, while the starting offset for subsequent line segments is slightly higher.

Enumerating all cycles in a graph is prohibitively time-consuming, and many of the resulting positive cycle expressions will be redundant. Instead, it is sufficient to return *some* positive cycle expression if an explored choice is infeasible. Our exploration algorithm, Algorithm 15, can then divide the polyhedron on that condition, to separate the decision regions on the outcome of that cycle. Additional positive cycles may appear during recursive evaluation of the smaller polyhedra.

Figure 5.9 shows how the Eager Scheduler performs for 25 identical products under variation of the minimum loop time, with a maximum buffer time of 3 seconds. The makespan per product increases as the minimum loop time increases, until another product fits in the loop. At that point, the scheduler can avoid some idle time by filling the loop more. Thus, the decision regions returned by the scheduler correspond to the number of products that fit in the re-entrant loop. As the minimum loop time increases, the slope in each region decreases; the minimum loop time is less and less often in the critical path. Performing such an evaluation for typical product queues will lead to a better understanding of which minimum loop time is productive.

Pattern Scheduler

If the products in a re-entrant flow-shop are coming in a regular pattern, then instead of optimizing a single product at a time, we can interleave patterns with each other. This is exploited by the Pattern Scheduler [123]. By aligning first and second passes of products from different instances of the pattern, large set-up times can be avoided. A good heuristic is to find the maximum number of patterns that fit in the loop and allow the alignment of the two passes. We analyse a scheduler that determines how many patterns fit in a 2-re-entrant loop, and then calculates which pattern interleaving schedule would be indefinitely repeatable. A pattern $Z = \langle z_1, \ldots, z_k \rangle$ is an ordered list of $|Z|$ product types. The pattern repeats r times in the job.

The number of patterns n that can be concurrently in the re-entrant loop is calculated in a way similar to the steady-state performance estimator of Waqas et al. [123] for re-entrant flow-shops, which models a steady-state pattern scheduler. It calculates how many interleaved patterns could be executed in the interval between the minimum and maximum loop time. The buffer range is denoted by $l_b = l_+ - l_-$). Figure 5.10a shows that the first passes (solid rectangles) of pattern k are interleaved with the second passes (dashed rectangles) of pattern $k-n$. Pattern time t_p is derived as a sum of set-up times, given this particular interleaving of first and second passes, and use the knowledge of the pattern that the product type of $i - n \cdot |Z|$ is the same as i:

$$\begin{aligned} t_p &= \sum_{z_i \in Z} (S(z_i, z_{i-n\cdot|Z|}) + S(z_{i-n\cdot|Z|}, z_n i + 1)) \\ &= \sum_{z_i \in Z} (S(z_i, z_i) + S(z_i, z_{i+1})) \end{aligned}$$

In steady state, this means that the first passes of a pattern k and the second passes of a re-entrant pattern $k-n$ are processed every t_p time units. For any product z_i, the deadline $D(z_i)$ (denoted with red dashed arrows in Figure 5.10b) between the first and second passes must be satisfied for a valid schedule to exist. The relative due date $D(z_i)$ is assumed constant, and parametrized by l_+. Before the *second* pass of a product of pattern k is executed, n patterns ($n \cdot |Z|$ products) must be processed, plus a *first* pass product from pattern $k+n+1$. That is, for each product $z_i \in Z$, the chosen loop time $t_l^* = n \cdot t_p + S(z_i, z_{i+n\cdot|Z|}) \leq D(z_i)$ should be less than the re-entrant due date $D(i)$. The set-up time $S(z_i, z_{i+n\cdot|Z|}) = S(z_i, z_i)$ in Figure 5.10a corresponds to the set-up time between a_1^{k+n}, and a_2^k. This constraint is shown as the cycle of highlighted blue edges in Figure 5.10b, and is the same for each instance of the pattern, but may be different for each product type within the pattern. Therefore, we can locally check whether the deadlines for each product type in the pattern can be met for a particular value of n.

The number n of patterns that fit in the loop is determined by the pattern time t_p, the maximum additional set-up time $t_s = \max_{z_i \in Z} S(z_i, z_i)$, l_- and l_+ as follows:

$$n_* = \left\lfloor \frac{l_-}{t_p} \right\rfloor \qquad n_{max} = \left\lfloor \frac{l_+ - t_s}{t_p} \right\rfloor$$

$$n = \begin{cases} 0 & \text{if } n_{max} \leq 0 \\ n_{max} & \text{if } 1 \leq n_{max} < n_* \\ n_* & \text{otherwise} \end{cases}$$

The maximum number of patterns that fit in the loop is then n_{max}. To maintain a high throughput, the scheduler tries to fit n_* patterns in the loop, such that $n_* \cdot t_p \geq l_-$, because then the re-entrant products can return exactly after finishing some previous pattern's first passes. However, if $n_* > n_{max}$ then there exists no n such that the pattern can return within the time window $l_- \leq n_* \cdot t_p \leq l_+$. In that case the scheduler uses the maximum number of patterns that fit in the loop $n = n_{max}$.

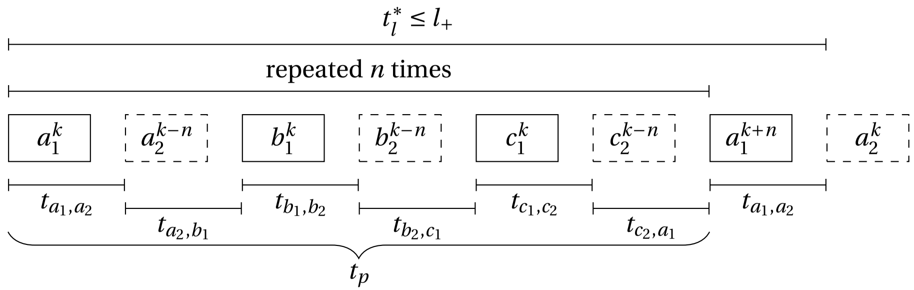
The actual timing of the operations is determined by first filling the loop with n products, then interleave products of pattern k with $k-n$ in steady-state, until the loop needs to be emptied again. Algorithm 16 first creates an event network as shown in Figure 5.10b, without any operation order enforced yet (Lines 2 to 4). It then determines the operation sequence; i.e., a sequence of first and second pass products. This operation sequence first requires to fill the loop with n products (lead-in on Line 7), then interleave products of pattern k with $k-n$ in steady-state (Line 8 to Line 10), until the loop needs to be emptied again: (lead-out on Line 11). The appropriate sequence-dependent set-up times for this sequence are added (Line 13), and the earliest realization times (i.e., a schedule) are calculated (Line 14). Finally, the schedule is returned along with the event network G. All conditions used in this decision process are captured symbolically as expressions in the minimum loop and buffer times l_- and l_b.

If no complete pattern fits in the loop (i.e., $n_{max} = 0$), then the interleaving procedure schedules both passes of a product to be processed before the next product starts its first pass. This is rather inefficient, so if $n_{max} = 0$ the previously mentioned Eager Scheduler is invoked instead.

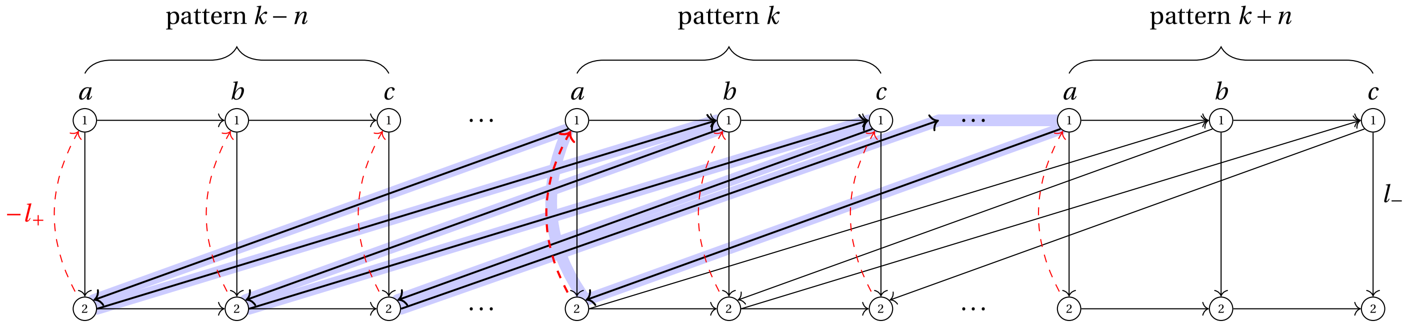
Algorithm 16 Pattern-scheduling

```
function InterleavePatterns(pattern $Z$, repetitions $r$, set-up times $SS : Z^2 \to \mathbb{R}$, number of patterns to interleave $n$)
    Create an empty event network $G$
    For each first and second pass operation $o \in \{1, 2\}$ of each job $j \in \{1, |Z| \cdot r\}$, insert an associated event $e_{j,o}$ into $G$
    Between each $e_{j,1}$ and $e_{j,2}$, add constraints for the minimum loop time $l_{min}$ and maximum loop time $l_{max}$ in $G$
    operation sequence $w = \langle\, \rangle$
    $d = \min(n, r)$ // determine number of products in transient phase
    for each $i \in \{1, d \cdot |Z|\}$ do append $e_{i,1}$ to $w$
    for each $i \in \{d \cdot |Z| + 1, r \cdot |Z|\}$ do
        append $e_{i,1}$ to $w$
        append $e_{i-n \cdot |Z|,2}$ to $w$
    for each $i \in \{\max(0, (r-n) \cdot |Z|) + 1, r \cdot |Z|\}$ do append $e_{i,2}$ to $w$
    for each $i \in \{1, r \cdot |Z| - 1\}$ do
        add edge $w_i \to w_{i+1}$ with weight $SS(w_i, w_{i+1})$ to graph $G$
    Compute earliest realization times $\underline{t}$ and feasibility $f$ of $G$
    if $f$ then return $G$, $\underline{t}$
    else return $G$, $\emptyset$
```

Figure 5.11 shows for a pattern of three products, repeated 5 times, how minimum loop time and buffer time impact the decisions made by the pattern scheduler. One pattern takes 15/2 time units, while the maximum first to second pass set-up time of the same product is $t_s = 2$. Figure 5.11 shows that for these combinations of minimum loop times and buffer times at most two patterns are used in the loop. The interpretation of triangle $\{(l_-, l_b) \mid 15 \leq l_- \wedge l_- + l_b < 17 \wedge l_b \geq 0\}$ is that one pattern less is interleaved because the maximum loop time (being the minimum loop time plus buffer time) is not sufficient yet to allow the product's second pass to start. In this region, the adjusted loop time $n \cdot t_p < l_-$ is less than the minimum loop time, and therefore some unproductive idle time has been inserted between patterns. The productivity is therefore decreased if the minimum loop time is increased and idle time is introduced, until an additional pattern fits again. In the bottom left, the more irregular structure of the regions is due to the Eager Scheduler making less regular decisions.



(a) A time representation of the interleaving of n patterns, starting at the first pass of pattern k.



(b) An event/constraint graph representation of the interleaving of n patterns.

Figure 5.10: A steady-state pattern interleaving example.

This result enables a designer to study the ability of the buffer to adapt to different patterns with respect to the physical layout of the printer. Improvements can then be explored on how a larger minimum loop time or a larger buffer would impact the ability to adapt to different patterns. For this particular scheduler, adding additional buffering capabilities above some threshold does not yield additional productivity. By performing such an analysis for typical use cases, a designer can make more informed decisions.

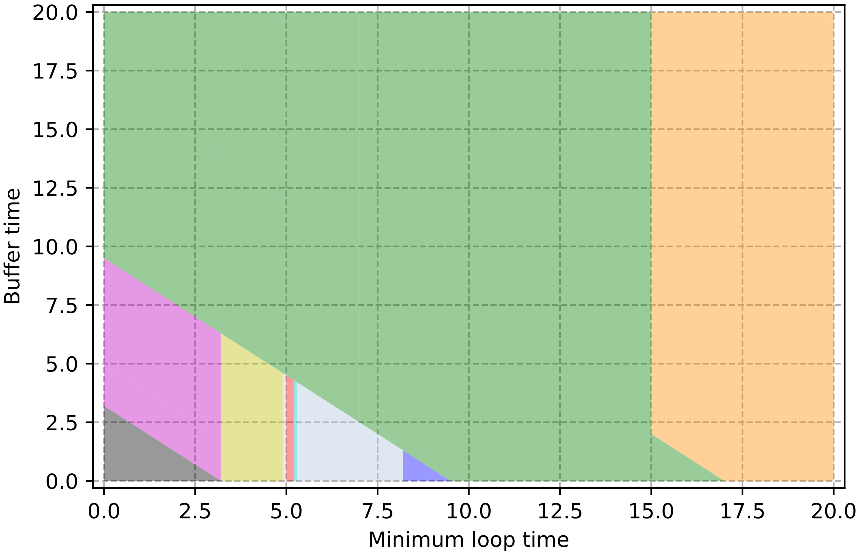


Figure 5.11: FMS pattern scheduling for three products, repeated 5 times. In the green area, one pattern is used in the loop, in the orange area two patterns are used. If no pattern fits in the loop (below left of the green area), then the eager scheduler decides on the operation ordering.

5.5.3 Scalability

Table 5.2 shows some runtime statistics for the example instances discussed in the previous subsections. The SPTF scheduler is very fast at evaluating single parameter points, and the task queues are rather small. These experiments only contain few different scheduler results and decisions. In these cases, our divide-and-conquer approach does not branch much more than strictly necessary in these cases, as the number of regions is close to or equal to the number of parametric scheduler results found. The Eager Scheduler and Pattern Scheduler require many more evaluations to cover the initial convex parameter region. In all experiments, almost all the time is spent in evaluating corner points, and the time spent gather-

Table 5.2: Parametric scheduling exploration statistics. The columns show per instance how many evaluations (E) were performed (i.e., symbolic schedule calls), how many regions (R) were generated, and how many different parametric scheduler results (PSR) were found. The last column shows the execution time (T) of the divide-and-conquer algorithm including all scheduling calls.

Instance	E	R	PSR	T
SPTF Scheduler Figure 5.3a	15	8	8	0.8s
SPTF Scheduler Figure 5.5	19	10	8	1.3s
SPTF Scheduler Figure 5.8	47	24	21	5.5s
Eager Scheduler Figure 5.9	54	14	14	289s
Pattern Scheduler Figure 5.11	366	45	11	948s

ing the conditions and choosing a branching condition were negligible.

The experiment of Figure 5.11 shows that many redundant regions were found, most leading to the same parametric scheduler results, which are in this case part of the same (green) decision region. The green area could have been detected by three cuts, but the cuts that were generated for the eager scheduler also cut the green area in more parts than necessary. These experiments can possibly be sped up by heuristically (instead of arbitrarily) choosing a separation condition that typically leads to fewer corner evaluations.

5.6 Related work

Schedulability regions have been determined for periodic task sets [14]. Task execution times in this work are parametric, and it provides scheduling models for rate-monotonic and fixed-priority scheduling. The results show that it is possible to use hyperplane representations for these scheduling models. The hyperplanes determine limits on the schedulability, i.e., for which task execution times the processors are proved to be capable of executing a workload. Their results map the parameter space to a binary outcome (schedulable or not schedulable), and the approach is specific to the rate-monotonic and fixed-priority scheduling.

The parametric scheduling work of Subramani [110] also focuses on schedulability queries for real-time systems, using the execution-time-constraints (ETC) framework. This dispatching mechanism only considers constraints from and to *past* events. It does not take into account that constraints from and to *future* events impact the scheduling decision, and computes a single range in which a task/job can be dispatched.

Parametric timed automata approaches such as [63] [44] also return a set of symbolic conditions, called zones. The typical query, however, is whether a *given* state is reachable. The problem in this chapter requires finding whether any accepting state is reachable under certain parameter evaluations, which has been shown to be PSPACE-Complete [3]. Current approaches [44], however, do not give an algorithm to *discover* any accepting states automatically and also capture the symbolic conditions for the decisions leading to these states (in the form of a decision region). Our approach simultaneously discovers the reachable scheduling results and the symbolic conditions under which they are reachable.

The convexity of parametric problems is often based on the resulting parametric affine expression, as in [47] and [119]. In these works, the throughput and critical-path expressions are determined for data-flow graphs and event networks respectively. Such expressions are valid on the border of regions, whereas our problem requires the regions not to overlap. Our approach finds symbolic conditions that are related only to the decision making, but are not necessarily derived from the scheduling result.

Scheduling problems can often be expressed as integer programming problems. Such integer programming problems are often not amenable to solving to optimality due to the complexity of the scheduling problem. The constraints of such integer programming problems can be expressed through parameters [38]. Their work solves a different kind of parametric scheduling problem; the decision making is performed by a general-purpose solver. As with the previously mentioned related work, the scheduling result relates directly to the symbolic conditions under which they are optimal, which is not required for our approach.

We recognize that the problem at hand resembles a classification problem from the machine learning domain. In particular, classification trees [16] at first glance seem to match the problem. The algorithms that build such decision trees, however, only create rules in which a single feature (i.e., a parameter) is compared to a constant, or selected from a known set of values. The condition $a > b$ would therefore either require that this relationship is linearised in a feature, defeating the purpose of learning such relations automatically, or that it be approximated through many constants. In addition, the number of samples needed to find these relationships would likely be much larger than in our approach. A minimally adequate oracle is also required for an active learning algorithm to terminate.

5.7 Conclusion

A product flow scheduler has a significant impact on the productivity of an FMS. We therefore want to find out how a design parameter in a given range influences the sequence of decisions made in a scheduler. These decisions lead to the activation or avoidance of timing constraints, and therefore lead to a particular system performance. In this chapter, we introduced a symbolic scheduling model that captures the conditions under which certain decisions are taken. We introduced a divide-and-conquer algorithm that uses such information from symbolic schedulers to determine which parameter combinations lead to the same scheduling decisions for deterministic scheduling algorithms. It covers all possible parameter combinations in a given range. It can be used to exhaustively test how the system behaves in this range.

We applied this technique to several schedulers, to show that different kinds of problems and schedulers can be characterised by this technique. We believe that many deterministic schedulers are amenable to this analysis. The symbolic scheduling can be introduced to an existing scheduler to include symbolic relations with relatively few modifications.

6

Conclusions and Future Work

This thesis contains techniques to perform online optimization of product flows in re-entrant flexible manufacturing systems (FMSs) and to perform design-time analysis of the performance of FMSs, including the behaviour of the employed scheduling algorithms.

6.1 Conclusions

Online scheduling of product flows in re-entrant FMSs requires fast and efficient decision making to optimize the system productivity. We have provided an online scheduling algorithm for interleaving product flows in a 2-re-entrant flow shop. Given a stream of products and their associated flow, the scheduling algorithm quickly determines good scheduling decisions. The algorithm uses longest-path computations to compute timing instructions for the FMS, and to check whether such timing instructions exist for different scheduling options. We have identified restrictions on operation orderings to avoid the time-consuming computation of the longest path. We generalize the Compositional Pareto-algebraic Heuristic (CPH) into Constructive Pareto Meta-Heuristic (CPMH); a multi-dimensional meta-heuristic that explores multiple options without backtracking. Exploring multiple options improves the quality of the schedules and reduces the need for manual tuning of the weights for the algorithm's decision metrics. This allows the scheduler to be effective for a wider range of products, as is desired for an FMS. We have also shown that the worst-case time complexity of our algorithm is related to the largest number of products that can ever be interleaved with each other. This leads to interesting design trade-offs, as it implies a limit to the maximum useful size of the re-entrant loop relative to the smallest product. At some point, the scheduling algorithm will itself become a bottleneck for the system productivity.

We have also contributed an online scheduling algorithm for batches of products that need to be processed by the re-entrant FMS, in the form of a multi-dimensional Generalized TSP (GTSP) algorithm. This algorithm explores ordering batches of (ordered) product streams and selection of system settings to find trade-offs between productivity and quality. The re-entrant FMS scheduler takes a predefined product output ordering into account. The GTSP algorithm has the freedom to choose the ordering of batches as input for the FMS. The algorithm is also based on CPMH and incrementally constructs trade-off spaces in an online manner. The incremental and compositional construction of trade-off spaces is a promising direction to research. The performance in runtime is better than the current state-of-the-art approaches, and the quality of solutions is similar. The meta-heuristic is particularly suited for online optimization problems, where information arrives incrementally. The algorithms for GTSP are not restricted to 2-re-entrant FMSs. It is applicable to any FMS for which the input batch order has an impact on performance.

The input interleaving and batch optimization algorithms improve the productivity of existing FMSs. As the design of such FMSs is continuously evolving, the next design iteration of an FMS benefits from understanding the impact of variations of the current design. In particular, it is useful to evaluate the effect of combinations of structural parameters, such as loop lengths and settling times, on system productivity. We have contributed two parametric algorithms that can be used together to fully characterize the system performance of an FMS for particular sequences of products. The algorithms together characterize how the structural parameters relate to the scheduling decisions and the bottlenecks of schedules.

The parametric scheduling algorithm allows a designer to identify the impact of different parameter combinations on the scheduling decisions that are taken. We have shown a method to divide the parameter combinations into regions where the parameters lead to the same scheduling decisions. This allows investigation of the scheduling algorithm for different design choices. This also helps to understand in which situations a particular scheduler outperforms another or is more robust against parameter changes.

Once the scheduling decisions determine the dependencies between events, our parametric critical path algorithm can find the bottlenecks, i.e., critical-path expressions. The critical-path expressions are found for each parameter combination. The combination of these techniques yields a performance perspective of the total system, including the discrete performance steps that are introduced by the scheduling behaviour. The parameters have a (physical) interpretation in another domain. The critical-path expressions therefore yield feedback information that can be translated back to the original domain. A designer can then assess what the impact of parameter changes is for typical jobs, and determine the right trade-off between cost and productivity. The parametric analysis algorithms advance the current state-of-the-art for design-time performance analysis of FMSs in terms of parameters, e.g., from the control, or physical domain.

The techniques and algorithms presented in this thesis have wider-ranging applications. CPMH is a promising multi-dimensional meta-heuristic that can be used for online scheduling and allocation problems. The parametric critical path algorithm can be applied to many different problem domains, as the underlying event network uses little assumptions. The parametric scheduling algorithm has applications in learning the decisions that a deterministic algorithm takes given different parameters. These last two algorithms are applicable as long as the relationships between events, and the conditions for decisions, are expressed as linear combinations of parameters. These parametrized models and algorithms provide a way to characterise the system performance of an FMS taking into account both continuous timing constraints as well as discrete scheduling decisions. These characterizations can be related to the original domain, by translating the annotated parameters.

6.2 Future work

The online flow shop scheduler works for 2-re-entrant systems and an obvious, but non-trivial, extension would be to include higher re-entrancy systems [121]. To guarantee that our list scheduler produces feasible results, any partially completed product sequence should lead to a set of set-up times that are sufficient to return to an empty manufacturing system. That is, for each feasible interleaving option, there is a feasible follow-up option, which is in the worst case the empty state. We assume that an FMS can always restart its operation from an empty state. For 3-re-entrant or higher-re-entrant systems it is possible that none of the follow-up options are feasible; if it is feasible to insert a second pass into a series of first passes, then two operations need to add set-up times, violating each other's deadlines, even though neither of these deadlines was violated by the previously taken decision. Although the schedule-space-exploration algorithms of [121] make it more likely to find feasible schedules, they are not online algorithms, and it would require significant effort to maintain a good balance between high quality and fast computation time.

Variable re-entrancy scheduling for industrial printers (i.e., mixing simplex and duplex pages) has been investigated by [116]. The impact of variable re-entrancy on online operation sequencing in higher re-entrancy systems is still unclear. Further research could focus on improving the understanding of the impact of operation ordering on exploring scheduling options.

The current re-entrant scheduling model assumes that the state of a machine is fully determined by its preceding operation. In many FMSs the state of a machine is influenced by many factors, such as the history of products, the time since a particular product was last processed, the idle time between two operations, or the environment. These constraints have not been modelled and the scheduling algorithm does not take them into account. The online scheduling problem can be extended to incorporate the states of the machines, and make decisions based on maintenance or system reconfiguration requirements.

The re-entrant flow shop scheduling algorithm could be made into an any-time algorithm, such that it provides scheduling solutions before completing the calculation of the new generation. One step in this direction would be to remove parameter k and evaluate options according to a given time budget. Another way to reduce the peaks in computation time is to reduce the maximum number of relaxations performed in the Bellman-Ford-Moore (BFM) algorithm. If the operation begin times corresponding to the scheduling option do not converge fast enough, one can consider the option to be infeasible.

The Pareto meta-heuristic CPMH is an interesting candidate for solving online problems. One of its steps is to approximate a large set of solutions by selecting diverse trade-offs, in the hopes that the search space still contains the optimal solutions. It would also be interesting to incorporate solution diversity in the reduction operator. The reduction operators used in this thesis select representatives based on the assessment of partial solutions, but do not consider diversity or distance in the solution domain.

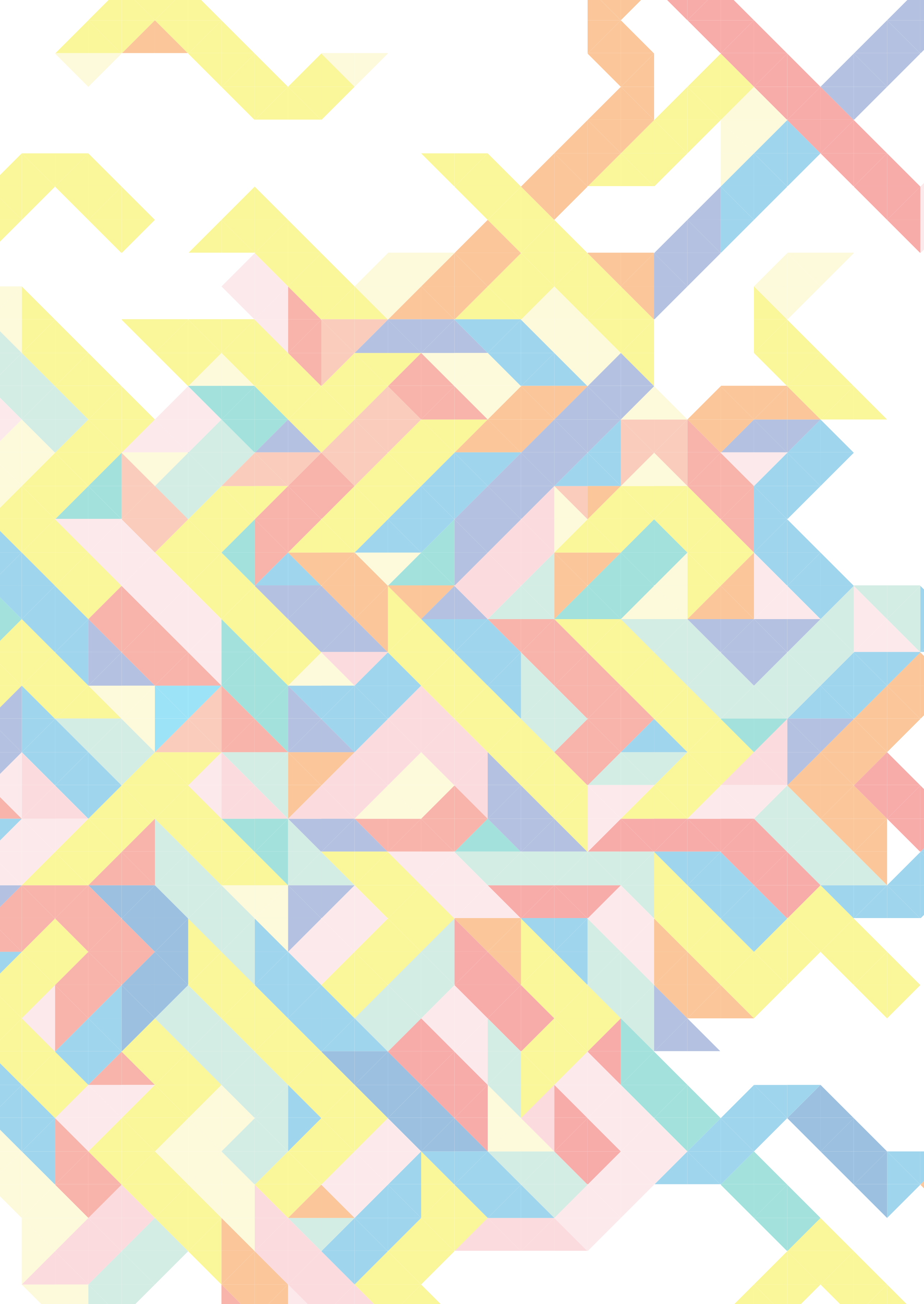
The batch scheduler currently does not take into account that particular batches may have deadlines, or precedence constraints. The inclusion of such properties may make the general problem harder. Yet it also yields opportunities to cut off parts of the solution space that definitely do not meet the constraints. The uncertainty in the prediction of batch processing times may be included by using a Pareto-dominance where the uncertainty in the prediction is explicitly taken into account [59].

The parametric critical path analysis has scalability issues when the parameter space has many dimensions, or when parameter combinations may lead to many different regions. We conjecture that it is possible to use lower and upper bounds on the critical-path expressions, based on the constraints imposed by neighbouring critical-path expressions. Such an approximation requires a measure of convergence, and an effective way to determine that measure. It would also be interesting to see whether the technique can be improved by taking into account certain non-linear relationships more effectively.

The parametric scheduling characterization problem resembles an active learning problem. The structure of the underlying decision tree then needs to be learned. This implies that the parametric scheduling problem actually solves a larger class of problems than we intended to solve. An interesting research direction would be to see whether this grey-box active learning can be applied to different kinds of deterministic algorithms as well.

Due to the generic nature of the characterization algorithm that is defined for a wide class of schedulers, it is difficult to give generic correctness proofs. It would be interesting to provide correctness proofs and a deeper analysis of the characterization algorithm for specific classes of schedulers. This would improve the understanding of the assumptions on the scheduler made in the characterization algorithm.

The results of the parametric analysis yield insight into the relationship between the multi-domain timing constraints, and the performance of the system. Currently, the analysis is applied to representative individual product sequences. The flexibility of the systems can be taken into account by aggregating these results in a meaningful way, for example by using weighted sums of piecewise linear functions.



Acronyms

2PPLS Two-Phase Pareto Local Search.

ALAP as-late-as-possible.

ASAP as-soon-as-possible.

AUGMECON2 Augmented Epsilon Constraint algorithm.

BFM Bellman-Ford-Moore.

BHCS Bounded HCS.

CPH Compositional Pareto-algebraic Heuristic.

CPM Critical Path Method.

CPMH Constructive Pareto Meta-Heuristic.

CPS cyber-physical system.

DSE design-space exploration.

EDF Earliest-Deadline-First.

FIN paper finishing module.

FMS flexible manufacturing system.

GPR generalized precedence relation.

GTSP Generalized TSP.

HCS Heuristic Constraint-based Scheduler.

ILS Iterated Local Search.

ITS image transfer station.

LST Least-Slack-Time.

MA-GTSP Memetic Algorithm for GTSP.
MCME maximum cycle mean expression.
MD-BHCS Multi-dimensional BHCS.
MIP mixed integer programming.
MMKP Multi-dimensional Multiple-choice Knapsack Problem.
MO-GTSP Multi-Objective Generalized TSP.
MO-TSP Multi-Objective Traveling Salesman Problem.

PD-TPLS Pareto Double Two Phase Local Search.
PIM paper input module.
PLS Pareto Local Search.

RHP Rolling Horizon Procedure.

SDF synchronous data-flow.
SLS Stochastic Local Search.
SPEA2 Strength-Pareto Evolutionary Algorithm.
SPTF Shortest Processing Time First.

TSP Traveling Salesman Problem.

Appendices

A

Bellman-Ford-Moore longest-path algorithm

This appendix describes the problem statement and some of the subtleties of choosing a longest-path algorithm. We first describe the shortest-path problem, and distinguish between two versions of the longest-path problem. We then describe in more detail the label-correcting algorithm BFM to find longest paths, and adapt this algorithm to incrementally determine the longest path for online algorithms.

A.1 Shortest-path problem

The shortest-path problem is a common problem in computer science: given a weighted graph $G = (V, E)$ with vertices V and edges $E \subseteq V \times V$ and a weight function $w : E \to R$ determine a shortest path $p \subseteq E$ with minimum weight $w(p) = \sum_{a \in p}(w(a))$ from a source vertex $s \in V$ to another vertex $t \in V$. A subset of E is a path from s to t if it is a finite sequence of edges which connect a sequence of vertices, starting at s and ending at t. If a vertex t is reachable from the source s, and if there are no negative-weight cycles then at least one of the paths in the graph has the shortest weight.

Common algorithms for the shortest-path problem include Dijkstra's shortest-path algorithm [31], A^* [56]. These algorithms are only suitable for directed acyclic graphs with positive weights on the edges.

A.2 Longest-(simple-)path problem

In contrast to the shortest-path problem, the longest-path problem is often defined as finding a *simple* path with *maximum* weight. A simple path is a path with no repeated vertices. We will call this version of the problem the longest-*simple*-path problem to distinguish from the longest-path problem where paths with repeated vertices are allowed. These problems are the same when graphs are not allowed to have cycles. A cycle is a path with at least one edge, whose first and last vertices are the same.

The longest-simple-path problem is NP-hard [25], as it reduces to the Hamiltonian path problem if all weights are unit weights, and there exists a polynomial time verifier whether a path is indeed simple. Any graph has a maximum weight simple path between two nodes as long as there exists a path between the two nodes. Determining such paths in general cannot be solved in polynomial time unless P=NP.

If there are cycles in the graph, and the problem does not restrict to simple paths, then there are infinitely many paths. A path that visits any part of a cycle can be extended by adding the cycle to the path, and can therefore be infinitely extended. If such a cycle has a positive weight, then the extended path is longer than the original path. As such cycles can be added without restriction, these paths can become infinitely long, and no bounded longest path exists. However, efficient polynomial time algorithms exist that either prove that no bounded longest path exists, or that return a longest path.

A.3 Systems of linear inequalities

The shortest-path and longest-path problems are equivalent to finding a solution to a system of linear inequalities, where each vertex $v_i \in V$ is labelled as l_i. For a given $G = (V, E)$ and weight function w, the system of linear inequalities is given by $l_j \geq l_i + w(v_i, v_j)$ for each $(v_i, v_j) \in E$. For this reason, a weighted graph with this interpretation of the weights is also called a *constraint* graph [25]. It is a special case of a linear program, where the objective is to minimize one of the variables.

The label-correcting longest-path algorithm BFM detects that either (i) there is a longest path from the source of the graph, or determines that (ii) a graph contains a positive cycle. The BFM algorithm iteratively updates distance labels of vertices, as shown in Algorithm 17. Before the first iteration, only the source contains a label and the rest is set to $-\infty$. At each iteration, the distance labels are relaxed once for each edge in the graph. At the beginning of the i^{th} iteration, the labels d contain the distance of the longest path of at least $i-1$ edges. The relaxation corrects/updates the labels d such that at least one additional edge is taken into account for the longest path to any target. After the relaxation, d therefore contains the distance of the longest path of at least i edges. Each iteration relaxes the distance labels by at least one edge; if no distance label needed to be changed, the labels were already sufficiently far apart, and have therefore converged to the final solution.

However, in case the graph contains a positive cycle, the labels will never converge, and will continue to increase. Any path that does not contain a cycle (i.e., a simple path) may contain at most $|V|-1$ edges. The BFM algorithm performs relaxations for $|V|-1$ iterations, and then checks whether any further relaxation is still possible. If it is, then there must exist a path of at least $|V|$ edges that is longer than any of the paths of at least $|V|-1$ edges. Any path of length $|V|$ edges must include at least one vertex that is visited more than once. Therefore, if the last relaxation loop detects that the distance labels have not converged, it will never converge. If there are no positive cycles, then the algorithm converges in at most

$\mathcal{O}(|V||E|)$ time. If there are positive cycles, then the algorithm detects it in its final pass, and returns false.

At each relaxation, the mapping 'pred[v]' is updated to the source u of an edge u, v when the target label must be corrected. This allows us to determine which edge relaxation was last executed, and therefore occurs in a longest path between the source and that node. By recursively following the predecessor, we can determine in linear time one of the longest paths in the graph. If the graph contains positive cycles, we perform a similar recursion to find the start of a cycle, and then continue until the start of the cycle occurs again.

Note that if the edges are sorted in order of a longest path, that the sequential relaxation will find the longest paths in the first relaxation of iteration. In the second relaxation iteration, none of the labels will change anymore. This means that in a constant three iterations (including the feasibility check), the algorithm terminates in time linear to $|E|$.

Algorithm 17 Bellman-Ford-Moore

```
function BFM(Graph $G(V,E)$, weight function $w$, source $v_s \in V$)
    for all $v_i \in V$ do
        $l_i = -\infty$ for each $v_i \in V$ // Initialize labels to 'unreachable'
        pred[$v_i$]= NIL // Initialize the predecessors subgraph
    $l_s = 0$ // Initialize the source to distance 0
    for $i \in \{1, \ldots, |V|-1\}$ do
        $c$ = true // Assume converged
        for all $(u, v) \in E$ do
            $c = c \wedge \neg$RELAX($d, u, v, w,$pred)
        if $c$ then break
    // Check whether the labels have converged
    for each edge $u, v \in E_a$ do
        if RELAX($d, u, v, w,$pred) then return false, $d$
    return true, $d$
```

Algorithm 18 Relax one edge

```
function RELAX($d, u, v, w,$pred)
    if $d(u) + w(u,v) > d(v)$ then
        $d(v) = d(u) + w(u,v)$
        pred[$v$] = $u$
        return true
    return false
```

A.4 Multiple sources BFM algorithm

For online problems, the running time of BFM can become a performance bottleneck. The graphs in such online problems can grow arbitrarily large. For online algorithms, we would like to reuse previously calculated distance labels to increase the efficiency of the computations. To do so, we extend the interpretation of the labels and the definition of feasibility.

Algorithm 19 Bellman-Ford-Moore with multiple sources

```
function BFM(Graph $G(V,E,w)$, subset of active vertices $V_a \subset V$, set of sources $V_s \subset V$, begin times for sources $d'$)
    for each $a \in V_s$ do
        $d[a] = d'[a]$
        for each $(a,v) \in E$ do
            RELAX($d, a, v, w$)
    $f$ = true // assume feasible
    $E_a = \{(x,y) \in E : x \in V_a\}$
    for each $i \in \{1,\dots,|V_a|-1\}$ do
        for all $(u,v) \in E_a$ do
            $r$ =RELAX($d, u, v, w$)
            if $v \in V_s \wedge r$ then $f$ = false
    for each edge $u, v \in E_a$ do
        if (RELAX($d, u, v, w$) then $f$ = false
    return $f, d$
```

The algorithm starts by relaxing all the source nodes, to find the initial labels of the active subgraph. It then determines the edges for which the subgraph needs to be relaxed. During the relaxation of each of these edges, it is checked whether any of the source nodes would need to be relaxed. If this is the case, then the result is infeasible; i.e., there is no valid longest path that allows the source nodes to keep their initial distance labels.

A.4.1 Correctness

The sources are used to find the initial distance to the active subgraph, and the modified BFM algorithm then relaxes for each edge in the active subgraph. The algorithm detects positive cycles of length $|E_a|$ in the subgraph induced by E_a, using the same argument as for the original BFM, except that there are multiple sources.

The timestamps of the sources are not allowed to be changed, and therefore any relaxation that would perform that, means that there are no distance labels possible for this active subgraph that allows the constraints of the active subgraph to be satisfied without changes in the source labels.

The begin times of operations in jobs before the eligible operation's job are not allowed to be changed (Line 10 in Algorithm 19), as changing them could lead to an infeasible schedule through a positive cycle that is outside the sub-graph.

When we consider the invariant that at the beginning of Algorithm 19 all times for the scheduled operations were feasible, then it can detect infeasibility within the horizon when the set of sources consists of the operations of the last-scheduled job plus operations containing sequence-dependent set-up times into the active vertices, and the set of active vertices consists of the operations of all jobs that cannot be delayed indefinitely any more. The jobs that can be delayed indefinitely are the ones starting from the insertion point, e.g., operation $o_{4,1}$ in Figure 2.2. At the end of Algorithm 19 we either find that the sequence is infeasible, or we find feasible begin times, which can be used in the next iteration as initial estimates to speed up convergence of the longest-path computations.

A.4.2 Time complexity

The time complexity of the algorithm is $\mathcal{O}(\max(|V_s|, |V_a||E_a|))$, as the first for-loop takes $\mathcal{O}(V_s|)$ time, and the second for-loop takes $\mathcal{O}(|V_a||E_a|)$.

B

Proofs for convex polyhedra

This appendix describes some properties of convex polyhedra, and shows some proofs related to this thesis.

B.1 Convex polyhedra

A polyhedron is an n-dimensional solid with flat surfaces.

Definition 7 (convex set)**.** *A set* C *is convex if and only if for each point* $\boldsymbol{x}, \boldsymbol{y} \in C$ *the point* $t\boldsymbol{x} + (1-t)\boldsymbol{y}$ *for all* $t \in (0,1)$ *is also a member of* C.

A convex polyhedron $h \subseteq R^n$ is a subset of an n-dimensional space, such that for any two points $\boldsymbol{x}, \boldsymbol{y} \in h \implies t\boldsymbol{x} + (1-t)\boldsymbol{y} \in h$ for all $t \in (0,1)$. That is, any point that lies on a direct line between them is also part of the convex polyhedron. Note that the empty set is also considered a convex set.

Definition 8 (convex hull)**.** *The convex hull of a finite set* S *of points is the set of all convex combinations of its points:* $conv(S) = \left\{ \sum_{i=1}^{|S|} \gamma_i \boldsymbol{x}_i \mid (\forall i : \gamma_i \geq 0) \wedge \sum_{i=1}^{S} \gamma_i = 1 \right\}$.

The vertices (i.e., the extreme points) of $\mathrm{conv}(S)$ are the points $v_i \in \mathrm{conv}(S)$ in of the convex hull that are not in the convex hull of the other points ($v_i \notin \mathrm{conv}(S \setminus \{v_i\})$.

B.2 Convexity of the maximum of expressions

An affine expression $\boldsymbol{e}(\boldsymbol{p}) = \alpha_1 p_1 \cdots + \alpha_d p_d + \alpha_{d+1}$ for a d-dimensional space can be written as the inner product of a vector and the point at which it is evaluated: $[\alpha_1 \ldots \alpha_{d+1}] \cdot [\boldsymbol{p}\ 1]$. Given a set $D \subseteq 2^{R^d+1}$ of affine expressions, let $f(\boldsymbol{p})$ be the maximum of each function at any point $\boldsymbol{p}$ $f_{\max}(\boldsymbol{p}) = \max_{e \in D}$.

An expression $\boldsymbol{e}$ is maximal at a point $\boldsymbol{p}$ if and only if $\boldsymbol{e}_{max}(\boldsymbol{p}) = \boldsymbol{e}(\boldsymbol{p})$. The set of points for which an expression is maximal then form a convex polyhedron (proposition and proof adapted from [47]):

Proposition 3. $\left\{ \boldsymbol{p} \in \mathbb{R}^d \mid M(\boldsymbol{p}) = \boldsymbol{e}(\boldsymbol{p}) \right\}$ *is a convex polyhedron for any (critical path) expression* $\boldsymbol{e}$.

Proof. Given a set of expressions D, for any expression $e \in D$ the polyhedron $h = \left\{ \boldsymbol{p} \in \mathbb{R}^n \mid e_{\max}(\boldsymbol{p}) = \boldsymbol{e}(\boldsymbol{p}) \right\}$ is a convex polyhedron if and only if for any two points $p_1, p_2 \in h$, the point $t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2 \in h$ for all $t \in (0,1)$. Then we need to show that for each $t \in (0,1)$, the point $t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2$ is also a member of h. That is: $\boldsymbol{e}(t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2) = \max_{\boldsymbol{e}_i \in D}(t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2)$ or equivalently:

$$\boldsymbol{e}(t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2) \geq \boldsymbol{e}_i(t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2) \text{ for all } \boldsymbol{e}_i \in D \tag{B.1}$$

We know that if the point $\boldsymbol{p}_1 \in h$, then the expression e is maximal in $\boldsymbol{p}_1$; $\boldsymbol{e}(\boldsymbol{p}_1) = \max_{\boldsymbol{e}_i \in D}(\boldsymbol{e}_i(\boldsymbol{p}_1))$. Similarly, for point $\boldsymbol{p_2} \in h$ the same expression e is maximal. This can be written as:

$$\boldsymbol{e}(\boldsymbol{p}_1) \geq \boldsymbol{e}_i(\boldsymbol{p}_1) \text{ for all } \boldsymbol{e}_i \in D \tag{B.2}$$

$$\boldsymbol{e}(\boldsymbol{p}_2) \geq \boldsymbol{e}_i(\boldsymbol{p}_2) \text{ for all } \boldsymbol{e}_i \in D \tag{B.3}$$

Using the linearity of the expression $\boldsymbol{e}$ in B.4 and B.6 and replacing $\boldsymbol{e}$ using the inequalities of Equations B.2 and B.3 in B.5, we get:

$$\begin{aligned}
\boldsymbol{e}(t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2) &= t\boldsymbol{e}(\boldsymbol{p}_1) + (1-t)\boldsymbol{e}(\boldsymbol{p}_2) && && \text{(B.4)}\\
&\geq t\boldsymbol{e}_i(\boldsymbol{p}_1) + (1-t)\boldsymbol{e}_i(\boldsymbol{p}_2) && \text{for all } \boldsymbol{e}_i \in D && \text{(B.5)}\\
&= \boldsymbol{e}_i(t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2) && \text{for all } \boldsymbol{e}_i \in D && \text{(B.6)}
\end{aligned}$$

Combining these results completes the proof:

$$\begin{aligned}
&\boldsymbol{e}(t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2) \geq \boldsymbol{e}_i(t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2) \text{ for all } \boldsymbol{e}_i \in D && \text{(B.7)}\\
&\iff \boldsymbol{e}(t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2) = \max_{\boldsymbol{e}_i \in D}(t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2) && \text{(B.8)}\\
&\iff (t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2) \in h && \text{(B.9)}
\end{aligned}$$

□

Any half-space s can be described by an affine expression $\boldsymbol{e}$ such that $s(\boldsymbol{e}) = \{\boldsymbol{p} \in \mathbb{R}^d \mid \boldsymbol{e}(\boldsymbol{p}) \geq 0\}$. A convex polyhedron can also be represented as the intersection of a finite set of half-spaces, each of which is represented by an expression or vector. I.e., a polygon can be represented by a subset $h \subset \mathbb{R}^{d+1}$. We lift the function s to sets of half-spaces: $s(h) = \bigcap_{\boldsymbol{e} \in h} s(\boldsymbol{e})$. A convex polyhedron can therefore be described by a set of expressions that correspond to half-spaces. Proposition 4 helps to determine the feasible parameter combinations.

Proposition 4. *If all corners of a convex polyhedron with half-space representation h are feasible, then all points in* $s(h)$ *are feasible.*

Proof. If there would be a parameter point $\boldsymbol{p}$ in the polyhedron (with half-space representation h) that is infeasible, then at that point a positive cycle with cumulative weight $V(\boldsymbol{x}) = \boldsymbol{b} \cdot \boldsymbol{x} + c$ must exist such that $V(\boldsymbol{p}) > 0$. The inequality $\boldsymbol{b} \cdot \boldsymbol{x} + c$ corresponds to a half-space that contains $\boldsymbol{p}$. The half-space containing $\boldsymbol{p}$ includes at least one corner point of h. Therefore, that corner point must be infeasible too. This is a contradiction and therefore proves the proposition. □

B.3 Transformations of convex polyhedra

A convex polyhedron in the parameter space can be transformed by an affine transformation $\boldsymbol{T}$. A polyhedron can be described by a collection of half-spaces, or by the convex hull of its extreme points [42].

Figure B.1 shows that the intersection of the transformation of the half-spaces is not sufficient to find the corresponding polyhedron in the transformation's target space. The polyhedron is described by the intersection of three half spaces. Each half space in the parameter space has a corresponding transformed half space in the cost-makespan space. Let the makespan expression of this polyhedron be $10 - p - q$ and the cost function $p + q$. The points that lie in the convex hull of the extremes in the parameter space (Figure B.1a) map to the line segment between the two red squares in Figure B.1b. However, this line segment requires four half spaces to intersect, otherwise points are included that do not have a counterpart in the parameter space.

Therefore, we first convert the extremes of the polyhedron in the parameter space to extremes of a polyhedron in the cost-makespan space.

Proposition 5. *Applying an affine transformation to all points of a convex region R_1 leads to another convex region R_2.*

Proof. Given an affine transformation $T(\boldsymbol{x}) = \boldsymbol{P}\boldsymbol{x} + \boldsymbol{p}$ the transformed region is determined by:

$$R_2 = \{\, T(\boldsymbol{x}) \mid \boldsymbol{x} \in R_1 \,\} \tag{B.10}$$

As R_1 is a convex region, it can be described as the convex hull of its vertices S_1:

$$R_1 = \operatorname{conv}(S_1) \tag{B.11}$$

Combining Equations B.10 and B.11 and expanding $\operatorname{conv}(S_1)$ (Definition 8), we can write:

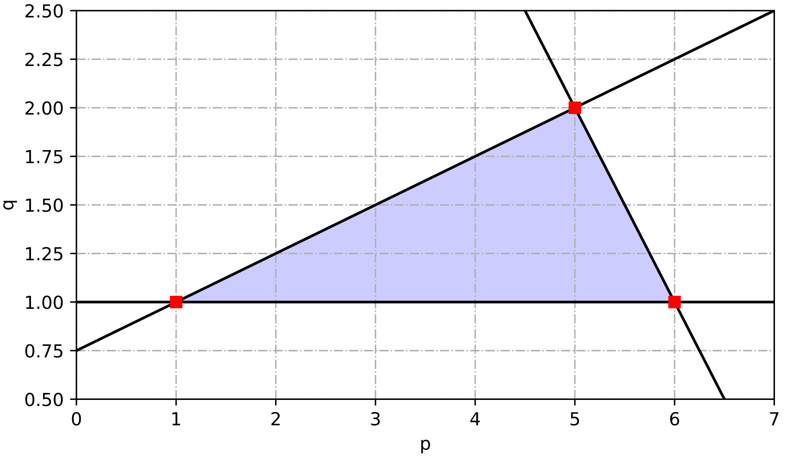
$$\begin{aligned}
R_2 &= \{\, T(\boldsymbol{x}) \mid \boldsymbol{x} \in \operatorname{conv}(S_1) \} && \text{(B.12)}\\
&= \left\{ T(\boldsymbol{x}) \mid \boldsymbol{x} \in \left\{ \sum_{i=1}^{|S_1|} \gamma_i \boldsymbol{v}_i \mid (\forall i : \gamma_i \geq 0) \wedge \sum_{i=1}^{S_1} \gamma_i = 1 \right\} \right\} && \text{(B.13)}\\
&= \left\{ T\left(\sum_{i=1}^{|S_1|} \gamma_i \boldsymbol{v}_i \right) \mid (\forall i : \gamma_i \geq 0) \wedge \sum_{i=1}^{S_1} \gamma_i = 1 \right\} && \text{(B.14)}
\end{aligned}$$

Then we apply the definition of affine transformation $T(\boldsymbol{x})$ and due to linearity of $\boldsymbol{P}$ and $\sum_{i=1}^{\gamma_i} \boldsymbol{p} = \boldsymbol{p}$ to obtain:

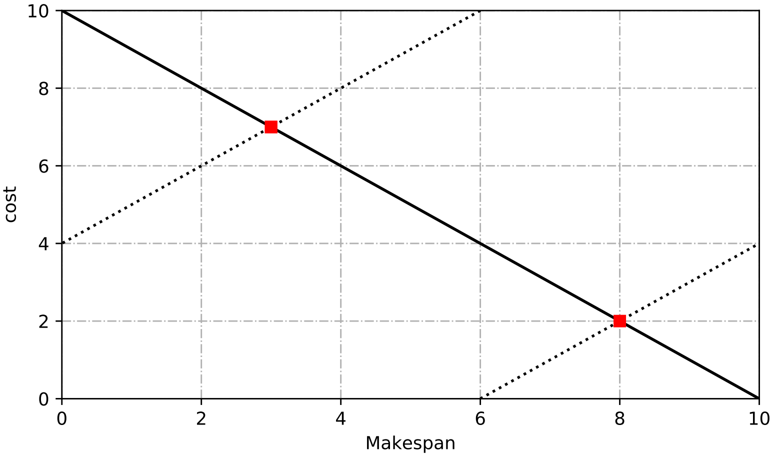
$$R_2 = \left\{ \boldsymbol{P}\left(\sum_{i=1}^{|S_1|} \gamma_i \boldsymbol{v}_i\right) + \boldsymbol{p} \mid (\forall \boldsymbol{i} : \gamma_i \geq 0) \wedge \sum_{i=1}^{S_1} \gamma_i = 1 \right\} \tag{B.15}$$

$$= \left\{ \left(\sum_{i=1}^{|S_1|} \gamma_i (\boldsymbol{P}\boldsymbol{v}_i + \boldsymbol{p})\right) \mid (\forall \boldsymbol{i} : \gamma_i \geq 0) \wedge \sum_{i=1}^{S_1} \gamma_i = 1 \right\} \tag{B.16}$$

$$= \left\{ \left(\sum_{i=1}^{|S_1|} \gamma_i T(\boldsymbol{v}_i)\right) \mid (\forall \boldsymbol{i} : \gamma_i \geq 0) \wedge \sum_{i=1}^{S_1} \gamma_i = 1 \right\} \tag{B.17}$$



(a) A convex region before applying an affine transformation.



(b) A convex region after applying an affine transformation.

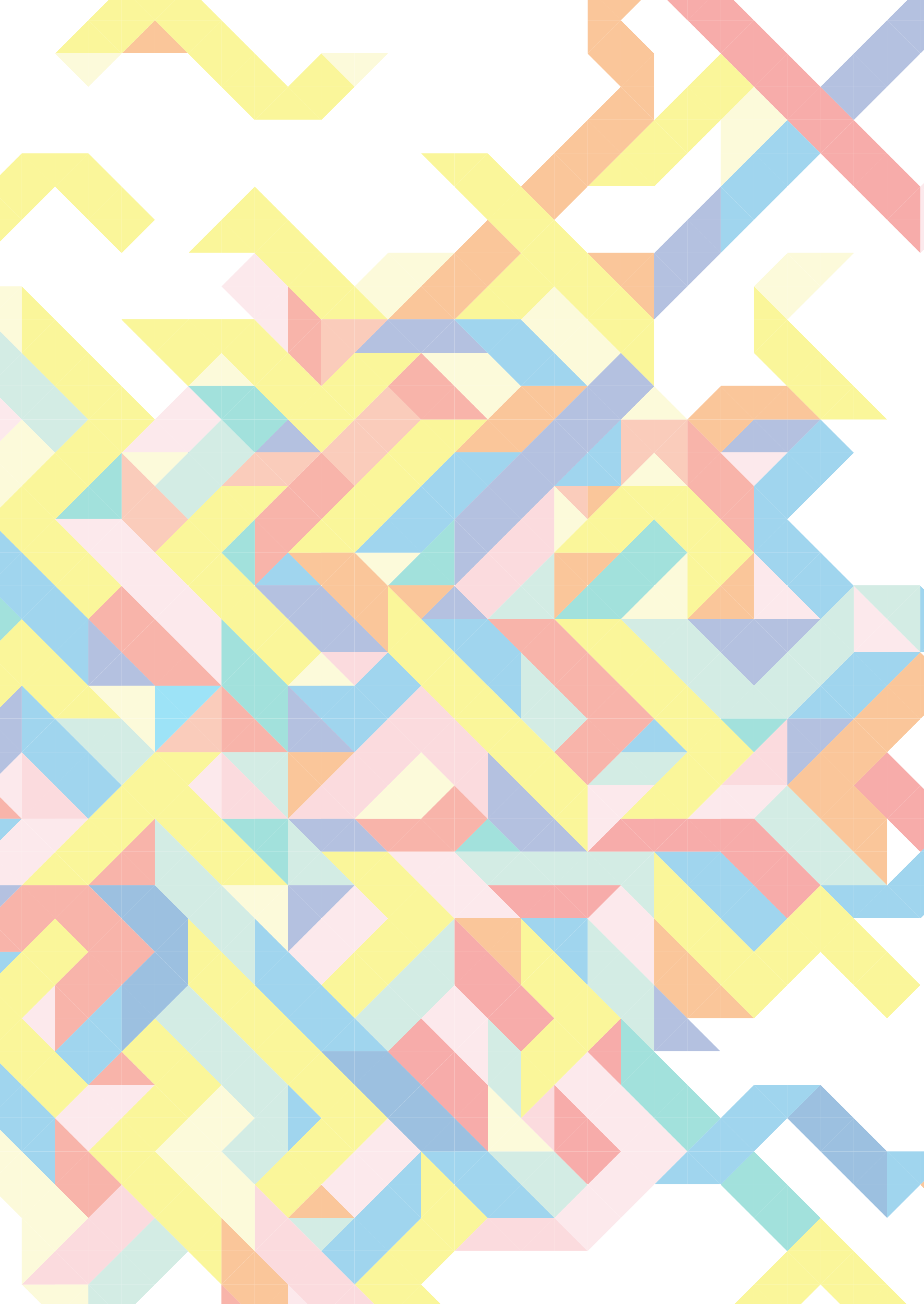
Figure B.1: Affine transformation of a convex polyhedron, from parameter space (p,q) to (Makespan, cost).

There are at most $|S_1|$ different transformed corners in the set S_2. Applying Definition 8 leads to:

$$R_2 = \text{conv}(\{\, T(\boldsymbol{v}_i) \mid \boldsymbol{v}_i \in S_1 \,\}) \tag{B.18}$$

We have therefore shown that the transformation of the convex hull of a set of points is equal to the convex hull of the transformation of those points. □

This proof also shows that it is sufficient to transform all vertices of a convex region, to find all points in the transformed region.



Bibliography

[1] Shreya Adyanthaya, Hadi Alizadeh Ara, João Bastos, Amir Behrouzian, Róbinson Medina Sánchez, Joost van Pinxten, Bram van der Sanden, Umar Waqas, Twan Basten, Henk Corporaal, Raymond Frijns, Marc Geilen, Dip Goswami, Martijn Hendriks, Sander Stuijk, Michel Reniers, and Jeroen Voeten. xCPS: A tool to eXplore Cyber Physical Systems. *ACM Special Interest Group on Embedded Systems (SIGBED)*, 2017. (Cited on page 173.)

[2] Shreya Adyanthaya, Hadi Alizadeh Ara, João Bastos, Amir Behrouzian, Róbinson Medina Sánchez, Joost van Pinxten, Bram van der Sanden, Umar Waqas, Twan Basten, Henk Corporaal, Raymond Frijns, Marc Geilen, Dip Goswami, Sander Stuijk, Michel Reniers, and Jeroen Voeten. xCPS: A tool to eXplore Cyber Physical Systems. In *Proceedings of 2015 Workshop on Embedded and Cyber-Physical Systems Education*, pages 3:1–3:8, 2015. (Cited on page 174.)

[3] Rajeev Alur and David Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994. (Cited on page 122.)

[4] Yash Aneja and Kunhiraman Nair. Bicriteria transportation problem. *Management Science*, 25(1):73–78, 1979. (Cited on pages 52, 53, and 55.)

[5] Eric Angel, Evripidis Bampis, and Laurent Gourvès. A dynasearch neighborhood for the bicriteria traveling salesman problem. In Xavier Gandibleux, Marc Sevaux, Kenneth Sörensen, and Vincent T'kindt, editors, *Metaheuristics for Multiobjective Optimisation*, pages 153–176, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. (Cited on pages 49 and 51.)

[6] David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde TSP solver, 2006. (Cited on page 50.)

[7] David Applegate, Robert Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2006. (Cited on page 50.)

[8] Twan Basten, Roelof Hamberg, Frans Reckers, and Jacques Verriet. *Model-based design of adaptive embedded systems*. Springer, 2013. (Cited on page 11.)

[9] João Bastos, Bram van der Sanden, Olaf Donkx, Jeroen Voeten, Sander Stuijk, Ramon Schiffelers, and Henk Corporaal. Identifying bottlenecks in manufacturing systems using stochastic criticality analysis. In *2017 Forum on Specification and Design Languages (FDL)*, pages 1–8, Sept 2017. (Cited on page 95.)

[10] Joao Bastos, Sander Stuijk, Jeroen Voeten, Ramon Schiffelers, Johan Jacobs, and Henk Corporaal. Modeling resource sharing using FSM-SADF. In *2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 96–101, Sept 2015. (Cited on page 10.)

[11] Gerd Behrmann, Ed Brinksma, Martijn Hendriks, and Angelika Mader. Production scheduling by reachability analysis: a case study. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 140–142, Los Alamitos, CA, USA, 2005. IEEE. (Cited on pages 18, 41, 42, and 77.)

[12] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958. (Cited on pages 18, 24, and 77.)

[13] Jon Jouis Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on computing*, 4(4):387–411, 1992. (Cited on page 60.)

[14] Enrico Bini and Giorgio Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, Nov 2004. (Cited on pages 96 and 121.)

[15] Pedro Borges and Michael Hansen. A basis for future successes in multiobjective combinatorial optimization. *Technical report, Department of Mathematical Modelling, Technical University of Denmark*, 1998. (Cited on page 51.)

[16] Leo Breiman. *Classification and regression trees*. Routledge, 2017. (Cited on page 122.)

[17] Jim Browne, Didier Dubois, Keith Rathmill, Suresh Sethi, and Kathryn Stecke. Classification of flexible manufacturing systems. *The FMS magazine*, 2(2):114–117, 1984. (Cited on page 17.)

[18] Frederico Busato and Nicola Bombieri. An efficient implementation of the Bellman-Ford algorithm for Kepler GPU architectures. *IEEE Transactions on Parallel and Distributed Systems*, 27(8):2222–2233, Aug 2016. (Cited on page 30.)

[19] Suresh Chand, Rodney Traub, and Reha Uzsoy. Rolling horizon procedures for the single machine deterministic total completion time scheduling problem with release dates. *Annals of Operations Research*, 70(0):115–125, 1997. (Cited on pages 17, 41, and 42.)

[20] Jen-Shiang Chen, Jason Chao-Hsien Pan, and Chien-Kuang Wu. Hybrid tabu search for re-entrant permutation flow-shop scheduling problem. *Expert Systems with Applications*, 34(3):1924–1930, 2008. (Cited on page 41.)

[21] Seong-Woo Choi and Yeong-Dae Kim. Minimizing makespan on a two-machine re-entrant flowshop. *Journal of the Operational Research Society*, 58(7):972–981, Jul 2007. (Cited on pages 18 and 43.)

[22] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976. (Cited on page 50.)

[23] Alessandro Cimatti, Luigi Palopoli, and Yusi Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *2008 Real-Time Systems Symposium*, pages 80–89, Nov 2008. (Cited on page 96.)

[24] McKinsey & Company. The great re-make: Manufacturing for modern times. https://www.mckinsey.com/~/media/McKinsey/BusinessFunctions/Operations/Our%20Insights/The%20great%20remake%20Manufacturing%20for%20modern%20times/The-great-remake-Manufacturing-for-modern-times-full-compenium-October-2017-final.ashx. (Cited on page 3.)

[25] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001. (Cited on pages 135 and 136.)

[26] National Research Council et al. *The changing nature of work: Implications for occupational analysis*. National Academies Press, 1999. (Cited on page 3.)

[27] David Craft, Tarek Halabi, Helen Shih, and Thomas Bortfeld. Approximating convex pareto surfaces in multiobjective radiotherapy planning. *Medical physics*, 33(9):3399–3407, 2006. (Cited on pages 55 and 60.)

[28] Morteza Damavandpeyma, Sander Stuijk, Marc Geilen, Twan Basten, and Henk Corporaal. Parametric throughput analysis of scenario-aware dataflow graphs. In *2012 IEEE 30th International Conference on Computer Design (ICCD)*, pages 219–226, Sept 2012. (Cited on page 96.)

[29] Ebru Demirkol and Reha Uzsoy. Decomposition methods for reentrant flow shops with sequence-dependent setup times. *Journal of scheduling*, 3(3):155–177, 2000. (Cited on pages 17, 41, and 42.)

[30] Stephen Devaux. *Total project control : a practitioner's guide to managing projects as investments*. CRC Press, Boca Raton, 2nd edition, 2015. (Cited on page 95.)

[31] Edsger Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959. (Cited on page 135.)

[32] Joseph Ecker, Nancy Hegner, and Issoufou Kouada. Generating all maximal efficient faces for multiple objective linear programs. *Journal of Optimization Theory and Applications*, 30(3):353–381, Mar 1980. (Cited on pages 84 and 96.)

[33] Matthias Ehrgott. Approximation algorithms for combinatorial multicriteria optimization problems. *International Transactions in Operational Research*, 7:5 – 31, 2000. (Cited on page 51.)

[34] Matthias Ehrgott. *A Multiobjective Simplex Method*, pages 171–196. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. (Cited on page 96.)

[35] Matthias Ehrgott. *Multiobjective Versions of Polynomially Solvable Problems*, pages 171–196. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. (Cited on page 56.)

[36] Matthias Ehrgott. A discussion of scalarization techniques for multiple objective integer programming. *OR*, 147:343–360, 2006. (Cited on pages 54 and 59.)

[37] Salah Elmaghraby and Jerzy Kamburowski. The analysis of activity networks under generalized precedence relations (GPRs). *Management science*, 38(9):1245–1263, 1992. (Cited on pages 72, 77, 94, 95, and 96.)

[38] Paul Feautrier. Parametric integer programming. *RAIRO Recherche Opèrationnelle*, 22(3):243–268, 1988. (Cited on pages 95 and 122.)

[39] Matteo Fischetti, Juan Jose Salazar Gonzalez, and Paolo Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Oper. Res.*, 45(3):378–394, 1997. (Cited on pages 49, 52, 53, 56, 57, 60, and 61.)

[40] Kostas Florios and George Mavrotas. Generation of the exact Pareto set in multi-objective traveling salesman and set covering problems. *Applied Mathematics and Computation*, 237, 2014. (Cited on pages 51 and 53.)

[41] Lester Ford Jr. *Network Flow Theory*. Rand Corp, Santa Monica, CA, USA, 1956. (Cited on pages 18, 24, and 77.)

[42] Komei Fukuda and Alain Prodon. Double description method revisited. *Combinatorics and Computer Science*, pages 91–111, 1996. (Cited on pages 82, 85, 110, 113, and 143.)

[43] Tomas Gal and Josef Nedoma. Multiparametric linear programming. *Management Science*, 18(7):406–422, 1972. (Cited on page 95.)

[44] Paul Gastin, Sayan Mukherjee, and B Srivathsan. Reachability in timed automata with diagonal constraints. *arXiv preprint arXiv:1806.11007*, 06 2018. (Cited on pages 121 and 122.)

[45] Marc Geilen and Twan Basten. A calculator for Pareto points. In *DATE'07*, pages 1–6. IEEE, 2007. (Cited on pages 31 and 34.)

[46] Marc Geilen, Twan Basten, Bart Theelen, and Ralph Otten. An algebra of Pareto points. In *Fundamenta Informaticae*, pages 88–97. IEEE Computer Society Press, 2005. (Cited on pages 31 and 32.)

[47] Amir Ghamarian, Marc Geilen, Twan Basten, and Sander Stuijk. Parametric throughput analysis of synchronous data flow graphs. In *2008 Design, Automation and Test in Europe*, pages 116–121, March 2008. (Cited on pages 76, 78, 80, 82, 96, 122, and 141.)

[48] Józef Grabowski, Ewa Skubalska, and Czesław Smutnicki. On flow shop scheduling with release and due dates to minimize maximum lateness. *Journal of Operational Research Society*, 34(7):615–620, 1983. (Cited on page 42.)

[49] Ronald Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132 – 133, 1972. (Cited on page 84.)

[50] Ronald Graham, Eugene Lawler, Jan Karel Lenstra, and Alexander Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979. (Cited on page 41.)

[51] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.1.2 edition, 2016. http://gmplib.org/. (Cited on page 85.)

[52] Gregory Gutin and Daniel Karapetyan. A memetic algorithm for the generalized traveling salesman problem. *Natural Computing*, 9(1):47–60, 2010. (Cited on pages 49, 52, 53, 55, and 60.)

[53] Gregory Gutin, Daniel Karapetyan, and Natalio Krasnogor. Memetic algorithm for the generalized asymmetric traveling salesman problem. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, volume 129 of *Studies in Computational Intelligence*, pages 199–210. Springer Berlin Heidelberg, 2008. (Cited on pages 52 and 53.)

[54] Gregory Gutin and Abraham Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006. (Cited on page 50.)

[55] Miklós Hajdu. *Network scheduling techniques for construction project management*, volume 16. Springer Science & Business Media, 2013. (Cited on page 95.)

[56] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968. (Cited on page 135.)

[57] Maurice Heemels and Gerrit-Jan Muller. *Boderc: model-based design of high-tech systems : a collaborative research project for multi-disciplinary design analysis of high-tech systems.* Embedded Systems Institute, 2006. (Cited on page 10.)

[58] Khaled Heloue, Sari Onaissi, and Farid Najm. Efficient block-based parameterized timing analysis covering all potentially critical paths. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(4):472–484, April 2012. (Cited on page 96.)

[59] Martijn Hendriks, Marc Geilen, and Twan Basten. Pareto analysis with uncertainty. In *Proceedings of the 2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing*, EUC '11, pages 189–196, Washington, DC, USA, 2011. IEEE Computer Society. (Cited on page 128.)

[60] Rob Hoogendijk and Jack Kandelaars. De knepen van de paperhandling. https://bits-chips.nl/artikel/de-knepen-van-de-paperhandling-46452.html, 2016. (Cited on page 10.)

[61] Jozef Hooman, Mooij Arjan, and Hans van Wezep. Early fault detection in industry using models at various abstraction levels. In *Integrated Formal Methods - 9th International Conference, IFM 2012, Pisa, Italy, June 18-21, 2012. Proceedings*, pages 268–282, 2012. (Cited on page 10.)

[62] Holger Hoos and Thomas Stützle. *Stochastic local search: Foundations and applications.* Elsevier, 2004. (Cited on page 51.)

[63] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits Vaandrager. Linear parametric model checking of timed automata. In Tiziana Margaria and Wang Yi, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 189–203, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. (Cited on pages 96 and 121.)

[64] IBM. *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual*, 2017. (Cited on page 37.)

[65] BongJoo Jeong and Yeong-Dae Kim. Minimizing total tardiness in a two-machine re-entrant flowshop with sequence-dependent setup times. *Computers and Operations Research*, 47:72 – 80, 2014. (Cited on pages 41 and 43.)

[66] Caixia Jing, Wanzhen Huang, and Guochun Tang. Minimizing total completion time for re-entrant flow shop scheduling problems. *Theoretical Computer Science*, 412(48):6712 – 6719, 2011. (Cited on page 41.)

[67] David Johnson and Lyle McGeoch. The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, 1:215–310, 1997. (Cited on page 60.)

[68] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The traveling salesman problem. *Handbook in Operations Research and Management Science: Network Models*, 1995. (Cited on page 50.)

[69] Jitti Jungwattanakit, Manop Reodecha, Paveena Chaovalitwongse, and Frank Werner. Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. *International Journal of Advanced Manufacturing Technology*, 37(3):354–370, 2008. (Cited on pages 41 and 42.)

[70] Daniel Karapetyan and Gregory Gutin. Efficient local search algorithms for known and new neighborhoods for the generalized traveling salesman problem. *Eur. J. of Oper. Res.*, 219:234–251, 2012. (Cited on page 60.)

[71] James Kelley, Jr and Morgan. Walker. Critical-path planning and scheduling. In *Eastern Joint IRE-AIEE-ACM Computer Conference*, IRE-AIEE-ACM '59 (Eastern), pages 160–173, New York, NY, USA, 1959. ACM. (Cited on pages 71 and 94.)

[72] Joep Kerbosch and Henk Schell. Network planning by the extended metra potential method (EMPM), 1975. (Cited on pages 72, 77, and 94.)

[73] Masanori Kobayashi and Shinsuke Odagiri. Tropical geometry of pert. *arXiv preprint arXiv:1202.6457*, 2012. (Cited on page 95.)

[74] Wieslaw Kubiak, Sheldon Lou, and Yingmeng Wang. Mean flow time minimization in reentrant job shops with a hub. *Operations Research*, 44(5):764–776, 1996. (Cited on pages 41 and 43.)

[75] Murat Köksalan and Diclehan Tezcaner Öztürk. An evolutionary approach to generalized biobjective traveling salesperson problem. *Computers & Operations Research*, 79:304 – 313, 2017. (Cited on page 53.)

[76] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345 – 358, 1992. (Cited on page 47.)

[77] Eugene Lawler, Jan Karel Lenstra, Alexander Kan, and David Shmoys. *The traveling salesman problem: a guided tour of combinatorial optimization.* New York (Wiley), 1987. (Cited on pages 50 and 59.)

[78] Jan Karel Lenstra, Alexander Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. In P.L. Hammer, E.L. Johnson, B.H. Korte, and G.L. Nemhauser, editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343 – 362. Elsevier, 1977. (Cited on page 101.)

[79] Eugene Levner and Vladimir Kats. A parametric critical path problem and an application for cyclic scheduling. *Discrete Applied Mathematics*, 87(1):149 – 158, 1998. (Cited on pages 82, 83, 95, and 96.)

[80] Alexis Linard, Rick Smetsers, Frits Vaandrager, Umar Waqas, Joost van Pinxten, and Sicco Verwer. Learning pairwise disjoint simple languages from positive examples. In *LearnAut 2017*, volume abs/1706.01663, 2017. (Cited on page 174.)

[81] Andreas Löhne and Benjamin Weißing. Equivalence between polyhedral projection, multiple objective linear programming and vector linear programming. *Mathematical Methods of Operations Research*, 84(2):411–426, Oct 2016. (Cited on page 96.)

[82] Thibaut Lust and Jacques Teghem. The multiobjective traveling salesman problem: A survey and a new approach. In *Advances in multi-objective nature inspired computing*, pages 119–141. Springer, 2010. (Cited on page 64.)

[83] Thibaut Lust and Jacques Teghem. Two-phase Pareto local search for the biobjective traveling salesman problem. *Journal of Heuristics*, 16(3):475–510, Jun 2010. (Cited on pages 49, 51, 52, 53, 54, 55, and 56.)

[84] Donald Malcolm, John Roseboom, Charles Clark, and Willard Fazar. Application of a technique for research and development program evaluation. *Operations research*, 7(5):646–669, 1959. (Cited on page 71.)

[85] Niall. McCarthy. Automation could eliminate 73 million u.s. jobs by 2030 [digital image]. https://www.statista.com/chart/12082/automation-could-eliminate-73-million-us-jobs-by-2030/, 2017. (Cited on page 3.)

[86] Hristina Moneva, Jurjen Caarls, and Jacques Verriet. A holonic approach to warehouse control. In Yves Demazeau, Juan Pavón, Juan M. Corchado, and Javier Bajo, editors, *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*, pages 1–10, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. (Cited on page 10.)

[87] Manuel Montoya Aguirre. Compositional Pareto-algebraic heuristic for packing problems. Master's thesis, Technische Universiteit Eindhoven, 2016. (Cited on page 32.)

[88] Arjan Mooij, Jozef Hooman, and Rob Albers. Early fault detection using design models for collision prevention in medical equipment. In *Foundations of Health Information Engineering and Systems - Third International Symposium, FHIES 2013, Macau, China, August 21-23, 2013. Revised Selected Papers*, pages 170–187, 2013. (Cited on page 10.)

[89] Marcin Mucha and Maxim Sviridenko. No-wait flowshop scheduling is as hard as asymmetric traveling salesman problem. In Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming*, pages 769–779, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. (Cited on page 47.)

[90] Klaus Neumann and Christoph Schwindt. Activity-on-node networks with minimal and maximal time lags and their application to make-to-order production. *Operations-Research-Spektrum*, 19(3):205–217, Sep 1997. (Cited on page 71.)

[91] Océ, a Canon Company. Océ Technologies website. http://oce.com/, August 2018. (Cited on page 8.)

[92] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991. (Cited on page 50.)

[93] Jason Chao-Hsien Pan and Jen-Shiang Chen. Minimizing makespan in reentrant permutation flow-shops. *Journal of the Operational Research Society*, 54(6):642–653, 2003. (Cited on page 41.)

[94] Luis Paquete, Marco Chiarandini, and Thomas Stützle. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In Xavier Gandibleux, Marc Sevaux, Kenneth Sörensen, and Vincent T'kindt, editors, *Metaheuristics for Multiobjective Optimisation*, pages 177–199, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. (Cited on pages 51, 54, and 56.)

[95] Luís Paquete and Thomas Stützle. Clusters of non-dominated solutions in multiobjective combinatorial optimization: An experimental analysis. In Vincent Barichard, Matthias Ehrgott, Xavier Gandibleux, and Vincent T'Kindt, editors, *Multiobjective Programming and Goal Programming*, pages 69–77, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. (Cited on pages 51 and 54.)

[96] Luís Paquete and Thomas Stützle. Design and analysis of stochastic local search for the multiobjective traveling salesman problem. *Comput. Oper. Res.*, 36(9):2619 – 2631, 2009. (Cited on pages 49, 51, 52, and 60.)

[97] Luis Paquete and Thomas Stützle. A two-phase local search for the biobjective traveling salesman problem. In CarlosM. Fonseca, PeterJ. Fleming, Eckart Zitzler, Lothar Thiele, and Kalyanmoy Deb, editors, *Evolutionary Multi-Criterion Optimization*, volume 2632 of *Lecture Notes in Computer Science*, pages 479–493. Springer Berlin Heidelberg, 2003. (Cited on pages 51, 52, and 61.)

[98] Wen Lea Pearn, S Chung, M Yang, and K Shiao. Solution strategies for multistage wafer probing scheduling problem with reentry. *Journal of the Operational Research Society*, 59(5):637–651, 2008. (Cited on page 18.)

[99] Gerhard Reinelt. TSPLIB A traveling salesman problem library. *ORSA J. Comput.*, 1991. (Cited on pages 47 and 61.)

[100] Nieke Roos. Multidisciplinair software ontwikkelen op een virtuele printer. https://bits-chips.nl/artikel/multidisciplinair-software-ontwikkelen-op-een-virtuele-printer-45358.html, 2015. (Cited on page 10.)

[101] Christoph Roser, Masaru Nakano, and Minoru Tanaka. Shifting bottleneck detection. In *Proceedings of the Winter Simulation Conference*, volume 2, pages 1079–1086 vol.2, Dec 2002. (Cited on page 94.)

[102] Bernard Roy. Graphes et ordonnancement. *Revue Française de Recherche Opérationnelle*, pages 323–333, 1962. (Cited on pages 77 and 94.)

[103] Robert Sedgewick and Kevin Wayne. *Algorithms, 4th Edition.* Addison-Wesley, Boston, Massachusetts, 2011. (Cited on pages 26 and 77.)

[104] Abdulrahman Seleim and Hoda ElMaraghy. Parametric analysis of mixed-model assembly lines using max-plus algebra. *CIRP Journal of Manufacturing Science and Technology*, 7(4):305 – 314, 2014. (Cited on page 95.)

[105] John Shewchuk and Colin Moodie. Definition and classification of manufacturing flexibility types and measures. *International Journal of Flexible Manufacturing Systems*, 10(4):325–349, 1998. (Cited on page 17.)

[106] Hamid Shojaei, Twan Basten, Marc Geilen, and Azadeh Davoodi. A fast and scalable multidimensional multiple-choice knapsack heuristic. *ACM Trans. Des. Autom. Electron. Syst.*, 18(4):51:1–51:32, October 2013. (Cited on pages 19, 31, 32, 34, 49, 53, 57, and 58.)

[107] Hamid Shojaei, Amir Ghamarian, Twan Basten, Marc Geilen, Sander Stuijk, and Rob Hoes. A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for CMP run-time management. In *DAC'09*, pages 917–922. ACM, 2009. (Cited on page 53.)

[108] Chantal Steimer, Jan Fischer, and Jan Aurich. Model-based design process for the early phases of manufacturing system planning using sysml. *Procedia CIRP*, 60:163 – 168, 2017. Complex Systems Engineering and Development Proceedings of the 27th CIRP Design Conference Cranfield University, UK 10th – 12th May 2017. (Cited on page 11.)

[109] Thomas Stützle and Holger Hoos. Analyzing the run-time behaviour of iterated local search for the TSP. In *III Metaheuristics Int. Conf.*, 1999. (Cited on pages 32, 49, 50, 59, and 60.)

[110] Krishnamurthy Subramani. Parametric scheduling — algorithms and complexity. In Burkhard Monien, Viktor Prasanna, and Sriram Vajapeyam, editors, *High Performance Computing — HiPC 2001*, pages 36–46, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. (Cited on page 121.)

[111] Lennart Swartjes, Pascal Etman, Asia van de Mortel-Fronczak, Koos Rooda, and Lou Somers. Simultaneous analysis and design based optimization for paper path and timing design of a high-volume printer. *Mechatronics*, 41:82 – 89, 2017. (Cited on pages 9, 10, 11, 72, 85, and 89.)

[112] Bram van der Sanden, João Bastos, Jeroen Voeten, Marc Geilen, Michel Reniers, Twan Basten, Johan Jacobs, and Ramon Schiffelers. Compositional specification of functionality and timing of manufacturing systems. In *2016 Forum on Specification and Design Languages, FDL*, Bremen, Germany, September 2016. (Cited on pages 72, 77, 85, and 86.)

[113] Bram van der Sanden, Joao Bastos, Jeroen Voeten, Marc Geilen, Michel Reniers, Twan Basten, Johan Jacobs, and Ramon Schiffelers. Compositional specification of functionality and timing of manufacturing systems. In *2016 Forum on Specification and Design Languages (FDL)*, pages 1–8, Sept 2016. (Cited on page 11.)

[114] Bram van der Sanden, Michel Reniers, Marc Geilen, Twan Basten, Johan Jacobs, Jeroen Voeten, and Ramon Schiffelers. Modular model-based supervisory controller design for wafer logistics in lithography machines. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 416–425, Sept 2015. (Cited on page 11.)

[115] Roel van der Tempel, Joost van Pinxten, Marc Geilen, and Umar Waqas. A heuristic for variable re-entrant scheduling problems. In *2018 Euromicro Conference on Digital System Design (DSD)*, 7 2018. 21st Euromicro Conference on Digital System Design (DSD 2018). (Cited on pages 21 and 174.)

[116] Roel van der Tempel, Joost van Pinxten, Marc Geilen, and Umar Waqas. A heuristic for variable re-entrant scheduling problems. Technical Report ESR-2005-04, Electronic Systems Group, Department of Electrical Engineering,

Eindhoven University of Technology, June 2018. (Cited on pages 43, 127, and 174.)

[117] Joost van Pinxten, Marc Geilen, and Twan Basten. Characterising parametric schedulers. [submitted]. (Cited on pages 13 and 173.)

[118] Joost van Pinxten, Marc Geilen, Twan Basten, Umar Waqas, and Lou Somers. Online heuristic for the multi-objective generalized traveling salesman problem. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*, pages 822–825. IEEE, March 2016. (Cited on pages 13, 47, 50, and 173.)

[119] Joost van Pinxten, Marc Geilen, Martijn Hendriks, and Twan Basten. Parametric critical path analysis for event networks with minimal and maximal time lags. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 2697 – 2708, October 2018. (Cited on pages 13, 71, 122, and 173.)

[120] Joost van Pinxten, Umar Waqas, Marc Geilen, Twan Basten, and Lou Somers. Online scheduling of 2-re-entrant flexible manufacturing systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):20, October 2017. (Cited on pages 12, 17, and 173.)

[121] Umar Waqas. *Scheduling and variation-aware design of self-re-entrant flowshops*. PhD thesis, Department of Electrical Engineering, 11 2017. (Cited on pages 10 and 127.)

[122] Umar Waqas, Marc Geilen, Jack Kandelaars, Lou Somers, Twan Basten, Sander Stuijk, Patrick Vestjens, and Henk Corporaal. A re-entrant flowshop heuristic for online scheduling of the paper path in a large scale printer. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, pages 573–578, 2015. (Cited on pages 10, 17, 18, 19, 22, 23, 27, 37, 41, 42, 43, 72, 77, and 115.)

[123] Umar Waqas, Marc Geilen, Sander Stuijk, Joost van Pinxten, Twan Basten, Lou Somers, and Henk Corporaal. A fast estimator of performance with respect to the design parameters of self re-entrant flowshops. In *2016 Euromicro Conference on Digital System Design (DSD)*, pages 215–221, Aug 2016. (Cited on pages 116 and 174.)

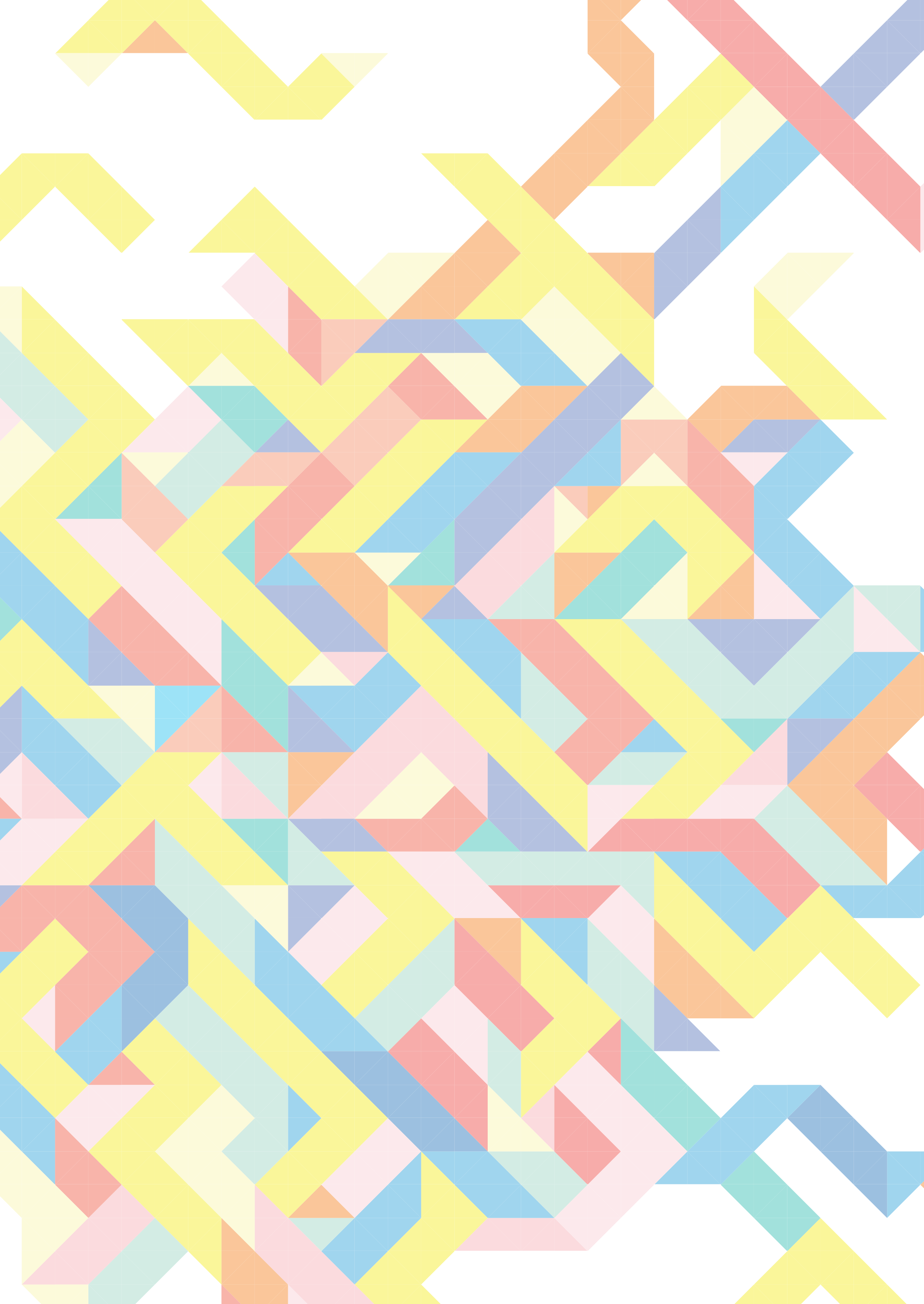
[124] Ian Wyatt and Daniel Hecker. Occupational changes during the 20th century. *Monthly Labor Review*, page 35, 2006. (Cited on page 3.)

[125] Michael Yukish. *Algorithms to Identify Pareto Points in Multi-dimensional Data Sets*. PhD thesis, The Pennsylvania State University, 2004. AAI3148694. (Cited on pages 34 and 84.)

[126] Eckart Zitzler, Dimo Brockhoff, and Lothar Thiele. The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration. In Shigeru Obayashi, Kalyanmoy Deb, Carlo Poloni, Tomoyuki Hiroyasu, and Tadahiko Murata, editors, *Evolutionary Multi-Criterion Optimization*, pages 862–876, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. (Cited on page 62.)

[127] Eckart Zitzler, Joshua Knowles, and Lothar Thiele. Quality assessment of pareto set approximations. In *Multiobjective Optimization*, pages 373–404. Springer, 2008. (Cited on page 62.)

[128] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm. Technical report, Swiss Federal Institute of Technology (ETH) Zurich, 2001. (Cited on pages 32, 35, 36, and 58.)



Samenvatting

Optimalisatie van productstromen in flexibele fabricagesystemen

Wafer scanners, gerobotiseerde assemblagelijnen, en industriële printers zijn voorbeelden van flexibele fabricagesystemen (FFS) die op maat gemaakte producten kunnen produceren. Wafer scanners stellen silicium wafers bloot aan UV-licht om computerchips te maken, en industriële printers dragen digitale afbeeldingen over op vellen papier. Elk product kan daarbij verschillende vereisten of karakteristieken hebben, wat kan leiden tot verschillende tijdsbeperkingen tussen de operaties. De productiviteit van een FFS is gelimiteerd door zowel cyber-beperkingen, zoals de rekentijd van een on-line schedulingalgoritme, alsmede fysieke beperkingen, zoals restricties op het transport van een product. Zowel de interactie tussen componenten in een FFS als de productiviteitseisen blijven toenemen. In dit proefschrift worden technieken uiteengezet voor het uitvoeren van online scheduling van FFS'en en voor het analyseren van de impact van systeemparameters op de prestaties van het totaalsysteem.

FFS'en kunnen vele vormen aannemen. Wij richten ons op schedulingalgoritmes voor FFS'en waarin producten tweemaal worden bewerkt door één van de machines. Een dergelijke geval vindt men in industriële printers, waarbij een vel (d.w.z., het product) aan twee kanten bewerkt moet worden, door tweemaal door een afbeeldingsoverdrachtstation te gaan. Het soort medium en de uitvoervolgorde van de vellen zijn bepaald door de klant, en mogen niet worden gewijzigd. Nadat een vel is bewerkt, is het mogelijk dat het afbeeldingsoverdrachtstation moet worden omgesteld voordat het volgende vel bewerkt kan worden. De omsteltijden hangen typisch af van de verschillen in lengte, dikte, afwerking, en andere karakteristieken van het medium. De karakteristieken van een bepaald product, en daarmee de omsteltijden, zijn pas enkele momenten voordat de FFS verwerkingsinstructies moet ontvangen beschikbaar. Daarom is het nodig dat een scheduler snel efficiënte verwerkingsinstructies kan berekenen voor iedere operatie. Het is daarbij de uitdaging om de stroom van ongeprinte vellen samen te voegen met de stroom van terugkerende bedrukte vellen, zonder dat er botsingen ontstaan of er omsteltijden worden geschonden, zodat de productiviteit van het systeem geoptimaliseerd wordt. De volgorde van de producten onder het afbeeldingsoverdrachtstation speelt een belangrijke rol in de productiviteit van industriële printers, omdat het de omsteltijden tussen de producten direct beïnvloedt. De prestatie wordt dus beïnvloed door de wisselwerking van gebruikersinvoer, de fysieke parameters van de systeemcomponenten, en de schedulingbeslissingen.

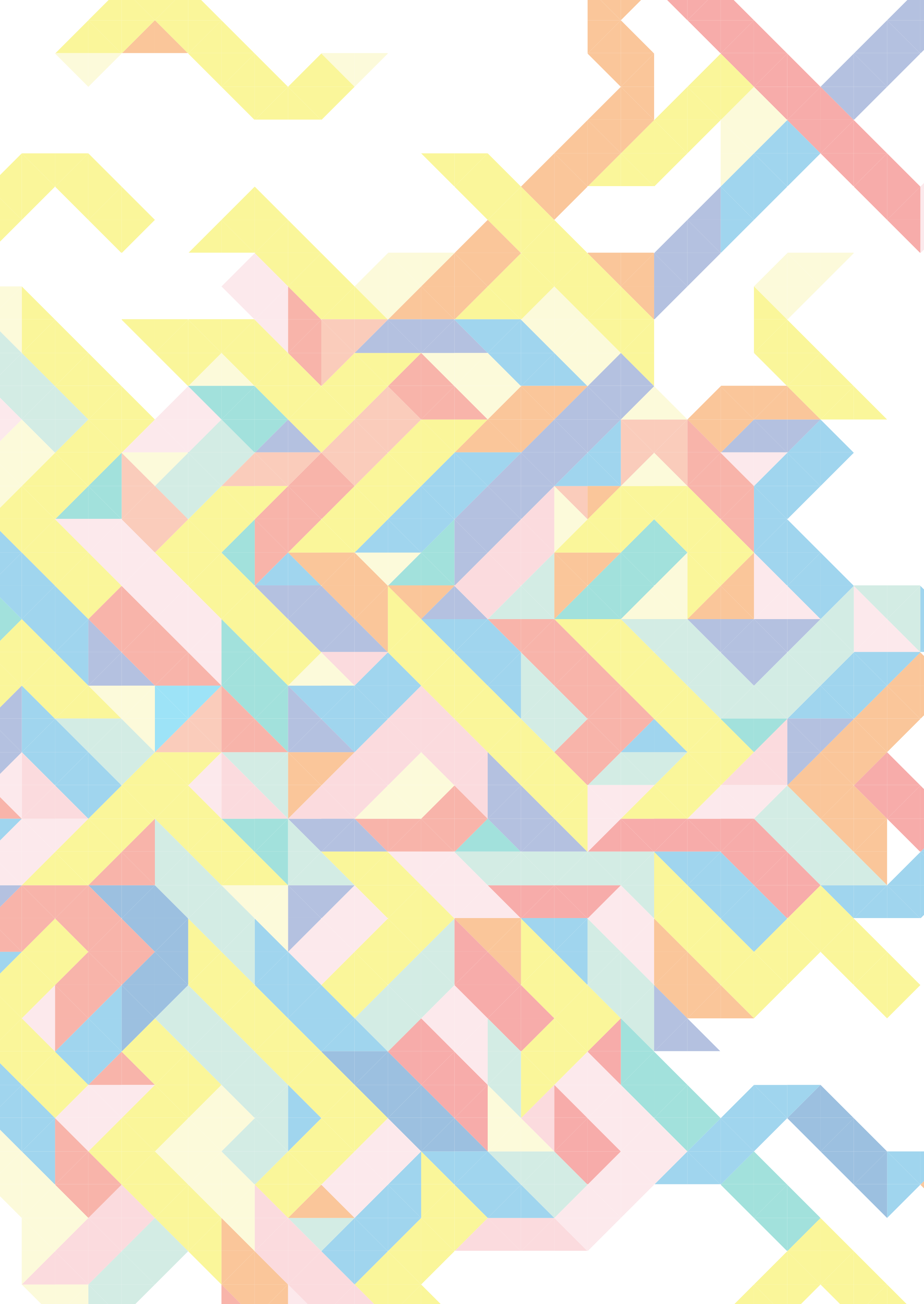
Ten eerste presenteren we een efficiënt on-line velscheduling algoritme voor industriële printers. Het opdrachtoptimalisatie probleem is gemodelleerd als een re-entrant flow shop met insteltijden, vervaltijden, en een vaste uitvoervolgorde. Wanneer een bepaalde volgorde van de producten (d.w.z., een printopdracht) is gegeven, dan bepaalt het velscheduling algoritme een uitvoerbare en efficiënte volgorde van operaties voor de re-entrant machine (d.w.z., het afbeeldingsoverdrachtstation). De tijdsinstructies voor elk van de operaties op de vellen kan daarna efficiënt gevonden worden. Ons heuristisch algoritme gebruikt meerdere doelen om tussentijdse opties in volgorde te verkennen zonder dat er keuzes teruggedraaid hoeven te worden. De heuristiek gebruikt verschillende eigenschappen van het schedulingprobleem, zoals de uitvoervolgorde van de vellen, en beperkingen op het aantal producten dat op enig moment gedeeltelijk bewerkt is. Voor het type FFS dat we beschouwen ligt de uitvoervolgorde van producten binnen een opdracht vast, maar de opdrachten moeten nog wel worden geordend.

De tweede bijdrage is een scheduling heuristiek dat de volgorde van de opdrachten bepaalt, en gelijktijdig de systeeminstelling voor elke opdracht selecteert, zodat Pareto-optimale productiviteit- en kwaliteitafwegingen worden gevonden. Het optimaliseren van de invoervolgorde van de opdrachten is met name relevant voor FFS'en waarvoor omsteltijden nodig zijn tussen verschillende producten, zoals industriële printers. Het probleem is gemodelleerd als een gegeneraliseerd handelsreizigersprobleem met meerdere doelen en het orderningsalgoritme is een online heuristiek met meerdere doelen. Het algoritme levert goede resultaten in minder rekentijd dan de best bekende aanpakken.

Ten derde presenteren we een parametrisch-kritisch-pad-analyse. Gedurende de onderwerpfase moet een ontwerper een fysiek ontwerp creëren die aan een bepaalde topologie voldoet, d.w.z., een set machines en de relaties daartussen. Een dergelijk ontwerp wordt beïnvloed door eisen uit vele domeinen. Zulke beperkingen bestaan bijvoorbeeld uit minimale en maximale opwarmtijden, minimale segmentlengte(s), of maximale snelheden. De ontwerper moet een realisatie kiezen die gelijktijdig zowel het gedrag als de kosten van het systeem optimaliseert. Ontwerpparameters zoals segmentlengte of transportsnelheden hebben directe invloed op het totaalgedrag van het systeem, doordat bijvoorbeeld het verwerken van een product sneller of langzamer wordt. Ons parametrisch-kritisch-pad-analyse identificeert relaties tussen fysieke parameters en de productiviteit van een FFS. De schedulinggrafen die worden gebruikt in de velschedulingheuristiek kunnen worden geannoteerd met affiene uitdrukkingen die tijdsbeperkingen relateren aan ontwerpparameters. De analyse identificeert knelpunten in de productiviteit van een FFS in termen van affiene uitdrukkingen voor verschillende parametercombinaties. Deze analyse stelt ontwerpers in staat om te identificeren hoe de productiviteit afhangt van de combinatie van parameters.

Als vierde bijdrage presenteren we een parametrische schedulerkarakterisatie algoritme dat de impact identificieert die parameters hebben op de scheduling. Veranderingen in de parameters kunnen ertoe leiden dat er andere schedulingkeuzes worden geselecteerd, waardoor andere schedules ontstaan. We laten zien dat het voor sommige klassen schedulers mogelijk is om informatie te geven over welke schedulingkeuzes zijn gekozen voor verschillende parameter combinaties. Deze informatie kan worden verkregen door de scheduling symbolisch uit te voeren. We kunnen daarna de parametrisch-kritische-pad-analyse toepassen op elk van de verschillende schedules. De totale systeemproductiviteit kan dan worden onderzocht in termen van deze parameters, waarbij ook de schedulingkeuzes in acht worden genomen.

Dit proefschrift verschaft ontwerpers en ingenieurs de hulpmiddelen om het systeemgedrag van FFS'en te analyseren en optimaliseren. Afwegingen tussen productiviteit en kwaliteit kunnen on-line worden gemaakt door de opdrachten in een effectieve volgorde te zetten. De on-line velscheduler verbetert de productiviteit door een effectievere verkenning van schedulingopties voor 2-re-entrant flow shops. De opdrachtoptimalisatie, de parametrisch-kritisch-pad analyse, en de parametrische schedulerkarakterisatie zijn toepasbaar op systemen waarbij gebeurtenissen met minimale en maximale tijdsverschillen worden beschreven.



Acknowledgements

The Eindhoven University of Technology has hosted me for the better part of thirteen years, first as a student, and later as a PhD researcher. I am indebted to the Dutch government and the Eindhoven University of Technology for providing me with the education required to finish this dissertation. I thank the society for creating the inter-subjective reality[1] that enabled me to pursue this career.

It takes a village to raise a child.

- African Proverb

In the same spirit, I feel that the following should become a proverb too:

It takes a society to write a dissertation.

It is interesting that these inter-subjective and subjective relations allow one to perform research to find objective, testable ideas. During my time as a Bachelor, Master and PhD student at Eindhoven University of Technology, many personal relationships have influenced me, making me the person I am today. I am well-aware that this piece of text is the piece of the dissertation that will likely be read most often. That being said, I want to highlight several personal relationships that have supported me throughout my studies and PhD research.

First and foremost, I want to extend my gratitude to professor Twan Basten for being my promotor and Marc Geilen for being my co-promotor. I recognize that you are among the few people in the world that have the intellect, experience and interpersonal skills to educate scientific researchers without alienating yourself on the personal level. Both of you managed to clearly separate the objective assessment of the research from the quirks of the process, while also building a personal relationship. I still feel lucky to be given the opportunity to work for and with you for the past five years. I am grateful to the promotion committee members prof. Sebastian Engell, prof. Frits Vaandrager, prof. Jeroen Voeten, and dr. Lou Somers for their time and effort to read, assess and provide valuable feedback on this dissertation.

I am also grateful to Umar Waqas, for extracting the academic version of the scheduling problem at Océ, and for collaborating to improve on his scheduling algorithm. His efforts have paved the way for me to continue with research on flow

---

[1] See *Sapiens* by Yuval Noah Harari.

shops in the Electronic Systems group. I have always enjoyed your inquisitive nature in all things. We have discussed many different topics while car-pooling from Eindhoven to Océ in Venlo. It was enjoyable to car-pool with you, Barath, Nick, Savvas, Roel, Mauricio (AKA Giovanni ;-)), Waheed, Ali, and Marijn (AKA Hank); we've used the travelling time for many philosophical, political, cultural, environmental, and technological discussions. I thank my fellow PhD's in the project, Alexis and Amir, for the nice discussions, talks, presentations. Alexis, I am still amazed by how fast you picked up the Dutch language!

The project with Océ would not have been possible without the continued efforts of Lou, Twan, and ESI to form projects around the scientific questions that tie into the development of Océ. I am grateful that NWO-TTW decided to finance the research project. I thank Lou for handling the Intellectual Property issues that inherently arise from joint research by industry and the university; the university requires us to publish novel ideas and applications, while the company prefers to keep novel ideas internal, and patent them. Walking this fine line has not made the project any easier.

The (bi-)weekly discussions at Océ with Patrick and Jack were invaluable to the success of my project. You both made a great effort to provide us with enough information without immediately drowning us in the nitty-gritty details of the real-world implementations. The discussions with Jack, Patrick, Peter, Eugen, Hristina, Oana, Amar, Amol, and many others at Océ were very insightful and enjoyable. I thank Michel and Rob for providing me with the use case for the third chapter in this thesis. In the collaboration with ESI, I have enjoyed the discussions and conversations with Jacques, Martijn, Peter, Tjerk, Roelof, Bas, Jozef, Arjan, Carmen, Jeroen, and many others.

My fellow colleagues[2] in the projects of the RCPS programme and the CPS subgroup of the ES-group, have greatly contributed to my enjoyment of my PhD time. Umar, Bram, Robinson, João, Amir, Hadi, Ruben, I feel that we've created a group that will keep in contact for the rest of our lives. The shared experience around the xCPS machine and the CPS meetings will never be forgotten!

Another unforgettable life experience originated from the Friday-afternoon conversations in Potentiaal (previously called E-hoog, now Luna) with Marcel, Raymond, Luc (x2), Maurice, Martijn, Sander, Sven, Marc, Mark, Yonghui, Juan (AKA “the Interruptor") and many others. The wide range of topics at the coffee table never ceased to amaze me. We had many laughs over the weirdest comments. It's a pity that we haven't managed to keep this show going in the new Flux building.

I thank the students who allowed me to try to instil some knowledge and skills in them. Ali, Sethuraman (AKA Sjors), Vishnu, and Roel were the ‘slachtoffers’ for the Océ projects. The project of Manuel, collaborating with Joost, Kostas, and Gaston, allowed us to explore some of our techniques to use cases outside of Océ. Joost, thanks for making these connections during the user committee meetings! Many students (Konrad, Arda, David, Max, Sjoerd, Luc, Alexander, Danny, Niek,

---

[2] ‘slachtoffers’ as my desk neighbour Robinson would often put it

Remy, Geert, Glenn, among many others) have contributed to the xCPS platform, its software, hardware, and the Blender visualization. I've enjoyed the lab sessions, exercises, assignments and the many, many reports for the Multiprocessors course. It was hard work, but also fun, to assist in the course, and take all the oral examinations together with Twan, Marc, Firew, Manil, Amir and Ruben.

One of the perks of the PhD is that you get to travel to all kinds of places, both domestic and international. The ICT-Open and ASCI trips always turned out nicely, with good food and nice people to talk to (and sometimes not so great, but funny, music, right Josh?). The trip to South Korea was memorable amongst other things due to the visit to the Trickeye Museum in Seoul, with Philip, Marco, and my wife Zlatka. The trip to India for Shreya and Ernest's wedding was a great excuse for additional travelling, we had a great time with Twan, Isabelle, Anne and Marc.

I thank all my ES colleagues for the nice lunch and cake breaks, the secretaries (Marja, Margot, Rian, Feyza) for providing the nice atmosphere in our group, and organizing the yearly ES days. In the United Kingdom, I was generously hosted by Louis in the YCCSA community at York, and had a great time being a part of the jolly bunch around the University of York Computer Science geeks; Sam, Simon, Rob, Richard, James, Ipek, Fiona, Chris, Dimitris, Kostas, Thanos, Mike, Imran, Adolfo, Antonio, Jason, José and not to forget Miss Sarah Christmas! I hope to find more excuses to travel to York/UK and visit you all!

Outside the academic life, the distractions from the research were plentiful, and provided in many ways; the international pub quiz on Thursdays with Liquid Knowledge and Solid Ignorance (Jeroen, Sjef, Lex, Silke, Gerald, Alan, Calin, Medhat, Amar, Alan, Zlatka, Niels, Marco, Eric, and Oana). The bi-weekly D&D sessions GM'd by Bas were a nice way to escape reality too; Melchior, Wouter, Boudewijn and Maarten, thanks for continuing all these years! Not to forget the many board games we managed to play with João, Martijn, Vladimir, Jovana, Nikola, Orlando, Nicolas, Marc, Sven, and many others! The (occasional) squash with the Quatsh trainings, and with Cris, Sofia, Juan, Eric, Marco, and Farooq, and the bouldering with Bram and Rose, provided some physical exercise combined with many funny conversations. We enjoyed the theater and media-related activities due to the overwhelming enthusiasm and energy of the van Esch family; Anita, Toine, Rosanne and Joris.

The ex-OOTIc family (Eric, Oana, Marco, Kiki, Mite, Aleksandra, Milosh, Bogdan, Andreaa, Cris, Andreaa, Francisco, Alejandra, Ivana, Max, Martin, Estella, to name a few) is a special bunch of people, with diverse international backgrounds, but with a passion for software, good food, good company, and travelling. We have enjoyed travelling to all kinds of places for weddings and other occasions. I hope that we will have many more of these joint travels or activities! At least the number of ex-OOTIc babies is growing fast, giving us ample excuses and opportunities to interact!

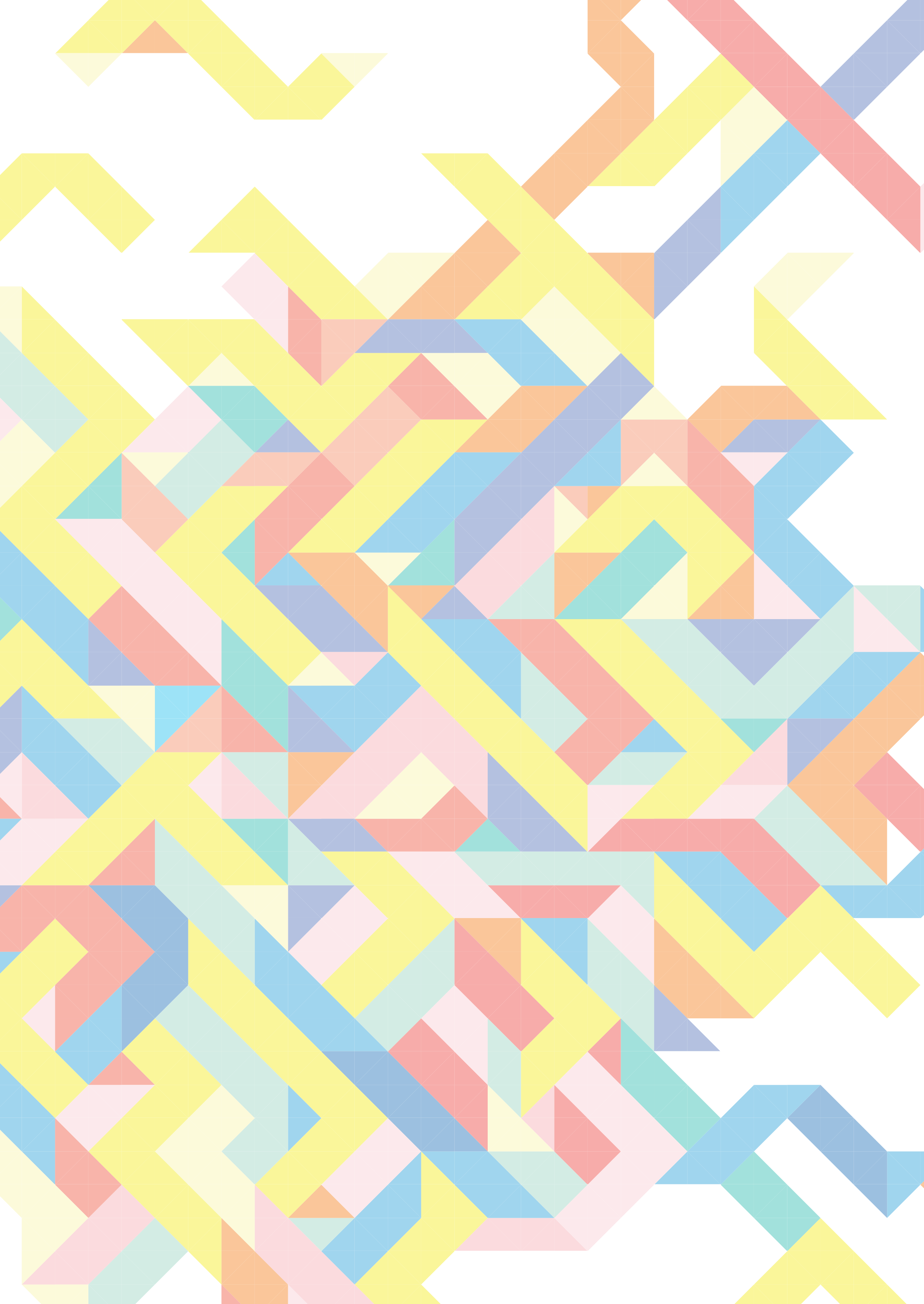
Last, but not least, I thank my parents, siblings, family, cats[3] and friends, in

---

[3] Timmy and Fluffy, the little purrito's!

particular Vic & Joris, for supporting me through my studies and the PhD. Without your support I am sure I would have dropped out somewhere along the line. My lovely wife Zlatka and our beautiful daughter Kaya provide me with plenty of opportunities to live life to the fullest, and I enjoy every waking moment with them. Additional thanks to Zlatka for putting up with my whimsies, whenever a deadline was approaching. I hope that we get to spend a lot more time together.

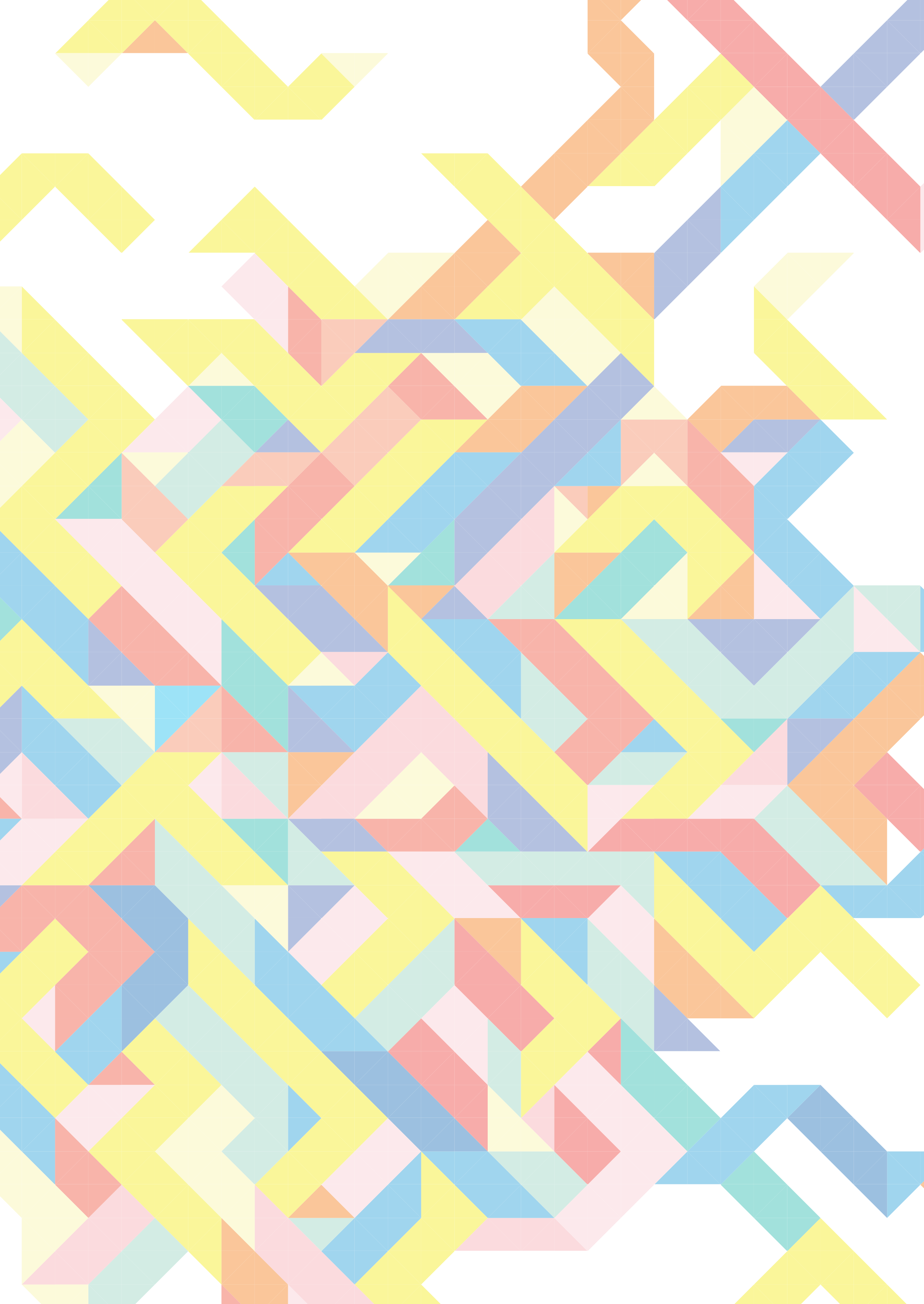
Everyone in my environment (my society) has not only enabled me to write this dissertation, but has also helped me shape to be the person who I am today.



Curriculum Vitae

Joost van Pinxten was born on December 13, 1986 in Sint-Michielsgestel, The Netherlands. In 2005 he graduated from Gymnasium Beekvliet, Sint-Michielsgestel. He studied at the Eindhoven University of Technology, where he obtained the B.Sc. and the M.Sc. degrees in Electrical Engineering in 2011 and 2013 respectively. As part of his Master's program, he did an internship at AuguSoft, Eindhoven. His Master's graduation project on Model Transformations for Design Space Exploration was performed at the Embedded Systems Institute (ESI), in cooperation with Océ Technologies. In 2013, he has visited York, United Kingdom, for an invited summer internship to extend EuGENiA Live, a flexible meta-modelling tool, with simulation aspects.

In 2014 Joost joined the Electronic Systems group of Eindhoven University of Technology as a PhD candidate in the *Integrated scheduling and control for cyber-physical systems* project under the NWO-TTW Perspectief programme *Robust Cyber-Pysical Systems*. The project focused on analysing, designing, scheduling and controlling the paper path of industrial duplex printers. The results of his research have led, among others, to several peer-reviewed publications and the contents of this dissertation. In 2019 he will start his new job in the Research and Development department of Océ Technologies.



List of Publications

First author

Journal papers

- Joost van Pinxten, Marc Geilen, Martijn Hendriks, and Twan Basten. Parametric critical path analysis for event networks with minimal and maximal time lags. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 2697 – 2708, October 2018

- Joost van Pinxten, Umar Waqas, Marc Geilen, Twan Basten, and Lou Somers. Online scheduling of 2-re-entrant flexible manufacturing systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):20, October 2017

Conference papers

- Joost van Pinxten, Marc Geilen, and Twan Basten. Characterising parametric schedulers. [submitted]

- Joost van Pinxten, Marc Geilen, Twan Basten, Umar Waqas, and Lou Somers. Online heuristic for the multi-objective generalized traveling salesman problem. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*, pages 822–825. IEEE, March 2016

Co-author

Journal papers

- Shreya Adyanthaya, Hadi Alizadeh Ara, João Bastos, Amir Behrouzian, Róbinson Medina Sánchez, Joost van Pinxten, Bram van der Sanden, Umar Waqas, Twan Basten, Henk Corporaal, Raymond Frijns, Marc Geilen, Dip Goswami, Martijn Hendriks, Sander Stuijk, Michel Reniers, and Jeroen Voeten. xCPS: A tool to eXplore Cyber Physical Systems. *ACM Special Interest Group on Embedded Systems (SIGBED)*, 2017

Conference papers

- Roel van der Tempel, Joost van Pinxten, Marc Geilen, and Umar Waqas. A heuristic for variable re-entrant scheduling problems. In *2018 Euromicro Conference on Digital System Design (DSD)*, 7 2018. 21st Euromicro Conference on Digital System Design (DSD 2018)

- Alexis Linard, Rick Smetsers, Frits Vaandrager, Umar Waqas, Joost van Pinxten, and Sicco Verwer. Learning pairwise disjoint simple languages from positive examples. In *LearnAut 2017*, volume abs/1706.01663, 2017

- Umar Waqas, Marc Geilen, Sander Stuijk, Joost van Pinxten, Twan Basten, Lou Somers, and Henk Corporaal. A fast estimator of performance with respect to the design parameters of self re-entrant flowshops. In *2016 Euromicro Conference on Digital System Design (DSD)*, pages 215–221, Aug 2016

- Shreya Adyanthaya, Hadi Alizadeh Ara, João Bastos, Amir Behrouzian, Róbinson Medina Sánchez, Joost van Pinxten, Bram van der Sanden, Umar Waqas, Twan Basten, Henk Corporaal, Raymond Frijns, Marc Geilen, Dip Goswami, Sander Stuijk, Michel Reniers, and Jeroen Voeten. xCPS: A tool to eXplore Cyber Physical Systems. In *Proceedings of 2015 Workshop on Embedded and Cyber-Physical Systems Education*, pages 3:1–3:8, 2015

Technical Reports (Non-Refereed)

- Roel van der Tempel, Joost van Pinxten, Marc Geilen, and Umar Waqas. A heuristic for variable re-entrant scheduling problems. Technical Report ESR-2005-04, Electronic Systems Group, Department of Electrical Engineering, Eindhoven University of Technology, June 2018