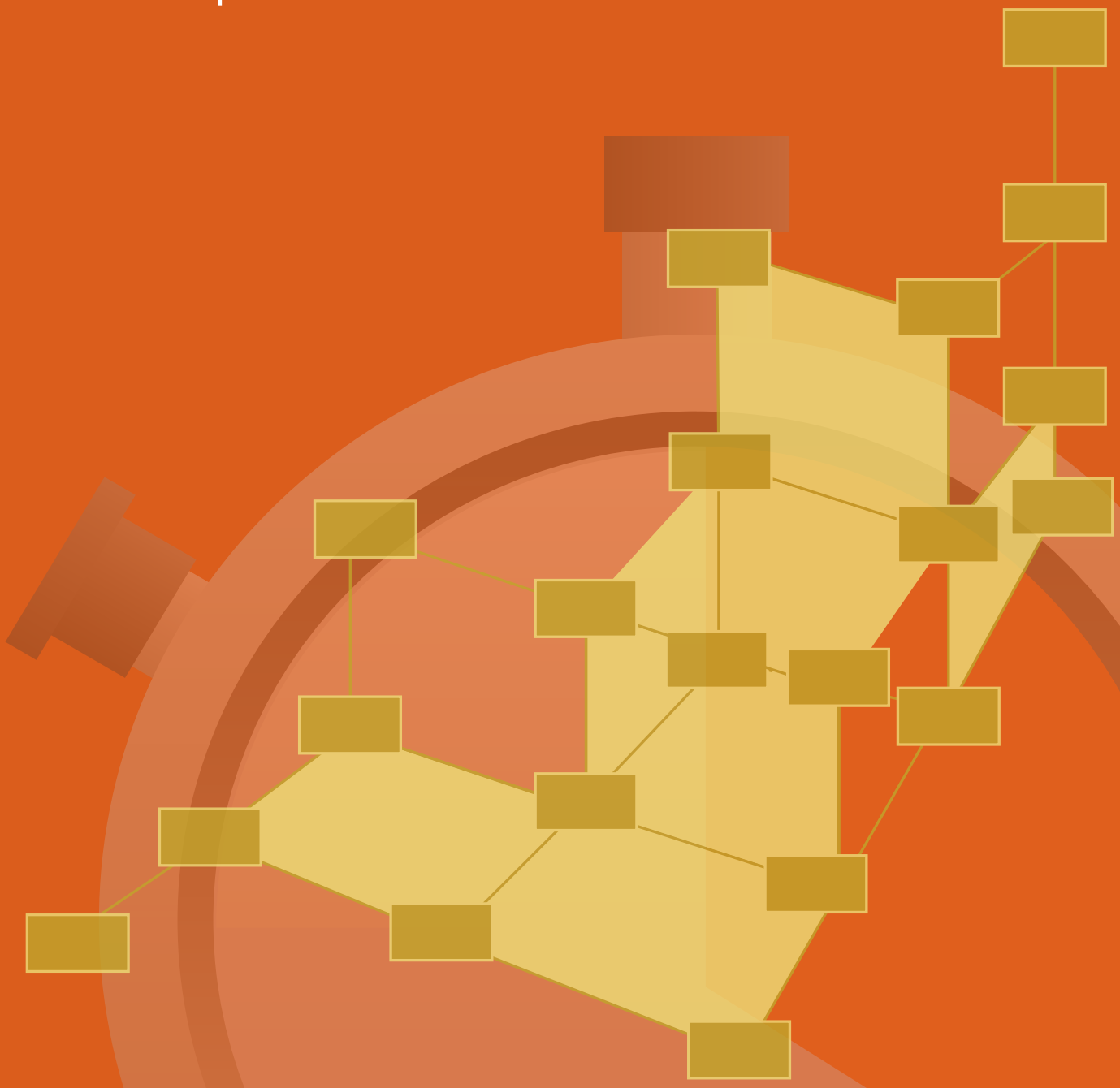
Scheduling and variation-aware design of self-re-entrant flowshops

Scheduling and Variation-aware Design of Self-re-entrant Flowshops

Umar Waqas



Scheduling and variation-aware design of self-re-entrant flowshops

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 23 november 2017 om 16:00 uur

door

Umar Waqas

geboren te Jhelum, Punjab, Pakistan.

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

voorzitter:	prof.dr.ir. J.H. Blom
1e promotor:	prof.dr. H. Corporaal
1e co-supervisor:	dr.ir. M.C.W. Geilen
2e co-supervisor:	dr.ir. S. Stuijk
leden:	prof.dr.sc. S. Chakraborty (TU München)
	prof.dr. J.J.M. Hooman (Radboud University - TNO-ESI)
	prof.dr.ir. A.A. Basten (TU Eindhoven - TNO-ESI)
	dr. L.J.A.M. Somers (TU Eindhoven - Océ Technologies)

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

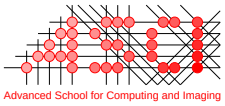
Scheduling and Variation-aware Design of Self-re-entrant Flowshops

Umar Waqas

Doctoral committee:

chairman:	prof.dr.ir. J.H. Blom
1^{e} promotor:	prof.dr. H. Corporaal
1^{e} co-supervisor:	dr.ir. M.C.W. Geilen
2^{e} co-supervisor:	dr.ir. S. Stuijk
members:	prof.dr.sc. S. Chakraborty (TU München)
	prof.dr. J.J.M. Hooman (Radboud University - TNO-ESI)
	prof.dr.ir. A.A. Basten (TU Eindhoven - TNO-ESI)
	dr. L.J.A.M. Somers (TU Eindhoven - Océ Technologies)

This work was supported in part by the Dutch Technology Foundation STW, project NEST 10346.



This work was carried out in the ASCI graduate school.
ASCI dissertation series number is 380.

A catalogue record is available from the Eindhoven University of Technology Library
Title: Scheduling and variation-aware design of self-re-entrant flowshops
Author: Umar Waqas
Eindhoven University of Technology, 2017.
ISBN: 978-90-386-4372-4
NUR: 993

Cover design: Yvonne Meeuwsen (www.yvonnemeeuwsen.nl).
Printed by: Gildeprint, The Netherlands (www.gildeprint.nl).

This thesis is dedicated to the memories of my father.
Without the support of my parents, this accomplishment was impossible.

یہ کتاب میرے والد محطرم کی یاد کو مختص کی جاتی ہے۔
یہ نتیجہ میرے والدین کی محنت اور دعاوں کے بغیرممکن نہ تھا۔

Scheduling and variation-aware design of self-re-entrant flowshops

Summary

Self-re-entrant systems consist of several machines that repeatedly process products. For example, in an industrial printer, a sheet is loaded by the first machine, gets printed on the front and the back side by the second machine, and unloaded by the third machine. The second machine is self-re-entrant i.e. a sheet comes back to the same machine to get printed on the back side. In a self-re-entrant system, based on the timing requirements of the products, a scheduling algorithm determines the times at which certain operations are performed on a product. Changing the structure of self-re-entrant systems changes the timing requirements that influence their performance. The performance of a self-re-entrant system may be improved by using a better scheduling algorithm as well as designing the system such that the timing requirements favour for performance. This thesis proposes new methods that by scheduling and better design improve the machine utilization in self-re-entrant systems.

A self-re-entrant flowshop is a model for the timing requirements of self-re-entrant systems. In this work, the model is used to determine schedules for self-re-entrant systems and to assess the impact of structural design choices over the performance of a system. As an example, self-re-entrant flowshops have been used to schedule sheets in an industrial printer. The model facilitates determination of possible scheduling alternatives, whether the alternatives are feasible or not and what is the effect of each alternative over the performance of a system. A designer of a self-re-entrant system can assess, using the flowshop model, whether the impact over performance is positive or negative.

The problem of scheduling self-re-entrant flowshops is NP-Hard. This thesis shows that scheduling of self-re-entrant flowshops remains to be NP-Hard even if the order of the products is pre-determined. The number of possible scheduling alternatives grows exponentially with respect to the number of products that need to be scheduled. The growth is due to different orders in which the operations can be performed over different jobs. Finding an optimal schedule for such a self-re-entrant flowshop is prohibitively compute intensive even for medium sized problem instances. The systems where such a schedule is required at runtime demand for an alternative approach to find good quality schedules, if not optimal ones. For example, in an online setup, where a schedule request arrives at runtime, a compute intensive algorithm will most likely become a bottleneck because processing of products cannot start before a schedule is computed.

A heuristic approach to schedule self-re-entrant flowshops is proposed in this thesis. The heuristic aims to improve the resource utilization by avoiding scheduling decisions which either have low performance or make a schedule infeasible. Decisions that have good performance are chosen based on the assessment of their impact considering performance and its feasibility. A complete schedule is created by the heuristic in several iterations. At the end of the iterations, the heuristic finds a schedule that determines when an operation is performed on a product. A

specialized version of the heuristic is also described in this thesis which is tailored for scheduling of sheets in an industrial printer at runtime. The case is a restricted version of the general scheduling problem where the system consists of a self-re-entrant machine that processes the products only twice. For this case, faster exploration of scheduling alternatives is possible which reduces execution time. Thus, the specialized heuristic provides a simplified method to solve a runtime challenge of finding a schedule for an industrial printer.

During early design phases, a designer of a self-re-entrant system encounters the question of dimensioning the system. Many parameters with a broad range of possible values require exploration. Modelling each design instance as a self-re-entrant flowshop and scheduling it to determine its performance is possible but time consuming. A fast method is required to assess the impact of design choices over the performance of the system. Due to the nature of products, it is highly likely that industrial self-re-entrant systems produce the products in a repeating pattern. For example, a book has a cover page followed by several body pages. This pattern of cover and body pages repeats one hundred times when one hundred copies of the book are printed. This thesis proposes a performance estimator that utilizes the knowledge of patterns to predict the scheduling behaviour and thus the performance. In this work, the estimator is used to quickly quantify the impact of design parameters over the system performance of an industrial printer and a research platform.

Slight variations in the design decisions lead to variations in performance of self-re-entrant systems. The final design, due to engineering challenges, might deviate from the desired design parameters. Therefore, it is of interest to measure how variation sensitive a design choice is considering the deviations that might occur. On the other hand, many times, high performing designs also have more variation in performance across different types of jobs. This variation occurs because the design of a self-re-entrant system favours the timing requirements of a specific type of job. This thesis describes a new method to measure variation in performance due to changes in design parameters. The measure utilizes the information from the schedules generated for the self-re-entrant flowshop instances corresponding to specific design choices. Interesting designs are the ones which combine high performance with low variation in performance due to deviations in design parameters.

The combination of the self-re-entrant flowshops as a modelling technique with the heuristic scheduling algorithm, the performance estimation and the variation-aware design assists during the design and operation of self-re-entrant systems. Together they aid a designer of a self-re-entrant system in making decisions which either increase the performance of the system, or find cheaper alternatives while maintaining the system performance.

Table of contents

"Wisdom begins in wonder" - Socrates

1 Introduction

Pre-industrial manufacturing was solely based on skilled personnel producing goods. Usually, a small group of people specialized to produce a certain type of products. The knowledge of the skills was passed through generations and occasionally there was apprenticeship. Productivity of the manufacturing process was limited. One of the key reasons of limited productivity was the dependence of human performance over several factors like health, age and motivation. Other reasons were availability of skilled work force and scalability of the manufacturing process or influence of varying aspects such as weather and seasons. Achieving higher performance remains to be an active area of research, though the design and operational challenges of manufacturing have significantly changed.

This thesis proposes solutions to a set of design and operational challenges for a class of modern manufacturing systems. Section 1.1 introduces areas where modern manufacturing is used. These areas have several aspects in common which make them closely related to each other. One such aspect is the order in which products are processed. Section 1.2 introduces a subclass of manufacturing systems, called *self-re-entrant flowshops*, in which a product passes through the system in a specific order. Section 1.3 describes the challenges in design and operation of self-re-entrant flowshops. The contributions of this thesis are listed in Section 1.4. Section 1.5 describes how this work is organized in different chapters.

1.1 Modern manufacturing systems

Modern manufacturing flourishes over the use of machines to increase productivity. Mostly, several machines work in parallel to process products. The increase in performance is mainly due to faster and longer availability of machines. The production could now continue for several days. Periodic maintenance and breakdowns can be compensated with additional machines that are ready to take over the tasks of others. Due to the enhanced scalability additional resources can be added when an existing manufacturing setup reaches its limits. No wonder today there is a surplus of production filling up the warehouses.

Depending on the application area, a modern manufacturing system consists of several resources. Figure 1.1 shows examples of resources like belts, robotic arms, personnel etc. The

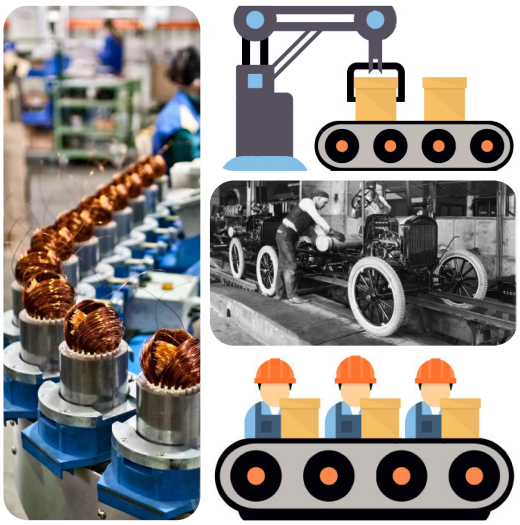


Figure 1.1: Typical resources in manufacturing systems (images from [1, 2]).

performance of a system depends on the performance of the resources available to the system. In order to scale a system it is required to increase the number of resources in demand. Once abundance of resources is achieved, finding out how to operate them efficiently increases the performance of the system. On the other hand, under-utilized resources usually add up to the cost of the system. Thus, achieving higher performance along with higher utilization is also of concern.

Humans are still part of modern manufacturing systems. Mostly, they are in the roles which require human intervention for decision making e.g. quality control, planning, as well as roles which cannot be fully automated. Many of such roles do not require human presence once the critical decisions have been made. For most of the repetitive tasks, machines have been designed to take over the load from people.

The products in a manufacturing system are transported between the machines using conveyor belts, robots or even human operators. These *transport mechanisms* have operational constraints. For example, human operators have constrained availability. Moving components of a robot require collision free operational space. Improving performance of a system with such constraints requires optimising the system considering the timing constraints originating from the operational constraints of the transport mechanisms. Timing constraints further arise from the availability and operational requirements of the resources available in a system. Any violation of these constraints may lead to a catastrophic event or might make the system inoperative.

The transport and processing of products are subject to deviations. The placement of products on a conveyor belt, even when done by robotic arms, has slight fluctuations between the desired position and where it was placed. Similarly, the etching of wafers in a lithographic machine requires the reticle to be exposed with an accuracy in the nano meter scale. Achieving the required accuracy or even designing a manufacturing system with corrective components is possible, but usually results in a higher cost. Such deviations, when present in a system, lead to variations in system performance. Sometimes, these variations can lead to a very underutilized system. For example, incorrectly placed products over a conveyor belt might require human intervention for which the complete system must first be stopped. Secondly, as the placement of a product might have violated certain constraints, reprocessing of certain products might be required leading to

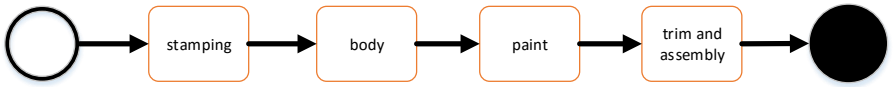


Figure 1.2: Different stages in a car manufacturing system [4].

performance losses.

Scheduling is the process of allocating limited resources to tasks over time while optimizing for one or more objectives [3]. The tasks in a manufacturing system are scheduled to use a set of resources for a certain amount of time. Different scheduling alternatives are compared to each other with respect to the scheduling objectives. For example, *makespan* minimization aims to finish the required set of tasks as soon as possible. *Latency* minimization aims to reduce the amount of time a product stays in the manufacturing system. Depending on the objective, different schedules might be considered for the same manufacturing system. The scheduling techniques proposed in this thesis aim to increase the performance of a system by minimizing the makespan when allocating resources while adhering to the timing constraints.

A modern *scheduler* computes schedules with the use of computers. The timing constraints, the optimization objectives and the details of the products are input to a scheduler. Given this information, a scheduler determines which tasks should be performed at what time, considering the availability of resources. The generation of a schedule must be completed before the manufacturing system requires it to start processing the products. Otherwise the system has to wait, which leads to performance losses. The time required to compute a schedule depends on the approach used. Heuristic approaches are usually used to find schedules quicker than the optimal approaches for the same problem. The schedulers proposed in this thesis optimize for makespan and use a heuristic approach to find schedules.

The products in a manufacturing system may follow a fixed route through machines. The term *flowshop* originates from such a route where products seem to flow through machines. The information about the flow is used by a scheduler when the order of products to be processed on a machine is determined. The scheduler might choose to process the products in different orders at different stages to achieve higher system utilization. Such freedom is still available to a scheduler. Usage of schedulers, that find the best choices out of possible ones, is key to increase the performance of modern manufacturing systems.

1.2 Self-re-entrant flowshops

A schematic of *stages*, each consisting of several resources, in a car manufacturing system is shown in Figure 1.2. A car starts at the *stamping* stage where the metal sheets for the body are power pressed to form the desired shape. Once the parts of the body are ready, they are put together in the *body* stage. In the *paint* stage, depending on the type of a part, different coating layers of paint are applied. *Trim* refers to the process of removal of the obstacles to the *assembly* procedure. For example, the doors of a car hinder the process of assembly of the car dashboard. Removing the doors makes the assembly easier and therefore trimming is performed before assembly.

All cars pass through the stages in Figure 1.2 in the same order as indicated by the arrows. The paint process is applied multiple times to a car. Coating layers are applied to a body part one by one. Similarly, in semi-conductor manufacturing, different layers of transistors are *etched* one by one. A wafer returns to a lithography machine several times. Machines to which a product returns one or more times are called *self-re-entrant* machines. Flowshops that consist of at least one self-re-entrant machine are called *self-re-entrant flowshops*. Self-re-entrancy is usually due to

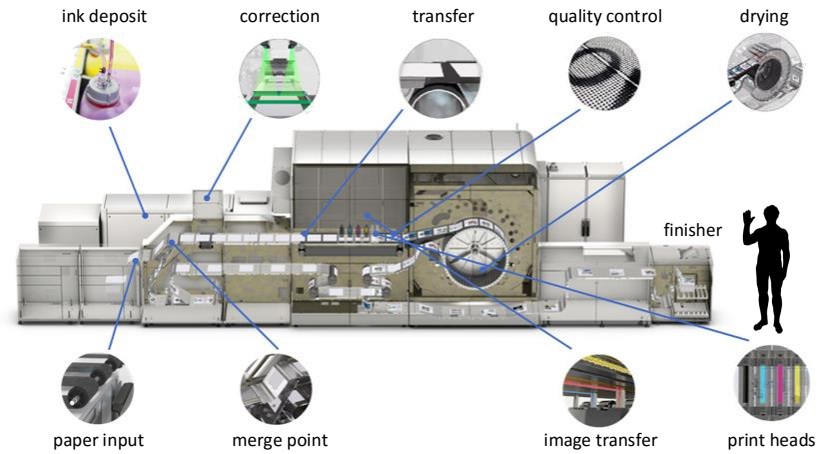


Figure 1.3: An example of an LSP: Canon VarioPrint i300 [8, 9].

a physical reason, for example, in a lithography machine, having a separate machine for each layer of the wafer is infeasible due to physical constraints of etching. Furthermore, using the same resource for a process increases its utilization and can reduce system costs.

Assembly lines [2], LED manufacturing [5], wafer sorting [6], TFT-LCD manufacturing [7] are industrial examples of re-entrant flowshops. In self-re-entrant flowshops, the products either get reprocessed at the current machine or they are passed on to the next machine. Products are never passed to a previous machine in the order of the flow. This thesis describes methods generally applicable to a class of self-re-entrant systems and specifically applied to industrial printing. Therefore, industrial printing will be repeatedly used to exemplify the concepts applicable to self-re-entrant flowshops.

An *Large Scale Printer* (LSP) transfers digital images from an electronic document to sheets of paper. *Canon's VarioPrint i300* is an example of an LSP, as shown in Figure 1.3. It has a capacity to produce 150 pages per minute leading to thousands of pages per day. This large capacity distinguishes an LSP from conventional desktop printers. Such a high performance comes from the complex blend of engineering in mechatronic, electronic and computational domains. To understand the essentials of the high performance, the details on how the products, which are sheets, are processed and their flow in the LSP needs to be understood.

The resources in VarioPrint i300 are shown in Figure 1.3. A sheet starts from the *paper input* unit and moves towards the *image transfer* station through the *merge point*. The paper input separates a sheet from a stack of sheets by air suction and by the use of mechanical rollers. Towards the transfer station, at the merge point, a sheet meets an incoming stream of sheets which were already present in the printer. Before an image can be transferred to the sheet, any errors in the positioning of the sheet must be corrected. This is performed by the *correction* unit. After the correction, a sheet is transferred to a belt which keeps the sheet stuck to itself by air suction. The belt takes a sheet through the image transfer station where an image gets transferred to it. The image transfer is done by a grid of *print heads*. These print heads spray the ink onto the sheet as tiny particles. Depending on the type of the sheet, the ink is required to be at a specific temperature. A glossy sheet requires the ink to be at a temperature which is different from a normal sheet type. In case two different types of sheets follow each other, the ink needs to be first at the required temperature and then sprayed. This additional action is called a

setup. The process of transferring the ink is closely monitored by the *quality control* unit. After the quality control, a sheet passes onto the *drying* drum. The drum is heated by several electrical lamps to dry the ink. If the ink is not properly absorbed, the printed image might get distorted. After the drying process, the sheets which need to be only printed on one side, called *simplex* sheets, leave the system towards the *finisher* which stacks the sheets and may perform additional tasks like stapling or binding.

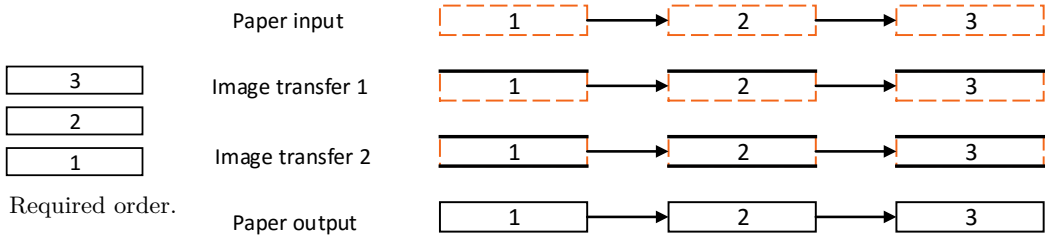
Sheets that need to be printed on both sides are called *duplex* sheets. The duplex sheets return back after drying on the *return path* towards the merge point. On the return path, the sheets are turned over such that the printed side is below. At the merge point, the returned sheets meet the incoming stream of new sheets. At this point a scheduling decision is made whether to first let a returning sheet join the queue to be printed or let a new sheet into the system. This decision is referred to as determining the *interleaving* of sheets. A scheduler must ensure that these two streams of sheets do not collide. The order in which these streams are merged together influences the performance of the printer. Merging similar sheets together will not require the temperature of ink to change and therefore no additional time is spent.

The additional time, if spent, is known as a *setup time* which is an expensive operation. For the LSP, such a setup requires in the order of 10 times additional time than nominal printing. This extra time is due to the fact that the heating or the cooling of the ink cannot be instantaneous: the physics behind the setup requires time. Thus, avoiding setups is key for a scheduler to gain performance as, if two consecutive sheets are of the same type then no setup is required. This conditional setup is known as *sequence dependent setup* [10]. An approach to reduce the number of setups in the LSP is to introduce a possibility to slow down the sheets on a segment on the return path. Special motors which facilitate variable rotational velocity are used to achieve programmable speed. This segment acts as a virtual *buffer* between the first pass and the second pass of a sheet. A scheduler can potentially slow down a set of returning sheets such that it merges with a similar type of incoming sheet. This merge does not require a setup, because the two sheets are of the same type and are consecutive. Increasing the number of such merges avoids setups which might be required otherwise. Buffering is thus used to improve the utilization of the image transfer and to increase the performance of the system.

The arrangement of resources in a paper path is similar to arrangement of machines in *shop* scheduling [10]. These resources are shared between the sheets. The path that sheets follow through an LSP is called a *paper path*. The re-entrant sheets flow back to the merge point over the *return path*. There are bounds on the speed it can travel on the return path. These bounds originate from the bounds over the rotational velocity that the motors in the buffer region can achieve. A *relative due-date* is the resulting upper bound between the first time a sheet is printed and the second time the sheet is printed. The combination of re-entrance, sequence dependent setup times and relative due-dates define the class of self-re-entrant flowshops addressed in this thesis.

A print request determines the order of the sheets at the output of an industrial printer. It is a requirement to output the sheets in the same order as they are in the document. Changing the order will violate the requirements on where certain pages must be in the output. Figure 1.4a shows an example of a required order of three sheets when they are stacked at the finisher. The paper output unit must adhere to this order as shown in Figure 1.4b. In order to be stacked in the required order, sheet number 1 must be output before sheet number 2. The arrows indicate the precedence: sheet number 1 must be processed before sheet number 2 if the arrow is $1 \rightarrow 2$. These constraints are referred to as *ordering constraints*.

The ordering constraints from the paper output translate to the constraints over the order in which image transfer 2 is performed. This translation is due to the fact that the sheets do not bypass each other on the paper path. The ordering constraints over the image transfer 2



(a) Required order.

(b) The order in which sheets are processed.

Figure 1.4: Example explaining the fixed order of sheets in an LSP.

further translate to image transfer 1 and the order of sheets in paper input. Therefore, the order of sheets at the paper input, the image transfers and the paper output are fixed and known at the time a job request is made. The machines performing these tasks must conform to this order. These ordering constraints are called a *fixed job order*.

The fixed job order makes the class of self-re-entrant flowshops different from classical flowshops. In flowshop scheduling the order in which a job gets processed at every resource is determined by the scheduler of the system. Self-re-entrant flowshops with fixed job order only need to determine the order in which the re-entering jobs interleave with the newly entering jobs. Therefore, a question arises whether the problem has become easier to solve compared to the classical flowshop scheduling problem? Due to availability of additional information and the ordering constraints the scheduling freedom has been reduced. Fixed order flowshops have polynomial time algorithms to find optimal schedules (see Appendix A for details). But did the fixed job order render the computational complexity of scheduling self-re-entrant flowshops with a fixed job order to be polynomial time? These questions are studied in this thesis.

The knowledge of the type of sheets and their required ordering becomes available when a print request arrives. Before that moment, a scheduler does not have sufficient information to compute a valid schedule. The set of jobs processed by a flowshop at once is called a *jobset*. For example, a request for printing several sheets is a jobset consisting of several jobs which are sheets. Once a jobset arrives, a flowshop must start processing it as soon as possible. Therefore a schedule must be computed quickly.

Designing self-re-entrant flowshops is an iterative process. A prototype is created to assess the feasibility of the idea. Then the prototype is further developed into a complete system. During the development, issues like the design of the paper path, placement and sizing of resources are considered. Determining the impact of the design decisions over the performance of a self-re-entrant flowshop is crucial. Creating several prototypes, one for each structural change, might be possible but might be expensive.

Buffers do increase the performance of the system, but also introduce performance variation. Jobs for which a scheduler can utilize the buffer has higher resource utilization. The jobs that do not benefit from the current size of buffers have less performance as they incur more setups. However, if the size of the buffer is increased then the performance might increase due to the fact that more jobs can benefit from the size of the buffer. Similarly, the time spent on the re-entrant path influences the available choices to the scheduler. A longer re-entrant path might benefit for the performance of one type of jobs and may introduce setups for other types. Such variation might introduce unpredictability which may incur additional costs due to unplanned logistic and storage demands. Therefore, design choices for self-re-entrant flowshops that have high performance and low variation are of high interest.

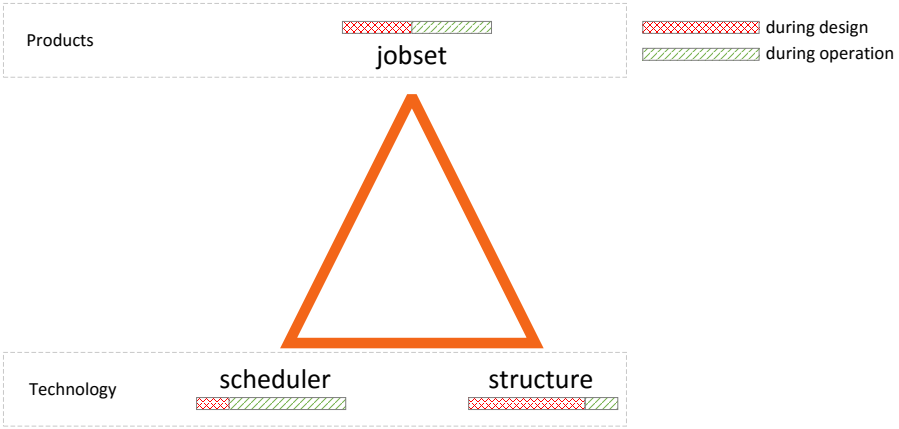


Figure 1.5: The aspects influencing the performance of self-re-entrant flowshops.

1.3 Research challenges

The performance of self-re-entrant flowshops is influenced by three aspects: 1) the *scheduler* 2) the *structure* of a flowshop and 3) the characteristics of a *jobset*. These aspects are shown in Figure 1.5. The characteristics of a jobset determine which operations are required to be performed on the products.

The structure of the flowshop influences the scheduling alternatives for the jobset. Certain alternatives might become infeasible due to changes in the structure. Furthermore, these alternatives lead to different timing constraints that may suite well for performance. Similarly, a different jobset can lead to different ordering and processing requirements for which the structure might be well suited and thus might lead to better performance. Tuning the scheduler to find good schedules for a class of jobsets or for a particular structure might have an increase in the performance. Thus, alternative configurations have trade-offs in performance.

The characteristics of a jobset influence the performance of a self-re-entrant flowshop and depend on the customers of a product. In most cases, the characteristics are specific to the market segment for which the products are being produced. For example, in car manufacturing, the shape of the cars, the metal used, and the paint coatings, depend on whether the car is for small family-use, SUV or for sport. Altering these characteristics is considered out of the scope of this thesis. However, different classes of the same products are studied. For example, for the case of the LSP, scheduling of different print requests like booklets, pamphlets and all sheets with same size is studied.

Most of the influences over the performance due to the structure of a self-re-entrant flowshop are at design time. For example, the dimensions of the system and the size of buffers used. Majority of the influences by a scheduler are made during its operation. For example, when a scheduler decides how much buffer capacity a product will use or when the ordering of products is determined. But, some influences, for example, the type and configurations of a scheduler might be decided at design time. The influences due to jobsets can either be at design time or during its operation. Decisions like which type of products a self-re-entrant flowshop will produce are made at design time. The jobsets influence the performance during the system operation due to the timing constraints. For example, timing constraints of jobsets from different market segments are different.

From the aspects shown in Figure 1.5, scheduler and structure both can be altered to increase the performance of self-re-entrant flowshops. The impact of the structure over the performance

becomes known when the scheduling behaviour of a system is determined. Therefore, solving the scheduling challenges become a precursor to solving the challenges of structural improvements in self-re-entrant flowshops.

The challenges in the design and operation of a self-re-entrant flowshop can be summarized in the following questions:

1. *How can a self-re-entrant flowshop with setup times, relative due-dates and fixed job order be quickly scheduled?*
2. *How likely is it to find an optimal schedule for such a flowshop quickly?*
3. *How can its performance be estimated?*
4. *How can its design be improved to reduce variation in its performance?*

The first challenge is to find a schedule to perform operations over a product. How can the timing requirements which originate from different sources be modelled and considered in the scheduling procedure? The model must support a scheduler in ensuring that the processing time, the setup time and the relative due-date constraints are not violated.

The second challenge is to determine what approach should be taken to generate schedules for self-re-entrant flowshops with fixed job order. The question is whether an optimal schedule can be found quickly. A yes to this question will render maximum performance for any number of products. If finding an optimal schedule quickly, for the scheduling problem, is highly unlikely, then a compromise on optimality needs to be made. How would that approach look like? How can a *good* schedule be found which still adheres to the timing constraints and can be computed quickly?

The third challenge is in the evaluation of early design decisions for a self-re-entrant flowshop. How can the impact of a design choice over the performance of the system be assessed? How does the system performance change when the structure and layout of the system are changed? This assessment must be fast as the design space of a self-re-entrant system is usually huge. Creating a separate model for each design choice and scheduling it is usually not a good choice due to limited time.

The fourth challenge is in the design of a self-re-entrant flowshop. Small changes in the design parameters of self-re-entrant flowshops produce variation in their performance. Certain designs yield high performance for a certain product type but might be less efficient for other product types. Designing a system for every product is possible but costly. In many cases, it might even be unknown which one of the possible types a system might be processing. How can the designs with high performance and low variation in performance be found? How can such a variation be measured? How can such a measurement be utilized in the design space exploration? Answers to these challenges will improve the performance of self-re-entrant flowshops.

1.4 Contributions

The contributions of this thesis are as follows. The first contribution proves that scheduling of self-re-entrant flowshops remains NP-Hard even if the jobs have a fixed order. Based on the proof, it is highly likely that a fast and scalable optimal scheduling algorithm is impossible. The contribution also presents, for self-re-entrant flowshops, a model of the timing constraints and a heuristic to find schedules. The second contribution specializes in finding schedules for an LSP with makespans that are, on average, shorter than the makespans of schedules found using the heuristic of the first contribution. The contribution achieves higher performance given the specific knowledge of the timing constraints in an LSP. For self-re-entrant flowshops, the third contribution describes a fast performance estimator for jobsets in which jobs repeat in a pattern. Contribution 4 uses contribution 1 to analyze the variation in the performance of self-re-entrant flowshops due to deviations in design parameters. The relation between contribution 4 and

contribution 1 is shown in Figure 1.6. The details of the contributions are described in the sequel.

1) A heuristic to schedule self-re-entrant flowshops with sequence dependent setup times, relative due-dates and fixed job ordering. This contribution shows that the problem of scheduling self-re-entrant flowshops remains to be NP-Hard even when the job order is fixed. In addition, it describes a heuristic for any self-re-entrant flowshop with setup times, due-dates and fixed job order. The computational complexity is proven by reducing the *source to target Travelling Salesman Problem* (st-TSP) to the self-re-entrant flowshop scheduling problem. Finding optimal solutions quickly, even for medium-size jobsets, is unlikely. Therefore a heuristic approach serves the purpose of scheduling the products at runtime. The heuristic is evaluated over a set of randomly generated instances of self-re-entrant flowshops. The quality of the schedules is assessed by comparing the makespan of the schedules with the lower bounds estimated using ILOG-OPL [11] constraint programming. The performance of the proposed heuristic is evaluated on a synthetic testset inspired from the literature. Given 60 seconds per testcase, the makespans of the schedules generated by the heuristic are 6%(median) larger than the estimated lower bounds. The contribution is submitted to:

[12] U. Waqas, M. Geilen, S. Stuijk, J. Pinxten, T. Basten, L. Somers, and H. Corporaal. "A heuristic to schedule self-re-entrant flowshops with sequence dependent setup times, relative due-dates and fixed job ordering". In: *Design of Embedded Systems, Springer* (2017, submission in review process).

2) A Re-entrant Flowshop Heuristic for Online Scheduling of the Paper Path in a Large Scale Printer. This contribution introduces a heuristic approach to schedule sheets in an LSP where the sheets arrive at the LSP at runtime. The LSP scheduling problem is solved as runtime scheduling of re-entrant flowshops with sequence dependent setup times, relative due-dates, with makespan minimization as the scheduling criterion. The heuristic is evaluated by testing on an industrial testset. The quality of the schedules generated by the heuristic are evaluated by a comparison to the lower bounds, to the schedules generated by the scheduler of the LSP and to the schedules generated by a modified version of the classical NEH (*Modified Nawaz Enscore and Ham* (MNEH)) heuristic [13]. On average, the heuristic schedules are 40% shorter than the schedules of the LSP scheduler and, on average, 89% shorter than the schedules generated by the MNEH heuristic. The heuristic schedules, on average, are 25% longer than the estimated lower bounds.

The second contribution also finds better schedules compared to the first contribution. The schedules are, on average, 11% shorter than the schedules found by the first contribution. However, on average, the heuristic of the first contribution is faster, as on average, it took 23% lesser time. This makes both the heuristics suitable for different tasks. On average, contribution 2 is suitable when schedules must be computed quickly and when schedules with slightly longer makespans are acceptable. Contribution 1 should be used when a compromise on the makespan is not possible. Both heuristics have cases where either runtime is higher than the other or the makespan are worse than the other. The second contribution is published in:

[14] U. Waqas, M. Geilen, J. Kandelaars, L. Somers, T. Basten, S. Stuijk, P. Vestjens, and H. 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.

3) Fast Performance Estimation for the Design of Self-re-entrant Flowshops. This contribution describes a method to estimate the performance of self-re-entrant flowshops with sequence dependent setup times, relative due-dates and fixed job order, in which the types of the products in a jobset repeat in a pattern. The information present in the pattern is used to compute the constraints enforced by the pattern over the scheduling freedom. Once the constraints are known, arrival times for products in the patterns can be determined. The estimator uses the knowledge of the arrival times to estimate the amount of time spent to process one pattern of products. The time for all patterns can then be estimated using the information for the time for one pattern. The estimator requires, on average, 1.1ms on an Intel i5 processor. It has applications during *Design Space Exploration* (DSE) of self-re-entrant flowshops. To get better performance while reducing costs, the estimator is used during the design process to explore the trade-offs between the structure and the performance of self-re-entrant flowshops. Using the estimator, the impact of the structural changes for an LSP and a research platform, *eXplore Cyber Physical Systems* (xCPS), are explored. This contribution is published in:

[15] U. Waqas, M. Geilen, S. Stuijk, J. Pinxten, T. Basten, L. Somers, and H. 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 2016*. 2016, pages 215–221. DOI: 10.1109/DSD.2016.26.

An extended version of the contribution is submitted to:

[16] U. Waqas, M. Geilen, S. Stuijk, J. Pinxten, T. Basten, L. Somers, and H. Corporaal. "A fast estimator of performance with respect to the design parameters of self re-entrant flowshops". In: *Microprocessors and Microsystems (MICPRO)* (2017, submission in review process).

4) A Measure of Variation for Design Choices of Self-re-entrant Flowshops. This contribution proposes a measure of variation in performance due to slight variations in the design decisions made for self-re-entrant flowshops. The final design may deviate from the desired design parameters. Using this measure, the design choices which have less fluctuation can be prioritized. Especially if the design choices with less variation also have high performance. The measure utilizes the information from the schedules generated for the self-re-entrant flowshop instances corresponding to specific design choices. Certain designs are good for performance as well as have less variation. A tool, based on this measure, can assist the designer of self-re-entrant flowshops to find less varying and high performing design choices. This contribution is submitted to:

[17] U. Waqas, M. Geilen, S. Stuijk, J. Pinxten, T. Basten, L. Somers, and H. Corporaal. "A Variation Measure for Design Choices of Self-re-entrant Flowshops". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2018, submission in review process).

The four contributions, together, aim to make it possible to perform runtime scheduling and fast DSE of self-re-entrant flowshops. They are described in detail in the following chapters.

1.5 Thesis outline

Chapter 2 formally introduces self-re-entrant flowshop scheduling problem and provides the details of the computational complexity of the problem. This chapter also describes a heuristic approach to compute schedules. A specialized approach to find schedules for an LSP is described is Chapter 3. The approach is dedicated for the restricted case where the sheets return only once in the re-entrant machine. Chapter 4 describes the performance estimation of self-re-entrant flowshops with a fixed job order. Variation-aware design of self-re-entrant flowshops is described in Chapter 5. Chapter 6 concludes this thesis and presents possible future work.

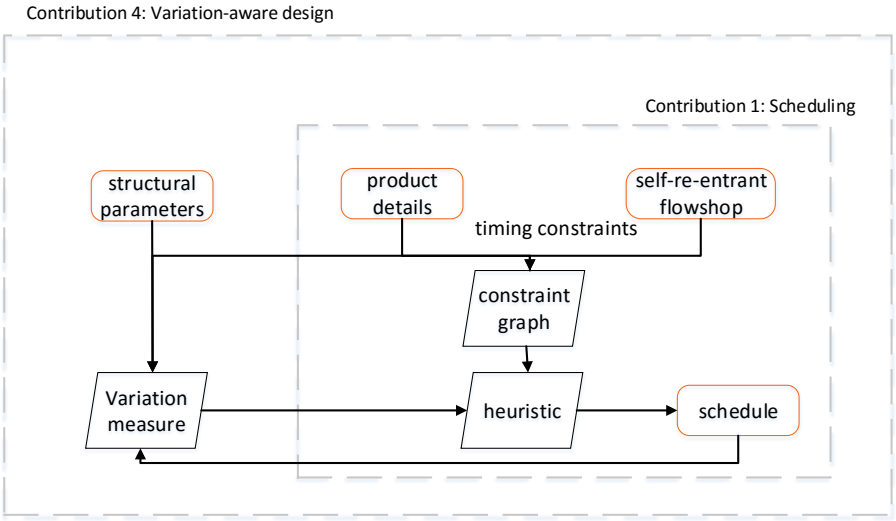


Figure 1.6: Relation between contribution 1 and 4.

"The key is not to prioritize what's on your schedule, but to schedule your priorities" - Stephen Covey

2

Scheduling of self-re-entrant flowshops

The machines in a self-re-entrant flowshop require scheduling of several operations performed on a job. One of the characteristics of a flowshop is that the jobs follow the same sequence through the machines. This flow is known upfront and depends on the products that are being made. For example, in a system that creates shirts, cloth is first cut into required segments of a shirt and then sewed. All shirts follow this order of cut and sew. A self-re-entrant machine has a choice whether to perform an operation on a returning job or to start processing a new job. This choice requires a decision that in turn has an impact over the performance of the flowshop. Processing a returning job might not require additional steps and thus might be better for performance. Scheduling of self-re-entrant flowshops requires assessment of such choices.

Re-entrant flowshops have many types based on which different scheduling techniques are formed. The types that are relevant to the scheduling of self-re-entrant flowshops are introduced in Section 2.1. Section 2.2 formally introduces the scheduling problem. The related work is reviewed in Section 2.3. The problem of finding optimal schedules for self-re-entrant flowshops and classical flowshops is known to be NP-Hard. However, if the order of the jobs is fixed, it is possible to derive polynomial time algorithms to find an optimal schedule for flowshops. This is not the case for self-re-entrant flowshops with sequence dependent setup times and due-dates, contrary to what the name may suggest. This work shows, in Section 2.4, that finding optimal schedules for such flowshops remains to be NP-Hard even if the order of jobs is fixed. Finding optimal schedules at runtime is thus unlikely to be feasible in a limited amount of time. A heuristic approach to find schedules for the scheduling problem is described in Section 2.5. Section 2.6 presents the results of the experimental evaluation of the heuristic approach. Section 2.7 concludes this chapter.

2.1 Introduction

Re-entrant flowshops are found in many industrial applications like Large Scale Printing [14], Wafer Sorting [6] and TFT-LCD manufacturing [5, 7]. In *flowshops*, the *flow* of jobs through the machines is the same for all jobs due to fixed sequence of machines. Every machine in a flowshop performs an *operation* on a job and passes the job to the next machine in the sequence. *Re-entrant* flowshops allow a job to be *re-processed* by a machine before passing it onto the next

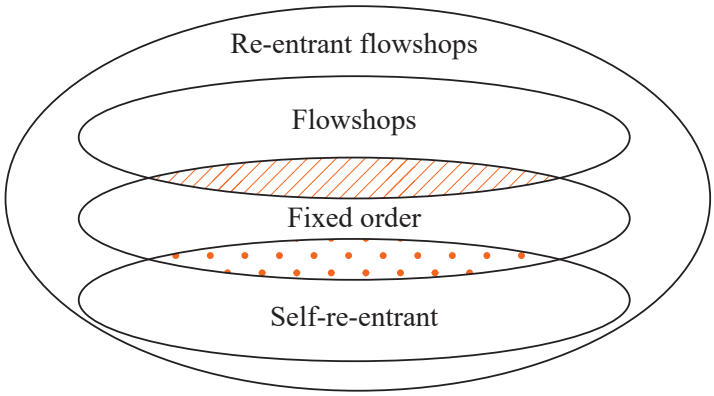


Figure 2.1: Different types of re-entrant flowshops.

machine. *Self-re-entrant* flowshops limit the jobs to re-enter only to the current machine where a job is. Once a job is passed on to a next machine, it cannot re-enter any of the previous machines. Self-re-entrant flowshops are a sub-class of re-entrant flowshops and are of interest due to their industrial applications like [5, 6, 7, 14].

Sequence dependent setup times require the processing of a job to be delayed on a machine for an amount of time depending on the type of two consecutive jobs in a machine. Such setup times arise from the fact that a job may leave a machine in a state that is required to be altered before the next job can be processed. The sequence dependent setup times impose a lower bound over the time passed between the processing of two jobs. *Relative due-dates* require that the time elapsed between the start of two jobs should not be more than a certain amount. Thus relative due-dates impose upper bounds over the amount of time elapsed between the start of processing of the two operations over the jobs. These due-dates arise, for example, from the minimum rotational velocity that could be achieved by a motor used to transport products. Furthermore, *minimization of makespan* is a scheduling criterion that aims to have high system utilization by having the end of a schedule, i.e. *makespan*, as soon as possible. Though significant research has been done to investigate scheduling of self-re-entrant flowshops, e.g. [6, 18], setup times [19, 20], due-dates [5, 14], the *combination* of self-re-entrance with sequence dependent setup time and relative due-dates for an arbitrary number of jobs and machines is studied for the first time in this thesis.

The physical characteristics of the available resources impose timing constraints on the operations performed by a self-re-entrant flowshop. A schedule not only needs to have high resource utilization but also must adhere to the timing constraints. The combination of *self-re-entrant* machines with *sequence dependent setup times* and *relative due-dates* is seen, for example, in an industrial printer. In the printer, reducing the number of setups improves the system performance as the setup times are around 10 times larger than the nominal processing times of an operation. The due-dates, on the other hand, impose a requirement that two operations must be performed within an interval relative to the start times of the operations. Failure to meet due-dates leads to collision of sheets. The timing constraints together with due-date constraints must be adhered by a scheduling technique when finding high performance schedules.

Different types of flowshops are shown in a Venn diagram in Figure 2.1. The optimal scheduling of re-entrant flowshops, self-re-entrant flowshops and flowshops is known to be NP-Hard [13, 18, 21]. Self-re-entrant flowshops are a subclass of re-entrant flowshops where jobs are only allowed to re-enter where they are being processed. Once a job leaves a machine, it never enters the machines

where it was previously processed. Note that flowshops are a subclass of re-entrant flowshops where the re-entrancy is zero i.e. a job is processed only once at a machine. Self-re-entrant flowshops require that there is at least one machine where jobs are processed at least twice. Classical flowshops do not fulfil this requirement. Thus self-re-entrant flowshop and classical flowshops are non-overlapping types of re-entrant flowshops, contrary to what their name may suggest. Both self-re-entrant flowshops and classical flowshops have subclasses where the order of jobs can be fixed.

The fixed order of jobs has a different impact over the computational complexity of classical flowshops and self-re-entrant flowshops. When the order of jobs in a classical flowshop is fixed, a polynomial time algorithm exists to find an optimal schedule (for intuitive proof refer to Appendix A). The subclass of classical flowshops with a fixed order is in P and is marked with a diagonal pattern in Figure 2.1. Section 2.4 shows that self-re-entrant flowshops with fixed job order, the subclass that is marked with a dotted pattern, remain NP-Hard even with fixed job order. For this subclass, a heuristic approach is described in Section 2.5. The heuristic is an extended version of the heuristic of [14] which was developed for scheduling industrial printers. The heuristic generates schedules for the scheduling problem with an arbitrary number of machines, jobs, operations and self-re-entrances.

2.2 Problem definition

A self-re-entrant flowshop consists of a set $\mathcal{M} = \{\mu_1, \ldots, \mu_i, \ldots, \mu_m\}$ of machines processing a jobset $\mathcal{J} = \{j_1, \ldots, j_n\}$. The machines process the jobs in an order described in a *flow vector* $\mathcal{V} = [v_1, \ldots, v_r]$ with $v_i \in \{1, \ldots, m\}$. In the flowshop, the processing of every job starts from the first machine, i.e. $v_1 = 1$ and ends at the last one, i.e. $v_r = m$. In a self-re-entrant flowshop a job is either re-processed by the same machine or passed on to the next machine. It means that in the flow vector $\mathcal{V}$, v_{i+1} either equals v_i or is the index of the next machine $v_i + 1$. Thus, the flowshop operates r times on a job with the set $\mathcal{O}_i = \{o_{i,1}, \ldots, o_{i,r}\}$ denoting the set of operations of a job j_i. The set $\mathcal{O}_\mathcal{J} = \bigcup_{j_i \in \mathcal{J}} \mathcal{O}_i$ contains the operations of all jobs in the jobset $\mathcal{J}$. The jobs in the flowshop have a *fixed job ordering* where the processing of jobs start from j_1 and ends with j_n. For each machine, the operations from different jobs require to be ordered as the machine is a *unary resource*.

The y^{th} operation of a job j_x is $o_{x,y}$ that takes $p(x,y)$ time units to execute. A machine can perform at most one operation at a time and preemptions are not allowed. A machine may require additional time, called *sequence dependent setup time*, to prepare for the processing of the next operation. The sequence dependent setup time between an operation $o_{x,y}$ and an immediately following operation $o_{u,w}$ is $s(x,y,u,w)$. Similarly, a *relative due-date* $d(x,y,u,w)$, is the maximum allowed amount of time difference between the start of the processing of an operation $o_{x,y}$ and the start of the processing of an operation $o_{u,w}$. For an operation $o_{x,y}$ its start time is $\mathcal{S}(x,y)$ where $\mathcal{S}$ is a schedule.

The operations in the flowshop have *precedence constraints* with a grid like structure as shown in Figure 2.2. The solid arrows denote the precedence constraints which should not be violated in any valid schedule. The dotted (red) arrows indicate the *conditional constraints* that must not be violated if an operation immediately follows another operation. For simplicity, only the conditional constraints originating from the operation $o_{1,r}$ are shown. These constraints can be used by a scheduler to enforce an order between two unordered operations. Precedence constraints between operations of a job arise from the flow of jobs in the flowshop. The precedence constraints between two consecutive jobs arise due to the fixed order of jobs. The conditional constraints arise from the execution and setup time requirements and may exist between an arbitrary pair of operations.

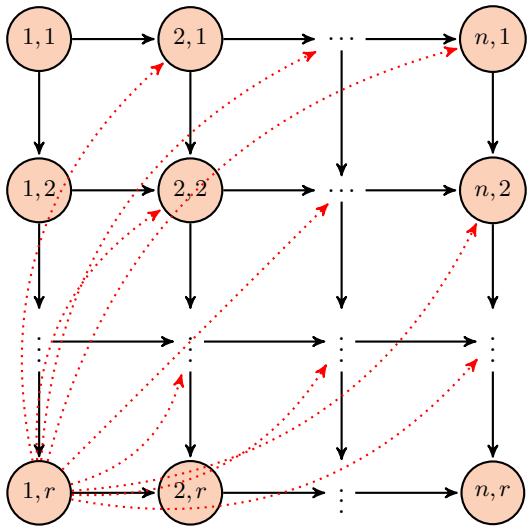


Figure 2.2: Precedence and conditional constraints in the scheduling problem.

Definition 2.1 The ***makespan***, $\mathcal{C}_{\mathcal{J}}$, of a schedule $\mathcal{S}$ for a jobset $\mathcal{J}$ is the start time of an operation that is performed the last in a schedule $\mathcal{S}$ computed as $\mathcal{C}_{\mathcal{J}} = \max_{o_{x,y} \in O_{\mathcal{J}}} \mathcal{S}(x,y)$.

The challenge is to find the start times for all operations such that the makespan $\mathcal{C}_{\mathcal{J}}$ of the schedule $\mathcal{S}$ is minimal and the following constraints hold. The operations are non-preemptive and every machine processes at most one operation at a time. For an operation $o_{i,j}$ that is immediately followed by an operation $o_{x,y}$ it must hold that $\mathcal{S}(x,y) \geq \mathcal{S}(i,j) + p(i,j) + s(i,j,x,y)$. The relative due-dates between two operations $o_{i,j}$ and $o_{x,y}$ must hold i.e. $\mathcal{S}(x,y) \leq \mathcal{S}(i,j) + d(i,j,x,y)$. This scheduling problem is called *self-re-entrant flowshop scheduling with sequence dependent setup times, relative due-dates and with fixed job order.*

2.3 Related work

Problems similar to the scheduling problem considered in this chapter exist. However, the combination of self-re-entrant machines with sequence dependent setup times, due-dates and with fixed job order is for the first time studied in work. Section 2.3.1 describes scheduling approaches relevant to the proposed heuristic which is a general version of the heuristic described in Chapter 3 and in [14]. Section 2.3.2 describes the work related to the study of the computational complexity of the scheduling problem.

2.3.1 Existing heuristics

Johnson was the first to invent an algorithm, called *Johnson Heuristic* (JH), that finds optimal solutions for two machine flowshops in polynomial time complexity. An extension to JH, namely *Extended Johnson* (EJ), was presented in [22]. It schedules re-entrant flowshops by applying JH to sub-problems and creating a schedule from sub-schedules. The EJ does not take sequence dependent setup times, due-dates and self-re-entrance into account. Searching for feasible sub-problems requires incorporation of the due-dates in the search and therefore limits the applicability of EJ on the scheduling problem under consideration.

The *Modified Nawaz Enscore and Ham* (MNEH) heuristic, described in [13], is used to schedule

re-entrant flowshops. The heuristic starts with a feasible, arbitrary input sequence of operations that obeys the precedence constraints and shuffles the input sequence to find an operation ordering that leads to better schedules. However, the heuristic assumes that any random schedule that obeys the precedence constraints is always feasible. This assumption does not hold in our case due to the relative due-dates. Failure to meet a single constraint makes a schedule infeasible. For randomly generated schedules it is highly likely that one or more constraints are not met. Our initial experiments with the MNEH heuristic show that given 10 random trials (which obey the precedence constraints) to find a feasible schedule, for a test case, the heuristic does not find a feasible schedule that could be improved with the classical MNEH heuristic of [13]. Therefore, for most of the test cases MNEH finds no feasible schedule.

The runtime of well known heuristics for variants of flowshop scheduling was compared in [23]. The heuristics were evaluated over a commonly accepted benchmark in literature. The heuristics took between 5 milliseconds to 39 seconds for small to large size jobsets. Similarly, the wafer probing problem solved in [6] took between 2 seconds to 50 seconds. Though these problems are structurally different (setup times, relative due-dates, etc.) than the problem under study, the runtime of the methods used to solve them show the accepted notion of how fast a heuristic should be.

2.3.2 Related work to the complexity of the problem

The computational complexity of the related variants of the scheduling problem are under investigation for a long time. Still, it is of interest because a slight difference of assumptions might make two similar scheduling problems fall under different complexity classes. This section outlines those variants having the objective of *makespan minimization*. Table 2.1 summarizes the variants whose details are as follows. Depending on the assumptions/constraints, as described below, the problems which are similar can either be in the class of NP-Complete/NP-Hard problems or problems for which polynomial time solutions are known. For example, the simplest of the variants of the problem is the scheduling of operations over a *single machine* with the objective of makespan minimization that can be performed optimally using the *Weighted Shortest Processing Time first* (WSPT) rule (computational complexity $O(nlogn)$). The optimality of the rule is proven in [18]. However addition of the sequence dependent setup times to the decision version of the single machine scheduling problem, called *Single Machine Scheduling with Sequence Dependent Execution Times* (SMS-SDET), makes it NP-Complete (for proof refer to Appendix B).

A larger class of problems to the problem solved in this chapter is *Precedence Constrained Scheduling* (PCS) that consists of a set of tasks $T = \{t_1, \ldots, t_n\}$, processors $P = \{p_1, \ldots, p_m\}$, a partial order $(\prec, T)$ and execution times $E : T \to \mathbb{Z}^+$. The problem is to find a schedule S defining that task t_i has start time $S(i)$ to *minimize* $\max_{1 \leq i \leq n} S(i)$. Furthermore, the following constraints must hold on S. (1) A processor executes at most one task. (2) The start times $S(i) < S(j)$ if $t_i \prec t_j$. (3) Every task t_i executes for $E(t_i)$ time units uninterruptedly. PCS is known to be NP-Complete [24]. The scheduling problem addressed in this paper is similar to PCS as there are multiple machines and it has an ordering originating from flowshops. Additionally, different from PCS, the scheduling problem addressed in this work has sequence dependent setup times and due-dates.

A restricted case of PCS is *PCS with Unit execution times* (PCS-U) that is $\forall t \in T : E(t) = 1$. PCS-U stays NP-Complete [25]. However *PCS with two processors, unit execution time - PCS-UM2* has a polynomial time optimal solution [26, 27]. The scheduling problem addressed in this paper has a unidirectional grid structure (explained in detail in the following section) of the orderings of the operations due to the fixed job orderings. As shown in this work the scheduling problem stays NP-Hard even with the additional restriction of fixed job orderings. Similarly, if

Problem	Type	Complexity	Ref
SMS	Single machine scheduling	Polynomial	[18]
SMS-SD	SMS, sequence dependent times	NP-Complete	Appendix B
PCS	Precedence constrained scheduling	NP-Complete	[24]
PCS-U	PCS, unit execution times	NP-Complete	[25]
PCS-UM2	PCS-U, 2 machines	Polynomial	[27]
PCS-UF	PCS-U, forest ordering	Polynomial	[21]
PCS-M2E	PCS, 2 machines, execution time 1 or 2 units	NP-Complete	[28]

Table 2.1: Similar scheduling problems with different complexity classes.

the mapping of operations are fixed to processors then both the PCS-U with either two machines and an arbitrary ordering or the PCS-U with arbitrary machines and a forest ordering (PCS-UF), an ordering where the constraints have a form as a set of trees, are known to be NP-Complete [28]. The interplay of assumptions and constraints make the computational complexity of the relevant scheduling problems fall in polynomial time or NP-Complete/NP-Hard complexity classes. The computational complexity analysis for the scheduling problem described in this chapter shows that even with additional constraints due to a fixed job order the scheduling problem stays NP-Hard. In contrast, for classical flowshops, the optimal schedule can be found using a polynomial time algorithm when the order of the jobs is fixed.

2.4 Computational complexity of the scheduling problem

The computational complexity of optimal scheduling of self-re-entrant flowshops with sequence dependent setup times, relative due-dates and fixed job order is shown to be NP-Hard, in several steps. First, it is shown that the decision version of the scheduling problem with a single machine and *the number of jobs equal to the number of operations* is NP-Complete by a reduction from the *source to target Travelling Salesman Problem* (st-TSP) [21, 29]. From the reduction it follows that the decision version of the scheduling problem with arbitrary number of machines, jobs and operations is NP-Complete. In the third step it is shown that the (optimization version of the) scheduling problem is NP-Hard. The st-TSP is first formally defined followed by the definition of the squared version of the scheduling problem.

Definition 2.2 (st-TSP [21]) Given a set $C = \{c_1, c_2, \ldots, c_u, c_s, c_t\}$ of cities with c_s and c_t as the start and end cities, distance between cities $dc(c_i, c_j) \in \mathbb{Z}^+$ for city pairs and a positive integer L, determine whether a permutation π over the set $\{1, \ldots, u\}$ exists such that the tour $< c_s, c_{\pi(1)}, \ldots, c_{\pi(u)}, c_t >$ has length at most L i.e.
$[dc(c_s, c_{\pi(1)}) + (\sum_{i=1}^{u-1} dc(c_{\pi(i)}, c_{\pi(i+1)})) + dc(c_{\pi(u)}, c_t)] \leq L$?

The decision version of the st-TSP is described in Definition 2.2 and is known to be NP-Complete [21]. The squared decision version of the scheduling problem to which st-TSP is reduced is as follows.

Square decision version: *Given n jobs with n operations each executing over a single machine with flow vector $\mathcal{V} = [1, \ldots, 1]$ with $|\mathcal{V}| = n$, processing times, setup times, relative due-dates and a non-negative integer I is there a schedule with makespan at most I?*

An intuitive illustration of the reduction of st-TSP to the squared decision version of

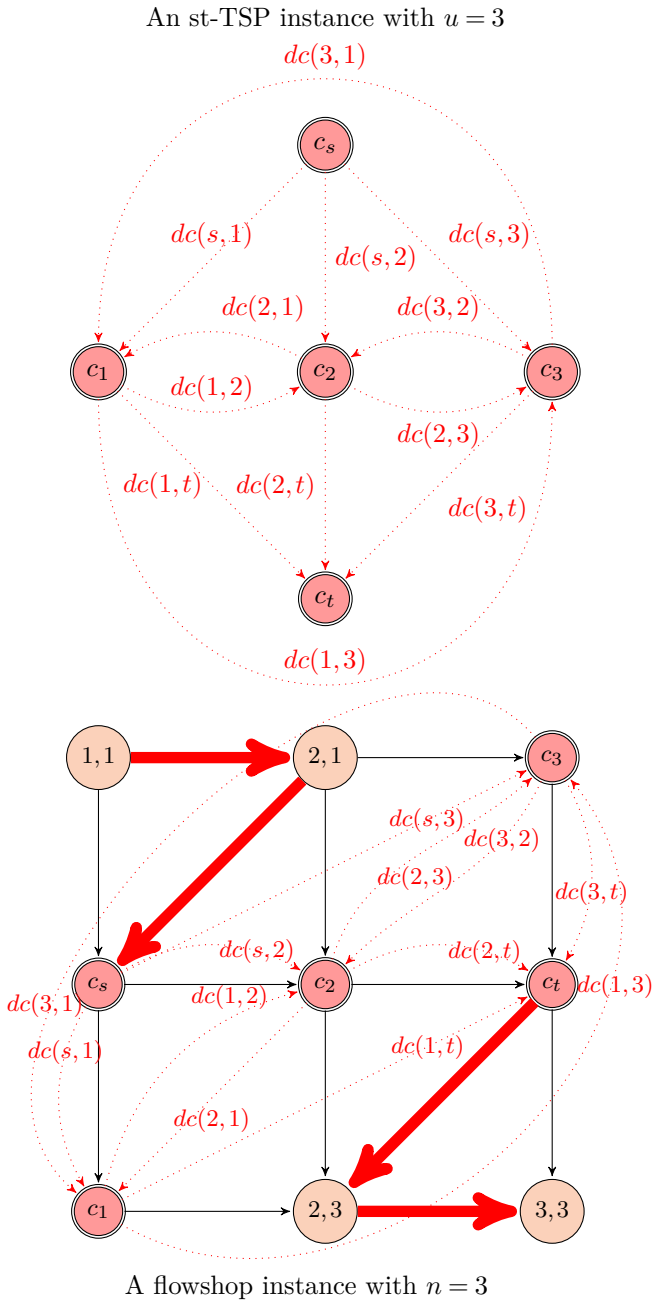


Figure 2.3: An illustration of the reduction of st-TSP to the squared decision version of the scheduling problem.

the scheduling problem is shown in Figure 2.3. An example of a self-re-entrant flowshop instance is taken with three jobs $n = 3$ i.e., 9 operations and an st-TSP instance with $u = 3$ i.e., in total 5 cities. The operations are split into 3 groups: *city operations* that have double outline and denote the cities encoded from the st-TSP instance, the operations *above* the city operations and the operations *below* the city operations. The operations above the city operations will be forced to assume a fixed ordering (shown as bold arrows) starting from $(1,1)$ then to $(2,1)$ and ending on $(1,2)$ (the starting city c_s). From $(1,2)$ there is a choice of visiting the city operations in any order. From any city operation there is a choice to go to operation $(3,2)$ from where the fixed ordering of the operations below the city operations starts. The fixed ordering ends with the last operation of the scheduling problem i.e. $(3,3)$. The edges corresponding to the edges from the st-TSP instance are shown as dotted edges with labels that have the same weight as the corresponding st-TSP edge e.g. $p(i,j)+s(i,j,x,y) = dc(a,b)$. The order over the operations in the scheduling problem (and thus the encoded operations) determines the ordering of the cities in the st-TSP. The fixed ordering in the operations above and below the encoded operations is achieved by either setting the sum of the execution and the setup time to be 0 for the allowed ordering or to $L+1$ for the prohibited ordering making any path using it longer than L. Then a schedule with a makespan at most I for the encoded problem determines an ordering for the st-TSP problem with a tour with length less or equal to L. If an st-TSP ordering with length at most L does not exist then a feasible schedule with makespan at most I does not exist.

The squared decision version and the decision version of the scheduling problem are in NP. The scheduling problem is in NP as an ordering τ of the tasks in the scheduling problem could be guessed non-deterministically. A polynomial space/time verifier exists that either finds a feasible schedule given an ordering or detects that the guessed ordering τ leads to an infeasible schedule as follows. Using the ordering, a schedule $\mathcal{S}$ could be computed using the longest path variant of the Bellman-Ford algorithm which has polynomial runtime and space requirements (space requirement $O(n^2)$ and computational complexity of $O(n^6)$). The variable n is the number of jobs in the scheduling problem. The distances found by the Bellman-Ford algorithm are the *As Soon As Possible* (ASAP) start times of the operations in the scheduling problem. To verify the feasibility of the schedule computed by Bellman-Ford all constraints must hold. The verification of a single constraint can be performed in $O(1)$ as the inequality can be confirmed/rejected in constant time complexity. Furthermore, there are at most n^4 processing and setup times constraints and at most n^4 relative due-date constraints which both are polynomial in number. Hence, evaluation of constraints can be performed in polynomial time.

st-TSP is polynomial time/space reducible to the squared decision version of the scheduling problem. Given an arbitrary st-TSP instance we encode the instance to the squared decision version of the scheduling problem with $n = u$ as follows. All deadlines equal $L+1$. The processing times $p(i,j)$ all equal 0. The deadlines and the processing times are effectively not used in the proofs but still the problem remains NP-Complete. The setup times are $L+1$ unless they are specified by following equations.

$$\forall 1 \leq r \leq n-2, \forall 1 \leq c \leq n-r-1 : \; s(c,r,c+1,r) = 0 \tag{2.1}$$

$$\forall 1 \leq r \leq n-2 : \; s(n-r,r,1,r+1) = 0 \tag{2.2}$$

$$\forall 1 \leq i \leq n : \; s(1,n-1,i,n-i+1) = dc(c_s,c_i) \tag{2.3}$$

$$\forall 1 \leq i,j \leq n : \; s(i,n-i+1,j,n-j+1) = dc(c_i,c_j) \tag{2.4}$$

$$\forall 1 \leq i \leq n : \; s(i,n-i+1,n,2) = dc(c_i,c_t) \tag{2.5}$$

$$\forall 3 \leq r \leq n, \forall n-r+2 \leq c \leq n-1 : \; s(c,r,c+1,r) = 0 \tag{2.6}$$

$$\forall 3 \leq r \leq n : \; s(n,r-1,n-r+2,r) = 0 \tag{2.7}$$

The fixed ordering for the operations above the city operations is enforced by Equations 2.1 and 2.2. Equation 2.3 encodes the distances outgoing from the c_s node. Equation 2.4 encodes all the distances between c_1, c_2, ..., c_u. Similarly, Equation 2.5 encodes the distances incoming to the c_t node. The fixed ordering below the city operations is encoded by the Equations 2.6 and 2.7.

The encoding of the st-TSP problem to the squared decision version of the self-re-entrant flowshop scheduling problem has polynomial time. A schedule with makespan at most L determines an ordering for the st-TSP cities with a tour length at most L as in polynomial time the encoded operations can be sorted (with respect to their start times) and the ordering can be determined. Thus, the problem can be encoded and the result can be derived in polynomial time.

The reduction preserves the correctness. The squared decision version of the scheduling problem will accept all 'yes' instances of the st-TSP problem. For any 'yes' instance there exists a permutation π of cities resulting in a tour with length at most L. Because such π exists there exists an ordering τ of tasks in the scheduling problem in which the ordering of the city operations is the same as the order of cities in the permutation π. Because the ordering of the non-city operations can be chosen with setup and execution times equal to zero and deadlines equal to $L+1$ the resulting schedule has makespan L thus accepting the 'yes' instance.

The squared decision version of the scheduling problem will reject all 'no' instances of the st-TSP problem. For a 'no' instance there does not exist an ordering π of cities resulting in a tour length at most L. That means, there is no ordering τ which can result in the encoded operations to be scheduled in an time interval of L time units. As the smallest interval in which the non-encoded operations can be scheduled is 0 time units the complete scheduling problem cannot be scheduled in L time units thus rejecting the 'no' instance. ■

It has been shown that the squared decision version of the scheduling problem is in NP, has st-TSP problem as an instance and preserves the correctness of st-TSP results thus showing that the squared decision version of the scheduling problem is NP-Complete. In the sequel it is shown that the decision version of the scheduling problem with arbitrary number of jobs and operations is also NP-Complete.

Decision version: *Given n jobs with r operations each executing over a set of machines $\mathcal{M}$ with self-re-entrant flow vector $\mathcal{V}$ with $|\mathcal{V}| = r$, non-negative processing and setup times, non-negative relative due-dates and a non-negative integer I is there a schedule with makespan at most I?*

The decision version of the scheduling problem is in NP as described for the squared decision version of the problem. Furthermore, the squared decision version of the scheduling problem is a sub-class of the decision version of the scheduling problem. All instances of the squared decision version of the scheduling problem are also instances of the decision version of the scheduling problem and thus the decision version of the scheduling problem is also at least NP-Complete.

Optimization version: *Given n jobs with r operations each executing over a set of machines $\mathcal{M}$ with self-re-entrant flow vector $\mathcal{V}$ with $|\mathcal{V}| = r$, non-negative processing and setup times, non-negative relative due-dates, find a schedule $\mathcal{S}$ with shortest makespan.*

To show that the optimization version of the scheduling problem is NP-Hard, it is shown that the decision version of the scheduling problem can be solved, in polynomial time, using a solution for the optimization version. An intuitive explanation of the polynomial transformation is as follows. The optimization version of the scheduling problem finds an optimal schedule. Let the makespan of the optimal schedule be $\mathcal{C}_o$. If $\mathcal{C}_o \leq I$ then a schedule smaller than I is known yielding a yes to the decision version of the problem. In case $\mathcal{C}_o > I$, no schedule shorter than I exists yielding a no to the decision problem. ■

2.5 Heuristic approach

This section describes an approach to perform runtime scheduling of products in a self-re-entrant flowshop with sequence dependent setup times, relative due-dates and fixed job order. The approach models the timing constraints in the scheduling problem as a *constraint graph*. Section 2.5.1 describes the details of the constraint graph and describes how the Bellman-Ford algorithm is used to compute the ASAP start times and to assess the feasibility of schedules. The start times are a valid schedule if the order of operations have been determined out of possible scheduling alternatives. The approach determines an order by modelling the *scheduling freedom* as a *Labelled Transition System* (LTS). The LTS is referred to as the *scheduling-space* of the problem. The details of the scheduling-space are described in Section 2.5.2. The heuristic described in this section evaluates the scheduling decisions as a *path* between the states in the scheduling space. Multiple paths may exist from the start state till the final state which represent different scheduling decisions. The heuristic aims to find good schedules by ranking different paths in the scheduling-space using the metrics, *productivity* and *flexibility* of the scheduling decisions. The details of the metrics are provided in Section 2.5.3.

2.5.1 The constraint graph

A constraint graph is a tuple $cg(V, E = E_C \cup E_O, w, G, m, v_s, v_d)$. The set V is a set of vertices that are the operations performed by machines over each job. Figure 2.4 shows a constraint graph for three jobs j_1, j_2, j_3 having two operations a_i, b_i each on a self-re-entrant flowshop with flow vector $\mathcal{V} = [1,1]$. The set $E \subseteq V \times V$ is a set of directed edges. An edge $(a_1, b_1) \in E$ with weight $w(a_1, b_1) = 6$ denotes a timing constraint between operations a_1 and b_1. Let t_{a_1} and t_{b_1} be the start times of a_1 and b_1 respectively then the timing constraint modelled by the edge is $t_{b_1} \geq t_{a_1} + w(a_1, b_1)$. The timing constraints due to the processing and the setup time requirements have a non-negative weight. The timing constraints due to the relative due-dates have a negative weight. For example, in Figure 2.4, the edge (b_1, a_1) has weight -10 with the constraint $t_{a_1} \geq t_{b_1} - 10$. The function $w : E \to \mathbb{Z}$ maps each edge in cg to its weight.

The set of edges E is partitioned into two sets E_C and E_O. The set E_C consists of *compulsory constraints* that must not be violated. The compulsory constraints are due to the relative due-dates, the order of operations due to the flowshop and due to the fixed job order. The constraint graph with only its compulsory edges is called *compulsory constraint graph* and consists of solid and dashed edges. The set E_O contains *optional constraints* that can be added to E_C to enforce an order between two unordered operations. The optional constraints are shown as dotted edges. The ordering is performed while computing a schedule. For example, the edges (a_3, b_1) and (b_1, a_3) are optional edges. A scheduler can add (a_3, b_1) to the set of compulsory edges when b_1 is processed immediately after a_3. The weights on the optional edges denote the timing constraints that must be enforced if an operation immediately follows the other. The function $G : V \to \{1, \ldots, m\}$ determines the group of a vertex in V. Operations that are processed at the same machine have the same group. The optional edges are only between vertices that belong to the same group. The source vertex v_s is a vertex with no incoming edge and has one outgoing edge with weight 0 to the vertex corresponding to the first operation of the first job. The destination vertex v_d is a vertex with no outgoing edge and has one incoming edge with weight 0 from the vertex corresponding to the last operation of the last job.

Machines in a self-re-entrant flowshop are assumed to be unary resources. The operations processed by a machine have corresponding vertices in the same group. These operations require to be totally ordered such that for any operation in a group all other operations are processed before or after its operation. Two vertices a and b are ordered with respect to each other when either of the vertices is reachable from the other by a path with non-negative edges from the set

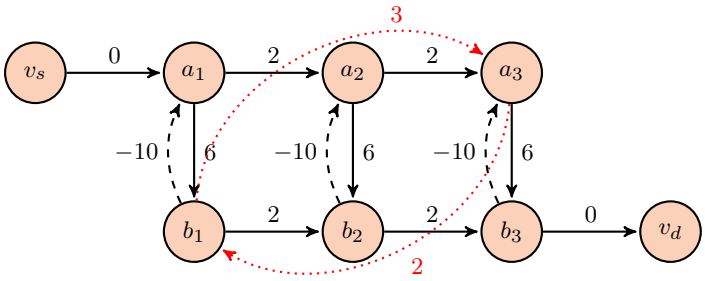


Figure 2.4: An example of a constraint graph.

of compulsory edges. For example, vertices b_1 and a_3 are ordered if either of the optional edges (a_3, b_1) or (b_1, a_3) are added to the set of compulsory edges. A group of vertices is totally ordered when all vertices in the group are ordered with respect to others. A constraint graph is considered totally ordered when all vertex groups in the graph are totally ordered. A totally ordered constraint graph either has valid start times or has a conflict between its timing constraints.

For a totally ordered constraint graph a longest path variant of Bellman-Ford algorithm is used to find a valid schedule or a conflict between the timing constraints. The distances between the source vertex and the remaining vertices are the ASAP start times for the operations. For example, assuming (a_3, b_1) is added to E_c in Figure 2.4, the start times are $a_1(0)$, $a_2(2)$, $a_3(4)$, $b_1(6)$, $b_2(8)$, $b_3(10)$, $v_d(10)$.

Algorithm 2.5.1: *RelaxEdges*$(cg, D, \overline{w})$

```
1 foreach (a,b) ∈ E_c of cg do
2 |   if D(b) > D(a) + w̄(a,b) then
3 |   |   D(b) ← D(a) + w̄(a,b)
4 |   end
5 end
6 return D
```

Algorithm 2.5.2: *BellmanFord*(cg)

```
1 w̄ ← negate all weights of w in cg
2 Set D(v_s) to 0 and to ∞ for all other vertices in V of cg
3 foreach v ∈ V do
4 |   foreach e ∈ E_c do
5 |   |   D ← RelaxEdges(cg, D, w̄)
6 |   end
7 end
8 D̄ ← RelaxEdges(cg, D, w̄)
9 if D̄ does not equal D then abort else D are start times.
```

The longest path variant of the Bellman-Ford algorithm to compute the ASAP start times for a totally ordered constraint graph is described in Algorithm 2.5.2. The algorithm repeatedly relaxes

edges in a graph to find distances between vertices. Algorithm 2.5.1 describes the relaxation of edges. Given a constraint graph, a vector of distances of each vertex from the source node and the weights of the edges, the algorithm checks whether a vertex is reachable by a smaller distance than its current distance. If the vertex is indeed reachable then the distance of the vertex is updated with the smaller value. The relaxation algorithm is used by Algorithm 2.5.2. In the start, the weights of the all edges in a constraint graph are negated. Due to the negation the algorithm finds a longest path from the source vertex to any other vertex. The algorithm iterates for each vertex, on line 3 and for each edge, on line 4, to compute the longest distance between the vertices in the graph and the source vertex. After the iterations if the result of one more relaxation is not equal to the outcome of the previous relaxation, then the constraint graph had conflicting timing constraints, e.g, due to a positive cycle. Otherwise, the distances are ASAP start times satisfying all the timing constraints. The distance of the destination vertex from the source vertex is the makespan of the schedule.

The start times computed by the Bellman-Ford algorithm are a valid schedule if the constraint graph is totally ordered. The heuristic proposed in this chapter first totally orders a constraint graph by modelling the scheduling freedom in the problem as a scheduling-space. While traversing the space, the heuristic uses the Bellman-Ford algorithm to assess whether certain scheduling alternatives are feasible or not and what the impact is of an alternative on the makespan.

2.5.2 Scheduling-space of the scheduling problem

The scheduling-space of a machine in the scheduling problem is a LTS in which a state represents the amount of *remaining work* i.e operations that are not processed yet. A transition between two states represents processing of an operation. Thus, a path consisting of several transitions represents an *order* in which certain operations can be performed.

Definition 2.3 (Scheduling-space) A scheduling-space (Q, Σ, δ) for a self-re-entrant flowshop with n jobs having r operations each and a fixed job order consists of set of states $Q = \{(s_1, \ldots, s_r) | s_i \in \{0, 1, \ldots, n\}\}$, a set of labels $\Sigma = \{o_1, \ldots, o_r\}$, a partial transition function $\delta : Q \times \Sigma \to Q$, a start state $(n, n, \ldots, n)$ and a final state $(0, 0, \ldots, 0)$.

Figure 2.5 shows the reachable part of the scheduling-space of a self-re-entrant flowshop problem with three jobs having two operations, each being processed by a single machine with the flow vector $\mathcal{V} = [1, 1]$. The *initial* state (shown with dotted outline) represents the initial state of the system i.e when all operations still need to be processed. The numbers inside a state represent the count of remaining operations per re-entrance of a job. From the start state only the last operation in the problem is possible due to the fixed job and operation ordering. In the example of Figure 2.5 operation o_2 of the third job is the last operation due to the precedence constraints between jobs and operations. From the state $(3, 2)$ there are two possible operations that can be processed immediately before. They are o_1 of the third job or o_2 of the second job. There are states where only operation o_2 is possible. For example, the state $(2, 2)$. The *final* state (shown with double outline) of a state-space is $(0, 0)$ and represents that all operations have been processed.

The reachable part of the state-space has an *invariant* that for a state $k \in Q$ the index $k(i)$ is always greater than or equal to $k(i+1)$. Furthermore, for any state there are at most r next states due to the fact that there are r operations. It is possible that from a given state k operations have their precedence constraints satisfied and thus ready for processing. These invariants are a direct consequence of the fixed operation order due to flowshops and fixed job order.

From a state all reachable states following it can be found by using the pseudo-code described in Algorithm 2.5.3. The block starting at line (2) finds whether decrement in last operation count for the given state is possible or not. If it is possible then a new state is created with a decrement

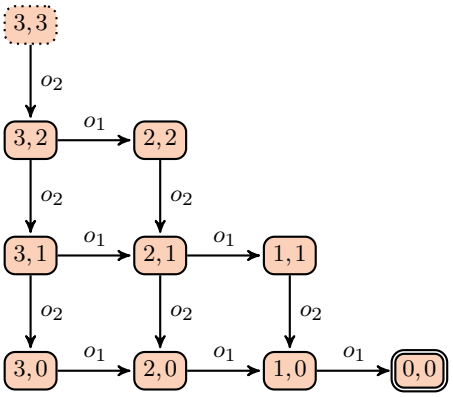


Figure 2.5: An example of a scheduling-space of a flowshop scheduling instance.

Algorithm 2.5.3: *FindNextStates*$(state, r)$

```
1  NextStates ← {}
2  if state(r) > 0 then
3  |  s ← state
4  |  s(r) ← s(r) - 1
5  |  NextStates ← NextStates ∪ {s}
6  end
7  i ← r - 1
8  while i ≥ 1 do
9  |  if state(i) ≥ state(i+1) then
10 |  |  s ← state
11 |  |  s(i) ← s(i) - 1
12 |  |  NextStates ← NextStates ∪ {s}
13 |  end
14 |  i ← i - 1
15 end
16 return NextStates
```

in the count for an operation o_i. Similarly, the block starting at line (9) iteratively constructs new states if a decrement between two consecutive operation counts is possible. For example, in Figure 2.5 for state $k = (3,2)$ where $r = 2$, the block starting at line (2) will create state $(3,1)$ as $k(i)$ is non-zero and thus it is decremented. The block starting at line (9) checks whether $k(1) \geq k(2)$ which is *true* for the example. Thus, another state is created with a decrement in $k(1)$ resulting in state $(2,2)$. Given the number of jobs and operations in the scheduling problem, the entire scheduling-space including the transition function δ is generated using the algorithm by starting from the initial state and finding next states till the final state is reached. Additionally, the algorithm is also used when computing different paths through the states to assess different scheduling decisions.

Definition 2.4 (Path) A path in a scheduling-space (Q, Σ, δ) is a sequence $p = \langle s_1, \ldots, s_x \rangle$ of states s_i from Q such that for all $1 \leq y < x$ there is some $\sigma \in \Sigma$ such that $\delta(s_y, \sigma) = s_{y+1}$. p is said to have length $|p| = x$.

A path through the states in a scheduling-space represents the order in which operations can be performed. An example of a path in the scheduling-space shown in Figure 2.5 is $\langle(3,3),(3,2),(3,1)\rangle$ of length 3. Different paths in a scheduling-space represent orderings of operations in the scheduling problem. Thus, evaluation of different paths leads to an evaluation of scheduling decisions. Starting from a given state, paths which end in the same state, called a *merge state*, represent different orderings of the same set of operations. *For a fair comparison of possible scheduling decisions, all paths that start from a state and merge at another state should be compared.* Such a comparison is fair because the paths represent the same amount of work achieved but with different orders.

Definition 2.5 (Merge State) For a state s, a merge state s_m is the state where all paths starting from the state s end.

In Figure 2.5, for the state $(3,2)$, following are two examples of merge states: $(2,1)$ is the merge state for paths $\langle(3,2),(3,1),(2,1)\rangle$ and $\langle(3,2),(2,2),(2,1)\rangle$. Similarly, state $(1,1)$ is also a merge state for the state $(3,2)$ with paths $\langle(3,2),(3,1),(2,1),(1,1)\rangle$ and $\langle(3,2),(2,2),(2,1),(1,1)\rangle$. In fact all states that are *reachable* from the state $(2,1)$ are merge states for the state $(3,2)$. The state $(2,1)$ is a merge state with the shortest paths originating from $(3,2)$ and is called the *nearest merge state*.

Definition 2.6 (Nearest Merge State) For a state s, out of the set of all merge states S_M, the nearest merge state is a state $s_n \in S_M$ where the paths starting from s and ending in s_n are the shortest compared to all paths starting from s and ending in any of the states in S_M.

Algorithm 2.5.4: NearestMergeState(state, r)

```
1 NextStates ← FindNextStates(state, r)
2 s_n ← state
3 foreach s ∈ NextStates do
4 |   d ← state − s
5 |   s_n ← s_n − d
6 end
7 return s_n
```

Algorithm 2.5.4 computes the nearest merge state for a given state. It uses the Algorithm 2.5.3 to compute the next states for the given state. Then, on line 4 it computes the member-wise difference between the given state and a next state. The difference is cumulatively subtracted from s_n which was initialized with the given state. An intuitive argument that the algorithm finds the nearest merge state is as follows. A transition in the scheduling-space represents the execution of an operation. All transitions from a state are possible executions of different operations. The nearest merge state is a state where all of those possible executions have happened (else the paths did not meet). Thus subtracting the difference of next states and the current state, to the current state, results in a state that represents the work between the current state and the merge state. Different paths originating from a given state and ending in the nearest merge state are different possible scheduling decisions to achieve the same work. The following section describes an approach to select paths through the scheduling-space in order to evaluate possible scheduling decisions.

2.5.3 Ranking of paths in the scheduling-space

The scheduling-space consists of states that represent the amount of work achieved by different transitions. From a given state there are multiple paths to achieve the same amount of work.

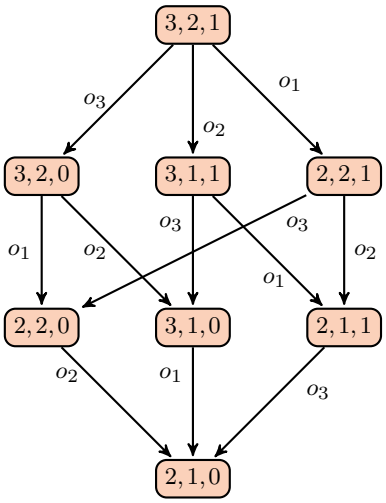


Figure 2.6: A partial scheduling space.

This section describes an approach to select a path from the initial state to the final state. Such a path represents the order of operations on a machine. Once all machines in the scheduling problem have the operations ordered a schedule corresponding to the ordering is computed by the longest path variant of the Bellman-Ford algorithm.

Figure 2.6 shows a part of the scheduling-space for three jobs having three operations each. The nearest merge state of the state $(3,2,1)$ is the state $(2,1,0)$ reachable by six paths in total. For example one out of the six paths is $\langle(3,2,1),(3,2,0),(2,2,0),(2,10)\rangle$. For an arbitrary instance of the scheduling problem with r operations there can be $r!$ number of paths between a state and its merge state. Thus, enumeration of all path is computationally intractable for even medium sized scheduling problems. Algorithm 2.5.7 constructs r paths out of the $r!$ paths between a given state and the nearest merge state. From a given state the algorithm selects the next operation by computing productivity and flexibility of possible next transitions. Based on the productivity and flexibility the next transitions are ranked. The transition with the minimum rank is select to add to the path.

Definition 2.7 (Productivity of the next transitions) Given the current state s_c, set of possible next states S from s_c the productivity $prod_i$ of the transitions from s_c to $i \in S$ equals $\frac{t_p+t_s}{m_{ps}}$ where t_p and t_s are the processing and setup times of the operation corresponding to the transition from s_c to i. m_{ps} denotes the maximum of the processing and setup times of all transitions from s_c. The productivity equals 1 if m_{ps} equals 0.

The productivity of a transition is a relative measure. It indicates how productive a transition is with respect to all possible transitions from a given state. For example, in Figure 2.6, consider the state $s(3,2,1)$. There are three possible next states $s_1(3,2,0)$, $s_2(3,1,1)$ and $s_3(2,2,1)$. Assume that the sums of the processing and the setup times for the transitions from s to s_1, s_2, s_3 are $9,0,7$ respectively. Then the productivities for the transitions are $\frac{9}{9}, \frac{0}{9}, \frac{7}{9}$ respectively. The smaller the productivity, the better.

Definition 2.8 (Flexibility of the next transitions) Given a previous state s_p, current state s_c, set of possible next states S from s_c the flexibility $flex_i$ of the transitions from s_c to $i \in S$

equals $\frac{d}{m_d}$ where d is the minimum due-date between the operations corresponding to the transitions between s_p to s_c and from s_c to i. m_d is the maximum of due-dates over all $i \in S$. Flexibility is 1 if $d = \infty$ or $m_d = 0$.

The flexibility of a transition assesses a scheduling decision such that it can be compared to other transitions indicating how much time is consumed from the due-dates. For example, in Figure 2.6, consider the state $s(3,2,1)$. There are three possible next states $s_1(3,2,0)$, $s_2(3,1,1)$ and $s_3(2,2,1)$. Assume that the due-dates for the transitions from s to s_1, s_2, s_3 are $3,6,9$ respectively. Then the flexibilities for the transitions are $\frac{3}{9}, \frac{6}{9}, \frac{9}{9}$ respectively. The smaller the flexibility, the better.

Algorithm 2.5.5: $SelectNextOperation(sls, ls, PendingOps, w_p, w_f)$

```
1  s_c ← ls
2  s_p ← sls
3  S ← PendingOps                          ▷ i.e. a set of possible next states.
4  for i ∈ S do
5  |  Compute productivity for the transition to i using s_c and S in Definition 2.7
6  |  Compute flexibility for the transition to i using s_c, s_p and S in Definition 2.8
7  |  rank[i] ← w_p × productivity + w_f × flexibility
8  end
9  s ← operation from PendingOps with minimum rank
10 return ls + s
```

From a state, the next state to visit is selected by Algorithm 2.5.5 by using the productivity and flexibility of transitions to next states. Using the measures, one of the possible next operations is selected while constructing a path through the scheduling space. Given a path with its second last state sls, the last state ls, states of possible next operations $PendingOps$ and the weights w_p, w_f , the algorithm ranks the states in $PendingOps$. Line 5 computes productivity that is a measure of how much the relative system utilization will be in case an operation is selected as the next operation. Similarly, the flexibility is computed on line 6 which indicates how much time is utilized from the due-date constraint in case an operation is selected as the next operation. The algorithm selects and returns an operation with minimum rank i.e. has minimum weighted productivity and flexibility. Different paths to order the same set of operations represent different scheduling choices. A selection out of these choices is made by computing ranks for the paths based on productivity and flexibility of paths defined as follows.

Definition 2.9 (Productivity of a path) For a path $p \in P$ its productivity is $\frac{x}{m}$ where x is the start time of the operation corresponding to the last transition in path p computed by Algorithm 2.5.6 and m is the maximum of the delay due to the paths in the set P.

Productivity of a path is a metric relative to other paths that start from the current state and reach the merge state. It is computed using the delay computed by Algorithm 2.5.6. The algorithm computes the start times of the operations as if the operations are performed in the order specified by the path. As a result the algorithm returns the start time of the operations corresponding to the last transition in the path. The result of the algorithm is used to estimate the length of the partial schedule that is due to the order specified by the path. The variable m is the maximum of the length of the partial schedules due to the individual paths in P. The normalization of x with m indicates how productive a path is compared to other paths which perform the same work.

Algorithm 2.5.6: $PathDelay(p)$

```
1 x ← S(o_1) where S is a partial schedule and o_1 is operation corresponding to transition
    s_0 → s_1 in p
2 for i = 2 to |p| do
3 |   o_1 ← operation corresponding to transition s_{i-2} → s_{i-1} in p.
4 |   o_2 ← operation corresponding to transition s_{i-1} → s_i in p.
5 |   x ← max(S(o_2), x + p(o_1) + s(o_1, o_2))
6 end
7 return x
```

Definition 2.10 (Flexibility of a transition in a path) The flexibility of the i^{th} transition in a path p is $\frac{increase}{ub-lb}$. Let $o_{x,y}$ and $o_{u,w}$ be the operations corresponding to the $(i-1)^{th}$ and the i^{th} transition in the path then $increase = max(0, \mathcal{S}(x,y) + p(x,y) + s(x,y,u,w) - \mathcal{S}(u,w))$, $lb = \mathcal{S}(x,y)$ and $ub = \max_{\forall o_{i,j} \in O} d(i,j,u,w)$.

The flexibility of the i^{th} transition in a path p is computed by estimation of how much the execution of the operation corresponding to the transition consumes from the remaining time allowed by the deadlines enforced on it. The *increase* denotes the estimated increase in the start time of the operation. The denominator is the difference between the lower and the upper bounds over the start time of operations in the transitions. In case the $increase > (ub - lb)$ then the transition leads to an infeasible scheduling decision. It is assumed that infeasible transitions have flexibility equal to ∞. Furthermore, if $ub - lb$ is 0 or ∞ then it is not possible to compute flexibility and the flexibility is assumed to be 0. For example, for the transition $(2,2,0)$ in the path $\langle (3,2,1),(3,2,0),(2,2,0)\rangle$ in Figure 2.6 the corresponding operations are $o_3 \to o_1$. Assuming $x = 3$, $S(o_3) = 10$ and $S(o_1) = 12$ in the partial schedule then the $increase = max(0, 10 + 3 - 12) = 1$. Similarly, assume that the $ub = 20$ and $lb = S(o_1) = 12$. Then the flexibility is $\frac{1}{20-12}$ indicating how much a transition consumes from the available freedom from the due-date constraints between the operations corresponding to the transition.

Definition 2.11 (Flexibility of a path) The flexibility of a path is the maximum of the flexibilities of the transitions in the path.

Selection of a path, out of possible paths, to reach the nearest merge state is performed by computing a rank for each path by Algorithm 2.5.7. On line (7), the algorithm computes the set of possible transitions from the current state to the next states. For example, for the state $(3,2,1)$ in Figure 2.6 the nearest merge state is $(2,1,0)$, then the $NextStates = \{(3,2,0),(3,1,1),(2,2,1)\}$ and $OpStates = \{(0,0,1),(0,1,0),(1,0,0)\}$. In the example $r = 3$ and the algorithm finds three paths out of 3! paths using the states in $OpStates$ and the states in $PendingOps$. The algorithm starts with a path for each operation and then selects an operation by selecting a state from $PendingOps$. The nearest merge state must be reached when all operations have been performed. The next operation on line 21 of Algorithm 2.5.7 is selected by Algorithm 2.5.5. Given the second last and the last state in a path the algorithm selects an operation by ranking them based on productivity and flexibility of transitions in the scheduling-space. The algorithm continues iteratively until all paths have reached the merged state.

A path from the initial state to the final state in the scheduling-space represents a possible partial schedule. Algorithm 2.5.8 iteratively builds such a path. The algorithm starts from the initial state of a scheduling-space and finds the next states for a given state till it reaches the final state. Furthermore, the algorithm adds edges to order the operations in a graph that represents

Algorithm 2.5.7: $PathsToNearestMergeState(Q, state, w_p, w_f, r)$

```
NearestMergeState ← NearestMergeState(state, r)
NextStates ← FindNextStates(state, r)
Paths[i] ← < state >        i ∈ 1,...,r
PendingOps[i] ← {}          i ∈ 1,...,r
OpStates ← <>
foreach s ∈ NextStates do
    Add state − s to OpStates
end
for i ← 1 to r do
    Append Paths[i][i+1] + OpStates[i] to Paths[i]
    for j ← 1 to r do
        if i ≠ j then
            Add OpStates[j] to PendingOps[i]
        end
    end
end
for i ← 1 to r do
    while The last state in Paths[i] is not NearestMergeState do
        sls ← second last state in Paths[i]
        ls ← last state in Paths[i]
        NextOp ← SelectNextOperation(sls, ls, PendingOps, w_p, w_f)
        Remove NextOp from PendingOps[i] and add it to Paths[i]
    end
end
return Paths
```

timing constraint between operations.

Algorithm 2.5.8: $InitialToFinalPath(Q, \Sigma, w_p, w_f, r)$

```
1  path ←<>
2  PrevState ← get the initial state from Q
3  CState ← the next state by FindNextStates(CurrentState, r)
4  Add PrevState and CState to path
5  Initialize the timing constraints in a graph CG
6  S ← Distances computed using Bellman-Ford on CG
7  while CState is not final state do
8  |  Paths ← PathsToNearestMergeState(Q, CState, w_p, w_f, r)
9  |  foreach p ∈ Paths do
10 |  |  prod ← Compute produtivity for p using P in Definition 2.9
11 |  |  flex ← Compute flexibility for p using S in Definition 2.11
12 |  |  rank[p] ← w_p × prod + w_f × flex
13 |  end
14 |  p ← a feasible path with minimum rank. If none then abort.
15 |  PrevState ← CState
16 |  CState ← first state in p
17 |  Add CState to path
18 |  Get operation o_c from PrevState and CState
19 |  States ← FindNextStates(PrevState, r)
20 |  foreach s ∈ States do
21 |  |  Get operation o from the transition between s and PrevState
22 |  |  Insert an edge from o to o_c in CG
23 |  end
24 |  S ← Distances computed using Bellman-Ford algorithm on CG
25 end
26 return path
```

In each iteration Algorithm 2.5.8 computes paths from the current state to the nearest merge state. For every path its rank is computed. For the ranked paths the algorithm checks for feasibility using the Bellman-Ford algorithm and by checking whether the due-date constraints hold across different machines. The algorithm aborts in case no feasible path is found. Since the algorithm is a best effort approach there might be feasible test cases for which the algorithm does not find a solution.

In the case when the algorithm finds feasible paths, the path with minimum rank is selected and the first state of the path is added to the final path. Such a selection by the algorithm represents ordering of the corresponding operation. The edges for the ordering are added to the corresponding graph with timing constraints with weight equal to the processing and the setup times between the operations. Once the edges are inserted the partial schedule $\mathcal{S}$ is updated by the longest path variant of the Bellman-Ford algorithm. The iteration of Algorithm 2.5.8 continues till the final state is reached.

Algorithm 2.5.8 is used to find ordering of operations for a single machine. In the heuristic approach described in this section the algorithm is applied on all machines in the scheduling problem one by one. Thus computing total order over each machine and the $\mathcal{S}$ is total i.e. for any pair of operations it defined whether an operation is processed before or after the other. Once operations over all machines are totally ordered a schedule is computed using the Bellman-Ford

Parameter	Value
# of Jobs (n)	1-100
# of machines (r)	1-100
Re-entrances per machine	1-10
Processing time	0-100
Setup time	0-100
# of random processing,setup and due-dates	total # of operations
α, β	{1,6,11,16,21}

Table 2.2: Specifications of the job sets used to test.

algorithm. In the next section the performance of the heuristic is experimentally evaluated.

2.6 Experimental evaluation

The aim of the evaluation is to assess the performance of the heuristic over a variety of randomly generated test-cases. In the following section the experimental setup is explained followed by the results section.

2.6.1 Experimental setup

The performance of the heuristic is evaluated using different test-cases with concerns as how fast the heuristic generates the results as well as to assess the quality of the produced results. The characteristics of the jobs and the test-cases are inspired from an *Large Scale Printer* (LSP) [8] and from the classical Taillard benchmark [30] from the literature.

Table 5.4 describes the characteristics of the cases. The number of jobs and the number of machines in a test-case is randomly generated from $[1,100]$. Similarly, re-entrances per machine is randomly selected from $[1,10]$. The number of machines and re-entrances per machine is used to compute the re-entrance vector. For each test-case the number of operations in a job equals the sum of re-entrances.

In each test-case, the processing times for randomly selected $n \times r$ operations is generated from $[0,100]$ where n is the number of jobs and r is the number of operations per job. Similarly, the setup times are generated for randomly selected pairs of operations from $[0,100]$. For the same pair, an integer dd is generated from $[0,100]$ to compute the deadline $d = (p+s) \times \alpha + dd$ or $d = (p+s) \times \beta + dd$ depending on whether the pair of operations belong to the same job or different jobs. $(p+s)$ is the sum of the processing and the setup time for the pair. The parameters α, β control the strictness of the deadline; the smaller the value the stricter a deadline is. Cases consisting of single job or single operation in a testcase are not considered because there is no scheduling freedom in the testcase. In total, there are 57825 test-cases generated in the experiments.

The experiments are performed on an Intel 3.1 Ghz CPU with a single thread running Ubuntu 14.04. For each test-case the wall clock time is reported in the experiments. Furthermore, the heuristic was allowed to only run for a maximum of 60 seconds.

The lower bounds on the makespans used in the experiments are computed for each test-case using IBM OPL modelling language (ILOG) [11] with a time-limit of maximum 60 seconds. ILOG also proved infeasibility for 52290 testcases out of the total of 57825 testcases within 60 seconds of the time limit. The OPL model of the scheduling problem only considers processing time and due-date constraints and thus provides a lower bound on the makespan of the scheduling problem which also considers the setup times. The model consists of the start times of the operations

Criterion	Minimum	Median	Maximum
makespan / lower bound	1.0	1.06	2.15
runtime (s)	0.009	0.3	58.52

Table 2.3: Results of the experiments.

as decision variables. The precedence constraints between operations and the unary machine constraints are modelled using the scheduling constructs provided by the OPL language. Including the setup time constraints makes the model prohibitively slow and thus does not provide many results.

2.6.2 Results

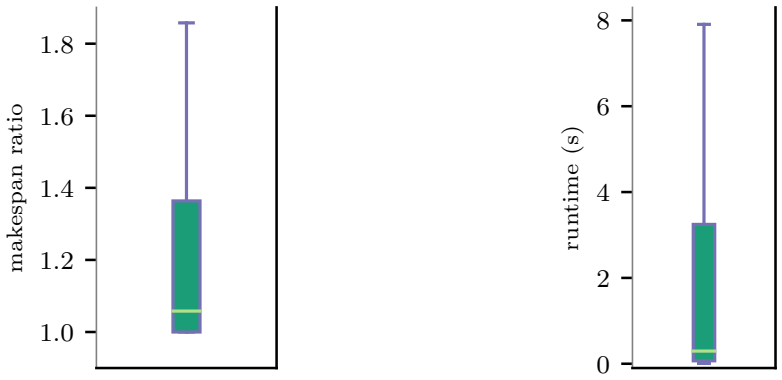
The evaluation of the heuristic is performed by considering the makespans of the schedules found and the runtime spent by the heuristic to find the schedules. The summary of the results is shown in Table 2.3. The first row describes the assessment of the quality of the schedules found by the heuristic. For each test case, the quality is the ratio between the makespan of the schedule and the estimated lower bounds. Similarly, the second row shows the amount of time per operation spent by the heuristic to find a schedule.

The minimum, median and maximum runtime of the heuristic is shown in Table 2.3 with seconds per operation as units. The statistics are from performance of the heuristic to find a schedule for each testcase. The minimum time that the heuristic spent is 0.009 seconds. The maximum amount of time the heuristic spent is 58.52 seconds per operation. The median time that the heuristic spent is 0.3 seconds. The maximum amount of time is usually spent for the testcases which are large and have strict due-dates. The statistics show the runtime of the heuristic when tested over a diverse set of testcases.

Out of 2320 schedules found by the heuristic, for 2137 cases ILOG estimated lower bounds within 60 seconds. 966 out of 2320 schedules, i.e. 41% of the schedules, found by the heuristic are optimal because their makespan equals to the lower bound for the testcase. However, for the cases for which the makespan of the schedules found do not equal to the lower bound, the optimality cannot be concluded because a schedule equal to the lower bound is not found. Such a conclusion can only be drawn when a schedule with a makespan equal to the lower bound is found.

If the heuristic found a schedule with a makespan same as the estimated lower bound then the ratio of the makespan with the lower bound is one. Even, for the case when the schedule has makespan equal to 0 (avoiding $\frac{0}{0}$) because, for some test cases, all operations may have processing and setup times equal to 0. The median, that is 1.06, indicates that the schedules are 6% longer the estimated lower bounds. The maximum ratio between the makespan of a schedule found by the heuristic and the lower bound is 2.15 times longer than the lower bound. The cases for which the lower bound was 0 but heuristic found a schedule with makespan larger than 0 (leading to a divide by 0) are not considered in the comparison.

It is not sufficient to only consider the minimum, median and makespan as metrics. Additionally, the distribution of the makespans and runtimes also provide insight. Figure 2.7a shows the distribution of makespan ratios. The figure is a *box-whisker* plot [31] in which the box shows the spread of the 50% of the ratios. The box has two whiskers one on the upper side of the box and one on the lower side of the box. The length of a whisker shows the spread of the 24.65% of the ratios. The line that splits the box into two halves marks the median (1.06) of the distribution. The ratios that are 2.698 times above the median are considered as outliers [31] as they are



(a) Makespans ratios.

(b) Runtimes.

Figure 2.7: Distribution of makespan ratios and runtimes.

statistically considered as outliers and are not shown in the figure. 41% of the schedules found were optimal and for 49% of the test cases the schedules found by the heuristic are less than or equal to 6% longer than the computed lower bounds. 25% of the schedules are between 6% and 38 % longer than the estimated bounds. 24.6% of the schedules are between 38% and 85% longer than the estimated lower bounds. 0.40% are between 85% and 2.15 times longer than the estimated bounds. These 0.40% are outliers not shown in the plot.

Figure 2.7b shows the spread of the runtime of the heuristic that was given a time limit of 60 seconds. The median runtime of the heuristic over the test-cases is 0.3 seconds. For 49% of the test-cases the heuristic takes 0.3 seconds or less. 25% of the schedules are found in between 0.3 and 3.25 seconds. 24.6% of the schedules are found in between 3.25 and 7.95 seconds per operations. 0.40% of the schedules are computed in between 7.95 and 58.52 seconds. The variation of the runtime is due to the different number of jobs in the test set. Furthermore, the number of re-entrances increase the number of operations in a job thus causing the variation in runtimes. The runtime of the scheduling methods of variants of the scheduling problems vary between 2 to 50 seconds [23] where the results presented in work were generated within maximum of 60 seconds per test case.

2.7 Conclusions and Future work

This chapter has shown that the problem of scheduling self-re-entrant flowshops with sequence-dependent setup times, relative due-dates and with fixed job order is NP-Hard. The proof of complexity consists of a reduction of the st-TSP problem to the version of the scheduling problem, called the squared version, that has the same number of jobs and operations. The squared version of the scheduling problem is a sub-class of the scheduling problem that has an arbitrary number of jobs and operations. The optimization version of the scheduling problem is then shown to be NP-Hard. On the other hand, finding an optimal schedule for classical flowshop with with fixed job order can be done in polynomial time.

A heuristic approach to schedule self-re-entrant flowshop with sequence dependent setup times, relative due-dates and fixed job order has been described in this chapter. The heuristic uses the scheduling-space to find possible scheduling alternatives. The alternatives are assessed by two metrics, namely, productivity and flexibility. Using these metrics, the heuristic finds a schedule

by constructing a path from the initial state to the final state in the scheduling-space satisfying the processing time, setup time, relative due-date and ordering constraints. The performance of the heuristic is assessed by testing over a randomly generated testset that is inspired from benchmarks from the literature and industrial testcases. The experiments show that the median increase in the heuristic schedules is 6% compared to the estimated lower bounds with median runtime of 0.3 seconds.

As a future work, computation of lower bounds is proposed. These lower bounds can be used for assessing the performance of the heuristic over the testcases for which the exact methods require intractable amount of time. One such lower bound is to use the distances computed by the Bellman-Ford algorithm on the partial schedules. However, such a lower bound is not tight as it ignores the possible setup times that may arise when operations are totally ordered. Considering the setup times in conjuction with the information from the distances from the Bellman-Ford algorithm should serve as a tighter bound than considering them separately.

As an extension to the heuristic, back tracking can be added. Back tracking might allow reversal of choices that the heuristic made after further exploration of the scheduling-space. Such an extension might be very useful for the testcases where the heuristic aborts because no feasible choice is found. The scheduling space can be extended to self-re-entrant flowshops without a fixed job order. Such an extension will make the heuristic proposed in this chapter applicable to other systems where fixed job order is not a requirement.

"Simplicity is the ultimate sophistication" -
Leonardo da Vinci

3 Specialised heuristic for LSPs

A *Large Scale Printer* (LSP) prints thousands of sheets per day at high quality for books, commercial letters etc. The total time spent to print the sheets determines the productivity of an LSP. When a print request arrives, a scheduler computes the time at which the operations for the request will be performed. Computation of the schedule should not become a bottleneck because processing of the request cannot start before a schedule is ready. This chapter describes a specialized version of the heuristic described in Chapter 2. The aim is to find out whether better quality schedules, i.e. with shorter makespans, can be found for the LSP by incorporating additional information.

The specialization simplifies the heuristic by computing the set of possible choices as *ordering tuples*. This specialization offers a simpler alternative to computing the scheduling-space thereby avoiding the need to find and rank paths in the scheduling-space. These tuples are then ranked using the metrics: *productivity*, *flexibility* and *distance*. The distance metric improves the quality of schedules in cases where other metrics have the same value. The heuristic is applicable to *2-self-re-entrant flowshops* i.e. flowshops where at least one machine processes a job exactly twice and the machines which do not process the jobs twice must only process them once. Section 3.1 introduces the path that sheets follow through an LSP. The problem of scheduling sheets in a printer is defined in Section 3.2. The related work is described in Section 3.3. The constraint graph for an LSP is defined in Section 3.4. The specialized heuristic is described in Section 3.5. The results of the experiments with the heuristic to schedule an LSP are described in Section 3.6. The specialized heuristic is compared with the general heuristic of Chapter 2 in Section 3.7. The comparison is to assess how the heuristics perform with respect to each other. This chapter is concluded and future work is described in Section 3.8.

3.1 Paper path of an LSP

A paper path in a printer is the path that sheets follow through different components. Figure 3.1 shows a paper path of a duplex printer (i.e. a printer that prints on both sides of a sheet). The arrows represent the flow of sheets and a hexagon is a component that processes sheets. A sheet enters from the Paper Input (PI) component, gets printed on its first side in the Image Transfer

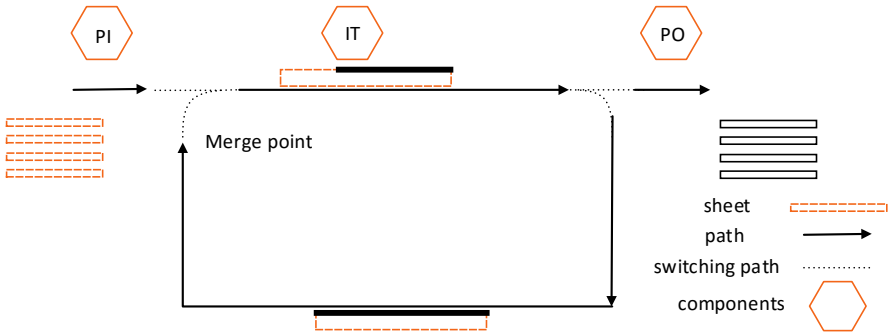


Figure 3.1: Flow of sheets through the paper path in an LSP.

(IT) component (referred to as the first pass) and returns back to get its second side printed (the second pass). After the second pass a sheet leaves through the Paper Output (PO) component. At the *merge point* the return path meets the input path and at this point a scheduling decision is made whether to first print a returning sheet or a new sheet. This decision is referred to as determining the *interleaving* of sheets. The second pass sheets re-enter the print section and are called *re-entrant sheets*. It is assumed for simplicity that all sheets are duplex sheets. Under this assumption, simplex sheets can still be printed by allowing the simplex sheets to re-enter but not print them for the second time.

The arrangement of components in a paper path is similar to arrangement of machines in *shop* scheduling [10]. The components are shared between the sheets. The IT component may need to perform additional steps, called *setups*, before printing the next sheet. For example, printing on a coated paper may require to heat up the ink. If the ink was already at the required temperature (because the previous sheet required the same temperature) then no setup is required. Thus it depends on interleaving decisions whether a setup is required or not. This conditional setup is known as *sequence dependent setup* [10]. In practice, for an industrial printer, the setup times are significantly larger (10-15 times) than the print times. The heuristic consecutively interleaves sheets with similar setup requirements to minimize the number of setups and to increase the productivity of an LSP.

The re-entrant sheets travel back to the merge point over the return path. There are bounds on the speed it can travel on the return path. A *relative due-date* is the upper bound between the first and the second passes of a sheet due to the lower bound on the travel speed. The combination of re-entrance, sequence dependent setup times and relative due-dates define the specific scheduling problem.

3.2 Problem definition

An LSP consists of components that perform operations on sheets. The operations are: input a sheet, print a sheet and finish a sheet. An LSP is a special case of re-entrant flowshops [10, 32] as shown in Figure 3.2.

Recall from Section 2.2 that a self-re-entrant flowshop consists of a set $\mathcal{M} = \{\mu_1, \ldots, \mu_i, \ldots, \mu_m\}$ of machines processing a jobset $\mathcal{J} = \{j_1, \ldots, j_n\}$. The circles in a job shown in Figure 3.2(a) are operations. The machines process the jobs in an order described in a *flow vector* $\mathcal{V} = [v_1, \ldots, v_r]$ with $v_i \in \{1, \ldots, m\}$ in which the processing of every job starts from the first machine, i.e. $v_1 = 1$ and ends at the last one, i.e. $v_r = m$. The flow vector for the LSP is [1,2,2,3]. In a self-re-entrant flowshop, a job is either re-processed by the same machine or passed on to the next machine. It

(a) 3 jobs having 4 operations each. (b) 3 machines.

Figure 3.2: The jobs (a) and the machines (b) in an LSP.

means that in the flow vector $\mathcal{V}$, v_{i+1} either equals v_i or is the index v_i+1 of the next machine. Thus the flowshop operates r times on a job with the set $\mathcal{O}_i=\{o_{i,1},\ldots,o_{i,r}\}$ denoting the set of operations of a job j_i and r is a positive integer that denotes the number of operations performed on each job. For the LSP every job has 4 operations i.e., $r=4$. The set $\mathcal{O}_\mathcal{J}=\bigcup_{j_i\in\mathcal{J}}\mathcal{O}_i$ contains the operations of all jobs in the jobset $\mathcal{J}$. In Figure 3.2(a) a job is a sheet in the LSP. The jobs in the flowshop have a *fixed job order* where the processing of jobs start from j_1 and ends with j_n i.e. $j_1\rightarrow j_2\rightarrow\cdots\rightarrow j_n$. For each machine, the operations from different jobs are required to be ordered as the machine is a *unary resource*. The LSP is an example of a 2-self-re-entrant system.

The y^{th} operation of a job j_x is $o_{x,y}$ that takes $p(x,y)$ time units to execute. A machine can at most perform one operation at a time and preemptions are not allowed. A machine may require additional time, called *sequence dependent setup time*, to prepare for the processing of the next operation. The sequence dependent setup time between an operation $o_{x,y}$ and an immediately following operation $o_{u,w}$ is $s(x,y,u,w)$. Similarly, a *relative due-date* $d(x,y,u,w)$, is the maximum allowed amount of time difference between the start of the processing of an operation $o_{x,y}$ and the start of the processing of an operation $o_{u,w}$. Operations are non-preemptive. A machine is available immediately after completion of an operation. For an operation $o_{x,y}$ its start time is $\mathcal{S}(x,y)$ where $\mathcal{S}$ is a schedule. Following is an example of a re-entrant flowshop with a single machine and 3 jobs.

The LSP is a restricted case of the class of self-re-entrant flowshops. Every job has four operations mapped on three machines. The first and the fourth operation are mapped on the first and the third machine. The second and the third operations are mapped on the second machine. There are no relative due-dates between operations of two jobs. The due-date between the first and the second operation of every job is same as the sum of the processing and the setup time between the operations. Similarly, the due-date between the third and the fourth operation of every job is same as the sum of the processing and the setup time between the operations. The due-dates between the second and the third operations are larger than the sum of the processing and the setup times between the operations. They are larger because of the lower bounded travel speed for the third operation. Given these constraints, makespan of a schedule is used to asses how well a schedule performs. Makespan is defined as follows.

Definition 3.1 The ***makespan***, $\mathcal{C}_\mathcal{J}$, of a jobset $\mathcal{J}$ is the start time of an operation that is performed the last in a schedule $\mathcal{S}$ computed as $\mathcal{C}_\mathcal{J}=\max_{o_{x,y}\in O_\mathcal{J}}\mathcal{S}(x,y)$.

The challenge is to find a schedule $\mathcal{S}$ such that the makespan $\mathcal{C}_\mathcal{J}$ of the schedule $\mathcal{S}$ is minimal. It suffices to consider start times, instead of the completion times, because the operation $o_{n,r}$ is the last operation to finish in a schedule due to the fixed job and operation order. Furthermore, a valid schedule always exists for any LSP and jobset. An intuitive argument is that a single job always has a schedule and that a schedule for a jobset can always be formed by considering each job separately without interleaving because the relative due-dates only exist between the operations of the same job and the timing constraints between the operations of a job are assumed to be always feasible. Thus, a non-interleaved schedule will always be feasible and therefore a valid schedule always exists. Although a non-interleaved schedule always exists, it will not be as

productive as a schedule with interleaving because, in the considered LSP, travelling back on the return path typically takes more time than interleaved printing.

3.3 Related work

Johnson provides in [33] a polynomial time algorithm to schedule flowshops with two machines and showed that scheduling more than two machines is NP-Complete. The CDS heuristic in [34] uses Johnson's polynomial time algorithm to find schedules for flowshops with more than two machines. However, Johnson and the CDS heuristic do not consider re-entrance, setup times and relative due-dates. *Branch and bound* based approaches presented in [35, 36] find optimal schedules for the flowshop scheduling problem. Similarly *mixed integer programs* are used by [37, 38] to solve different variants of flowshop scheduling problem to optimality but are not suitable for scheduling at runtime.

Several heuristics to solve flowshop scheduling problems faster are proposed. For example, by using simulated annealing [39] or genetic algorithms [40]. However, because of the due-dates in the LSP scheduling problem, it is not guaranteed that these algorithms will find a schedule. Other methods to find the schedules faster use a ranking function (as the heuristic presented in this paper). For example, *slope index* based ranking in [41], *rapid access* algorithm in [42] or the NEH heuristic in [43]. These methods rank the jobs to find the near optimal job orderings and do not focus on combination of re-entrance, setup times and due-dates. Less attention has been given to re-entrant flowshops. A branch and bound algorithm is used in [44] to schedule a re-entrant flowshop, but it is not suitable for scheduling at runtime. The heuristic described in this chapter is compared with the *Modified Nawaz Enscore and Ham* (MNEH) heuristic of [32] particularly for the LSP flow shop instances. The heuristic outperforms MNEH because it considers partial schedules that favor rescheduling if better for the later stages of iteration.

3.4 Constraint graph model for an LSP

Recall, from Section 2.5.1, that a constraint graph is a tuple $cg(V, E = E_C \cup E_O, w, G, m, v_s, v_d)$. The set V is a set of vertices that are the operations performed by machines over each job. For instance, c_1 and d_1 in Figure 3.3 are examples of vertices in a constraint graph for three jobs j_1, j_2, j_3 having four operations a_i, b_i, c_i, d_i each on a self-re-entrant flowshop with flow vector $\mathcal{V} = [1,2,2,3]$. The set $E \subseteq V \times V$ is a set of directed edges. An edge $(c_1, d_1) \in E$ with weight $w(c_1, d_1) = 6$ denotes a timing constraint between operations c_1 and d_1. Let t_{c_1} and t_{d_1} be the start times of c_1 and d_1 respectively then the timing constraint modelled by the edge is $t_{d_1} \geq t_{c_1} + w(c_1, d_1)$. The timing constraints due to the processing and the setup time requirements have a non-negative weight. The timing constraints due to the relative due-dates have negative weights. For example, in Figure 3.3, the edge (c_1, b_1) has weight -12 with the constraint $t_{b_1} \geq t_{c_1} - 12$. The function $w : E \to \mathbb{Z}$ maps each edge in cg to its weight.

The set E of edges is partitioned into two sets E_C and E_O. The set E_C consists of *compulsory constraints* that must not be violated. The compulsory constraints are due to the relative due-dates, the order of operations due to the flowshop and due to the fixed job order. The constraint graph with only its compulsory edges is called *compulsory constraint graph* and consists of solid and dashed edges. The set E_O contains *optional constraints* that can be added to E_C to enforce an order between two unordered operations. In Figure 3.3 the optional constraints are denoted as dotted edges. For example, the edges (c_1, b_2) and (b_2, c_1) are optional edges. A scheduler can add (c_1, b_2) to the set of compulsory edges when b_2 is processed immediately after c_1. The weights on the optional edges denote the timing constraints that must be enforced if an operation immediately follows the other. The function $G : V \to \{1, \ldots, m\}$ determines the group of a vertex.

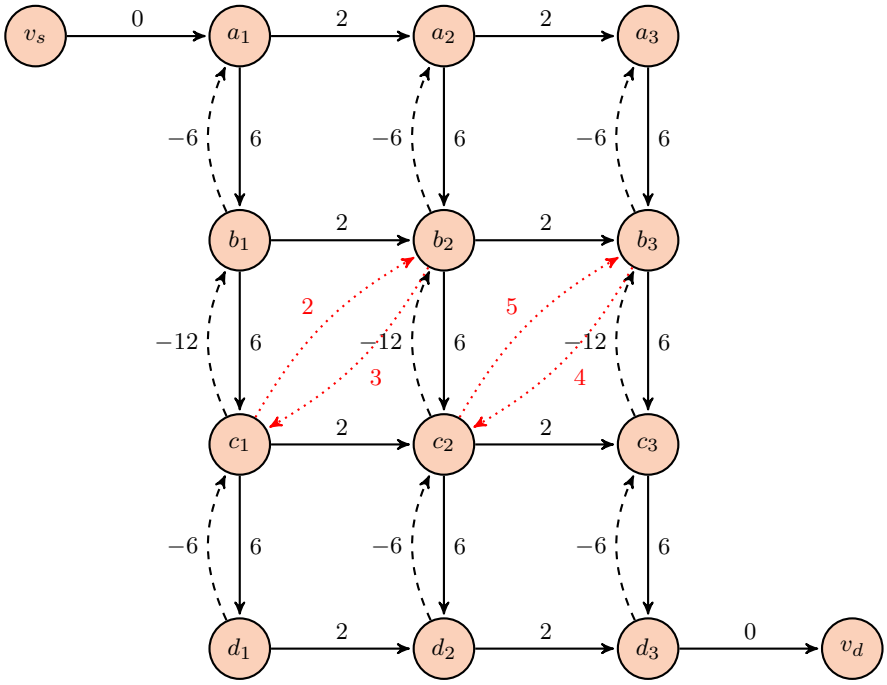


Figure 3.3: An example of a constraint graph for an LSP.

Operations that are processed at the same machine have the same group. The optional edges are only between vertices that belong to the same group. The source vertex v_s is a vertex with no incoming edge and has one outgoing edge with weight 0 to the vertex corresponding to the first operation of the first job. The destination vertex v_d is a vertex with no outgoing edge and has one incoming edge with weight 0 from the vertex corresponding to the last operation of the last job.

The example in Figure 3.3 has four operations for each of the three jobs. The columns show the operations of a job. Operations a_i belong to the first group, b_i and c_i to the second group and d_i to the third group. For each group it needs to be decided by a scheduler when to perform an operation. The time instant at which an operation starts is called the *occurrence time* of the operation. It is defined as follows.

Definition 3.2 A *occurrence time* of vertex $a \in V$ is $t_a \in \mathbb{N}$ such that all compulsory constraints hold. Schedule $\mathcal{S}$ has for each operation its occurrence time.

A schedule $\mathcal{S}$ for a constraint graph is feasible if and only if all compulsory constraints hold on the schedule. A schedule can be computed by finding the longest path between the source vertex and the other vertices by only using the compulsory edges. The longest path variant of the Bellman-Ford algorithm can be used to find such schedules as specified in Section 2.5.1. Operations may require resources when their processing starts. For example, printing a sheet requires that the IT component is free. Therefore operations in the same group must be *ordered* and the constraints must ensure that the resources are used exclusively. The vertices are ordered by enforcing edges from the set of optional edges E_O. The set E_O is assumed to contain the constraints for all possible orderings.

Definition 3.3 A *path* p in a constraint graph is a non-empty sequence of connected edges $(a,b) \to \cdots \to (c,d)$ from the set E. The length l_p of path p is the sum of the weights of the edges in p i.e., $l_p = w(a,b) + \cdots + w(c,d)$. The path starts from vertex a and ends at vertex d. A *cycle* is a path starting and ending at the same vertex.

Definition 3.4 A *non-negative* path p is a path consisting of only edges with non-negative weights i.e., for any edge (a,b) in p its weight $w(a,b) \geq 0$.

In Figure 3.3 $(v_s,a_1) \to (a_1,b_1) \to (b_1,b_2)$ is a non-negative path with length 8. The path $(b_1,b_2) \to (b_2,c_1) \to (c_1,b_1)$ is a cycle with length $2+3-12 = -7$. Cycles in the compulsory constraint graph with length greater than 0 cannot be satisfied by schedules and are thus prohibited.

Definition 3.5 A vertex a *precedes* vertex b if there exists a non-negative path of compulsory edges from E_C starting from a and ending at b. A vertex a *immediately precedes* vertex b if there exists an edge $(a,b) \in E_C$ with non-negative weight $w(a,b) \geq 0$.

Choices in interleaving of jobs are different possibilities to order a set of vertices in the same group. They represent in what order a unary machine can perform a set of operations. In Chapter 2, these choices were presented as a transition between two states. In the specialized heuristic described in this chapter, a choice is represented as an ordering tuple defined as follows.

Definition 3.6 A tuple $ot((a,b),(b,c))$ of edges (a,b) and (b,c) from E is called an *ordering tuple* for vertex b if a immediately precedes c. An ordering tuple is *feasible* if and only if adding the edges in the tuple to E_C does not introduce a cycle with positive length in the compulsory constraint graph.

It is assumed that E contains all edges for an ordering tuple for all possible interleavings. The set of ordering tuples for vertices in the groups is defined as follows.

Definition 3.7 The set $OT_b^x = \{((a,b),(b,c)) \mid a,c \in V_x\}$ is the set of all feasible ordering tuples for vertex b in its group x. $OT^x = \bigcup_{b \in V_x} OT_b^x$ is the set of all orderings of all vertices in V_x. $OT = \bigcup_{x \in \{1,\ldots,m\}} OT^x$ is the set of all ordering tuples for all groups.

Once the set of all feasible ordering tuples is known, the question is how to find the set of ordering tuples that lead to an optimal schedule. This problem is called the *vertex ordering problem* described in the following problem statement.

Definition 3.8 *(Vertex Ordering Problem)* Find a feasible set of ordering tuples $OT^{sol} \subseteq OT$ such that vertices in all vertex groups are totally ordered i.e., For every x in $\{1,\ldots,m\} : (\preceq, V_x)$ is a total order and the time of occurrence t_{v_d} of the destination vertex v_d is minimal.

Once the vertex ordering problem is solved, a schedule can be computed using the longest path variant of the Bellman-Ford algorithm. The heuristic proposed in the following section uses the constraint graph to find schedules for an LSP.

3.5 Proposed heuristic approach

A schedule for a re-entrant flowshop is the occurrence times of all totally ordered vertices in all groups. The makespan of a schedule depends on the selection of ordering tuples. Three metrics are described, namely, *productivity*, *flexibility* and *distance* to assess the impact of an ordering tuple on the constraints and on the makespan. Using these metrics, the heuristic ranks and selects ordering tuples to compute a schedule.

Productivity quantifies the impact of an ordering tuple on the makespan of a schedule. To compute productivity, Equation 3.1 first computes $d_{x,b}$, which is the maximum possible increase in time of vertex c that can result from any ordering tuple $ot((a,b),(b,c))$ in the set OT_b^x.

$$d_{x,b} = \max_{ot \in OT_b^x} \max(t_c, \max(t_b, t_a + w(a,b)) + w(b,c)) \tag{3.1}$$

Intuitively, productivity quantifies how restrictive the constraints in the ordering tuple are compared to all possible ordering tuples for a vertex. The ordering tuples that require setups have larger weights $w(a,b)$ and $w(b,c)$ than the ordering tuples not requiring a setup. The productivity of an ordering tuple $ot((a,b),(b,c))$ in a set OT_b^x for vertex b is computed in Equation 3.2 where $P_{ot} = 0$ indicates a maximally productive ordering tuple..

$$P_{ot} = \frac{\max(t_c, \max(t_b, t_a + w(a,b)) + w(b,c))}{d_{x,b}} \tag{3.2}$$

An ordering tuple enforces new constraints consuming the time between the due-date and the start time of a vertex. Flexibility of an ordering tuple $ot((a,b),(b,c))$ quantifies the effect of the edges over the due-date constraints between vertex b and other vertices. Let D_b contain all edges with negative weight (due-dates) originating from vertex b, and let x be the vertex to which the due-date is the smallest out of all dates in D_b, then Equation 3.3 computes the flexibility of the tuple ot.

$$F_{ot} = \frac{max(0, t_a - t_b + w(a,b))}{-w(b,x) - w(x,b)} \tag{3.3}$$

Equation 3.3 denotes the ratio between the excess amount of time used by an ordering tuple and the total allowed time by the due-date constraint. t_b and t_a are the occurrence times of vertices b and a computed from the partial solution (by considering the edges in E_C and the set OT^{sol}). The less the excess usage the more operations might be feasible later on due to relatively more scheduling freedom. $F_{ot} = 0$ indicates the most flexible ordering tuple.

Let $dist_c$ be the least number of edges required on a path from v_s to vertex c. The distance for the ordering tuple $ot((a,b),(b,c))$ denotes length of the interleaving and is computed as follows.

$$DT_{ot} = \frac{|V| - dist_c}{|V|} \tag{3.4}$$

The distance metric DT_{ot} is additional to the metrics described in Chapter 2. For 2-self-re-entrant flowshops, distance improves the utilization of a machine for the cases where productivity and flexibility of two ordering tuples is same. For such cases, the improvement is because the ordering tuple with more sheets on the return loop are preferred. Thus, the ordering tuple represent a choice with higher machine utilization.

Using the three metrics a rank (lower is better) is computed for ordering tuples in Equation 3.5.

$$R_{ot} = \kappa_P \times P_{ot} + \kappa_F \times F_{ot} + \kappa_{DT} \times DT_{ot} \tag{3.5}$$

The relative weights (κ_P,κ_F,κ_{DT}) sum up to 1 and indicate the relative importance of the metrics when determining the ordering of events.

Algorithm 3.5.1: A heuristic to determine OT^{sol} while minimizing t_{v_d}.

$OT^{sol} = \phi$
compute vertex groups V_x for $x \in \{1,\ldots,m\}$ using G
$T = \phi$
for *each* $x \in \{1,\ldots,m\}$ **do**
  while $\exists b,c \in V_x$ *not ordered in* V_x **do**
    compute and update T with occurrence
      times of $v \in V_x$
    compute set of ordering tuples OT_b^x
    compute rank R_{ot} for all $ot \in OT_b^x$
    select a feasible ot with minimum R_{ot}
    add the edges (a,b) and (b,c) from ot to E_C
    add the tuple ot to OT^{sol}
  end
end
return OT^{sol}

The proposed heuristic approach in Algorithm 3.5.1 ranks and picks different ordering tuples to minimize the occurrence time of the destination vertex v_d. The selected ordering tuples are in the set OT^{sol} and denote a feasible schedule. The algorithm starts by computing the vertex groups V_x using the grouping function G. The heuristic iterates for each vertex group and finds a pair of vertices a,b that are not ordered. To order the pair it computes the feasible ordering tuples, ranks them and selects the tuple with minimum rank. The edges in the selected tuple are added to E_C and the tuple is added to OT^{sol}. The algorithm terminates and returns OT^{sol} which is a feasible schedule. The Bellman-Ford algorithm has complexity $O(|V|^3)$ and is used by the heuristic to find whether a schedule is feasible or not. Furthermore, per group, the heuristic iterates with complexity $O(|V|^2)$. Hence, the complexity of the proposed heuristic is $O(m|V|^5)$ where m is the number of machines). The heuristic is evaluated on testcases from the operation of an LSP in Section 3.6.

3.6 Experimental results

The experiments aim to assess the quality of the heuristic by comparing schedules generated by the scheduler used in *Cannon VarioPrint i300* (referred to as the *Eager scheduler*), the Mixed Integer Programming (MIP) scheduler, the modified NEH heuristic from [32] (MNEH) with restricted iterations for use at runtime and estimated lower bounds on the optimal makespan. The experimental setup is first described followed by the details of schedulers, the details of the testset and the experimental results.

3.6.1 Experimental setup

The experiments are performed on two platforms (because of restrictions of the software to different platforms with node locked licenses). The lower bounds, our heuristic scheduler and the MIP were solved using an Intel Core i7 CPU running at 2.67GHz with Ubuntu 10.10. The Eager scheduler and the MNEH [32] were tested on a Intel Core i7 CPU running 3.1GHz with Windows 7. The Eager scheduler is proprietary software and is only available on Windows. However, the differences of the platforms have no effect on the makespans of the schedulers (the main aim of the comparison). The MIP is solved using CPLEX 12.6, which also provided the lower bounds.

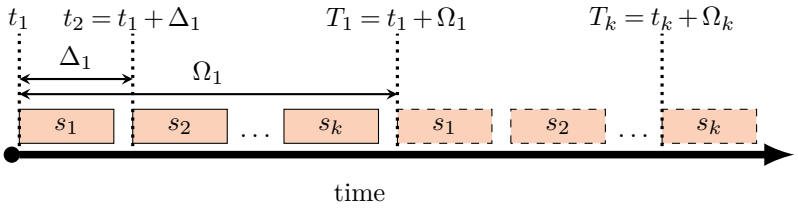


Figure 3.4: The operation of the Eager scheduler.

CPLEX was run for at most 300 seconds per testcase.

3.6.2 Details of schedulers, heuristics and lower bounds

The Eager scheduler currently in use the LSP is a First Fit (greedy) scheduler. Figure 3.4 shows the computation of a schedule for sheets $s_1, \ldots, s_k$. Let t_1 be the time when sheet s_1 is at the merge point for the first time. The eager scheduler assigns $T_1 = t_1 + \Omega_1$ as the time instant when sheet s_1 will be at the merge point for the second time after travelling on the return path for Ω_1 time units. The time Ω_1 is chosen based on the fastest speed s_1 can travel back to the merge point. The first pass of the next sheet, s_2, can be printed Δ_1 time units later than t_1. Δ_1 is the sum of the processing time of s_1 and the setup time between s_1 and s_2. Similarly, the start times, t_i and T_i for $1 \leq i \leq k$ are computed. If the first or the second pass of a sheet cannot be interleaved then the sheet is scheduled at the end of the partial schedule. The simplicity of the Eager scheduling motivates its use in practice.

The MIPs are solved with CPLEX [45] to compare the heuristic to optimal solutions. However, in many cases, CPLEX fails to find a feasible solution within the time bound. The Lagrangian relaxation of the MIP program is less compute intensive and provides a lower bound (LB) on the makespan of the optimal schedule. Note that the lower bounds do not necessarily have corresponding schedules that achieve a makespan of the lower bound.

The heuristic described in this chapter is also compared to the state-of-the-art greedy heuristic in literature, for which the MNEH heuristic of [32] is used to generate schedules in only a single iteration. The MNEH heuristic performs a backward iteration over the given initial seed sequence (i.e. a schedule without interleaving). The original MNEH continues to iterate over the (partial) schedule found till no further improvement is possible. In contrast, the heuristic proposed in this chapter only performs single backward iteration. For fair comparison, the MNEH heuristic was also run in a greedy fashion, i.e., restricted to single backward iteration. In the iteration, the MNEH heuristic starts from the last sheet in the seed sequence and selects the ordering tuple that results in minimal makespan for the partial schedule. The heuristic continues to find ordering tuples in backward direction.

3.6.3 Test set

The experiments test the heuristic on a set of 701 schedule requests varying in size between 18 and 800 sheets. Sheets in a schedule request can be one of 19 different types. Examples are sheets of different thickness, length etc. The types of sheets with patterns are shown in Table 3.1. These patterns are representatives of typical schedule requests for the LSP. Each pattern is a type of schedule request with a specific pattern of sheets. The patterns are shown as regular expressions with literals $a - s$ where each literal represents a different type of sheet.

The testset consists of 5 categories of patterns. The first category *Repeating A* (RA) consists of cases with *repeating patterns* of sheets in which an a type sheet is followed by several b type sheets. In category *Repeating B* (RB) a testcase is one, two or three a type sheets followed by

Category	*Pattern*
RA	$(a(b)^+)^+$
RB	$((a \mid aa \mid aaa)(b)^+)^+$
BA	$((c^{10} \mid c^{20}) \mid (d^{10} \mid d^{20}) \mid \cdots \mid (g^{10} \mid g^{20}))^5$
BB	$((h^{10} \mid h^{20}) \mid (i^{10} \mid i^{20}))$
H	$j^+ \mid k^+ \mid \cdots \mid s^+$

Table 3.1: Categories and patterns in the testset.

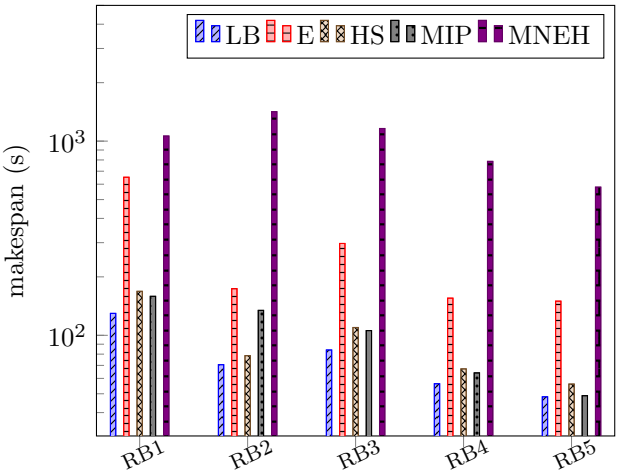


Figure 3.5: Comparison of different schedulers for the RB category.

several b type sheets. In category *block A* (BA) sheets are grouped in 5 blocks with each block having 10 or 20 sheets of type $c-g$. In category *block B* a testcase has 5 blocks of sheets. Each block consists of 10 or 20 sheets of type h or i. In the *Homogeneous* (H) category a testcase consists of all sheets of the same type. The different type of sheets when processed one after another require setups and have different due-dates.

3.6.4 Results

The number of setups encountered can be reduced by interleaving (recall that setups take significantly more time compared to print time). 4 out of 5 categories in the testset have different types of sheets potentially leading to setups. The heuristic uses the ranking function with weights $\kappa_P = 0.8$, $\kappa_F = 0.15$, $\kappa_{DT} = 0.05$. The weights have been determined by experimentation. Flexibility ensures that the heuristic generates partial schedules that prefer later on scheduling of remaining operations over higher performance. Overall, the heuristic generates schedules that have an average difference of 25% from the lower bound. The performance of the heuristic is relatively best for category RB.

Figure 3.5 shows the comparison of the makespans for 5 (randomly selected) testcases from RB shown on the x-axis. The makespan of schedules generated by the schedulers (Eager = E, our heuristic = HS, lower bound = LB) in logarithmic scale is shown on the y-axis. The Eager scheduler does not interleave the sheets based on their types but based on a fixed delay per sheet. The fixed delay does not necessarily minimize the number of setups. Similarly, the MNEH heuristic generates partial schedules that are not flexible resulting in schedules with larger

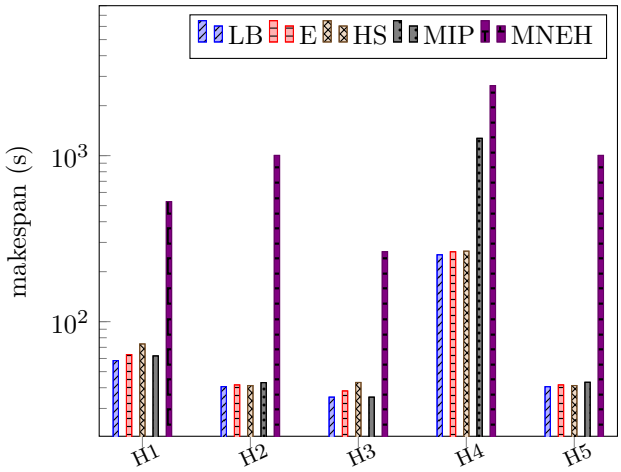


Figure 3.6: Comparison of different schedulers for the H category.

makespan. The heuristic, when computing productivity, aims to minimize the number of setups. In 56% of the cases our heuristic generates schedules with the same or better makespans as the MIP but in much less time (MIP was running for at most 300 seconds).

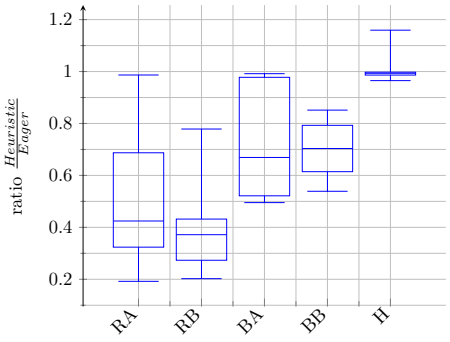
Figure 3.6 compares the makespans for category H consisting of 5 (randomly selected) test cases shown on the x-axis and the makespan shown on the y-axis. Due to the similarity of sheets in a job there are no setups required and thus flexible (partial) schedules are not required.

Figure 3.7a compares the heuristic directly to the Eager scheduler. The categories are shown on the x-axis and the y-axis shows the ratio of heuristic makespans to the Eager makespan. For each category, the rectangles show the region where the ratio of makespans of 50% of the test cases in the category fall. The bottom end of the line is the minimum and the top end of the line is the maximum of the ratios observed in the problem set. The line splitting a rectangle is the median of the ratios. A ratio less than 1 indicates that the heuristic performs better. The heuristic out-performs the Eager scheduler in 4 out of 5 categories (i.e. 92% of total test cases).

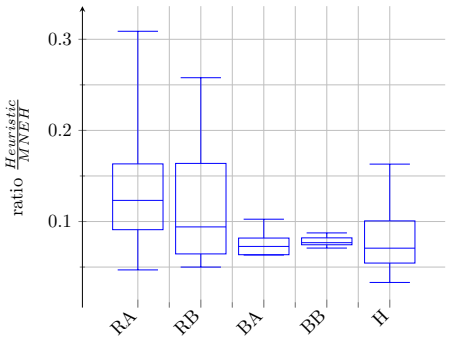
Similarly, Figure 3.7b compares the makespan ratios of our heuristic with the MNEH heuristic. Our heuristic out-performs the MNEH heuristic in all categories because the MNEH heuristic greedily focuses on maximum productivity and not flexibility. Figure 3.7c compares the ratios of our heuristic makespans to the lower bounds. The comparison indicates that the heuristic may have room for improvement for the RA category. Recall, however, that the lower bounds are estimates and might not be tight. The heuristic, to calculate schedules, requires on average 0.3 seconds per sheet and in worst case (for largest schedule request) 6.95 seconds per sheet. On average, over all categories, the proposed heuristic generates schedules that are 40% shorter than the Eager scheduler, have an average difference of 25% compared to the estimated lower bound and generates schedules with less than 67% of the makespan of schedules generated by the MNEH heuristic. The MNEH heuristic requires on average 0.01 seconds per sheet and is faster than the proposed heuristic trading off runtime over higher quality schedules.

3.7 Comparison of the specialized heuristic with the general heuristic

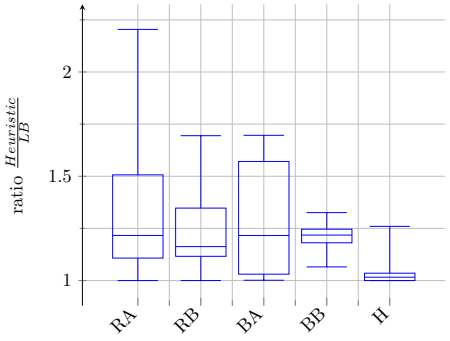
It is of interest to compare the performance of the specialized heuristic with the general heuristic of Chapter 2. How do they compare considering the quality of the schedules and the time that it took to find them. The question arises because the general heuristic can also solve the testset



(a) Eager scheduler.



(b) MNEH heuristic.



(c) Lower bounds.

Figure 3.7: Comparison of different schedulers with the heuristic proposed in this chapter.

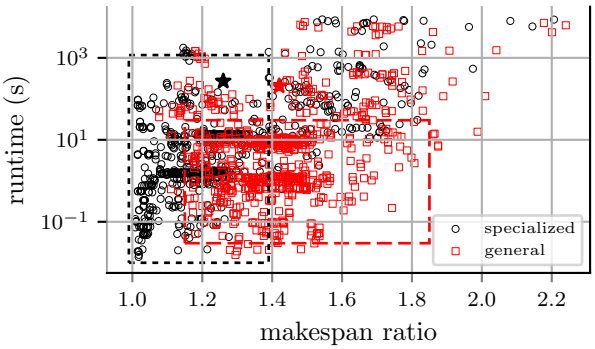


Figure 3.8: The comparison of the general heuristic with the specialized heuristic.

described in Section 3.6.3. To compare the heuristics, we find schedules for each of the testcases using both heuristics and evaluate how they perform.

Figure 3.8 shows the results of the comparison. The makespan of the schedules found by each of the heuristic are normalized by the lower bounds computed using CPLEX. These normalized makespans are shown in the x-axis. The absolute runtime is shown on the y-axis in logarithmic scale. For each testcase, the schedules found using the specialized heuristic are shown as a circle. The schedules found using the general heuristic are shown as boxes. The stars mark the average makespan ratios and average runtime for each of the heuristics. The average makespan ratio is 1.26 and 1.42 for the specialized and the general heuristic respectively. The average runtime is 269.48 seconds and 205.69 seconds for the specialized and the general heuristic respectively. On average, the specialized heuristic finds shorter schedules but takes more time compared to the general heuristic.

It is not sufficient to only consider average behaviour of the heuristics. There is clearly a larger concentration of schedules towards the lower left corner of the figure. The dotted and dashed rectangles show the concentration of the testcases for the specialized and the general heuristic respectively. The center of the concentrations are around the medians. The median makespan ratios are 1.21 and 1.41 for the specialized and the general heuristic respectively. The median runtimes are 8.24 seconds and 5.86 seconds for the specialized and the general heuristic respectively. As a trend in the schedules within concentrations, the specialized heuristic finds shorter schedules but also takes more time. This trend is indicated by the height and the width of the bounding rectangles for schedules generated by both heuristics.

Additional to the trends outlined by the concentrations, there are schedules, for both heuristics, which take more runtime and, compared to others, have longer makespans. For these cases, as indicated by the increased runtime, the heuristics have a larger solution space to explore. Moreover, as the lower bounds are not exact, having higher makespan ratio does not always indicate that the heuristics performed poorly. There might be no schedules for those testcases that is as small as the lower bound. Recall that the lower bounds are computed using the Lagrangian relaxation for the mixed integer program for the scheduling problem. Furthermore, many jobsets in the testset consist of more jobs than others. The relaxation for such jobsets does not consider the setups that may arise when a schedule is computed. In that case, the lower bound does not consider the setups and will be significantly smaller than an exact lower bound which is computed from an optimal schedule. This may in turn result in a larger makespan ratios than reality.

The way the heuristics work provides an explanation to their respective trends. Recall from

Chapter 2 that the general heuristic creates a scheduling-space to explore and find solutions. In a solution space, at any state, the maximum number of choices is equal to the number of operations mapped on the machine for which ordering of the operations is being explored. For the testset of this chapter, the testcases represent jobsets on an industrial printer. In which, there are two operations mapped on the IT component. Therefore, at any state, there are at most two choices from which the general heuristic selects one. The selection is made, for the cases in the testset, in a binary fashion; from two choices, pick one. That is why, on average, it is quicker but finds longer schedules. Recall that the specialized heuristic, for each sheet, out of all possible ordering tuples finds the locally best one. The number of tuples, for each sheet, are equal to one plus the number of sheets in a jobset. Therefore, the specialized heuristic, has a larger choice set from which one interleaving is made. The relatively high runtime and better quality of the specialized heuristic is the direct consequence of the larger choice set per interleaving decision. Both heuristics are useful in different circumstances; specialized when higher performance is required and general when slightly longer schedules are acceptable that are computed faster than the specialized heuristic.

3.8 Conclusions and Future work

A 2-self-re-entrant flowshop scheduling problem is presented with a heuristic to generate schedules for it. The heuristic is a greedy strategy which ranks local scheduling decisions and enforces the choice that ranks the best. The ranking is performed using three metrics, productivity, flexibility and distance. The performance of the heuristic is evaluated by experiments on a testset consisting of schedule requests of industrial relevance. The experiments show that the heuristic outperforms a greedy version of the state-of-the-art MNEH heuristic in all cases. Moreover, the heuristic out-performs the Eager scheduler (for 92% of the cases). Compared to the general heuristic of Chapter 2, the specialized heuristic explores a larger choice set and thus, on average, finds better solutions. The general heuristic, on the other hand, finds schedules, on average, faster than the specialized heuristic. When scheduling quality is of high importance, it is proposed to use the specialized heuristic. When finding schedules quickly is a requirement then the general heuristic is a better choice.

Recently, the heuristic proposed in this chapter was extended by [46] for online scheduling. The work schedules a part of a constraint graph and later on schedules the rest. In this way, the runtime of the heuristic is less than scheduling the complete constraint graph. In [47] with several runtime optimizations the heuristic has the worst case observed runtime of 325ms. Furthermore, [47] has extended the heuristic to cases where a job request comprises of simplex and duplex sheets. Due to this extension, jobs either have three or four operations. As a future work, generation of timing constraints from an LSP at runtime is proposed. At the moment, in this chapter and in the work of [46, 47], it is assumed that the timing constraints are given. However, for each type of job, the timing constraints need to be acquired from different components over the paper path of an LSP.

The computational complexity of optimal scheduling of an LSP is unknown. The reduction specified in Chapter 2 is not applicable to the LSP scheduling problem. Further research is required to find out whether this case, where one machine processes jobs twice, has an efficient algorithm to find optimal schedules or not. A starting point might be to reduce the set partition problem to the LSP scheduling problem.

Tighter lower bound computation is proposed as a future work. Such a bound is useful in comparing the quality of the schedules generated by the heuristic. One such lower bound can be computed using the following argument. There will be at least $x-1$ number of setups if there are x different sheets in a jobset. Thus, considering these setups together with the information from the timing constraints will result in a tighter bound making it possible to better assess the

quality of schedules generated by the heuristic.

"It is in your moments of decision that your destiny is shaped" - Tony Robbins

4 Performance estimation

The structure of a self-re-entrant system influences its performance. To get better performance while reducing costs, a fast performance estimation method can be used during the design phase to explore the trade-offs between the structure and the performance. A performance estimator is described in this chapter that uses the information in the jobs being processed to analyse the trade-offs. The estimator assumes that the jobsets consist of repeating patterns of jobs. The timing constraints of the jobs in the pattern are used to estimate the performance of a given system. Using this estimator, the impact of the structural changes for a *Large Scale Printer* (LSP) and a research platform are studied.

A performance estimation method for self-re-entrant flowshops with sequence dependent setup times, relative due-dates and fixed job order is described in this chapter. The structure of self-re-entrant flowshops is introduced in Section 4.1. The estimation problem is formally defined in Section 4.2. Section 4.3 compares the estimator with approaches from literature. Section 4.4 describes the estimator and shows how the estimator can be used to perform *Design Space Exploration* (DSE). A case study performed on an LSP is described in Section 4.6. The study demonstrates how the performance estimator can be used for industrial scale systems. Furthermore, the case study in Section 4.7 over the *eXplore Cyber Physical Systems* (xCPS) platform, shows how the estimator can be used to analyse systems with arbitrary but fixed number of re-entrances. This chapter is concluded and future work is described in Section 4.8.

4.1 Structure and operation of self-re-entrant flowshops

Performance estimation of a self-re-entrant flowshop is crucial during its design. Systems like [5, 7, 48] consist of several machines whose timing constraints are modelled as flowshops. Figure 4.1 shows a typical arrangement of a self re-entrant flowshop consisting of re-entrant machines (shown as boxes) that process jobs. The arrows indicate the flow of jobs through the machines. In a self-re-entrant flowshop, jobs are only allowed to re-enter the same machine or move to the next machine. The number on the top of a re-entrant loop indicates the number of times a job returns back to the same machine to get reprocessed. A re-entrant loop abstracts from components such as conveyor belts, robotic arms or human operators. Similarly, a machine is an abstraction of

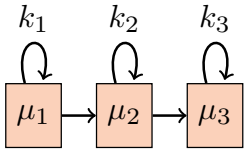


Figure 4.1: A self-re-entrant flowshop with three machines.

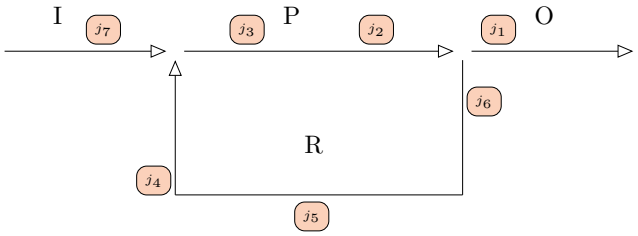


Figure 4.2: Jobs flowing in a self re-entrant machine.

a set of embedded systems that collaboratively schedule and control physical and mechanical processes. Decisions about the structure of a flowshop made during the design of a re-entrant loop influence the performance of the flowshop. For example, as later shown by the case study, changing the length, and hence duration, of the re-entrant loop of a LSP impacts its performance.

A machine in a self-re-entrant flowshop usually consists of several components. The operation of such a machine is shown in Figure 4.2. The jobs (shown as rounded rectangles) enter the machine from the Input component (I), get processed by the Processor component (P) and return back on the re-entrant loop (R) to get processed again. A job leaves the machine at the Output component (O). The P and the R component form the re-entrant loop of the machine. There are possibly many jobs in the machine and all of them re-enter the machine at least once (i.e. every job is at least processed twice). For many industrial applications (e.g. wafer sorting [6], industrial printing [14], LED manufacturing [5] and TFT-LCD assembly [7]) the processor component may require adjustment before processing a job leading to *sequence dependent setup times*. It is assumed that the setup times are significant and they arise due to the differences between jobs. In a wafer sorter [6], handling memory ICs and logic ICs are examples of different jobs that require setups, i.e., changes in the settings of the machine due to the nature of the process before the processor component, P, switches from one type of job to the other type. The structure of a flowshop (physical part) has an influence on the freedom a scheduler has (cyber part). Thus there is an interplay between the cyber and physical aspects of self-re-entrant flowshops.

On the re-entrant loop, it is assumed that the velocity of a job has a lower and an upper bound that result from mechanisms used to transport the job, such as, motors on a conveyor belt with programmable (but bounded) rotational velocity. The bounds in conjunction with the length of the loop is a deadline, referred to as a *relative due-date*, between the time when a job is processed for the first time and the time when the job is processed for the second time leading to a minimum and a maximum loop time. The difference between the minimum and the maximum loop time acts as a virtual buffer that can be used to reduce the number of setups to increase performance by smart interleaving of jobs from the input component with the jobs on the re-entrant loop. It is assumed that the cost of a flowshop is directly proportional to the minimum loop time and the buffer time. Reducing the minimum loop time and the buffer time reduces the cost of the flowshop. Hence there exist trade-offs between the cost and the performance of a re-entrant flowshop with setup times and relative due-dates.

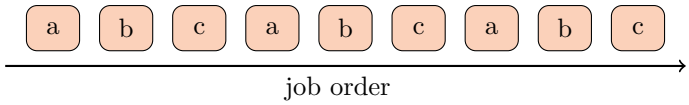


Figure 4.3: A regular job set with a pattern of jobs.

Many jobsets in an industrial operation of a re-entrant system either contain repeating patterns of jobs or only jobs of the same kind. Systems that process regular job patterns are for example a wafer sorter or a printer that repeatedly prints the same jobset. Figure 4.3 is an example of a jobset containing nine jobs with jobs a, b, c, as a pattern repeated three times with a as a first job in the pattern. The pattern repeats in a *tandem* fashion i.e. a pattern immediately follows after the previous pattern ends and the patterns are never partial. The influence of the timing constraints in a jobset on the performance of a flowshop is specific to the jobset. Given the jobsets a challenge is to determine the minimum loop time and the buffer time for maximal performance.

4.2 Problem definition

Recall, from Section 2.2, that a self-re-entrant flowshop consists of a set $\mathcal{M} = \{\mu_1, \ldots, \mu_i, \ldots, \mu_m\}$ of machines processing a job set $\mathcal{J} = \{j_1, \ldots, j_n\}$. The machines process the jobs in an order described in a *flow vector* $\mathcal{V} = [v_1, \ldots, v_r]$ with $v_i \in \{1, \ldots, m\}$ in which the processing of every job starts from the first machine, i.e. $v_1 = 1$ and ends at the last one, i.e. $v_r = m$. In a self-re-entrant flowshop a job is either re-processed by the same machine or passed on to the next machine. It means that in the flow vector $\mathcal{V}$, v_{i+1} either equals v_i or is index of the next machine $v_i + 1$. Thus the flowshop operates r times on a job with the set $\mathcal{O}_i = \{o_{i,1}, \ldots, o_{i,r}\}$ denoting the set of operations of a job j_i. The set $\mathcal{O}_\mathcal{J} = \bigcup_{j_i \in \mathcal{J}} \mathcal{O}_i$ contains the operations of all jobs in the job set $\mathcal{J}$. The jobs in the flowshop have a *fixed job order* where the processing of jobs start from j_1 and ends with j_n. For each machine, the operations from different jobs require to be ordered as the machine is a *unary resource*.

The y^{th} operation of a job j_x is $o_{x,y}$ that takes $p(x,y)$ time units to execute. A machine can at most perform one operation at a time and preemptions are not allowed. A machine may require additional time, called *sequence dependent setup time*, to prepare for the processing of the next operation. The sequence dependent setup time between an operation $o_{x,y}$ and an immediately following operation $o_{u,w}$ is $s(x,y,u,w)$. Similarly, a *relative due-date* $d(x,y,u,w)$, is the maximum allowed amount of time difference between the start of the processing of an operation $o_{x,y}$ and the start of the processing of an operation $o_{u,w}$. For an operation $o_{x,y}$ its start time is $\mathcal{S}(x,y)$ where $\mathcal{S}$ is a schedule. A tuple $f(\mathcal{M}, \mathcal{V}, \Lambda)$ denotes a flowshop with a set of machines $\mathcal{M}$ processing jobs as specified in the flow vector $\mathcal{V}$ with minimum loop times and buffer times specified in the vector Λ. The vector $\Lambda = < (l_1, b_1), \ldots, (l_m, b_m) >$ contains m pairs of minimum loop time and buffer time (respectively) for the machines in the flowshop.

Let the throughput, i.e. the number of jobs processed per unit of time, of a machine m in a flowshop f for a jobset $\mathcal{J}$ be denoted by $n_{m,f,\mathcal{J}}$. Then, the throughput of the re-entrant flowshop f for the jobset is $n_{f,\mathcal{J}} = \min_{m \in \mathcal{M}} n_{m,f,\mathcal{J}}$ assuming that the machine with the least throughput is the bottleneck.

The performance estimator defined in this work assumes that the jobsets consist of a tandemly repeating pattern of jobs. It is also assumed that the jobsets for which the estimations are made have feasible schedules. For example, for any operation the processing and setup times must be significantly smaller than the minimum loop time. Additionally, it is assumed that the setup times are significantly larger than the processing times and therefore a scheduler can minimize the

number of setups required to gain significant performance. It is also assumed that the minimum loop time and the buffer time must be positive real numbers.

The performance of a jobset $\mathcal{J}$ on a flowshop f is defined as $\xi_{f,\mathcal{J}} = \frac{n_{f,\mathcal{J}}}{n^*_{\mathcal{J}}}$ where $n^*_{\mathcal{J}}$ denotes the maximum observed throughput for the jobset $\mathcal{J}$ over different flowshops considered during the DSE.

Definition 4.1 A jobset $\mathcal{J}$ is regular if it consists of tandem repeats of the form ω^k (denoting k repeats of ω) where $k > 1$ and $\omega \in \Sigma^*$ is a word over an alphabet set Σ.

The example job set in Figure 4.3 has pattern $\omega = abc$ with $k = 3$ and alphabet set $\Sigma = \{a, b, c\}$. The performance estimator assumes that a jobset consists of one and only one pattern $w \in \Sigma^*$ repeated a finite number of times in a jobset. Given the performance of a jobset, a challenge is to find the performance of a flowshop on the jobsets that are grouped in different categories. Let c be a category where $c = \{\mathcal{J}_1, \ldots, \mathcal{J}_u\}$ of u jobsets. Then the performance of a re-entrant flowshop is defined as follows.

Definition 4.2 The performance of a self-re-entrant flowshop f on jobsets in a category c is the weighted sum $\xi_{c,f} = \sum_{\mathcal{J} \in c} w_{\mathcal{J}} \times \xi_{f,\mathcal{J}}$ where $w_{\mathcal{J}}$ is the given weight of the jobset $\mathcal{J}$.

The weight $w_{\mathcal{J}}$ denotes the relative importance of the jobset $\mathcal{J}$ in the category c modelled, for example, by a statistical distribution of the frequency of the occurrence of such a jobset as shown in the LSP case study in Section 4.6. The weights for the jobsets in a category sum to one i.e. $\sum_{\mathcal{J} \in c} w_{\mathcal{J}} = 1$. The higher the importance of a jobset the more it contributes to the overall performance of the flowshop for the category. Let C be the set of all categories of jobsets of interest. Then the performance of a re-entrant flowshop (called the *total performance*) over a set of categories is defined as follows.

Definition 4.3 (Problem definition) The total performance of a re-entrant flowshop f over a set C of categories of jobsets is defined as $\xi_{T,f} = \sum_{c \in C} \xi_{c,f} \times w_c$ where w_c is the weight for category c.

The *total performance* indicates how productive a flowshop is over all job sets. The higher the total performance, the more productive the flowshop is.

Definition 4.4 A flowshop f_o has maximum total performance over the set of flowshops F under consideration if the total performance $\xi_{T,f_o} = \max_{f \in F} \xi_{T,f}$.

The following section describes a performance estimation method which is used to estimate the performance of f_o.

4.3 Related work

Re-entrant flowshop scheduling problems are found in looms in textile industry [19], wafer sorters [6], photo-lithography, printed circuit board manufacturing, assembly of circuits ([10, 49] provides detailed lists). They have been shown to be NP-Complete [50], thus, existing exact scheduling algorithms are prohibitively slow to be used for fast performance estimation of re-entrant flowshops. Even the heuristic techniques [32, 43, 44, 51] are too slow to explore the design space of a self re-entrant flowshop and thus do not satisfy the need of fast performance estimators. The work of [48] finds an optimal re-entrant loop size for an LSP without sequence dependent setup times. Extending the work of [48] for sequence dependent setup times requires solving an LP program several times which results in a mixed integer program that is compute intensive to solve. Traditionally, performance estimation of re-entrant flowshops has been performed using simulation

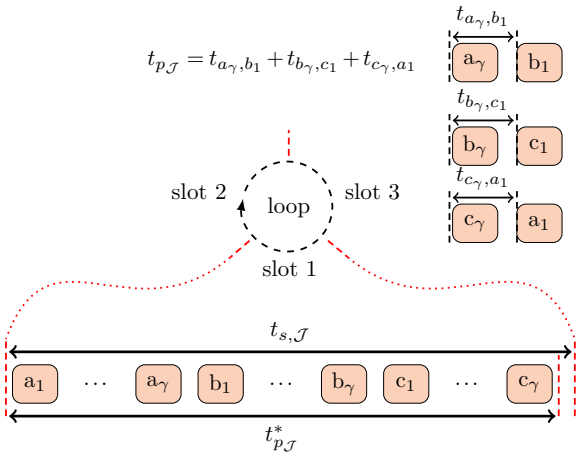


Figure 4.4: Place allocation for the regular jobset $\{a, b, c\}$.

[52], mean value analysis [53] and probabilistic methods [54]. Simulation based approaches are time consuming [52] which prohibits their usage as fast performance estimators. Mean value based estimators (e.g. [53]) assume that the processing times are known for jobs under consideration. However, in a flowshop with sequence dependent setup times the processing times depend on the state in which a machine is left by the previous job(s) and thus the total time, including the setup time, to process a job is unknown to perform mean value analysis.

The mean value based approach [53] and the probabilistic method [54] assume scheduling policies which either require priority scheduling for buffers or assume fixed scheduling policies. The assumption of a fixed scheduling policy and buffers with priority allow faster estimation but it does not optimize to reduce sequence dependent setups and thus will be less accurate. The estimator described in this work uses the knowledge of the patterns in a jobset to estimate the number of setups required. The computational complexity of the estimator is $O(n)$ which is similar to the mean value estimates in [53] as they are estimates that use the information about n jobs in a jobset.

The estimator described in this work is an extension of the work in [15]. The extension is for arbitrary number of re-entrances in a self-re-entrant flowshop. The number of re-entrances are highly dependent on the end-product of the self-re-entrant flowshop under study. The extension broadens the applicability of the estimator. Furthermore, [15] only estimates steady state performance. In this work, the steady state performance is extended to estimate performance of jobsets with finite number of jobs. The extension makes it possible to compare estimates with a scheduler which computes makespans for a given jobset. The difference of the steady state estimation with respect to the estimate for finite length jobs becomes negligible for large jobsets. However, in many application areas, smaller jobsets might be of high interest and therefore require estimation for finite length jobsets.

4.4 Performance estimation

The method in this section estimates the performance by dividing the loop in a self re-entrant machine into segments called *slots* that process repetitions of a pattern completely. Within a slot, for each job an allocation where it will be processed is determined based on the pattern of the jobs in the jobset and the timing constraints. Figure 4.4 shows the split of the loop into slots

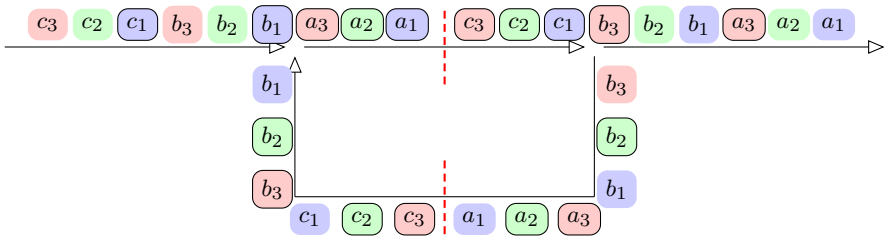


Figure 4.5: A loop with 2 slots for a job with pattern {a,b,c} having two re-entrances.

and the allocation of places for jobs with pattern a, b, c with every job re-entering γ times. The first place in a slot is allocated for a_1 (the subscript denotes that the allocation is fixed for the first occurrence of a in the slot), followed by the re-entering occurrences of job a to avoid setups. Similarly, for job b the allocations for the first and the re-entrant occurrences are determined. The performance estimation is performed by arranging multiple slots in a re-entrant loop. Figure 4.5 shows the slots in the re-entrant loop of a machine for the pattern shown in Figure 4.3 that has jobs a, b and c re-entering the loop twice in a steady state. The color of a job shows its re-entrance count. For example, all jobs that enter the machine for the first time are blue, green indicates that the job enters the second time and red for the third time. The jobs without a border indicate absent jobs but their place has been allocated to allow the merging of re-entering jobs with newly arriving jobs. For example, only first pass jobs are present while entering the machine. Similarly, only third pass jobs leave the machine. On the return path, before leaving the machine, the first pass jobs become the second pass and the second pass becomes the third pass. Therefore, there is no first pass job on the return path. The exact time when the pass of a job is changed to its next pass has no influence over the estimation but for clarity it is assumed that the passes change at the end of the processing component.

A slot s produces a number of jobs equal to the length $|p_{\mathcal{J}}|$ of the pattern $p_{\mathcal{J}}$ in the jobset $\mathcal{J}$. For example, in Figure 4.5 from every slot three jobs leave the machine assuming steady state. Therefore the performance of a re-entrant machine is the ratio between the work processed in a slot and the time span $t_{s,\mathcal{J}}$ of the slot. The *time span* of slot is the time that a slot occupies from the total loop time (minimum loop time + buffer time).

Definition 4.5 The estimated performance of a jobset $\mathcal{J}$ on a re-entrant machine m in a flowshop f is $\eta_{m,f,\mathcal{J}} = \frac{|p_{\mathcal{J}}|}{t_{s,\mathcal{J}}}$ where $|p_{\mathcal{J}}|$ is the number of jobs in the pattern $p_{\mathcal{J}}$ and $t_{s,\mathcal{J}}$ is the time span of the slot s.

Given a job set for a machine in a flowshop the performance is estimated using Definition 4.5 based on its pattern and an estimate of $t_{s,\mathcal{J}}$. The time span of a slot depends on the timing constraints between jobs in a pattern and thus in many cases will result in slot times which do not exactly fit in the loop. For such cases the slot time is adjusted such that the loop time becomes an integer multiple of the time span of the slot as the estimator assumes that the complete re-entrant loop in a machine m can be sub-divided into slots of identical time spans such that the loop time is an integer multiple of the time span of a slot. The assumption ensures that slots do not interfere with each other.

Let the pattern time $t_{p_{\mathcal{J}}}$ be the sum of the timing constraints between the last re-entrance of a job and the first entrance of the next job in the pattern $p_{\mathcal{J}}$ as shown in Figure 4.4. Let the re-entrant pattern time $t^*_{p_{\mathcal{J}}}$ be the sum of the timing constraints between the jobs in a pattern $p_{\mathcal{J}}$ considering the preallocated locations for re-entrant jobs. Then, for a given minimum loop time $t_l > 0$ and a buffer time $t_b > 0$ there are three possible cases that are described in Equation 4.1.

The first case is when the re-entrant pattern time is such that it fits an integer number of times in the loop. This case happens when the number of slots $n_{min} = \lfloor \frac{t_l}{t^*_{p_\mathcal{J}}} \rfloor$ that fit in the loop without buffering is not equal to the number of slots $n_{max} = \lfloor \frac{t_l + t_b}{t^*_{p_\mathcal{J}}} \rfloor$ that fit in the loop with using the maximum buffer time indicating that the range of the loop back times due to the buffer allow the re-entrant loop to accommodate an integer number of slots by selection of an appropriate buffer time.

$$t_{s,\mathcal{J}} = \begin{cases} t^*_{p_\mathcal{J}} & \text{if } n_{min} \neq n_{max} \\ \frac{t_l}{n_{min}} & \text{if } n_{min} \neq 0 \text{ and } n_{min} = n_{max} \\ t_{p_\mathcal{J}} + t_l \times |p_\mathcal{J}| \times (\gamma - 1) & \text{otherwise} \end{cases} \tag{4.1}$$

The second case is when at least one pattern fits in the re-entrant loop but the minimum loop time and the buffer time is not enough to accommodate a complete re-entrant pattern. Therefore there is no performance gain by using the buffer. Thus the time span of a slot $t_{s,\mathcal{J}} = \frac{t_l}{n_{min}}$ only utilizes the minimum loop time. Using only the minimum loop time will always result in an integer number of slots as follows. Substituting n_{min} gives $t_{s,\mathcal{J}} = \frac{t_l}{\lfloor \frac{t_l}{t^*_{p_\mathcal{J}}} \rfloor}$ and shows that the ratio of the loop time and the slot time results in an integer number i.e. $\frac{t_l}{t_{s,\mathcal{J}}} = \lfloor \frac{t_l}{t^*_{p_\mathcal{J}}} \rfloor$.

The third case is when $n_{min} = 0$ i.e. the re-entrant pattern time is larger than the given minimum loop time thus dividing the re-entrant loop into slots is not possible. In such a case, the re-entrant machine will process each job separately i.e. without interleaving the slots. The time $t_{p_\mathcal{J}} + t_l \times |p_\mathcal{J}| \times (\gamma - 1)$ denotes the total time to process the jobs in a pattern where t_l is minimum loop time, $|p_\mathcal{J}|$ is the number of jobs in a pattern and γ is the number of entrances. Thus the time is the total time for all jobs in a pattern.

The estimation described in Definition 4.5 assumes a steady state requiring infinitely long jobsets. During the case studies performed in this work the comparison is performed with jobsets containing a finite number of jobs. Initially a self-re-entrant machine does not contain any job. Several jobs are required such that the machine reaches the steady state. Similarly, when all jobs have entered a machine, they require processing before they leave the machine. States before and after the steady state are called *transient states*. The performance of a system is lower in the transient states compared to the steady state. Furthermore, for simplicity, it is assumed that the performance in the start and the end transient state is the same. The equations below estimate the makespan in the steady state and the makespan in the transient phases which are used to estimate the performance of a jobset with finite number of jobs.

$$MK = \frac{|J|}{\eta_{m,f,j}} \tag{4.2}$$

The duration of the jobs processed in the steady state is estimated by Equation 4.2. It is the ratio between the estimated steady state performance and the number of jobs processed in the steady state. To estimate the makespan considering a finite number of jobs we add the duration of two transient phases.

$$FMK = MK + 2.t_l.(\gamma - 1) \tag{4.3}$$

The makespan considering a finite number of jobs in a jobset is estimated using Equation 4.3. The duration of the transient states, that $\gamma - 1$ times the minimum loop time, is added twice because there are two transient states, one in the start and one in the end of the processing of the jobset.

Definition 4.6 The performance for a finite length jobset $\mathcal{J}$ over a re-entrant machine m in a flowshop f is $\eta'_{m,f,j} = \frac{|\mathcal{J}|}{FMK}$.

The performance for a jobset is estimated using Definition 4.6. Addition of the length of the transient states improves the estimate for smaller jobsets. For large jobsets, the estimate should converge with the estimate assuming steady state behavior.

The computational complexity of the estimator for n jobs in a job set is $O(n)$ which is explained with an intuitive argument. Equation 4.1 has time complexity $O(n)$ which is due to the computation of $t_{p_\mathcal{J}}$ and $t^*_{p_\mathcal{J}}$. The remaining variables in the equation are either derived from others in constant time or are parameters coming from the description of the considered job. Once the equation is computed, the performance is estimated through Definition 4.5 which has constant time complexity. Thus the total complexity of the estimator is $O(n)$. The usage of the estimator is described in the following section.

4.5 Using the estimator during DSE

The estimator described in the previous section computes the performance of a flowshop for a given jobset. This section demonstrates how the estimator can be used to perform DSE to find a flowshop which performs the best on all jobsets considering their importance. The relative importance of a jobset is a sample drawn from a probability distribution χ based on a specific property of a given jobset. For example, in many application areas (such as the LSP case study) larger jobsets occur less frequently than shorter jobsets and thus the number of jobs in a jobset could be used as a property to find the relative importance of the jobsets. Moreover, application specific distributions can be used where the relative importance is a more complicated function of a given jobset.

The estimator is used by the DSE method as described in Algorithm 4.5.1. The DSE method explores the design space of a flowshop given four sets: a set of minimum loop times, a set of buffer times, a set of machines and a set of categories of jobsets. The output of the algorithm are the estimates for the performance of jobsets over the provided loop and buffer times on all machines. Then the estimated performance of maximally productive flowshop f_o can be computed using the performance estimates for a jobset, the jobset weights and Definitions 4.2 and 4.3.

Algorithm 4.5.1: The design space exploration method

```
Input: Four sets: minimum loop times L, buffer times B, machines M and categories C
Output: Estimates of performance η_{m,f,J}
for m ∈ M, l ∈ L, b ∈ B do
    Let f be the flowshop with machine m having minimum loop time l and buffer time b.
    for c ∈ C do
        for J ∈ c do
            Compute η_{m,f,J} by Definition 4.5.
            Record η_{m,f,J}.
        end
    end
end
return recorded values of η_{m,f,J}.
```

4.6 Case study: Large scale printer

This section describes a case study to find the trade-offs between the performance of an LSP, the minimum loop time and the buffer time of the re-entrant loop in the printer. An LSP is a system printing sheets of paper in a re-entrant loop where every sheet is processed twice, i.e. to print the first side and the second side of a sheet. A job set consists of many jobs where each job is a sheet. The estimator described in this work finds efficient design decisions by comparing relative performance of different design choices therefore the actual units of the performance estimation does not influence the outcomes. The length of the loop is measured in the number of seconds spent by a sheet travelling from the merge point (the merge point is where the input component and the return loop meet) and to come back to the merge point. Similarly, the buffer time is the difference between the minimum loop time and the maximum loop time.

The case study is performed over three categories of jobsets with: *homogeneous* patterns, *booklet* patterns and *unstructured* randomly generated patterns. These categories represent the jobs observed in an industrial operation of the printer. The details of the testset are described in Section 4.6.1. Section 4.6.2 describes the experimental setup. The accuracy of the estimation is described in Section 4.6.3 followed by Section 4.6.4 that describes the results of the case study.

4.6.1 Test set

The testset to explore the design space of an LSP consists of jobsets which represent print requests of industrial importance. The jobsets are categorized based on the pattern in each category. The *homogeneous* jobsets have jobs with identical characteristics. Therefore the pattern in a homogeneous jobset consists of a single sheet only. The homogeneous category is of most importance as the majority of industrial jobsets are homogeneous in nature. Examples of homogeneous jobsets are found in printing of forms and information pamphlets. The second category is of the *booklet* type where there is a cover sheet followed by several body sheets of different type. Examples of booklets are books and brochures. The cover sheet in a booklet is often thicker than a body sheet requiring sequence dependent setups between printing a cover and a body sheet and vice versa. The third category consists of jobsets having *unstructured* randomly generated patterns repeating a number of times in the jobset.

Table 4.1 lists, for each category, the parameters of the testset. The number of jobs in a jobset is the first parameter which indicates what is the minimum and maximum number of jobs in a jobset (sheets in a print request). The second parameter indicates how many jobsets are created with a sheet count uniformly chosen between the minimum and maximum number of sheets. The jobs in a jobset are sheets of specific length and thickness. The pattern length indicates the range of the number of jobs in a pattern. The total jobsets indicates the total number of jobsets generated by the combination of different parameters for each category. In total there are 2100 jobsets in the testset. The design space of the LSP consists of the minimum loop time between $1-18$ seconds and $1-5$ seconds for the buffer time (in steps of 1 second) resulting in 189000 testcases to be explored. Minimum loop time of 10 seconds and buffer time of 2 seconds are considered as default values in the experiments if not mentioned explicitly. The ranges of the minimum loop time, the buffer time and the default values are of interest to the designers of the LSP.

4.6.2 Experimental setup and run-time

The DSE algorithm and the performance estimation were implemented in Python. The experiments presented in the next section are performed on a Windows 7 Enterprise Edition running on an Intel i7 at 2.90 GHz. The estimator took on average 1.1 ms and maximum of 32 ms per testcase. The scheduler used in the LSP (proprietary scheduler used by the company which manufactures

Parameter	homogeneous	booklet	unstructured
Jobs in a pattern/job set	between 15-115 sheets	between 3-12 sheets	between 3-12 sheets
No. of random job set size selection	70	5	5
Sheet length (mm)	{177.8, 210, 420, 431.8, 500}	c ∈ {355.6, 420} and b ∈ {177.8, 210}	{177.8, 210, 420, 500}
Sheet thickness (mm)	{0.1, 0.2}	b ∈ {0.1}, c ∈ {0.2}	{0.1, 0.2}
Pattern length	-	between 2 -36 sheets	between 1-8 sheets
Total job sets	700	700	700

Table 4.1: Parameters for the testset used in the case study.

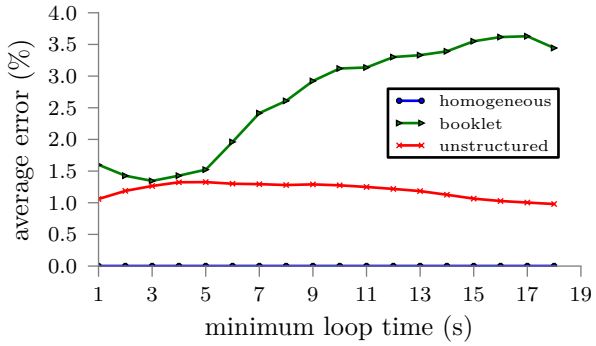


Figure 4.6: The absolute error between estimation and measured performance of the LSP.

the LSP) requires on average 200 ms and minimum 150 ms per test case. Thus the estimator is relatively fast.

4.6.3 Accuracy of performance estimation

The accuracy of the performance estimation influences the outcomes of the DSE. Incorrect performance estimation might lead to a design which has worse performance in reality. To assess the error, for each test case, the performance estimate is compared to the performance reported by the print scheduler (written as *real*) of the LSP. The reported performance is the length of sheets (in millimetres) printed per second by the printer (to normalize for different paper sizes). To compute the error, the estimate is also converted to millimeters per second. For all testcases, Figure 4.6 shows the percentage of average error illustrating how good the estimator performs on the entire testset. For instance, for a minimum loop time l and for a jobset $\mathcal{J}$ the ratio of error is $e_{l,\mathcal{J}} = \frac{real_{l,\mathcal{J}} - estimate_{l,\mathcal{J}}}{real_{l,\mathcal{J}}}$. Then the percentage of average error for a category C and minimum loop time l is $error_{l,C} = \frac{\sum_{\mathcal{J} \in C} e_{l,\mathcal{J}}}{|C|}$. As the figure shows, the homogeneous category has the least error because of the predictability of the scheduling behaviour of the LSP for a homogeneous jobset. The error increases with the increase in the dynamic structure in a jobset. The error

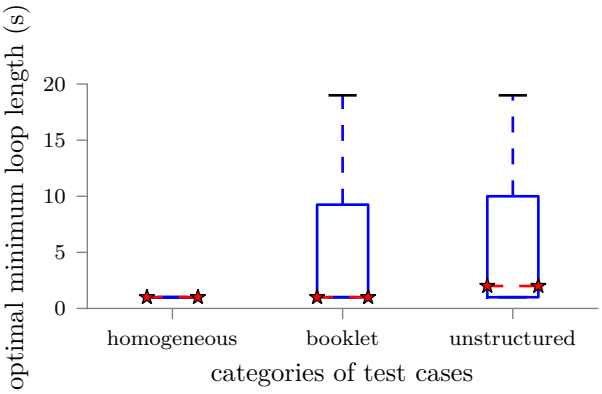


Figure 4.7: The variation in the optimal minimum loop time of the LSP.

in the booklet category increases with the increase in the minimum loop time. The increase is because, for testcases with few jobs and a large loop it is less likely to reach a steady state assumed by the estimation method. However, the average accuracy of at least 96% is promising to evaluate early design decisions and hence the estimator will sufficiently facilitate the trade-off analysis.

4.6.4 Trade-off analysis

The trade-offs between the length, the buffer time and the performance of an LSP arise because of the sequence dependent setup times, the relative due-dates and the buffering behaviour of the re-entrant loop. Furthermore the trade-offs are specific to jobsets. Figure 4.7 shows the variation in the optimal minimum loop time for all jobsets as a box-plot. The height of a box shows the span of the variation for 50% of the cases closest to the median in a category. The span of a whisker (the lines going out of the boxes) indicates the variation for 25% of the cases. The two stars inside a box is the median for a given category and splits the variation into two halves with each half representing variation for 50% of the cases. For the homogeneous category, the optimal minimum loop time is the smallest possible minimum loop time because, due to the homogeneity of the jobs, the minimum loop time does not affect the performance and the DSE method chooses the shortest re-entrant loop as the optimal one. The optimal minimum loop time for the booklet and the unstructured category varies over almost the entire design space. Over all categories, the experiment shows that the optimal minimum loop time is job and category specific. Thus, fixing the minimum loop time to a certain value would make the LSP productive for some jobsets but will lose performance for other jobsets.

The relative importance of jobsets in different categories can be modelled using distributions as shown in Figure 4.8. The relative importance is assumed to be normally distributed. In this case study, the importances were derived from the profiles of the customers of the LSP. For a given number of sheets in a pattern (x-axis) its relative importance is given by the height of the normal curve which is used as a weight to compute the total performance.

Figure 4.9 shows the total performance for the categories of the jobsets. For a given minimum loop time and for all jobsets, the performance is the weighted sum of the relative importance of a jobset and the ratio of performance with the maximum performance. The booklet category performs relatively poorly on small minimum loop times because a booklet does not fit and then the setups cannot be avoided thus losing performance. Furthermore, the weighted sum of the total performance over all categories (according to the importance of a category i.e $booklet = 0.7$,

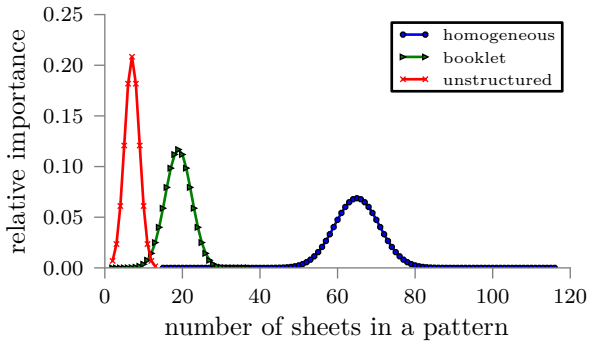


Figure 4.8: Distributions used to model the importance of jobsets in different categories.

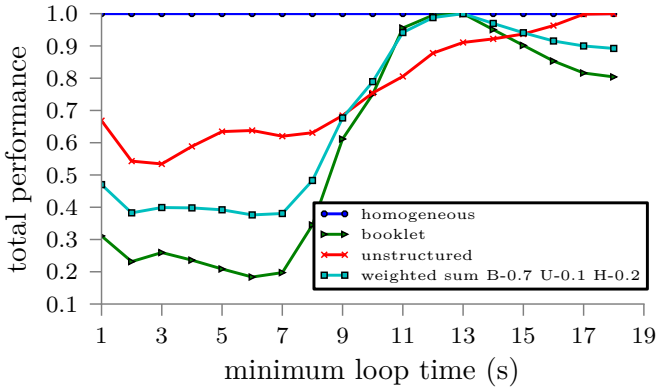


Figure 4.9: Total performance for different categories of jobsets for the LSP.

$unstructured = 0.1$, $homogeneous = 0.2$) indicates the total performance of the LSP. The printer has maximum performance at 13 seconds of minimum loop time and default buffer time.

Figure 4.10 shows the weighted sum total performance (normalized by maximum of estimated performances) for the printers with different buffer times. An increase in the buffer time increases the total performance because of the increase in the scheduling freedom to minimize the number of setups. However, as shown in the figure, at minimum loop time of 11 seconds the performance is almost the same for buffer times of 4 and 5 seconds. That means a cost reduction could be achieved by having a smaller buffer without significant loss of performance. Similarly, for buffer time of 1 second the minimum loop time of 13 seconds performs better than 15 seconds; thus a shorter loop is beneficial. However, as shown in the figure, an increase of a single second is very beneficial between 7 to 11 seconds. The designer of the LSP can assess, using these trade-offs unveiled by the DSE, whether adding additional cost to the LSP brings performance gains.

4.7 Case study: eXplore Cyber Physical Systems Platform

The xCPS platform [55] is a flexible self-re-entrant flowshop used for research and educational purposes. The system, shown in Figure 4.11, consists of belts and motors that transport objects with constant velocity to get processed at the processing unit. Figure 4.12 shows the flow of the jobs in xCPS. The jobs start at the *entry* point and move along the path towards the *processing*

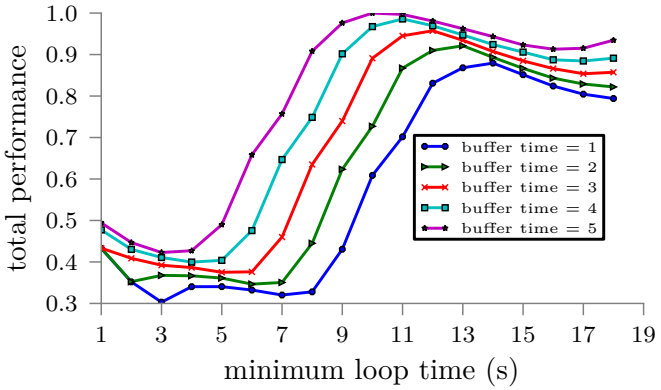


Figure 4.10: Total performance over the design space of the LSP.

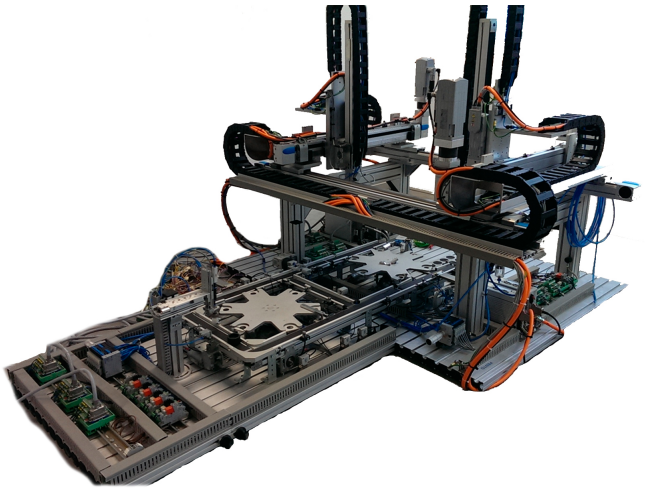


Figure 4.11: The xCPS platform.

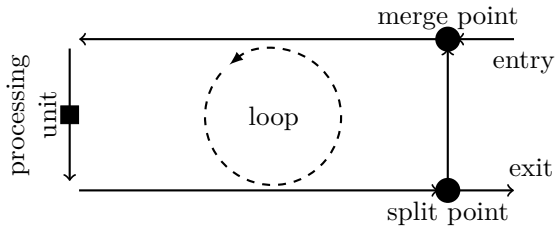


Figure 4.12: An abstraction of the xCPS platform.

Parameter	values
job types	{A,B,C}
patterns	{AAB, ABB, ABC }
# of jobs	102
minimum loop times	{50, 60, 70, . . . , 130}
re-entrances	{4,5}
buffer times	{6,12,18,24}
processing time	2
setup time	20

Table 4.2: Parameters for the testset used in the xCPS study.

unit. The processing unit performs an operation on a job. Depending on whether a job requires further processing, it either leaves the system at the *exit* point or moves towards the *merge point*. At the merge point the scheduler has to determine whether to let a new job in the system from the entry point or allow an existing job in the system to proceed for re-processing. Furthermore, the scheduler needs to ensure that two jobs do not collide at the merge point. It is assumed that all jobs require the same amount of processing and setup time in case a setup is required. The minimum loop time is a constraint between the consecutive reprocessing of a job. The sum of the minimum loop time and the buffer time is a relative due-date between the consecutive reprocessing of a job. The reprocessing of jobs is commonly seen, for example in lithography machines [56] and wafer sorters [6].

The flow of jobs in the xCPS is circular and is shown as a *loop* in Figure 4.12. This case study finds out what is the optimal time to travel on the loop to get good performance. Furthermore, this case study answers whether to add buffering capacity between the split point and the merge point will increase the overall system performance when the jobs are allowed to re-enter several times. The performance estimator proposed in this work is used to explore these design choices.

4.7.1 Test set

The case study was performed using a testset whose details are summarized in Table 4.2. The goal of the case study is to use the performance estimator to predict the impact of the structural changes over the performance of xCPS. The jobsets consist of at maximum three jobs with a tandem repeating pattern out of three possible patterns as described in the table. Similarly, the number of jobs in a jobset is an integer multiple of the length of a pattern.

The case study is performed over a minimum loop times of {50, 60, 70, . . . , 130} and buffer time of {6,12,18,24} time units. The jobs in the case study re-enter xCPS either 4 or 5 times. The processing time for all operations is 2 time units. The setup time, which occurs between any

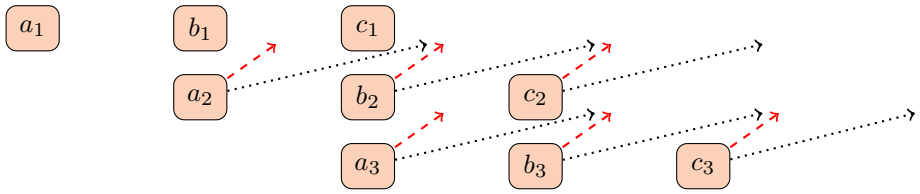


Figure 4.13: Overview of the adapted MNEH heuristic for a jobset {a,b,c}.

two consecutive jobs of different types, is 20 time units. Thus, when switching between different job types, the processing unit takes additional 20 time units.

4.7.2 Experimental setup and run-time

The DSE method and the performance estimation were implemented in Python. The experiments presented for xCPS were performed on a Windows 7 Enterprise Edition running on an Intel i7 at 2.90 GHz. The estimator took on average 0.016 ms and maximum 1.1 ms per test case. To evaluate the accuracy of the estimation, schedules were generated using an adapted version of the *Modified Nawaz Enscore and Ham* (MNEH) heuristic described in [13].

The overview of the adapted version of the MNEH heuristic for a sample jobset {a,b,c} with three re-entrances is shown in Figure 4.13. The heuristic, for each re-entrance, generates sequences of the operations in the flowshop by merging them to rows above with *shifting*. For example the dashed arrows show locations with a shift of one job. Dotted arrows indicate the locations with a shift of two jobs. The sequences generated by the merges are further merged with the next re-entrant operations until the last re-entrance. Out of all sequences generated a feasible sequence with minimal makespan is found and reported by the heuristic.

Algorithm 4.7.1 describes the adapted MNEH heuristic. Given a job-set with R re-entrances, minimum loop time and buffer time, and minimum of processing times of all operations it finds a schedule. The heuristic, for each re-entrance, iteratively merges sequences and generates new sequences. Out of all the possible sequences, the feasible sequences with minimal makespan is used to generate a schedule. The feasibility of a sequence and the makespan are computed using the Bellman-Ford algorithm. The MNEH heuristic of [13] shifts operations to find their best positions. However, for most of the cases considered for xCPS, the precedence constraints between jobs make the individually shifted operations lead to infeasible schedules. Instead, in the adapted version, all operations are shifted in a given re-entrance with the same amount of shift. The same amount of shift abides by the precedence constraints. Notice that many of the sequences generated by the adapted version might still be infeasible due to the relative due-date constraints.

Another benefit of fixed shifts per re-entrance is that the search space is reduced. Thus given the fixed amount of time the heuristic can explore more solutions and probably find better schedules. Our attempts to generate optimal schedules with, for example, MiniZinc [57] did not yield good solutions even when allowed to run for weeks. On the other hand, visual inspection of the schedules generated by the adapted MNEH reveals that most of the schedules are highly efficient and are therefore used for evaluation of the estimation in the next sub-section.

In the start of Algorithm 4.7.1 it is checked whether a fully interleaved schedule of jobs is possible or not. If not then the algorithm returns the trivial solution which is to process the jobs separately. Such a schedule will have very low performance. Note that it is possible that better schedules might still exist. For example, an optimal solver might still find partially interleaved

Algorithm 4.7.1: Adapted MNEH heuristic for self-re-entrant flowshops.

Input: a jobset $\mathcal{J}$ with R re-entrances, minimum loop time mlt, buffer time bt, minimum processing time pt
Output: a schedule
if $t_{s,\mathcal{J}} > (mlt + bt)$ **then**
    return the trivial solution: process each job separately.
end
$a \leftarrow$ operations of the first re-entrance of $\mathcal{J}$
$sequences \leftarrow \{a\}$
for $\gamma \in \{2, \ldots, R\}$ **do**
    $x \leftarrow$ operations of the γ^{th} re-entrance of $\mathcal{J}$
    $maxshifts \leftarrow \frac{mlt+bt}{pt}$
    for $sh \in \{1, \ldots, maxshifts\}$ **do**
        $new \leftarrow \{\}$
        for $a \in sequences$ **do**
            $c \leftarrow$ merge x with a with a shift of sh jobs
            add c to new if feasible
        end
        $sequences \leftarrow new$
    end
end
Find a from $sequences$ that is feasible and has smallest makespan
return a schedule found using a

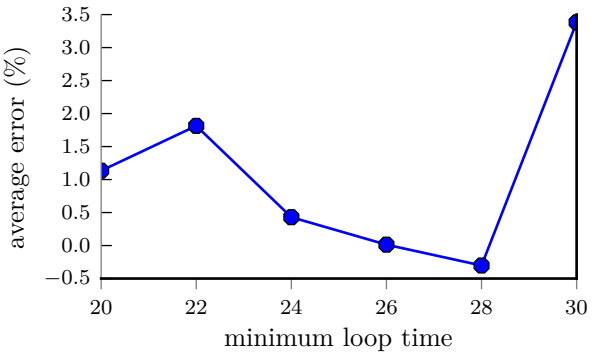


Figure 4.14: The accuracy of the estimation for the xCPS study.

schedules which are better than the trivial solution but finding such an optimal schedule is time consuming. Furthermore, designing such a scheduler during the early design space exploration is not a practical choice. Therefore, we assume the trivial solution as a possible schedule for the case where interleaved operation is not possible.

4.7.3 Accuracy of performance estimation

The estimation is evaluated by comparing it to the makespans of the schedules found for the testcases using the adapted version of the MNEH heuristic as described in Section 4.5. The average of the ratio of the error over all jobsets in the estimation is shown in Figure 4.14. The minimum loop time is shown on the x-axis and the percentage of the average error is shown on the y-axis. For each job set $\mathcal{J}$ and for a minimum loop time l the percentage of error is $e_{l,\mathcal{J}} = 100 \times \frac{abs(real_{l,\mathcal{J}} - estimate_{l,\mathcal{J}})}{real_{l,\mathcal{J}}}$. For a minimum loop time the average and the maximum across all the test cases are shown in the plot. The error is at most 3.5% due to the fact that the adapted MNEH heuristic generates start times in the transient states which are irregular but highly efficient. On the other hand, the estimation method assumes a simple abstraction on how long the transient states can be.

The adapted version of the MNEH heuristic, for job sets with 30 jobs, takes at most 7 hours. Thus, the set of test cases used for evaluation is smaller than the set of test cases for the DSE. However, once the evaluation is performed, the estimation method can be used for larger jobs and over a larger design space.

4.7.4 Trade-off analysis

The case study explores the impact of the structural changes over the xCPS platform, namely, the minimum loop time, buffer time and different number of re-entrances of job sets. The total performance of the xCPS platform where every job re-enters 4 times is shown in Figure 4.15. Unlike the LSP case study the job sets in the xCPS case study have the same importance and a single category; thus all weights in the total performance computation are the same.

The figure shows that the increase of the buffer time significantly influences the performance. The xCPS platform is most productive at minimum loop time 60 with buffer time of 24 seconds. Any decrease in the buffer size decreases the performance because many patterns do not fit in the loop and therefore, for patterns which do not fit in the loop the performance is decreased. Furthermore, the increase in the minimum loop time decreases the performance. This happens because the buffers are not active at all and therefore, in most of the cases the trivial schedule is

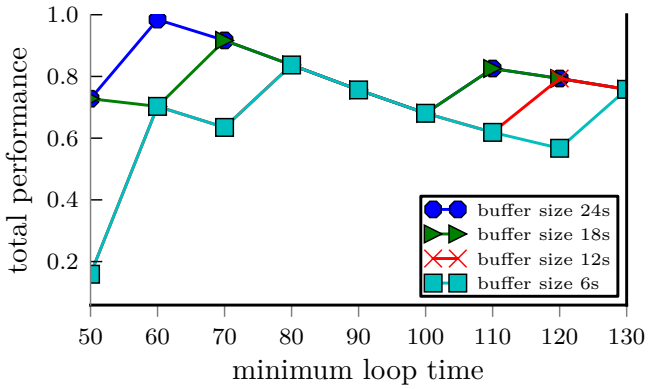


Figure 4.15: Total performance for 4 re-entrances of the xCPS platform.

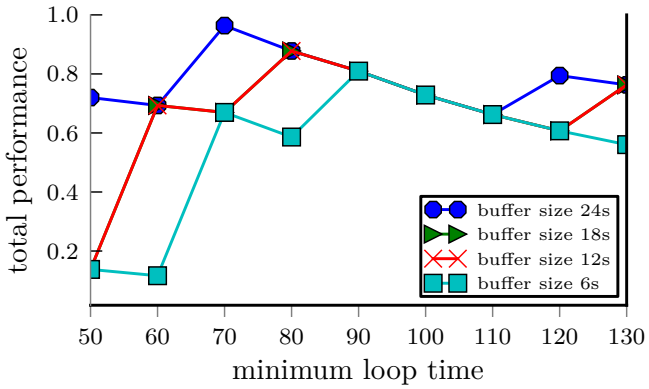


Figure 4.16: Total performance for 5 re-entrances of the xCPS platform.

produced (case 3 of Equation 4.1). In such a case increase in the minimum loop time increases the slot time and therefore decreases the performance. This experiment shows to the architect of the system that roughly an increase of 30% of the performance can be achieved with larger buffers. For minimum loop times between 80 and 100 buffers do not influence performance because reduction of setup times is not possible. Therefore, the smallest buffer can be used to reduce the cost of the overall system.

The total performance of xCPS with 5 re-entrances is shown in Figure 4.16. Compared to the previous case of 4 re-entrances, the best performance is observed at the minimum loop time of 70. The minimum loop times between 50 to 80 significantly benefit from additional buffering capacity; therefore it is worthwhile to spend costs to achieve performance. Minimum loop times between 90 to 100 do not get improved performance when additional buffering capacity is provided. Therefore, if the system architect decides to have a minimum loop time between 90 and 110 then smallest buffer can be used.

As shown by the experiments, the estimation method quantifies the impact of the structural design decisions over the performance. The architect of the xCPS platform can find out whether spending more cost over the buffers or having a smaller or a larger system is beneficial for performance or not. Depending on the type of self-re-entrant flowshops being produced, it may

also form a basis to perform customer specific customization with specific job patterns.

4.8 Conclusions and Future work

The structural design decisions of a self re-entrant flowshop influence its performance. Traditional approaches to estimate performance are either too slow or do not take sequence dependent setup times into account. The performance estimator described in this chapter, for regular jobsets, allows to explore the relation between the design parameters and performance without using compute intensive algorithms. The estimator splits the re-entrant loop into slots and finds whether the timing constraints of the jobs in a pattern fit into a slot. The designs where the constraints fit into a slot have high performance.

The maximally productive length of the re-entrant loop, as shown by the LSP case study, highly depends on the jobsets under consideration. Larger loops are suitable for jobsets that incur setups. Smaller loops are suitable for jobsets with fewer to no setups. For the LSP, the estimator has an average compute time of 1.1 milliseconds with an average accuracy of not less than 96%. The accuracy and fast computation of the estimator facilitates a designer to evaluate structural decisions during the design of self re-entrant flowshops with sequence dependent setup times and due-dates.

The performance estimator assumes that a jobset consists of repeating patterns of jobs. The assumption holds in many industrial applications but estimation of performance for jobsets without a pattern is left as future work. Furthermore, the design parameters explored in this work are the loop time and buffer time because they significantly influence the performance. The study of which other design parameters influence the performance is another interesting topic for future work.

An extension to the estimator for flowshops with arbitrary re-entrances other than self-re-entrances is proposed as a future work. Such an extension is suitable for the systems where job mixes from different market segments are processed at once. For example, in the printing industry, recently, prints of books, letters, and magazines are printed at once. Such a combination of prints are done by businesses that provide mass printing as a service to third parties.

The estimator can be further extended for computation of lower bounds over the performance. The estimator assumes that a job repeating in a pattern has timing constraints that also repeat in a pattern. The first and the second case of Equation 4.1 are already a lower bound, probably a tight lower bound but not exact. The third case is an upper bound. The knowledge of the patterns in the timing constraints might be used to get a lower bound for the third case. After the extension, the estimate will be a lower bound for the performance of self-re-entrant flowshops.

"You have to be careful to react when you start to deviate from your course" - Carlos Ghosn

5 Variation-aware design of self-re-entrant flowshops

The design parameters influence the performance of self-re-entrant flowshops. Such variation in performance might decrease the system predictability. The performance might vary due to a change in the dimensions of the system, due to a scheduler or due to the jobsets. The design process itself may cause variation in the desired system parameters. Factors like humans in the loop, available off-the-shelf components and complications due to precision and accuracy might change the design parameters. The designed flowshop might end-up to be different from the intended design due to the construction process. Availability of hardware and knowledge of the market segment might increase over time and cause changes in the designs. A final design might deviate from the desired design parameters. Variation-aware self-re-entrant system design can be achieved cheaper in early design phases than later [58]. This cost saving is one of the main reasons that early design space exploration is performed. This chapter presents a measure to estimate how much the performance may vary if a design deviates from planned parameters. This measure can be combined with other design space exploration methods, accounting for variation in performance due to deviations.

Section 5.1 introduces the structure of self-re-entrant systems. Different sources of variation are introduced in Section 5.2. These sources must be taken into account while measuring variation in performance. The problem is formally defined in Section 5.3. Section 5.5 describes a measure to estimate performance variation. The measure takes into account how small changes in a design parameter changes the performance. The results of using the measure are described in Section 5.6. This chapter is concluded and future work is described in Section 5.7.

5.1 Structure and variation in performance

Self-re-entrant systems come in different sizes, scales, and setups. Still, several aspects are in common. A car manufacturing system comprises of several stages possibly present at different locations. There might be several hundred people working in all stages together with many automation robots. Adding or removing human operators and robotic arms influences the performance of the manufacturing system. A lithography machine has dimensions fitting in a large room. Several pre and post processing machines around the lithography machines are

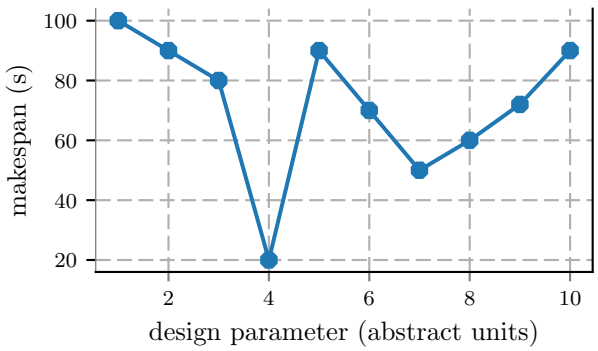
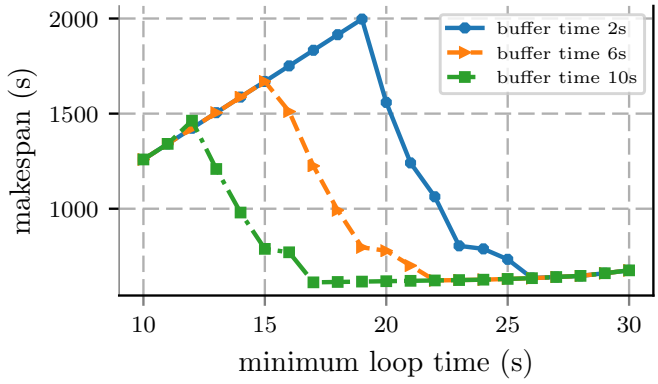


Figure 5.1: Variation in the performance of a system due to a structural parameter.

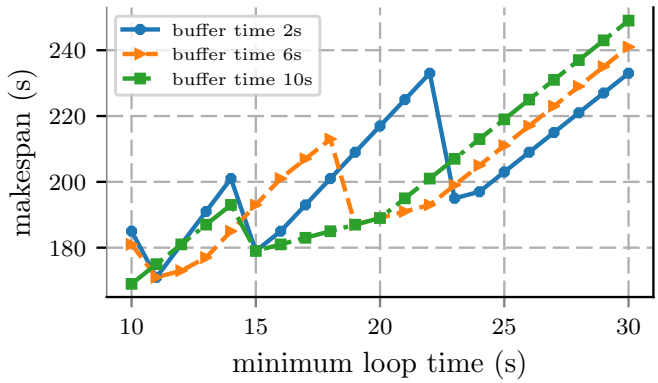
required to complete the wafer etching process. The machine consists of many high precision components. Adding more of these components is possible but is expensive. Relative to car manufacturing, lithography has fewer humans in the manufacturing loop. A *Large Scale Printer* (LSP) on the other hand fits entirely in a room with usually few human operators. These operators mostly monitor the operation of the LSP and intervene only if necessary. These examples show different flavours of self-re-entrancy and their associated structural aspects. The common aspects are self-re-entrant loops processing products in a machine and passing objects to the next stages possibly via buffers.

The self re-entrant loop in a flowshop brings the products back to the same machine for processing. The time that products spend travelling on the loop, for example, influences the timing constraints between the processing and re-processing operations of the same products. Similarly, the buffering capacity of the self-re-entrant loop influences the due-dates between the processing and reprocessing operations of the same product. Analysing the influence of the these structural decisions at the design phase along with the effect of deviations from the decisions may assist in having a system which has timing constraints for which a scheduler can find high quality schedules.

Variation usually reduces predictability of a system. For example, when several self-re-entrant flowshops work in a pipeline, variation in performance might make a self-re-entrant flowshop produce its products faster than expected. The buffers between the flowshop might require additional capacity to store the products before the next flowshop is ready. Consider the example in Figure 5.1 that shows the performance for a jobset over ten different system designs having different values for a design parameter. The value of 4 units yields the shortest makespan of 20 time units. A deviation of a single unit in the design parameter from 4 results in a system that has makespan of 80 or 90 time units which is around 4 times more than the makespan at value 4. The design with the parameter value of 7 units has the makespan of 50 units. The designs with value 6 and 8 have 70 and 60 time units respectively. Though the design with value 7 has larger makespan than of the design with value 4, its performance varies less with small deviations in parameter values. Such a design is more predictable, thus is preferred, compared to the design with value 4. An interesting question is how to measure such a variation at design time? Do designs with high performance having less variation exist in self-re-entrant systems?



(a) A jobset with many setups.



(b) A jobset with few setups.

Figure 5.2: Different sources of variation in performance in sample jobsets.

5.2 Sources of variation in performance

The variation in self-re-entrant flowshop has several structural sources. Changing the length of the re-entrant loop as well as its buffer capacity influences its performance. Figure 5.2 shows such sources. The makespan of a jobset consisting of mostly different jobs is shown in Figure 5.2a. Due to the differences most jobs require setups. This figure shows the makespan of the jobset with varying design parameters: minimum loop time and buffer time. For the jobset, the makespan linearly increases with the increase of the minimum loop time, but, beyond 11 time units, the increase depends on the size of the buffer. Smaller buffers in a design have less capacity and thus contribute less to minimize the number of setups. The longest makespan for the jobset is observed for a design with buffer time of 2 time units and minimum loop time of 19 time units. Additional buffer time of 4 time units decreases the makespan and thus increases the performance. Different minimum loop time and buffer time combinations lead to significantly different performances. The performances differ because a design might be well suited to the timing constraints of a jobset compared to other designs. For such a design, better schedules are possible where as for other designs the timing constraints may not have high performing schedules or a scheduler might not be able compute them.

The results of the experiment shown in Figure 5.2a show that the performance of a jobset varies with the change in the structural parameters of a self-re-entrant flowshop. Furthermore, the parameters are not directly or inversely proportional to the performances. For certain values, the increase in the minimum loop time increases the performance but beyond certain parameter values the performance is negatively influenced due to the increase in the minimum loop times. This parameter value, where the change occurs, depends on the characteristics of jobset. This is further exemplified by Figure 5.2b.

The performance of a jobset consisting of mostly similar jobs is shown in Figure 5.2b. The general trend is that the makespan linearly increases and then reduces beyond certain minimum loop time. The additional buffer capacity does not significantly increase the performance. Beyond 15 units of minimum loop time, the buffer of size 6 seconds has slightly larger makespan than the buffer with size of 2 seconds. This is counter-intuitive as the additional capacity of the larger buffer can be ignored by the scheduler and thus generate a schedule similar to the smaller buffers. However, in this experiment, the makespan from best of 5 different heuristic scheduling strategies is shown. As the heuristics do not guarantee optimal schedules, optimal usage of the available resources is also not guaranteed. As shown in this thesis, finding optimal schedules for the class of self-re-entrant flowshops is NP-Hard. It is highly likely that heuristics are used to perform runtime scheduling of self-re-entrant flowshops. In such a case, the variation in performance will occur due to the non-optimality of scheduling methods used.

The performance of a self-re-entrant flowshop depends on how well the structure of the flowshop suites the characteristic of a jobset. This was shown in Figure 5.2. This is not a problem if a flowshop has to process the same type of jobsets. However, in industrial applications where mixed job types are processed, the analysis of the relation between the structure and the variance is required. Similarly, the application areas where dedicated heuristic schedulers are used, the variation in performance due to the schedulers must also be taken into account.

5.3 Problem Definition

Recall from Section 2.2 that a self-re-entrant flowshop consists of a set of machines $\mathcal{M} = \{\mu_1, \ldots, \mu_i, \ldots, \mu_m\}$ processing a jobset $\mathcal{J} = \{j_1, \ldots, j_n\}$. The machines process the jobs in an order described in a *flow vector* $\mathcal{V} = [v_1, \ldots, v_r]$ in which the processing of every job starts from the first machine, i.e. $v_1 = 1$ and ends at the last one, i.e. $v_r = m$. In a self-re-entrant flowshop a job is either re-processed by the same machine or passed on to the next machine. It means that in the

flow vector $\mathcal{V}$, v_{i+1} either equals v_i or is index of the next machine v_i+1. Thus the flowshop operates r times on a job with the set $\mathcal{O}_i=\{o_{i,1},\ldots,o_{i,r}\}$ denoting the set of operations of a job j_i. The set $\mathcal{O}_\mathcal{J}=\bigcup_{j_i\in\mathcal{J}}\mathcal{O}_i$ contains the operations of all jobs in the jobset $\mathcal{J}$. The set $\mathbb{J}$ contains all jobsets under study.

The y^{th} operation of a job j_x is $o_{x,y}$ that takes $p(x,y)$ time units to execute. A machine can at most perform one operation at a time and preemptions are not allowed. A machine may require additional time, called *sequence dependent setup time*, to prepare for the processing of the next operation. The sequence dependent setup time between an operation $o_{x,y}$ and an immediately following operation $o_{u,w}$ is $s(x,y,u,w)$. Similarly, a *relative due-date* $d(x,y,u,w)$, is the maximum allowed amount of time difference between the start of the processing of an operation $o_{x,y}$ and the start of the processing of an operation $o_{u,w}$. For an operation $o_{x,y}$ its start time is $\mathcal{S}(x,y)$ where $\mathcal{S}$ is a schedule.

Definition 5.1 The ***makespan***, $\mathcal{C}_\mathcal{J}$, of a jobset $\mathcal{J}$ is the start time of an operation that is performed the last in a schedule $\mathcal{S}$ computed as $\mathcal{C}_\mathcal{J}=\max_{o_{x,y}\in O_\mathcal{J}}\mathcal{S}(x,y)$.

The makespan is a measure of how fast a jobset is processed by a self-re-entrant system. For the same jobset, the schedule with shorter makespan has higher system utilization because the same set of operations have been performed is a less amount of time on the same resources leading to better resource utilization. Therefore, it is desired to have as short makespan as possible to achieve high system performance.

The architects of self-re-entrant systems, when designing a system, usually aim to achieve high performance together with other criteria. As discussed in Section 5.2, variation in performance of a self-re-entrant system can occur due to the characteristics of a jobset, the structural design choices or due to a specific scheduler. An architect performs design space exploration to find a good design for the system. However, due to the process of designing a self-re-entrant system, the final design parameter values might deviate from the initial ones decided by the architect. The deviation might lead to undesired reduction in performance because the structural characteristics like the length of the self-re-entrant loop could not be fully utilized by the scheduler for a jobset.

Definition 5.2 (Problem Statement) Given a flowshop with jobsets, find out which parameter values lead to high performance that varies the least due to changes in the values of the parameters. What is the relation between jobsets and variance in performance? How do different types of schedulers influence the variation in performance?

Section 5.5 proposes a measure to solve the above specified problem. The measure uses the makespans of different schedules and the variation in performance when parameters are varied. The variation is then considered together with the performance of the design to find pareto-optimal designs to find high performing designs with less variation. The proposed measure is used to perform variation-aware design on a testset inspired from real-world self-re-entrant systems (LSP, *eXplore Cyber Physical Systems* (xCPS)) in Section 5.6.

5.4 Related work

The effect of an increase in variability over performance of self-re-entrant flowshops is analysed in [59]. It is noted that the increase in the variation of input parameters of flowshops lead to decreased performance. The type of variation considered, for example, is in the processing times. In our case, the variation in performance is not due to processing times, or other input parameters of flowshops but is because of the relation between the timing constraints and the possible schedules.

It is also noted in [59] that variability is cumulative due to the combination of inventory,

capacity of buffers and time [59]. In terms of self-re-entrant flowshops, this means that variability due to one source adds up to the variability of other sources. Furthermore, in case of self-re-entrant flowshops, our experiments in Section 5.6.4 show that the relation of variation in performance and the sources is rather strongly coupled. Change in one of the sources, e.g jobset characteristics, leads to a different performance and might have a different structure well suited for high performance.

In this chapter, the trend of variation caused by different sources are analysed. Different aspects causing variability in flowshops are analyzed by [60]. Those aspects are fluctuation in processing times, random setup times and break downs in the system. The flowshop considered consists of several buffer to balance the effect of variation. Main focus is to analyse the non-uniformity of processing time under several objective functions. As a solution, the work of [60] proposes alteration of processing time by different optimization models. However, [60] only considers permutation flowshops without sequence dependent setup times and relative due-dates. Furthermore, the flowshop instances considered are small in size with only 7 jobs in total. The variation sources described for self-re-entrant flowshop in this work are considered to be given and non-alterable.

Minimizing completion time variance problem for a single machine is NP-Hard [61] which aims to provide non-varying waiting times to users [62]. Most of the works in the direction of the minimizing completion time variance aims to find schedules in which the products produced by the machines have the least variation in the times at which they are produced. In case of design of self-re-entrant flowshops, the aim is to identify designs where the performance does not vary if deviations from original designs occur. Similarly, [63] describes methods for performance prediction of stochastic systems. The methods described can be used at the design time to predict how a certain design performs under deviations. However, stochastic self-re-entrant flowshops are out of the scope of this study and thus we do not have stochastic parameters to explore.

Mean flowtime and mean waiting time, for the single machine scheduling problem, achieve their minima on same job sequences. However, when variance is added as shown in [64], both criteria have different optimal job sequences. Inclusion of variance in the parameters might lead to different optimal schedules depending on the scheduling criterion. In our case, the parameters are deterministic. Only the effect of neighbouring parameter values vary the performance significantly. Furthermore, it has been noted, minimization of makespan and flowtime increases the variances in the inter-departure times [65]. Thus variability aware objective functions are focus area of current research [65]. The problem considered in this work is not to develop objective functions which optimize for makespan under variation but is to find designs which have less variation compared to other alternative designs.

In [66] it was found during a field study of job shops that the static definition of job shops is far from the dynamic requirements of the job shops in reality. The authors comment that modern scheduling failed to adequately address scheduling in uncertain and dynamic environments. Our contribution takes a step further by learning from performance variation at design time and thus reducing the variation in performance which might occur at runtime. Furthermore, there are two types of approaches to deal with uncertainty: reactive and proactive. Reactive approaches re-plan when an uncertain event has occurred. Proactive approaches foresee uncertain events by (statistical) knowledge from the domain. Our approach falls under the class of proactive approaches. An example of another proactive approach is slack based approaches of [67] that perform ALAP, ASAP analysis to change the processing times of operations such that any unseen circumstances could be accommodated without rescheduling. The authors analyse the effect of using different processing time modification techniques over the mean tardiness of a set of job shop instances. In our case the preference is to not change the processing times but assess variation in performance due to deviations in design parameters.

Parameter	Choices
minimum loop time	{1,5,9}
buffer time	{2,3,4}

Table 5.1: Example of a possible values for design parameters.

Robustness analysis of multiprocessor schedules has been performed by [68]. In the work, robustness is defined as an ability to tolerate fluctuations. The authors quantify how robust a given *Directed Acyclic Graph* (DAG) is and further use metrics to perform robustness analysis. They compute completion time distributions which are derived from stochastic execution times. The deadline for every task in the DAG is used in the computation of robustness. However, they have not studied the impact of change in the parameters that can cause fluctuation. For example, bounded setup times. Contrary to robustness analysis, variation analysis proposed in this chapter aims to find designs of the system such that the performance does not vary. The analysis is based on the knowledge acquired from the models and or the outcomes of the models. Robustness analysis was an alternative to the variation analysis done in this chapter. Robustness analysis requires knowledge of the parameters and their distribution. In many practical cases, most of these distributions are very difficult to find. Additionally, the robustness analysis requires that either one has to take assumptions or perform intensive simulations to learn the statistical properties of the system. A slight change in the system requires to test the consistency of the previous findings.

The variation in performance due to design parameter variability is similar to *process variation* in semiconductor fabrication [69, 70]. The design process itself induces changes to the physical properties of the circuit under construction. Similarly, in self-re-entrant flowshops, change of the design parameters due to the process of building self-re-entrant flowshops causes deviations from the intended design which may lead to a design with lower performance. Both in semi-conductor manufacturing and flowshops it is noticed that reduction of variation is desirable and achievable in early design phases.

5.5 Measuring the variation of a design

Different choices of design parameters like minimum loop time, buffer time, number of machines and number of re-entrances per machine lead to different designs of a self-re-entrant flowshop. The performance of a self-re-entrant flowshop might vary when these design choices vary due to the process of constructing these flowshops. This section presents a method to estimate how much variation in performance will occur if design choices in the design vary.

Consider the parameters of a design for an LSP as shown in Table 5.1. A structural design choice is how much time is minimally spent by a sheet on the self-re-entrant loop of an LSP. This time is denoted as the *minimum loop time*. The minimum loop time can be either of the possible choices $\{1,5,9\}$. Similarly, the time which a sheet can be delayed in the buffer of the self-re-entrant loop called the *buffer time*. The designer of the LSP selects one of the choices to create a design.

Definition 5.3 A ***design*** g is an element from the set of possible designs $\mathcal{G} = \mathcal{P}_1 \times \mathcal{P}_2 \times \cdots \times \mathcal{P}_k$ defined over k sets of structural parameters $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$.

A design $g \in \mathcal{G}$ is a selection for each parameter. Any combination of the parameter values shown in Table 5.1 is a design. $(1,2),(5,3),(9,4)$ are examples from a total of 9 possible designs. The different parameter values lead to different timing constraints. Depending on the

Design	**Jobset**	S_1	S_2	**makespan** $min(S_1, S_2)$	**normalized makespan**
(1,2)	$\mathcal{J}_1$	250	210	210	0.913
(9,4)	$\mathcal{J}_1$	270	230	230	1
(1,2)	$\mathcal{J}_2$	130	180	130	1
(9,4)	$\mathcal{J}_2$	110	160	110	0.846

Table 5.2: Example of makespans for jobsets over different designs.

characteristics of a jobset, a small change in the parameter value might lead to a large variation in performance.

Definition 5.4 A ***step*** is the smallest possible change between two consecutive values of a parameter $\mathcal{P}_i$ and is denoted as ϵ_i. A step is assumed to be discrete and uniform between all values of a parameter.

The example of parameters, namely minimum loop time and buffer time is shown in Table 5.1. The step for minimum loop time is 4 units and for buffer time is 1 unit. A step is the discretization factor of the design space which might be different for different parameters. For different self-re-entrant systems the step can be different depending on the industry, specifications or available components to build a system.

A design can be evaluated for its performance given a jobset using a scheduler. Heuristic schedulers do not guarantee optimality and thus might not be able to fully utilize the resources a design offers. The impact of this limitation of heuristic schedulers can be reduced by using multiple schedulers with different scheduling approaches. For example, heuristics like *Shortest Processing Time first* (SPT) and the heuristic approach described in Chapter 2 find schedules by different approaches. SPT finds a schedule in a greedy fashion for high performance whereas the heuristic approach of Chapter 2 balances the greediness of finding high performing schedules with flexibility of not missing relative due-dates. One of these approaches might find a schedule that utilizes the resources better than others. Therefore the best found schedule by multiple approaches is used to evaluate the performance of a design.

Definition 5.5 The ***makespan*** of a design g for a jobset $\mathcal{J}$ is the makespan of a shortest schedule found by different scheduling approaches.

Consider the example in Table 5.2. There are two designs that are processing two jobsets for which schedules are found using two schedulers S_1 and S_2. The makespan of a design for a jobset is the makespan of the shortest schedule computed by any of the two schedulers. The scheduler S_2 finds the shortest schedule for the jobset $\mathcal{J}_1$ and S_1 for $\mathcal{J}_2$. The design $(1,2)$ achieves higher performance for jobset $\mathcal{J}_1$ compared to the design $(9,4)$. The design $(9,4)$ is however well suited for jobset $\mathcal{J}_2$ as it achieves higher performance compared to the design $(1,2)$. The examples illustrates that there can be designs which are well suited for specific jobsets as well as there can be schedulers which achieve higher system utilization for a jobset. The makespan of a jobset $\mathcal{J}$ on a design g with parameters values $(q_1, q_2, \ldots, q_k)$ is denoted as $\mathcal{C}_{\mathcal{J}}(q_1, q_2, \ldots, q_k)$. For simplicity, when the description of the parameter values of a design is irrelevant, the makespan of a design for a jobset $\mathcal{J}$ is denoted as $\mathcal{C}_{\mathcal{J}}(g)$. For example, for a jobset $\mathcal{J}$, a design g_1 has better makespan than another design g_2 if $\mathcal{C}_{\mathcal{J}}(g_1) < \mathcal{C}_{\mathcal{J}}(g_2)$. Furthermore, for a jobset, the makespan of a design is normalized to assess how well a design performs with respect to other designs.

Design	normalized makespan
(1,2)	0.4
(1,3)	0.5
(5,2)	0.8
(5,3)	1
(5,4)	0.6
(9,3)	0.7

Table 5.3: Example of normalized makespans for a jobset over different designs.

Definition 5.6 The ***normalized makespan*** of a design g is the ratio of the makespan of the design for a jobset $\mathcal{J}$ with the maximum observed makespan for the jobset $\mathcal{J}$ across all designs. It is denoted as $\overline{\mathcal{C}_{\mathcal{J}}}(g)$ and computed as in Equation 5.1.

$$\overline{\mathcal{C}_{\mathcal{J}}}(g) = \frac{\mathcal{C}_{\mathcal{J}}(g)}{\max\limits_{x \in \mathcal{G}} \mathcal{C}_{\mathcal{J}}(x)} \tag{5.1}$$

Consider the normalized makespans for the example in Table 5.2. The designs with normalized makespan 1 took the most time to process a jobset compared to other designs. The smaller the normalized makespan the higher system utilization is achieved as for the same jobset less time was taken. When the parameter values are relevant, the normalized performance is denoted as $\overline{\mathcal{C}_{J}}(q_1, q_2, \ldots, q_k)$. The total normalized makespan is computed in order to assess how a design performs for all jobsets in the set $\mathbb{J}$ of jobsets .

Definition 5.7 The ***total normalized makespan*** of a design is the weighted sum of normalized makespans for the all jobsets over the design and is denoted as $\overline{\mathcal{C}}(g)$.

$$\overline{\mathcal{C}}(g) = \sum_{\mathcal{J} \in \mathbb{J}} w_{\mathcal{J}} \times \overline{\mathcal{C}_{\mathcal{J}}}(g) \tag{5.2}$$

Consider the example in Table 5.2. The total normalized makespan for design $(1,2)$, assuming the weights are 0.5, is $0.5 \times (0.913 + 1) = 0.9565$. Similarly, for the design $(9,4)$, the total normalized makespan is 0.923 indicating that on average design $(9,4)$ achieves higher system utilization given the jobsets of interest. The weights of the jobsets are considered to be the same as, in this example, it is assumed that both jobsets are of equal importance. However, during design space exploration, a designer of a self-re-entrant system can decide the weights of the jobsets based on their importance.

Definitions 5.5, 5.6 and 5.7 describe the makespan of a design with respect to jobsets as well as parameter values which can be used to analyse the variation in performance due to changes in the design parameters. The gradient $\nabla_{g,\mathcal{J}}$ of a design g with parameter values $(q_1, q_2, \ldots, q_k)$ and for a jobset $\mathcal{J}$ measures the rate of change in performance with steps in parameter values using the definitions in the following equations.

$$\nabla^{+}_{g,\mathcal{J}} = \begin{bmatrix} \overline{\mathcal{C}_{\mathcal{J}}}(g) \\ \overline{\mathcal{C}_{\mathcal{J}}}(g) \\ \vdots \\ \overline{\mathcal{C}_{\mathcal{J}}}(g) \end{bmatrix} - \begin{bmatrix} \overline{\mathcal{C}_{\mathcal{J}}}(q_1 + \epsilon_1, q_2, \ldots, q_k) \\ \overline{\mathcal{C}_{\mathcal{J}}}(q_1, q_2 + \epsilon_2, \ldots, q_k) \\ \vdots \\ \overline{\mathcal{C}_{\mathcal{J}}}(q_1, q_2, \ldots, q_k + \epsilon_k) \end{bmatrix} \tag{5.3}$$

Equation 5.3 computes the variation in makespan that occurs due to increments in the values of the parameters. Consider the example of normalized makespans for a jobset in Table 5.3. The positive gradient $\nabla^{+}_{g,\mathcal{J}}$ for the design $(5,3)$ is computed as $[1\ 1]^T - [0.7\ 0.6]^T = [0.3\ 0.4]^T$. The positive gradient measures the rate of change in the normalized makespan when the parameter values increase from the current parameter values of a design.

$$\nabla^{-}_{g,\mathcal{J}} = \begin{bmatrix} \overline{\mathcal{C}_{\mathcal{J}}}(g) \\ \overline{\mathcal{C}_{\mathcal{J}}}(g) \\ \vdots \\ \overline{\mathcal{C}_{\mathcal{J}}}(g) \end{bmatrix} - \begin{bmatrix} \overline{\mathcal{C}_{\mathcal{J}}}(q_1 - \epsilon_1, q_2, \ldots, q_k) \\ \overline{\mathcal{C}_{\mathcal{J}}}(q_1, q_2 - \epsilon_2, \ldots, q_k) \\ \vdots \\ \overline{\mathcal{C}_{\mathcal{J}}}(q_1, q_2, \ldots, q_k - \epsilon_k) \end{bmatrix} \tag{5.4}$$

The variation in the normalized makespan when parameter values decrease from the current values is denoted as $\nabla^{-}_{g,\mathcal{J}}$ and is computed in Equation 5.4. For the design $(5,3)$ in Table 5.3 its negative gradient is $[1\ 1]^T - [0.5\ 0.8]^T = [0.5\ 0.2]^T$. The parameters for which the previous or the next step is not defined, namely, the first and the last parameter value, the performance is computed using the current value of the parameter. For example, for the design $(1,3)$, the first value of the minimum loop times of interest is 1. The negative step from 1 is -3 which does not exist in the set of possible minimum loop times. Still, the negative gradient for design $(1,3)$ is computed as $[0.5\ 0.5]^T - [0.5\ 0.4]^T = [0\ 0.1]^T$. It is assumed that the steps which are outside of the parameter sets are ignored by a designer of a self-re-entrant flowshop because they are not possible or irrelevant for the design process, else, they would have been considered in the parameter set.

$$\nabla_{g,\mathcal{J}} = \begin{bmatrix} \sqrt{(\nabla^{-}_{g,\mathcal{J},1})^2 + (\nabla^{+}_{g,\mathcal{J},1})^2} \\ \sqrt{(\nabla^{-}_{g,\mathcal{J},2})^2 + (\nabla^{+}_{g,\mathcal{J},2})^2} \\ \vdots \\ \sqrt{(\nabla^{-}_{g,\mathcal{J},k})^2 + (\nabla^{+}_{g,\mathcal{J},k})^2} \end{bmatrix} \tag{5.5}$$

The negative and the positive gradient for variation in performance are combined in Equation 5.5 that is the index-wise square root of the squared components of the gradient vectors. The combined gradient vector measures how varying a design is considering increments and decrements simultaneously. For the positive and negative gradient vectors $[0.5\ 0.2]^T$ and $[0.3\ 0.4]^T$ of the design $(5,3)$ the gradient is $[\sqrt{0.5^2+0.3^2}\ \sqrt{0.2^2+0.4^2}]^T$ which equals to $[0.583\ 0.447]^T$. The square root of squared components is performed, as opposed to sum of components, to amplify the variation due to the components and then sum them up. The effect of lesser variation in a component compensating for another component with higher variation is reduced by the squared sum. This leads to a single vector that captures the variations due to all parameters for a design.

Definition 5.8 The ***variation*** $\mathcal{R}_{g,\mathcal{J}}$ of a design g and of a jobset $\mathcal{J}$ is the ℓ^2-*norm* of the gradient $\nabla_{g,\mathcal{J}}$.

The ℓ^2-*norm* of a gradient vector $\nabla_{g,\mathcal{J}}$ is defined as $|\nabla_{g,\mathcal{J}}| = \sqrt{\sum_{i=1}^{k} (\nabla_{g,\mathcal{J},k})^2}$. For example, for a jobset $\mathcal{J}$, given the two gradient vectors $\nabla_{g_1,\mathcal{J}} = [0.05, 0.05, 0.3]^T$ and $\nabla_{g_2,\mathcal{J}} = [0.1, 0.1, 0.2]^T$ their ℓ^2-*norm* is 0.0950 and 0.06 respectively. A design with smaller ℓ^2-*norm* has less variation compared to a design with a larger ℓ^2-*norm* . The design g_1 has a higher ℓ^2-*norm* than g_2 because the third component of the gradient vector of g_1 has significantly higher variation compared to other components. A vector v_1 is smaller than another vector v_2 if every component $v_1(i)$ is smaller than $v_2(i)$. A gradient vector v_1 that is smaller than a gradient vector v_2 i.e. $v_1 < v_2$

Parameter	Value
# of machines	1 (single), 3 (many)
Re-entrance vectors	wafer scanner [1 1 1 1 1], LSP [1 2 2 3], xCPS [1 2 2 2 3]
# of jobs in a jobset	50
processing time	1
minimum loop time	{10, 11, ..., 70}
buffer time	{2, 3, ..., 10}
setup time	0 or 10 times the processing time
deadlines	sum of min. loop time and buffer time
job types	structured repeat or random

Table 5.4: Specifications of the testcases used in experiments.

always has its ℓ^2-*norm* $|v_1|$ smaller than $|v_2|$. During the design space exploration, the designs with smaller ℓ^2-*norm* must be preferred over the designs having higher ℓ^2-*norm* . The overall variation of a design can be computed using the variation of a design for all the jobsets.

Definition 5.9 The ***total variation*** $\mathcal{R}_g$ of a design is the weighted sum of variation for the design for all jobsets in $\mathbb{J}$.

$$\mathcal{R}_g = \sum_{\mathcal{J} \in \mathbb{J}} w_{\mathcal{J}} \times \mathcal{R}_{g,\mathcal{J}} \tag{5.6}$$

The total variation for a design g is computed in Equation 5.6. For example, for a design g and for two jobsets $\mathcal{J}_1$ and $\mathcal{J}_1$ with variations $\mathcal{R}_{g,\mathcal{J}_1} = 0.3$ and $\mathcal{R}_{g,\mathcal{J}_2} = 0.2$ respectively the total variation is 0.25 assuming that the jobs have weights equal to 0.5. The total variation measures the performance of a design for the entire set of jobsets. In the following section, total variation is used to measure variation in the design spaces of self-re-entrant systems.

5.6 Experimental evaluation

The measure described in the previous section is used to assess the amount of variation in performance of self-re-entrant systems. This section uses the measure during the *Design Space Exploration* (DSE) of three self-re-entrant systems, namely, the xCPS platform, an LSP and a wafer scanner. These systems are real-world examples of systems where change in the design parameters cause variation in the system performance. Such systems become inspiration for the test set described in Section 5.6.1. The setup to perform experiments is described in Section 5.6.2. Section 5.6.3 describes experiments to evaluate the quality of the variation measure. The results of the experiments are described in Section 5.6.4.

5.6.1 Test set

The test parameters of self-re-entrant flowshops are described in Table 5.4. The inspiration of the values of the parameters comes from machines like LSP [14], wafer scanner [71] and the xCPS platform [72, 73]. In the testset, a testcase either has a single machine or many. Having many machines with the possibility of setup times might cause large variation in the overall performance. Thus, by having them in the testset ensures that schedulers and designs that perform the best are explored and found. The number of re-entrances on the other hand present the scheduler with

different amounts of freedom. Many re-entrances with appropriate minimum loop time and buffer time might lead to fluctuations in performance due to the interplay of precedence constraints and the arrival of jobs at a machine.

In the testset, the interplay between the number of jobs, the nominal processing times and the minimum loop times determines whether a machine has plenty to do i.e. steady state or is under utilized due to unavailability of work. In the experiments it is assumed that there are 10 jobs, the processing times are 1 and the minimum loop times are between 1 and 20. In such a setting, for many testcases, a machine is under utilized beyond 10 times units of minimum loop time. It suffices to fix two of the these parameters and explore the third one to explore the variation in steady state performance.

The setup times in a testcase require conditional operation to be performed by a machine. Performance variation might arise from presence of setup requirements. When a machine has to process a job different than the current one, a setup is required. The setup times are either 0 (insignificant) or 10 times the processing time. A jobset has two possible types of jobs; whether a job is of the same type as its predecessor or is different. The type is determined randomly or by a structured repeat. In a structured repeat there are three possibilities; one where every third job is different than its predecessor called *Most Different* (MD) or every tenth job is different called *Few Different* (FD) or all jobs are the same called *No Different* (ND). Different jobs might incur setups and a scheduler has a freedom to reduce the setups. Similarly, deadlines arise because of the buffers in a self-re-entrant machine and therefore they are equal to the sum of the processing and the buffer time. Furthermore, deadlines only exist between consecutive operations of a job executing on a machine.

5.6.2 Setup

The proposed method is implemented in Python. The experiments presented in this chapter were performed on a Windows 10 Enterprise Edition running on an Intel i7 at 2.90 GHz. Finding the makespans of jobsets over different designs took 3.42 seconds on average and maximum of 10 seconds. The makespans are input to compute total variation that took on average 55 microseconds and 1 millisecond per design.

In order to measure performance 5 different scheduling strategies were employed. The strategies were *Shortest Processing and Setup-time First* (SPSF), *Longest Processing and Setup-time First* (LPSF), and the heuristic of Chapter 2 with productivity and flexibility as (0.8,0.2), (0.2,0.8) and (0.7,0.3). The best found schedule by these strategies is used to compute the performance of the system. These strategies were employed because finding optimal schedules in reasonable amount of time is intractable. Therefore, due to absence of an optimal schedule, the best found schedule is considered to be a reference point.

5.6.3 Assessment of the variation measure

The proposed variation measure compares a design with others to assess how much variation insensitive a design is. In this section we assess the accuracy of the measure by introduction of random noise to the design parameters and measure the variations in the performance of a design. The noise represents the deviations which might occur during the design process. For example, when the minimum loop time differs from the desired value due to the available motor speeds on the mechanism used to transport products in a flowshop. A variation-insensitive design will have the least fluctuations in its performance when exposed to deviations.

The noise in a parameter is introduced by deviating the required value of the parameter with a few steps. The number of steps is decided based on random selection between a range. The width of the range determines how large deviations are allowed. Considering the entire set of possible values for the parameter means that the deviation will lead to any possible parameter

value. Generally, such a large deviation does not occur, but, for completeness, we include such deviations in our experiments along small deviations which generally occur in practice.

A fluctuation to a parameter q_i is denoted by α_i and is randomly chosen from the set $\mathcal{P}_i$. Then, given vector of random fluctuations $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_k]$ for a design g and a jobset $\mathcal{J}$ the fluctuations in performances is a vector defined as follows.

$$\Delta_{g,\mathcal{J},\alpha} = \begin{bmatrix} \overline{\mathcal{C}_{\mathcal{J}}}(g) \\ \overline{\mathcal{C}_{\mathcal{J}}}(g) \\ \vdots \\ \overline{\mathcal{C}_{\mathcal{J}}}(g) \end{bmatrix} - \begin{bmatrix} \overline{\mathcal{C}_{J}}(\alpha_1, q_2, \ldots, q_k) \\ \overline{\mathcal{C}_{J}}(q_1, \alpha_2, \ldots, q_k) \\ \vdots \\ \overline{\mathcal{C}_{J}}(q_1, q_2, \ldots, \alpha_k) \end{bmatrix} \tag{5.7}$$

The $\Delta_{g,\mathcal{J},\alpha}$, called as *performance deviation vector*, records how the performance of a design varies when each of the parameter is varied by a random deviation. For example, assume $\alpha = [10, 12]$ for $k = 2$ parameters. Then Equation 5.7 is evaluated as $[\overline{\mathcal{C}_{\mathcal{J}}}(g)\ \overline{\mathcal{C}_{\mathcal{J}}}(g)]^T - [\overline{\mathcal{C}_{J}}(10, q_2)\ \overline{\mathcal{C}_{J}}(q_1, 12)]^T$. When the performance deviation vector $\Delta_{g,\mathcal{J},\alpha}$ is a null vector then it indicates that with the applied deviation the performance of the self-re-entrant flowshop does not change. The $\Delta_{g,\mathcal{J},\alpha}$ vector is used to compute the average squared fluctuation in performance in Equation 5.8.

$$\mathcal{A}_{g,\mathcal{J},\alpha} = \frac{1}{k} \sum_{i=1}^{k} (\Delta_{g,\mathcal{J},\alpha}(i))^2 \tag{5.8}$$

The $\mathcal{A}_{g,\mathcal{J},\alpha}$ for a given difference vector measures how much variation in performance is caused due to random deviations. For example, for a difference vector $[0.5\ \ 0.9]^T$ for $k = 2$ parameters the $\mathcal{A}_{g,\mathcal{J},\alpha}$ is $\frac{(0.5)^2 + (0.9)^2}{2}$. The term fluctuation in performance in used to differentiate that the deviation is of random nature as opposed to the variation measure where deviation of a single step is used to assess a design. To assess the variation with respect to a set A of random deviations, the average fluctuation in performance is computed as follows.

$$\mathcal{A}_{g,\mathcal{J}} = median(\{\mathcal{A}_{g,\mathcal{J},\alpha} | \alpha \in A\}) \tag{5.9}$$

The choice of computing median over many deviations, rather than minimum or maximum is empirically made. Choosing minimum makes the fluctuation optimistic. On the other hand, considering maximum makes the fluctuation conservative. Median serves as an alternate indicator of variation due to deviations. Once such variations are considered, the variation due to different jobsets is considered by having a weighted sum of the fluctuations for all jobsets as specified in Equation 5.10. The sum is used to estimate error in the variation measure proposed in this chapter.

$$\mathcal{A}_g = \sum_{\mathcal{J} \in \mathbb{J}} w_{\mathcal{J}} \mathcal{A}_{g,\mathcal{J}} \tag{5.10}$$

A design g_1 is considered to be less varying compared to another design g_2 if $\mathcal{A}_{g_1} < \mathcal{A}_{g_2}$. Ranking the designs with their respective fluctuation provides an estimate of how much the performance of a design varies. This ranking is used to compute the error in the variation measure. The error is computed by comparing how did the variation measure rank a design versus how it was ranked by the weighted sum of fluctuations computed in Equation 5.10. Consider the example of three designs g_1, g_2, g_3. Assume that when sorted with respect to variation measure, their order is g_2, g_3, g_1. The sorted list of designs with respect to weighted sum of fluctuations is g_1, g_2, g_3. Then a *distance vector* between these two lists is $2, 1, 1$. The distance vector indicates the difference in how the variation measure and the weighted sum of fluctuations consider a design

with respect to other designs. A null difference vector will show that the variation measure and the weighted sum of fluctuations consider the designs similar and thus the selection using variation measure leads to designs having the lowest variations compared to other designs. Computation of the weighted sum of fluctuations require a large number of samples to compute random deviations. The variation measure is less compute intensive compared to the weighted sum of fluctuations.

Figure 5.3 shows the *Cumulative Density Function* (CDF) of the distance vectors for the wafer scanner. The plots show the cumulative percentage of the designs for distance of designs. For a design, the distance is the difference of the ranks, computed using the variation metric and the average squared fluctuation of Equation 5.10. A distance of zero indicates that the variation measure and the squared fluctuation considers the variation of the design to be the same. An ideal situation is when the distance vector is a null vector, the CDF will show 100% at distance 0 and cumulatively stays at 100% for the rest of the plot.

Consider the plot shown in Figure 5.3. The y-axis shows the cumulative percentage of indices at a particular distance specified on the x-axis. The plot shows the results of the experiments where random deviations are allowed, from a current parameter value, in steps from a range. For example, the line of $-2,2$ indicates that for each parameter value, a value was randomly picked from $x-2s_k, x-1s_k, x, x+s_k, x+2s_k$ where x is the current value and s_k is the step size. This random selection from a range in the neighbourhood of the current value mimics deviation occurring in a design due to processes.

With the increase of the range of deviation, as the CDF indicates, the percentage of indices with larger distances in the difference vector increases. Having larger distances in the difference vector between the variation metric and the mean squared error indicates that increased number of designs were considered differently. Nevertheless, as the plot shows, at least 60% of the designs had 0 distance in the difference vector. Notice, even for the range $-2,2$ the percentage quickly reaches towards 100%. Thus, the proposed measure achieves similar quality as the median based fluctuation analysis of Equation 5.7.

In the CDF, 62% of the designs are always ranked the same by the variation measure as the median based fluctuation analysis. This is irrespective of how large the range of deviation. This is only possible when designs are insensitive to variation, indicating that a large part of the design space is insensitive. However, the remaining part is where gains from variation-aware designs exist.

In the median based fluctuation analysis, the computation of the weighted sum of fluctuations require a large number of samples to compute random deviations. The variation measure is less compute intensive compared to the weighted sum of fluctuations. Therefore, compared to the median based fluctuation, the proposed measure achieves slightly less quality but without requiring a large number of computations. Therefore, the proposed method is suitable for variation analysis for deviations in design parameters of self-re-entrant flowshops.

5.6.4 Results

The structure of a self-re-entrant flowshop, the scheduler used to compute the schedules and the characteristics of a jobset have an impact over the performance and thus might be sources of variation. Using the variation measure proposed in this chapter, this section finds designs with high performance and with less variation in performance. Additionally, this section explores the potential sources to further investigate the reasons of variation. Several experiments are performed for each source. For the systems under study, namely the xCPS, wafer scanner and the LSP, results of the experiments are shown. The results from one of the systems are presented where the results for each system show similar trends.

The pareto-optimal designs for the design space of the wafer scanner are shown in Figure 5.4. The x and y axis are the structural parameters of the wafer scanner. The plot shows the

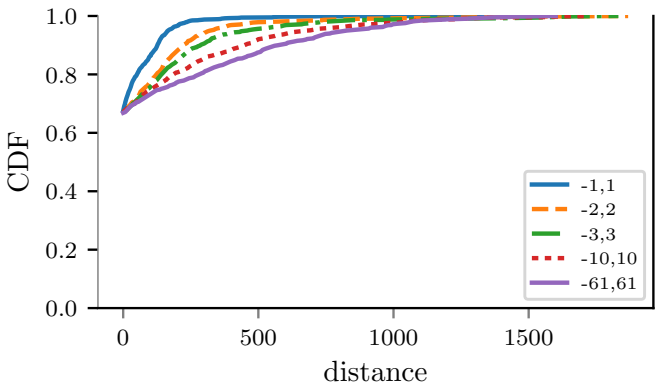


Figure 5.3: CDF of distances for the wafer scanner.

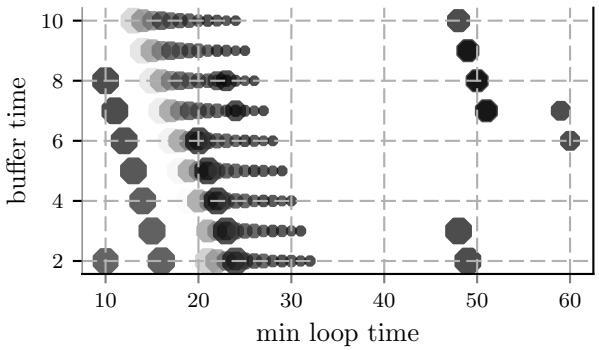


Figure 5.4: The pareto-optimal designs for the design space of the wafer scanner.

pareto-points in 4 dimensions: the minimum loop time, the buffer time, the total performance and the total variation. In the plot, the smaller a design point is the smaller makespan it generates for a job on average. Similarly, the decrease in the brightness of a design point shows the decrease in total variation. Small and dark design points are preferred points as they indicate high performing and least varying designs.

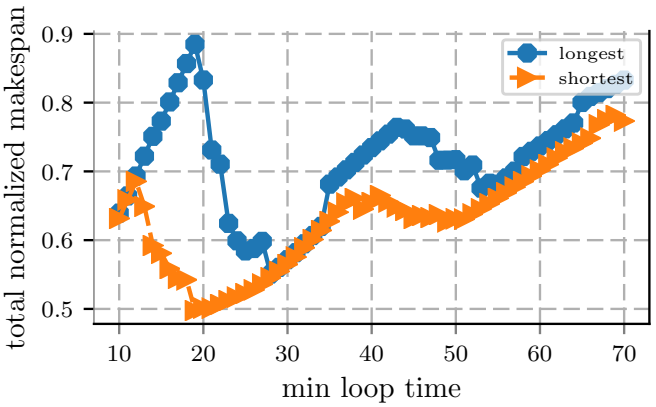
Figure 5.4 shows a general trend of the fact that an additional 2 units of buffer time improves the performance of the wafer scanner reduces variation for the same minimum loop time. This trend can be observed in the figure as the design points for a minimum loop time become smaller in size with an increase of buffer time. As an alternative perspective, increasing the minimum loop time can reduce the minimum buffer time required to achieve the same performance over the same amount of variation. Starting from the left side of the plot in Figure 5.4, with minimum loop time 21, the wafer scanner has larger makespans with high variation. With the increase in the minimum loop time, on average, the variation decreases and the makespan is shorter. For larger buffer times, the same effect of decrease in variation and makespans is observed on smaller minimum loop times. The trend continues till buffer time of 10 time units. Increase of one unit of buffer time leads of having the same performance with the same variation one time unit of

minimum loop time earlier. The constraints of the jobsets give an explanation to this trend due to the structural parameter values and why they are pareto-optimal. As specified in the testset details, there are five operations of each of the 50 jobs being processed in the scanner. These operations have a processing time of one time unit each. The loop with minimum loop time 21 can interleave, at most, 21 operations before the second operation of the returning job is performed. Even in the MD category consisting of the most different jobs, where every third job is different with job types specified as a regular expression $(1,0,0,1,0)^{+}$, interleaving operations from the first job leads to a high performance schedule as the operations from the first job fills up the loop with timings $10*2$ for processing and setups of different operations plus 5 for same operations summing to 25 time units. The highly varying designs on minimum loop time of 21 and buffer time of 2 time units is because some of the jobs can achieve high performing schedules, and for many jobs, high performance schedule cannot be achieved. Addition of more jobs to the interleaving will result in additional setup which the buffer time cannot facilitate and thus is not a solution. Having only one job in the loop incurs larger makespan for the complete jobset and thus has less performance than the pareto-optimal design found.

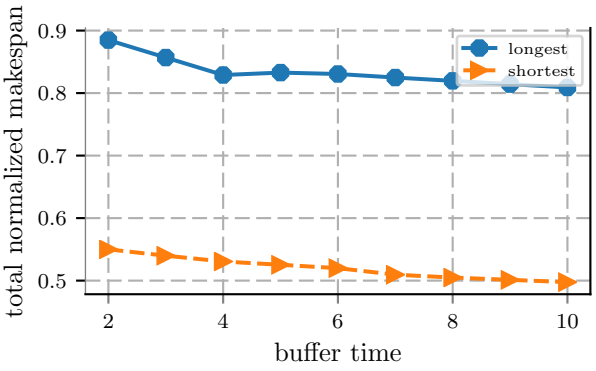
There are exceptions to the general trend of Figure 5.4 at the extremes of the design space. For example, the combination of 10 units of minimum loop time and 2 units of buffer time is an exception. Notice that it has low total performance. It is because the combination of the minimum loop time and the buffer time does not allow interleaving of two jobs because interleaving requires at least two setups which require minimum loop time plus the buffer time to be at least 20 time units for interleaved operation. Similarly, minimum loop times larger than 47 time units only have low total performance and least varying designs. This is possible because processing a wafer separately is always possible. However, when interleaving becomes possible, similar performance can be achieved on smaller minimum loop time and thus are not pareto-optimal.

The trend of variation in performance depends on the type of a design parameter. Analysis of the trend provides an insight on impact of the parameter over the system performance. Such an impact of the structural parameters over the variation in performance for the LSP is shown in Figure 5.5. The impact of the increase in the minimum loop time over the total performance is shown in Figure 5.5a. Initially over a smaller minimum loop time, there are many jobsets for which effective interleavings are not facilitated by the minimum loop time and buffer time combination. However, for many others, such a combination is possible where high performance can be achieved leading to a high total performance. This leads to a large variation between the total normalized makespans due to the shortest and the longest makespans found for a jobset. On larger minimum loop time the variation in total performances decreases because the combinations of the minimum loop times and the buffer times for most of the times permits a high performance interleaving. Note that on larger minimum loop times, the maximum total performance is also less than the smaller minimum loop times. The additional capacity offered by the combination of the minimum loop time and the buffer time is not utilized by a scheduler as interleaving more operations leads to timing constraints which cannot benefit from the additional capacity. The trend continues to a point where additional minimum loop time allows shorter makespans for some jobs. This pattern repeats but with a smaller magnitude of variation. On the other side, large minimum loop times lead to larger total normalized makespan because larger loops require more travel time for sheets in the LSP.

Similarly, Figure 5.5b shows the impact of the buffer time over the variation in total performance of the LSP. The increase in the buffer time decreases the variation in total performance as well as produced smaller makespans. Additional buffering capacity might allow interleaved schedules for jobsets for which interleaving was not possible before which in turn decreases the total normalized performance. Thus a decrease in makespans for both the shortest and the longest schedules for a jobset is observed.



(a) The variation due to minimum loop time.



(b) The variation due to buffer time.

Figure 5.5: The impact of changing the structure of an LSP over its total performance.

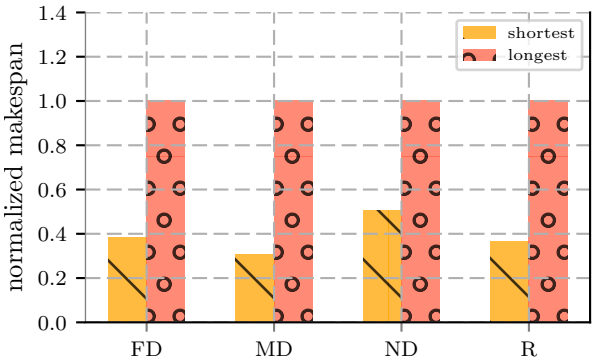


Figure 5.6: The variation in performance due to different job types for the xCPS.

The variation due to different job types in a jobset in the xCPS platform is shown in Figure 5.6. On the x-axis, four job types shown. They are FD, MD, ND and *Random (R)*. These job types indicate how different types of jobs are there in a jobset. The FD has the job types as in regular expression $(1,0,0,0,0)^+$. The category MD has job types as in regular expression $(1,0,0,1,0)^+$. The category, *Random*, contains job whose type is determined randomly.

The jobsets with most setups have the most variation as the difference between the normalized minimum and the normalized maximum performance is the largest. The variation is due to the fact that with most different job types, the chances of incurring setups are also high. Suitable minimum loop time and buffer time leads to timing constraints which a scheduler can exploit and optimize to achieve higher performance. However, for many designs such freedom is not available to a scheduler and thus the performance might be lower than the other cases.

The second most variation is in the FD job type. The amount of setups due to few different jobs is also small. Therefore, the optimization freedom to a scheduler is also less leading to smaller variation in performance. The least amount of variation is observed in the no different job type. Since all jobs are of the same type, setups cannot occur. The scheduling freedom is very less for such job types and thus the variation in performance is also smallest. Note the fact that even tough there are no setups, a scheduler still has the freedom to schedule operations of the same type in different orders. These differences of order still produce variation but relatively smaller than the variation produced due to setups.

The random job type has a number of differences between the number of differences in the FD and the MD categories. The variation in performance for the random type is less than the MD category but is significantly more than ND category. Upon inspection, it is found that the R category had at most the same number of different jobs in a jobset as the MD category. Thus, the variation in performance is also almost the same.

The normalized performance achieved by different schedulers for all designs and all jobsets is shown in Figure 5.7. Five different scheduling strategies were used because of absence of an optimal scheduler. These strategies have different ways to find schedules and a best found schedule is considered as an indicator of how a given design can process a jobset. In case an optimal scheduler with tractable runtime existed then it would have been known, for every jobset, how a design performs the best.

The box-whisker plot shown in Figure 5.7 illustrates which scheduling strategies resulted in finding shorter makespans. Each box has the normalized makespan values for all jobsets over all designs scheduled using the respective scheduling strategy. The dotted line inside the box depicts

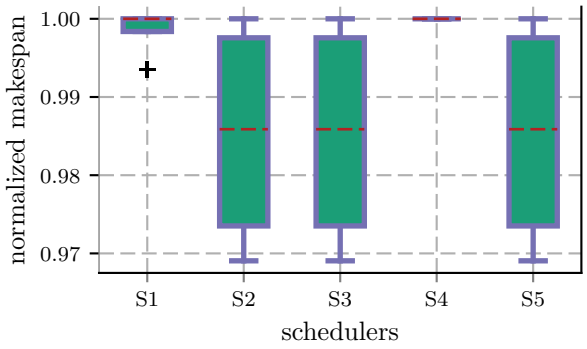


Figure 5.7: The variation in performance due to different schedulers for the xCPS platform.

the median normalized performance. A lower median indicates that the scheduling strategy has found shorter makespans for a jobset compared to other scheduling strategies. The strategy, S4, consistently performed the worse compared to other strategies because the schedules it found were the longest among other schedules for a jobset. Therefore, the median, minimum and maximum (the whiskers are absent) are 1. Other scheduling strategies have sometimes found a schedule for a jobset which was better than others. The plot shows that different scheduling strategies introduce variation in performance as a strategy might be well suited for a design or not.

The experiments of this chapter has shown that designs with higher performance and less variation due to parameter deviation exist. This is mainly due to buffers. Larger buffers provide freedom to find high performing schedules. When the size of buffers is not enough, the performance of the system may reduce. The experiments have also shown two other sources of variation. The number of different jobs in a jobset is one of them. The larger the number of different jobs the higher the variation is. The second source of variation is different heuristics used to compute the schedules. On average, the strategies that do not find best schedules have less variation compared to the others. At design time, the designer of a self-re-entrant system must consider these sources of variations.

5.7 Conclusions and Future work

The variation in the performance of self-re-entrant systems has several sources. The characteristics of a jobset, the structure of the system and the scheduling algorithm used to compute the schedules influence the performance and causes variation. The construction process might deviate the design of a system from the desired structural parameters. Due to this deviation, the neighbouring designs of a design might have very low performance. Thus the current design might not be ideal considering deviations. At design time, upfront knowledge of these highly varying designs can lead the designer of self-re-entrant systems to consider the variation.

This chapter introduced a measure of variation in the performance of a design due to change in design parameters. The measure utilizes the knowledge of the variation occurring when the values of structural parameters are increased or decreased. Using the measure, a case study was performed to explore the relation of different sources of variation and performance over three self-re-entrant systems, namely, the xCPS platform, a wafer scanner and an LSP. The DSE is performed in a space with 4 dimensions: total normalized makespan, total normalized variation, minimum loop time and buffer time. The pareto-optimal designs in the 4D space, for most of

the designs, have a trend common across the self-re-entrant system under study. The dominant trend is that a design operates at high performance when interleaving is possible. The interleaved operation increases the system utilization. Many of the designs, which do not follow the general trend, are cases where such an interleaved operations is not possible. Those designs have low performance but also less variation. The variation is less because such designs consistently perform poorly.

Different structural parameters influence the performance of a self-re-entrant system differently. The increase in the minimum loop time increases the variation for several steps and then decreases periodically. Every occurrence of the period reduces the magnitude of variation and eventually the variation becomes significantly compared to initial magnitude. Additionally, beyond certain a value, the system performance becomes poorer when additional minimum loop time is added to designs. Therefore, during the design phase the impact of the minimum loop time must be explored. Similarly, it must be analysed whether a scheduler can utilize the additional buffer capacity.

The jobsets with the most differences have the most variation in normalized performance. The jobsets without different job types show the least variation indicating that setups are the reason of variation. When a scheduler optimizes and reduces the number of setups high performance is achieved. When setups cannot be avoided due to the constraints or due to the available resources, e.g. the buffer times, low performance is yielded. These sources, all together contribute to variation in performance for self-re-entrant flowshops.

As a future work, positive variation, that is a type variation where the performance of a system improves, can be considered beneficial. A measure could be designed that favours positive variation. A designer might only want to disregard designs where the performance degrades. Any improvement in performance, even due to deviations might be acceptable. Such a measure can be useful where system performance is more important than predictability.

Variation can be classified as static and dynamic variation. Static variation is due to the choices at the design time. Dynamic variation occurs due to the environment in which a system operates as well as system configurations. Modelling the dynamic variation might increase the accuracy of the variation measure.

The variation measure can be extended for continuous design parameters. Such an extension will alleviate the requirement of a discrete step. Though the computation of the performance for such a continuous design parameter might become compute intensive.

"Every new beginning comes from some other beginning's end" - Seneca

6 Conclusions and Future work

This thesis has described techniques to schedule and perform variation-aware design of self-re-entrant systems. Section 6.1 concludes this thesis. The future work is described in Section 6.2.

6.1 Conclusions

The problem of scheduling self-re-entrant flowshops with sequence-dependent setup times and relative due-dates is known to be NP-Hard. This thesis has shown that the problem remains to be NP-Hard even when the order of jobs is pre-determined. The proof of complexity is a reduction of the *source to target Travelling Salesman Problem* (st-TSP) to the decision version of the scheduling problem. In contrast, finding an optimal schedule for classical flowshops with a fixed job order can be done in polynomial time. This work shows that finding optimal solutions efficiently is highly unlikely due to self-re-entrancy together with sequence dependent setup times and due-dates. Therefore, for such flowshops, a quick method to find good quality schedules is required.

A heuristic approach to schedule self-re-entrant flowshops with sequence dependent setup times and relative due-dates has been described in Chapter 2. The heuristic uses the scheduling-space to find possible scheduling alternatives. The alternatives are assessed by two metrics, namely, productivity and flexibility. Using these metrics, the heuristic finds a schedule by constructing a path from the initial state to the final state in the scheduling-space satisfying the processing time, setup time, relative due-date and ordering constraints. The performance of the heuristic is assessed by testing over a randomly generated testset that is inspired from benchmarks from the literature and industrial testcases. The experiments show that median ratio between the schedules generated by the heuristic and the estimated lower bounds is 6%. A small fraction, i.e. 0.40%, of the schedules generated by the heuristic are between 0.85 times and 2.15 times longer than the estimated lower bounds.

A specialized heuristic is presented in Chapter 3 that schedules an *Large Scale Printer* (LSP). The heuristic finds, on average, better schedules than the general heuristic of Chapter 2. The schedules are better because the heuristic explores the solution space differently. During the

exploration of the solution space, for an operation, the specialized heuristic considers more interleaving choices compared to the general heuristic. However, this increase in quality of schedules comes over the price of increased runtime. The specialized heuristic is a greedy strategy which ranks local scheduling decisions and enforces the choice that ranks the best. The ranking is performed using three metrics, productivity, flexibility and distance. The performance of the heuristic is evaluated by experiments on a testset consisting of schedule requests of industrial relevance. The experiments show that the heuristic out-performs a greedy version of the state-of-the-art *Modified Nawaz Enscore and Ham* (MNEH) heuristic in all cases. Moreover, the heuristic out-performs the Eager scheduler, i.e. the scheduler of the LSP, for 92% of the cases. On average, the specialized heuristic finds better schedules for all categories where significant number of sequence dependent setups occur. The Eager scheduler outperforms the heuristic on the testcases that do not have sequence dependant setups. Both the specialized and the general heuristics contribute in finding better schedules for self-re-entrant flowshops.

Performance improvement for a self-re-entrant flowshop with setup times, due-dates and fixed job order can also be achieved by exploring its design space. The design parameters of such flowshop influence its performance. Traditional approaches to estimate performance are either too slow or do not take sequence dependent setup times into account. The performance estimator described in Chapter 4, for regular jobsets, allows to explore the relation between the design parameters and performance without using compute intensive algorithms. As shown by the LSP case study, the maximally productive length of the re-entrant loop highly depends on the jobsets under consideration. For the LSP the estimator has an average compute time of 1.1 milliseconds with an average accuracy of not less than 96%. The accuracy and the fast computation of the estimator facilitate a designer to evaluate structural decisions during the design of self re-entrant flowshops with sequence dependent setup times and due-dates.

The variation in the performance of self-re-entrant systems has several sources. The characteristics of a jobset, the structure of the system and the scheduling algorithm used to compute the schedules influence the performance and cause variation. The construction process might deviate the design of a system from the desired structural parameters. Due to this deviation, the neighbouring designs of a design may have very low performance. Thus the current design may not be ideal. At the design time, upfront knowledge of these highly varying designs can lead the designer of a self-re-entrant system to consider the variation. Chapter 5 describes a measure of variation in the performance of a design due to changes in design parameters. The measure utilizes the knowledge of the variation that occurs when the values of structural parameters are increased or decreased. Using the measure, a case study was performed to explore the relation between different sources of variation and performance over three self-re-entrant systems, namely, a research platform called as *eXplore Cyber Physical Systems* (xCPS), a wafer scanner and an LSP.

The case study is performed in a design space with 4 dimensions: total normalized makespan, total normalized variation, minimum loop time and buffer time. The pareto-optimal designs in the 4D space, for most of the designs, have a trend common across the self-re-entrant system under study. The dominant trend is that a design operates at high performance when interleaving of products is possible. The interleaved operation increases the system utilization. Many of the designs, which do not follow the general trend, are cases where such an interleaved operation is not possible. Those designs have low performance but also less variation. The variation is less because such designs consistently perform poorly.

6.2 Future work

The computational complexity of the LSP scheduling problem of Chapter 3 is unknown. The reduction specified in Chapter 2 is not applicable to the LSP scheduling problem. Further research is required to find out whether this case, where one machine processes the jobs twice has an efficient algorithm to find optimal schedules or not. A starting point might be to reduce the set partition problem to the LSP scheduling problem.

The development of dedicated heuristics, for self-re-entrant systems, is a necessity as the algorithms to schedule such systems are usually compute intensive. However, the time that is spent to create such heuristics is usually enormous. A person either requires tons of experience to quickly develop practically viable solutions or simply needs a lot of time to master such a skill. Re-use of heuristics can be performed, for example, when two scheduling problems are similar. The question is how to assess the similarity? As a future work, it is proposed to compare the scheduling-spaces of two scheduling problems to find out what are the differences in scheduling choices. Once the difference is known, heuristics can be tuned to find schedules considering the differences.

The Bellman-Ford algorithm is the core of the scheduling techniques presented in this thesis. The algorithm is used to find a schedule, once an ordering of vertices is given. Additionally, the algorithm is also used to verify the feasibility of the scheduling alternatives. In many experiments, e.g. in [47], it is observed that the algorithm consumes between 60%-80% of the runtime. It is of high interest to optimize the Bellman-Ford algorithm for runtime. Software optimization techniques such as loop unrolling and the usage of dedicated hardware are candidates.

Computation of lower bounds over the makespan of a schedule for a self-re-entrant system is proposed. These lower bounds can be used for assessing the performance of a scheduler over the testcases for which the exact methods require intractable amount of time. One such lower bound is to use the distances computed by the Bellman-Ford algorithm on the partial schedules. However, such a lower bound is not tight as it ignores the possible setup times that may arise when operations are totally ordered. There will be at minimum $x-1$ number of setups if there are x different sheets in a jobset. Thus, considering these setups together with the information from the timing constraints will result in a tighter bound making it possible to better assess the quality of schedules generated by the heuristic.

As an extension to the heuristic, back propagation can be added. Back propagation might allow reversal of choices that the heuristic made after further exploration of the scheduling-space. Such an extension might be very useful for the testcases where the heuristic aborts because no feasible choice is found.

The scheduling-space can be extended to self-re-entrant flowshops without a fixed job order. Such an extension will make the heuristic proposed in Chapter 2 applicable to other systems where fixed job order is not a requirement.

Recently, the heuristic proposed in Chapter 3 was extended by [46] by addition of windowed scheduling. The work schedules a part of a constraint graph and later on schedules the rest. In this way, the runtime of the heuristic is less than scheduling the complete constraint graph. In [47], with several runtime optimizations, the extended version of the heuristic of [46] is shown to have a worst case runtime of 325ms. Furthermore, [47] has extended the heuristic to cases where a job request comprises of simplex and duplex sheets. Due to this extension, jobs either have three or four operations. At the moment, in this thesis and in work of [46, 47], it is assumed that the timing constraints are given. As a future work, generation of timing constraints from an LSP at runtime is proposed. The constraints depend on the characteristics of jobs and such information only becomes available once the jobset arrives at the LSP. Moreover, the timing constraints need to be acquired from different components of the paper path of an LSP. Generating the timing constraints at runtime poses an additional challenge of extracting constraints in an efficient way.

The performance estimator assumes that a jobset consists of repeating patterns of jobs. The assumption holds in many industrial applications but estimation of performance for jobsets without a pattern is left as a future work. Furthermore, the design parameters explored in this work were the minimum loop time and buffer time because they significantly influence the performance. The study of which other design parameters influence the performance is another interesting topic for future work.

Positive variation, that is a type of variation where the performance of a system improves, can be considered beneficial. A measure could be designed that favours positive variation. A designer might only want to disregard designs where the performance degrades. Any improvement in performance, even due to variation might be acceptable to some applications. Such a measure can be useful where system performance is more important than predictability. This thesis has assumed that predictability is important to save unexpected costs of storage and logistics of products produced by self-re-entrant systems.

Variation in performance can be classified as static or dynamic variation. Static variation is due to the choices at the design time. Dynamic variation occurs due to the environment in which a system operates as well as system configurations. Modelling the dynamic variation might increase the accuracy of the variation measure proposed in Chapter 5.

The variation measure described in Chapter 5 can be extended for design parameters with continuous values. Such an extension will alleviate the requirement of a discrete step. Though the computation of the performance for such a continuous design parameter might become compute intensive.

The extensions proposed as future work will increase the applicability of the contributions in improving the performance of self-re-entrant flowshops.

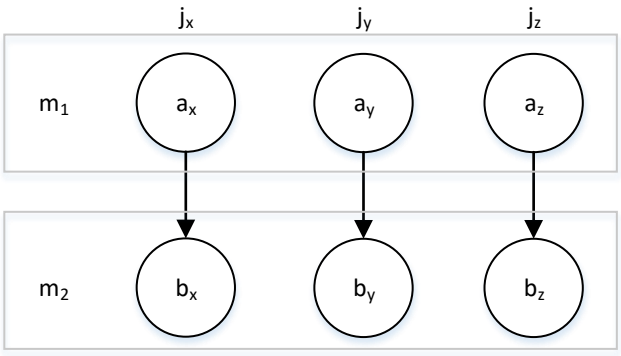
A

Flowshops with fixed job order

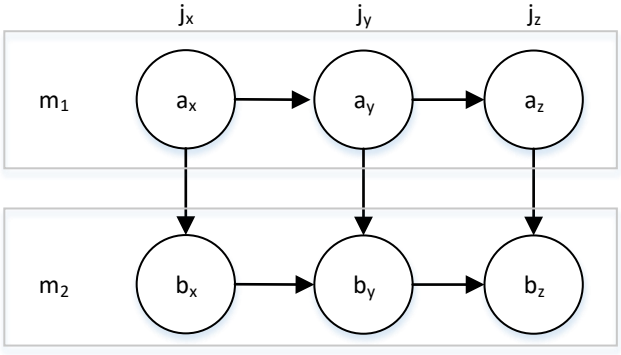
Finding optimal schedules for classical flowshops and self-re-entrant flowshops is known to be NP-Hard. Chapter 2 has shown that self-re-entrant flowshops remain to be NP-Hard even when the job order is fixed. However, when job order is fixed for classical flowshops, optimal schedules for them can be found using a polynomial time algorithm. An intuitive argument is given through the following example.

Three jobs x, y, z having two operations each, namely a and b, are executing over two machines m_1, m_2 in Figure A.1. Lets first consider the case in Figure A.1a where the job order is not fixed. In such a case, a scheduler needs to determine the order in which each machine processes the operations. The scheduler on machine m_1 needs to determine the order in which the operations a_x, a_y, a_z should be performed. Similarly, machine m_2 requires to order its operations.

The second case is when the job order is fixed as shown in Figure A.1b. The order in which operations are performed on machines is also known and fixed. On machine m_1 the order of operations is $a_x \rightarrow a_y \rightarrow a_z$. The remaining task for the scheduler is to compute the start times. Using Bellman-Ford algorithm, as described in this work, as soon as possible start times can be computed. The as soon as possible start times are optimal because any delay in the start of any operation will never decrease the makespan of the schedule.



(a) Operations of jobs.



(b) Operations of jobs with fixed job order.

Figure A.1: Operations over machines with non-fixed and fixed job order.

B

SMS-SDET is NP-Complete

Though *Single Machine Scheduling with Sequence Dependent Execution Times* (SMS-SDET) is accepted in literature [21, 29] to be NP-Complete, its proof of computational complexity is not found. Therefore, for completeness, the proof is described in this appendix.

SMS-SDET consists of Tasks $T = \{t_1, \ldots, t_n\}$, Processors $P = \{p_1\}$, Execution times $E : T \rightarrow \mathbb{Z}^+$ where $E(t_i, t_j)$ is the processing time of a task t_j if t_j is processed immediately after a task t_i. The problem is to find a schedule S such that every task t_i has start time $S(i)$ satisfying *with* $\max_{1 \leq i \leq n} S(i) \leq d$ where d is a given integer. The problem is to decide whether a schedule can be found that fits in d time units. The SMS-SDET is NP-Complete as st-TSP polynomially transforms to SMS-SDET. st-TSP is defined as given a graph $G = (V, A)$ with a set of vertices $|V| = n$, a source vertex s and a target vertex t in V, is there a *hamiltonian path* starting from s and ending at t? st-TSP is already known to be NP-Complete [29].

st-TSP is in NP as a path from the source to the target can be guessed non-deterministically. A polynomial time verifier exists that given a path finds whether the path starts from the source and ends in the target while visiting all nodes. Similarly, in case the path does not visit all nodes or does not start at the source or does not end in the target node then such a path is rejected in polynomial time.

st-TSP ($G(V, A)$) polynomially transforms to SMS-SDET. Given an arbitrary instance of st-TSP problem a SMS-SDET instance is created as follows. $T = \{t_s, t_1, \ldots, t_n, t_t\}$. $P = \{p_1\}$ *and* $d = n + 2$. *Execution time are* $E(t_s, t_1) = 1$, $E(t_n, t_t) = 1$, $E(t_i, t_j) = 1$ *for all* $(i, j) \in A$ *and* $E(t_i, t_j) = d + 1$ *for others. Then a schedule with makespan* $\leq d$ *only exists if and only if there exists a hamiltonian path between s and t.*

C
Benchmarks

Most of the testsets for experiments in this work have inspired from industrial testcases and testcases from literature. In the research community, mainly, there exist 3 benchmarks to test shop scheduling problems. Namely they are Taillard [74], Structured [75] and ORLib [76] benchmarks. These benchmarks have different goals. The Taillard benchmark is the most used benchmark in the literature. The benchmark targets permutation flowshop, flowshop, open shop and job shop scheduling problems. The original benchmark does not have setup times though later on researchers have extended the benchmark to add setup times. Similarly, the benchmark does not have relative due-dates and nor it was extended for relative due-dates prior to our work. For details and download of the benchmark refer to [74].

The second benchmark for shop scheduling problems is the structured benchmark. As the name suggests, the goal of the benchmark is to have test instances having a structure additional to the testcases with random instances. It contains all the instances from the Taillard benchmark. Further details and the benchmark can be found at [75].

The third benchmark focuses on collecting test cases found difficult (time consuming, poor quality) to solve in industry. We refer to this benchmark as ORLib benchmark. It contains instances from Taillard benchmark and testcases from industry. Benchmark can be found at [76].

Taillard benchmark was the most used in the shop scheduling research community and therefore it has served as the main source of inspiration. For example, the number of jobs, the interval in which processing times are assumed and the number of machines in an shop environment are considered when creating testcases for this work. However, for the testcases where industrial application of this work is tested, for example in the LSP case, the benchmark focuses on the testing for the application area.

D

Remarks over scheduling-space

Several observations over the structure of the scheduling-space of self-re-entrant flowshops are described in this appendix. These observations can be used in extension of the future heuristics for the problem. The observations are summarized in the following remarks.

R The total number of distinct orders of operations, for a given jobset, is a Catalan number of n where n is the number of jobs in the jobset.

R The structure of the self-re-entrant flowshop scheduling problem is the same as the Dyck Paths problem. An order of operations is similar to a walk over a Dyck Path.

There are Catalan number of Dyck Paths (analogous to orders) for a scheduling-space of a jobset but many of them are infeasible due to due-dates.

R With a fixed job order, a jobset with c distinct jobs will have at least c setups even in the optimal case.

If there are c distinct jobs, there is a setup required when a machine starts processing another job. Since all jobs need to be processed, c setups are always required.

R The Bellman-Ford algorithm has been used for infeasible schedule detection. Note that, situations like cyclic orderings of operations and due-dates across machines will not always be detected by the algorithm.

The cyclic orderings are not detected because Bellman-Ford is not a cycle detection algorithm. There can be cycles, for which, the bellman algorithm might still terminate with no further change in the distances.

The cross machine due-dates might arise in specific problems but not in the LSP scheduling problem. Thus, for such problems explicit check of cross machine due-dates is required. The Bellman-Ford algorithm does not detect infeasibility in case where the constraints are between

two operations which are unreachable from each other. Therefore, they are not ordered leading to the fact that there is no path between them. Having no path between them will not let a positive cycle to be formed and thus the infeasibility will not be detected.

R On small loops, the buffers might not be effective.

On loops that have their minimum loop times smaller than the setup time, processing all operations of the same job and then processing the next job becomes a viable heuristic for a good quality schedule. Such a heuristic does not utilize the buffer, thus, the impact of the buffer is not observed over the performance of the system.

Acronyms

LSP	*Large Scale Printer*	4, 32, 37, 53, 74, 93
st-TSP	*source to target Travelling Salesman Problem*	9, 18, 93
MNEH	*Modified Nawaz Enscore and Ham*	9, 16, 40, 67, 94
DSE	*Design Space Exploration*	10, 53, 83
xCPS	*eXplore Cyber Physical Systems*	10, 53, 77, 94
JH	*Johnson Heuristic*	16
EJ	*Extended Johnson*	16
WSPT	*Weighted Shortest Processing Time first*	17
SMS-SDET	*Single Machine Scheduling with Sequence Dependent Execution Times*	17, 99
PCS	*Precedence Constrained Scheduling*	17
PCS-U	*PCS with Unit execution times*	17
ASAP	*As Soon As Possible*	20
LTS	*Labelled Transition System*	22
DAG	*Directed Acyclic Graph*	79
SPT	*Shortest Processing Time first*	80
MD	*Most Different*	84
FD	*Few Different*	84
ND	*No Different*	84
SPSF	*Shortest Processing and Setup-time First*	84
LPSF	*Longest Processing and Setup-time First*	84
CDF	*Cumulative Density Function*	86

List of Symbols

$\mathcal{M}$	A set of machines	15, 38, 55, 76
μ_i	The i^{th} machine	15, 38, 55, 76
$\mathcal{J}$	A jobset	15, 38, 55, 76
$\mathcal{V}$	A flow vector	15, 38, 55, 76
m	Number of machines	15, 38, 55
$\mathcal{O}_i$	A set of all operations of job i	15, 39, 55, 77
j_i	The i^{th} job	15, 39, 55, 77
$\mathcal{O}_\mathcal{J}$	A set of all operations	15, 39, 55, 77
$o_{x,y}$	The y^{th} operation of the job j_x	15, 39, 55, 77
$p(x,y)$	The processing time of $o_{x,y}$	15, 39, 55, 77
$s(x,y,u,w)$	The setup time between $o_{x,y}$ and $o_{u,w}$	15, 39, 55, 77
$d(x,y,u,w)$	The due-date between $o_{x,y}$ and $o_{u,w}$	15, 39, 55, 77
$\mathcal{S}(x,y)$	The start time of operation $o_{x,y}$	15, 39, 55, 77
$\mathcal{S}$	A schedule	15, 39, 55, 77
$\mathcal{C}_\mathcal{J}$	The makespan of the jobset $\mathcal{J}$	16, 39, 77
n	Number of jobs in a jobset	16, 57
r	Number of operations in a job	16
cg	A constraint graph	22, 40, 107
V	The set of vertices in cg	22, 40
E	The set of edges in cg	22, 40
w	Weights of edges in cg	22, 40
G	A function to determine group of a vertex in cg	22, 40
(Q,Σ,δ)	The scheduling space of a self-re-entrant flowshop	24, 108

Q	The set of states in (Q, Σ, δ)	24
Σ	The set of labels in (Q, Σ, δ)	24
m	Number of groups in cg	40
f	A flowshop	55
ω	A word representing a pattern of jobs	56
γ	The number of times a job re-enters a machine	57
$t_{s,\mathcal{J}}$	Timespan of a slot for a jobset $\mathcal{J}$	58
$\eta_{m,f,\mathcal{J}}$	The estimated performance	58
MK	Duration of jobs processed in a steady state	59
FMK	Duration of a jobset having a finite number of jobs	59
$\eta'_{m,f,j}$	Performance of a finite length jobset	60
χ	Probability distribution for weights of jobsets	60
$\mathbb{J}$	The set of all jobsets	77
g	A design	79
$\mathcal{G}$	A set of designs	79
$\mathcal{P}_i$	Set of possible values of the i^{th} parameter	80
ϵ_i	The step of the i^{th} parameter	80
$\mathcal{C}_{\mathcal{J}}(q_1, q_2, \ldots, q_k)$	The makespan of a design for a given jobset and parameter values	80
$\mathcal{C}_{\mathcal{J}}(g)$	The makespan of a design g	80
$\overline{\mathcal{C}_{\mathcal{J}}}(g)$	The makespan of a design g	81
$\overline{\mathcal{C}}(g)$	The makespan of a design g	81
$\nabla_{g,\mathcal{J}}$	Rate of change in makespan with steps in parameter values	81
$\nabla^{+}_{g,\mathcal{J}}$	Rate of change in makespan with a positive step in parameter values	82
$\nabla^{-}_{g,\mathcal{J}}$	Rate of change in makespan with a negative step in parameter values	82
$\mathcal{R}_{g,\mathcal{J}}$	The variation of a design g and a jobset $\mathcal{J}$	82
$\mathcal{R}_g$	The total variation for a design g	83
q_i	A value for the i^{th} parameter	85
α_i	A fluctutation to i^{th} parameter	85
α	vector of random fluctuations	85
$\Delta_{g,\mathcal{J},\alpha}$	vector of fluctuations for design g and jobset $\mathcal{J}$	85, 108
$\mathcal{A}_{g,\mathcal{J},\alpha}$	Average fluctuation in performance of a design g for a jobset $\mathcal{J}$ and random deviation vector $\Delta_{g,\mathcal{J},\alpha}$	85
A	A set of random deviations	85
$\mathcal{A}_{g,\mathcal{J}}$	Average fluctuation in performance of a design g for a jobset $\mathcal{J}$	85
$\mathcal{A}_g$	Average fluctuation in performance of a design g	85

Bibliography

[1] *Assembly lines of technical gadgets.* URL: http://www.wisegeek.com/what-is-an-assembly-line.htm (cited on page 2).

[2] *The Ford assembly line.* URL: http://estefanyluv.blogspot.nl/ (cited on pages 2, 4).

[3] K.R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley & Sons, 1974 (cited on page 3).

[4] J.C.A.C. Alegre. "Performance Measurement on Automotive Assembly Line". PhD thesis. University of Porto, 2012 (cited on page 3).

[5] H.J. Shin. "A dispatching algorithm considering process quality and due dates: an application for re-entrant production lines". In: *The International Journal of Advanced Manufacturing Technology* 77.1-4 (2015), pages 249–259 (cited on pages 4, 13, 14, 53, 54).

[6] W.L. Pearn, S.H. Chung, M.H. Yang, and K.P. Shiao. "Solution strategies for multi-stage wafer probing scheduling problem with reentry". In: *Journal of the Operational Research Society* 59.5 (2008), pages 637–651 (cited on pages 4, 13, 14, 17, 54, 56, 66).

[7] H.J. Shin and Y.H. Kang. "A rework-based dispatching algorithm for module process in TFT-LCD manufacture". In: *International Journal of Production Research* 48.3 (2010), pages 915–931 (cited on pages 4, 13, 14, 53, 54).

[8] *Vario Print i300.* URL: http://www.canon.nl/for_work/products/professional_print/digital_colour_production/varioprint_i300/ (cited on pages 4, 32).

[9] *Human size comparison.* URL: https://upload.wikimedia.org/wikipedia/commons/b/b1/Human-staurikosaurus_size_comparison.svg (cited on page 4).

[10] H. Emmons and G. Vairaktarakis. *Flow shop scheduling: theoretical results, algorithms, and applications.* Volume 182. Springer, 2012 (cited on pages 5, 38, 56).

[11] *IBM ILOG OPL Modeling.* URL: www-01.ibm.com/software/commerce/optimization/modeling/ (cited on pages 9, 32).

[12] U. Waqas, M. Geilen, S. Stuijk, J. Pinxten, T. Basten, L. Somers, and H. Corporaal. "A heuristic to schedule self-re-entrant flowshops with sequence dependent setup times, relative due-dates and fixed job ordering". In: *Design of Embedded Systems, Springer* (2017, submission in review process) (cited on pages 9, 115).

[13] S. Choi and Y. Kim. "Minimizing total tardiness on a two-machine re-entrant flowshop". In: *European Journal of Operational Research* 199.2 (2009), pages 375–384 (cited on pages 9, 14, 16, 17, 67).

[14] U. Waqas, M. Geilen, J. Kandelaars, L. Somers, T. Basten, S. Stuijk, P. Vestjens, and H. 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 (cited on pages 9, 13–16, 54, 83, 115).

[15] U. Waqas, M. Geilen, S. Stuijk, J. Pinxten, T. Basten, L. Somers, and H. 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 2016*. 2016, pages 215–221. DOI: 10.1109/DSD.2016.26 (cited on pages 10, 57, 115).

[16] U. Waqas, M. Geilen, S. Stuijk, J. Pinxten, T. Basten, L. Somers, and H. Corporaal. "A fast estimator of performance with respect to the design parameters of self re-entrant flowshops". In: *Microprocessors and Microsystems (MICPRO)* (2017, submission in review process) (cited on pages 10, 115).

[17] U. Waqas, M. Geilen, S. Stuijk, J. Pinxten, T. Basten, L. Somers, and H. Corporaal. "A Variation Measure for Design Choices of Self-re-entrant Flowshops". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2018, submission in review process) (cited on pages 10, 115).

[18] M. Pinedo. *Scheduling, Theory, Algorithms and Systems*. Springer, 2015 (cited on pages 14, 17, 18).

[19] A. Allahverdi, C.T. Ng, T.C.E. Cheng, and M.Y. Kovalyov. "A survey of scheduling problems with setup times or costs". In: *European Journal of Operational Research* 187.3 (2008), pages 985–1032 (cited on pages 14, 56).

[20] A. Allahverdi. "The third comprehensive survey on scheduling problems with setup times/costs". In: *European Journal of Operational Research* 246.2 (2015), pages 345–378 (cited on page 14).

[21] M.R. Gary and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman and Company, New York, 1979 (cited on pages 14, 18, 99).

[22] C. Jing, G. Tang, and X. Qian. "Heuristic algorithms for two machine re-entrant flow shop". In: *Theoretical Computer Science* 400.1 (2008), pages 137–143 (cited on page 16).

[23] P. Semančo and V. Modrák. "A comparison of constructive heuristics with the objective of minimizing makespan in the flow-shop scheduling problem". In: *Acta Polytechnica Hungarica* 9.5 (2012) (cited on pages 17, 34).

[24] R.M. Karp. *Reducibility among combinatorial problems*. Springer, 1972 (cited on pages 17, 18).

[25] J.D. Ullman. "NP-complete scheduling problems". In: *Journal of Computer and System sciences* 10.3 (1975), pages 384–393 (cited on pages 17, 18).

[26] M. Fujii, T. Kasami, and K. Ninomiya. "Optimal sequencing of two equivalent processors". In: *SIAM Journal on Applied Mathematics* 17.4 (1969), pages 784–789 (cited on page 17).

[27] E.G. Coffman Jr and R.L. Graham. "Optimal scheduling for two-processor systems". In: *Acta Informatica* 1.3 (1972), pages 200–213 (cited on pages 17, 18).

[28] D.K. Goyal. *Scheduling processor bound systems*. Computer Science Department, Washington State University, 1976 (cited on page 18).

[29] N. Christofides. "The shortest Hamiltonian chain of a graph". In: *SIAM Journal on Applied Mathematics* 19.4 (1970), pages 689–696 (cited on pages 18, 99).

[30] E. Taillard. "Benchmarks for basic scheduling problems". In: *european journal of operational research* 64.2 (1993), pages 278–285 (cited on page 32).

[31] Wikipedia. *Box plot — Wikipedia, The Free Encyclopedia*. [Online; accessed 18-October-2016]. 2016. URL: https://en.wikipedia.org/wiki/Box%5C_plot (cited on page 33).

[32] S. Choi and Y. Kim. “Minimizing total tardiness on a two-machine re-entrant flowshop”. In: *EJOR* 199.2 (2009), pages 375–384 (cited on pages 38, 40, 44, 45, 56).

[33] S.M. Johnson. “Optimal two- and three-stage production schedules with setup times included”. In: *Naval Research Logistics Quarterly* 1.1 (1954), pages 61–68. ISSN: 1931-9193. DOI: 10.1002/nav.3800010110 (cited on page 40).

[34] H.G. Campbell, R.A. Dudek, and M.L. Smith. “A heuristic algorithm for the n job, m machine sequencing problem”. In: *Management science* 16.10 (1970), B–630 (cited on page 40).

[35] G.B. McMahon and P.G. Burton. “Flow-shop scheduling with the branch-and-bound method”. In: *Operations Research* 15.3 (1967), pages 473–481 (cited on page 40).

[36] J.N.D. Gupta. “A general algorithm for the n× m flowshop scheduling problem”. In: *The International Journal of Production Research* 7.3 (1968), pages 241–247 (cited on page 40).

[37] R.Z. Ríos-Mercado and J.F. Bard. “Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups”. In: *Computers & Operations Research* 25.5 (1998), pages 351–366 (cited on page 40).

[38] E.F. Stafford Jr and F.T. Tseng. “Two models for a family of flowshop sequencing problems”. In: *EJOR* 142.2 (2002), pages 282–293 (cited on page 40).

[39] H. Ishibuchi, S. Misaki, and H. Tanaka. “Modified simulated annealing algorithms for the flow shop sequencing problem”. In: *EJOR* 81.2 (1995), pages 388–398 (cited on page 40).

[40] C.R. Reeves and T. Yamada. “Genetic algorithms, path relinking, and the flowshop sequencing problem”. In: *Evolutionary Computation* 6.1 (1998), pages 45–60 (cited on page 40).

[41] D.S. Palmer. “Sequencing jobs through a multi-stage process in the minimum total time–a quick method of obtaining a near optimum”. In: *OR* (1965), pages 101–107 (cited on page 40).

[42] D.G. Dannenbring. “An evaluation of flow shop sequencing heuristics”. In: *Management science* 23.11 (1977), pages 1174–1182 (cited on page 40).

[43] M. Nawaz, E.E. Enscore Jr, and I. Ham. “A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem”. In: *Omega* 11.1 (1983), pages 91–95 (cited on pages 40, 56).

[44] D. Yang, W. Kuo, and M. Chern. “Multi-family scheduling in a two-machine reentrant flow shop with setups”. In: *EJOR* 187.3 (2008), pages 1160–1170 (cited on pages 40, 56).

[45] *IBM ILOG CPLEX*. URL: www-01.ibm.com/software/commerce/optimization/modeling/ (cited on page 45).

[46] S.V.V Pasupula. “Scheduling and Optimization of Heuristic Production Printer Scheduler”. Master’s thesis. Eindhoven University of Technology, 2016 (cited on pages 50, 95).

[47] R.V.D Tempel. “Mixed-Plexity Heuristic and Optimization for On-Line Scheduling in a Large Scale Printer”. Master’s thesis. Eindhoven University of Technology, 2017 (cited on pages 50, 95).

[48] L. Swartjes, L.F.P. Etman, J.M. Mortel-Fronczak, L.J.A.M. Somers, and J.E. Rooda. *Model-based Constrained Optimization for Paper Path Layout and Timing Design of Printers (Master Thesis)*. Eindhoven University of Technology, 2012 (cited on pages 53, 56).

[49] L. Danping and C.K.M. Lee. “A review of the research methodology for the re-entrant scheduling problem”. In: *International Journal of Production Research* 49.8 (2011), pages 2221–2242 (cited on page 56).

[50] M.R. Garey, D.S. Johnson, and R. Sethi. "The complexity of flowshop and jobshop scheduling". In: *Mathematics of operations research* 1.2 (1976), pages 117–129 (cited on page 56).

[51] E. Taillard. "Some efficient heuristic methods for the flow shop sequencing problem". In: *EJOR* 47.1 (1990), pages 65–74 (cited on page 56).

[52] M. Chincholkar and J.W. Herrmann. "Estimating manufacturing cycle time and throughput in flow shops with process drift and inspection". In: *International Journal of Production Research* 46.24 (2008), pages 7057–7072 (cited on page 57).

[53] Y. Park, S. Kim, and C. Jun. "Performance analysis of re-entrant flow shop with single-job and batch machines using mean value analysis". In: *Production Planning & Control* 11.6 (2000), pages 537–546 (cited on page 57).

[54] S. Pradhan, P. Damodaran, and K. Srihari. "Predicting performance measures for Markovian type of manufacturing systems with product failures". In: *European Journal of Operational Research* 184.2 (2008), pages 725–744 (cited on page 57).

[55] S. Adyanthaya, H.A. Ara, J. Bastos, A. Behrouzian, R.M. Sánchez, J. Pinxten, B. Sanden, U. Waqas, T. Basten, H. Corporaal, et al. "xCPS: a tool to eXplore cyber physical systems". In: *Proceedings of the WESE'15: Workshop on Embedded and Cyber-Physical Systems Education*. ACM. 2015, page 3 (cited on pages 64, 115).

[56] *Photolithography*. https://en.wikipedia.org/wiki/Photolithography. [Online; accessed 11-11-2016]. 2016 (cited on page 66).

[57] N. Nethercote, P.J. Stuckey, R. Becket, S. Brand, G.J. Duck, and G. Tack. "MiniZinc: Towards a standard CP modelling language". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2007, pages 529–543 (cited on page 67).

[58] A.S. Sandhu. "A robust production system design using robust engineering methodology". Concordia University, 2005 (cited on page 73).

[59] W.J. Hopp and M.L. Spearman. *Factory physics*. Waveland Press, 2011 (cited on pages 77, 78).

[60] K. Benkel, K. Jørnsten, and R. Leisten. "Variability aspects in flowshop scheduling systems". In: *Industrial Engineering and Systems Management (IESM), 2015 International Conference on*. IEEE. 2015, pages 118–127 (cited on page 78).

[61] W. Kubiak. "Completion time variance minimization on a single machine is difficult". In: *Operations Research Letters* 14.1 (1993), pages 49–59 (cited on page 78).

[62] J.A. Ventura and M.X. Weng. "Minimizing Single-Machine Completion Time Variance". In: *Management Science* 41.9 (1995), pages 1448–1455. ISSN: 00251909, 15265501. URL: `http://www.jstor.org/stable/2633040` (cited on page 78).

[63] H. Zhang, Q. Chen, and N. Mao. "System performance analysis of flexible flow shop with match processing constraint". In: *International Journal of Production Research* 54.20 (2016), pages 6052–6070 (cited on page 78).

[64] A.G. Merten and M.E. Muller. "Variance minimization in single machine sequencing problems". In: *Management Science* 18.9 (1972), pages 518–528 (cited on page 78).

[65] R. Leisten and C. Rajendran. "Variability of completion time differences in permutation flow shop scheduling". In: *Computers & Operations Research* 54 (2015), pages 155–167 (cited on page 78).

[66] K.N. McKay, F.R. Safayeni, and J.A. Buzacott. "Job-shop scheduling theory: What is relevant?" In: *Interfaces* 18.4 (1988), pages 84–90 (cited on page 78).

[67] A. Davenport, C. Gefflot, and C. Beck. “Slack-based techniques for robust schedules”. In: *Sixth European Conference on Planning*. 2014 (cited on page 78).

[68] S. Adyanthaya, Z. Zhang, M. Geilen, J. Voeten, T. Basten, and R. Schiffelers. “Robustness analysis of multiprocessor schedules”. In: *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on*. IEEE. 2014, pages 9–17 (cited on page 79).

[69] M. Onabajo and J. Silva-Martinez. “Process Variation Challenges and Solutions Approaches”. In: *Analog Circuit Design for Process Variation-Resilient Systems-on-a-Chip*. Boston, MA: Springer US, 2012, pages 9–30. ISBN: 978-1-4614-2296-9. DOI: 10.1007/978-1-4614-2296-9_2. URL: https://doi.org/10.1007/978-1-4614-2296-9_2 (cited on page 79).

[70] A.K. Choudhary, J.A. Harding, and M.K. Tiwari. “Data mining in manufacturing: a review based on the kind of knowledge”. In: *Journal of Intelligent Manufacturing* 20.5 (July 2008), page 501. ISSN: 1572-8145. DOI: 10.1007/s10845-008-0145-x. URL: https://doi.org/10.1007/s10845-008-0145-x (cited on page 79).

[71] B. Sanden, J. Bastos, J. Voeten, M. Geilen, M. Reniers, T. Basten, J. Jacobs, and R. Schiffelers. “Compositional specification of functionality and timing of manufacturing systems”. In: *2016 Forum on Specification and Design Languages (FDL)*. Sept. 2016, pages 1–8. DOI: 10.1109/FDL.2016.7880372 (cited on page 83).

[72] *xCPS*. http://www.es.ele.tue.nl/cps/xCPS/. [Online; accessed 31-5-2017]. 2017 (cited on page 83).

[73] S. Adyanthaya, H.A. Ara, J. Bastos, A. Behrouzian, R.M. Sánchez, J. Pinxten, B. Sanden, U. Waqas, T. Basten, H. Corporaal, et al. “xCPS: a tool to explore cyber physical systems”. In: *ACM SIGBED Review* 14.1 (2017), pages 81–95 (cited on pages 83, 115).

[74] *The Taillard benchmark*. URL: http://www.lifl.fr/~liefooga/benchmarks/benchmarks/ (cited on page 101).

[75] *The structured benchmark*. URL: http://www.cs.colostate.edu/sched/generator/ (cited on page 101).

[76] *ORLIB benchmark*. URL: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/flowshopinfo.html (cited on page 101).

[77] J. Pinxten, M. Geilen, T. Basten, U. Waqas, and L. Somers. “Online heuristic for the Multi-Objective Generalized traveling salesman problem”. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. IEEE. 2016, pages 822–825 (cited on page 115).

[78] J. Pinxten, U. Waqas, M. Geilen, T. Basten, and L. Somers. “Online Scheduling of 2-Reentrant Flexible Manufacturing Systems”. In: *ACM Trans. Embedd. Comput. Syst.* (2017) (cited on page 115).

[79] U. Waqas, J. Kandelaars, P. Vestjens, and L. Somers. “Windowed scheduling and optimization of production printers”. European Patent Office, (Filed). (Cited on page 115).

List of publications

First author, peer reviewed

1. U. Waqas, M. Geilen, J. Kandelaars, L. Somers, T. Basten, S. Stuijk, P. Vestjens, and H. 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.
2. U. Waqas, M. Geilen, S. Stuijk, J. Pinxten, T. Basten, L. Somers, and H. Corporaal. "A heuristic to schedule self-re-entrant flowshops with sequence dependent setup times, relative due-dates and fixed job ordering". In: *Design of Embedded Systems, Springer* (2017, submission in review process).
3. U. Waqas, M. Geilen, S. Stuijk, J. Pinxten, T. Basten, L. Somers, and H. 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 2016*. 2016, pages 215–221. DOI: 10.1109/DSD.2016.26.
4. U. Waqas, M. Geilen, S. Stuijk, J. Pinxten, T. Basten, L. Somers, and H. Corporaal. "A fast estimator of performance with respect to the design parameters of self re-entrant flowshops". In: *Microprocessors and Microsystems (MICPRO)* (2017, submission in review process).
5. U. Waqas, M. Geilen, S. Stuijk, J. Pinxten, T. Basten, L. Somers, and H. Corporaal. "A Variation Measure for Design Choices of Self-re-entrant Flowshops". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2018, submission in review process).

Co-author, peer reviewed

1. S. Adyanthaya, H.A. Ara, J. Bastos, A. Behrouzian, R.M. Sánchez, J. Pinxten, B. Sanden, U. Waqas, T. Basten, H. Corporaal, et al. "xCPS: a tool to eXplore cyber physical systems". In: *Proceedings of the WESE'15: Workshop on Embedded and Cyber-Physical Systems Education*. ACM. 2015, page 3.
2. S. Adyanthaya, H.A. Ara, J. Bastos, A. Behrouzian, R.M. Sánchez, J. Pinxten, B. Sanden, U. Waqas, T. Basten, H. Corporaal, et al. "xCPS: a tool to explore cyber physical systems". In: *ACM SIGBED Review* 14.1 (2017), pages 81–95.
3. J. Pinxten, M. Geilen, T. Basten, U. Waqas, and L. Somers. "Online heuristic for the Multi-Objective Generalized traveling salesman problem". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. IEEE. 2016, pages 822–825.
4. J. Pinxten, U. Waqas, M. Geilen, T. Basten, and L. Somers. "Online Scheduling of 2-Re-entrant Flexible Manufacturing Systems". In: *ACM Trans. Embedd. Comput. Syst.* (2017).

Patent

U. Waqas, J. Kandelaars, P. Vestjens, and L. Somers. "Windowed scheduling and optimization of production printers". European Patent Office, (Filed).

Acknowledgements

Many people have directly or indirectly contributed to my research. Hereby, I would like to pay my regards to them.

First, I thank professor Henk Corporaal for being my promotor. Henk, right from the first day of joining the ES group, I have learned a lot from you. Your emphasis on making my explanations simpler and well defined, has played an important role in shaping my research. You have many times spotted areas, where, I now have core contributions. Your questions on my presentations were interesting, thoughtful and sometimes funny; especially, "Once more, what is the difference between a flow-shop and a job-shop?".

I highly appreciate the contributions of professor Twan Basten, not only in the academic part of my work but also in the industrial collaboration. I have greatly benefited from your vision on how high-tech systems can be improved and what are the missing pieces. I remember one of our discussions, in the very start of my project, when my modelling techniques were bit unclear, you showed great patience over my frustration and rather "boiling argumentation". Truly, your leadership and ability to see the bigger picture have contributed in my work. And you did all this while being my "unofficial promotor". Thank you!

Two persons, my daily supervisors, Marc Geilen and Sander Stuijk, have spent a lot of time in helping me to develop. I just did a rough estimate; 520 hours of only one-to-one meetings and many more hours! From them I learned how to improve communication and writing skills. My first draft paper was a complete disaster and I thank them for their patience, guidance and kindness. Specially, I greatly admire Marc's theoretical rigour and support in understanding the computational complexity of my work. I did three failed attempts to get to the acceptable proof of complexity. We have the proof now, but, it was not possible without hope and support. I highly appreciate Sander's management skills. He has helped me a lot in balancing industrial collaboration and research. Many thanks to Sander for the leave arrangements he initiated, when, my father was sick. The arrangements made it possible for me, during tough times, to spend additional time with my family.

I thank the members of my thesis committee for providing me useful feedback. I was inspired from the detailed report, written by professor Samarjit Chakraborty, as feedback. I also thank professor Jozef Hooman for the extra time he has spent in the form of a meeting to understand my work. Thank you!

My humble regards to the people that I worked with at Océ Technologies. Lou Somers, I still remember the first interview presentation that I did before my selection for the collaboration; and there was a small typo leading to a theoretical blunder. For some reason, you still gave me a go! I also appreciate your overall support in getting access to information, and most of all, your help in getting things going at Océ. Huge bundle of thanks to Jack Kandelaars for the tooling and integration support. Your hacks and fixes in the system allowed me to balance research and collaboration. I also appreciate the nice discussions we had; I am gonna miss the lekker dropjes from the JACK-POT! I appreciate the patience and support from Patrick Vestjens. Many of your small jokes made my time at Océ full of life. Also, sorry for the extra coffee that you needed to stay through the boring parts of my presentations. I thank Henri Hunnekens for arranging

access to the main nodes and lab models. I am in debt to the great advices from Amol Khalate, especially, to always draw in TikZ. Also I thank to the GANG (Barath, Nick, Ali, Eugen, Ketan, Marijn, Sunder, Kemal, Nithin) for amazing discussions during the lunch breaks and walks around the river Maas.

Joost van Pinxten has spent significant energy in improving the outcomes of my work and in the collaborative work that we have done. I thank him for being very thorough in proof reading manuscripts and brain storms. I also thank him for his explanations why Dutch language has ...; I remember our discussion in Bonn about why sometimes "maakt niet uit" does not mean "I do not care". I would like to continue such discussions!

In the start of my project, a guru in optimization recommended me to resign from the project. His main objection was that it is highly unlikely that I can make algorithms which can solve the scheduling problem in less than 400 milliseconds. Two of my students, Vishnu Pasupula and Roel van der Tempel contributed in showing that the scheduling algorithms produce good results in less than 350 milliseconds. Meeting this requirement was fundamentally a success of my project for which I am grateful to my students for their contributions.

Most of my breaks were with Rehan. And what do we do? discussions and more discussions; We never get tired. If I think of the number of innovations that we have on paper, if 10% of them became reality, we would have most of the current world problems solved. But additional to that, we talked over our projects, startups, Elon Musk etc. I thank him for making my breaks interesting. I am thankful to the xCPS colleagues (Shreya, Medina, Bastos, THE REAL HADI, Amir, Meneer van der Sanden) for the cool time, and colleagues (Mark Wijtvliet, Roel Jordans, Francesco, Shubhendu, Yahya, Maurice, David, Manil, and others) and friends (Asma, Komal, the Rajas, Roxana, Mian Ahsan Iqbal, Tausif, Usman, Farooqi) for the fun time that we had; for the very many hangouts and dinners. I am grateful to Marja and Rian for their help at multiple occasions for problems that seemed unsolvable. But, they always manage to fix everything!

Going back to my school times, Sir Rashid told me to work hard. I am thankful to Sir Jahangir to point me out, early in high school, that I should pay attention to studies. To not waste my time (yes, me waza lazy kid). I am grateful to the Sufi's that I have met in my life because they brought my attention towards basics of Sufism. Thanks to the teachers, the mentors, the well wishers. These people, somehow had an idea of what is good for me; they did help me carve my way out.

I feel very blessed to have a brother, Awais, who can think of impact together with engineering. Many of our discussions might have shaped the way my research went. I have also learned a lot from my lil Sis, Mehak; things like the faith that future will turn out to be good! And how to sleep long hours. My father had taught me some of the earliest lessons of my life. Collaboration is one of them; I remember very well, some of the earliest memories of you, when we were fixing our car together. I always had continuous support from my mother. No matter what! perseverance is a lesson learned from you. You persisted that I will not study but play games on the computer. Here am I writing a thesis related to computers. I thank all friends and family for the nice time and well wishes. Finally, I thank Ruxandra; for standing right next to me. In all ups and downs.

Umar Waqas,
September 2017.

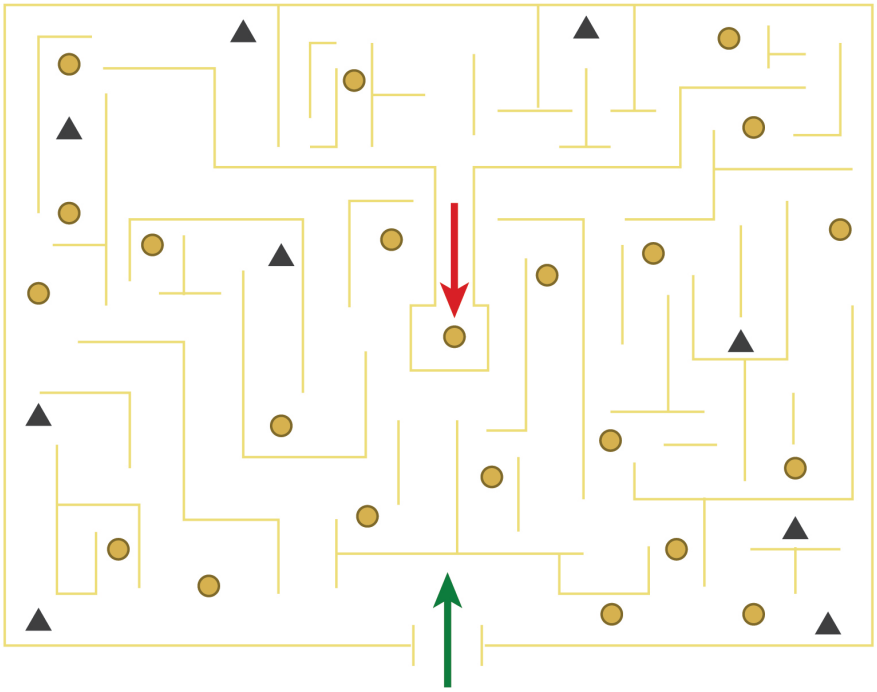
Curriculum vitae

Umar Waqas was born on 26 August, 1987 in Jhelum, Punjab, Pakistan. He completed his bachelors (with distinction) in Computer Science in 2009 and Masters (with distinction) in Embedded Systems in 2012. His master thesis focused to enhance model based design of software defined radios. In the master thesis he worked on modelling, designing, mapping and validation of real-time applications onto resources available in MPSoCs.

He joined the Electronic Systems Group at Eindhoven University of Technology in 2012 to start his PhD in the NEST project. In his project, he started with developing a heuristic to schedule *Canon's VarioPrint i300*. He got inspired from the scheduling problem and looked into scheduling a broader class of self-re-entrant systems. Additional to heuristics, Umar also developed performance estimators and measures for variation-aware design. He also implemented tandem pattern recognizer using suffix trees which is currently used in a printer.

A mini game version of a problem solved in this thesis, with rules:

i. Start from the bottom and find your way till center.
ii. Collect as many coins ( ) as possible.
iii. Avoid triangles ( ). You die if you get hit by three or more.
iv. Repeated visits to the same area are not allowed.



The game lets you encounter one of the main problems solved in this thesis. Additional to the challenges of the problem, in the research, it is solved quickly.