N° d'ordre NNT : 2016LYSEM014

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON

opérée au sein de

l'École des Mines de Saint-Étienne

École Doctorale N° 488

Sciences, Ingénierie, Santé

Spécialité de doctorat : Génie Industriel

Soutenue publiquement le 20/09/2016, par :

Sebastian Knopp

Complex Job-Shop Scheduling with Batching in Semiconductor Manufacturing

---

Devant le jury composé de :

Christelle Jussien-Guéret, Professeur, Université d'Angers	Présidente
Christian Artigues, Directeur de Recherche, LAAS-CNRS, Toulouse	Rapporteur
Scott J. Mason, Professeur, Clemson University, États-Unis	Examinateur
Lars Mönch, Professeur, FernUniversität in Hagen, Allemagne	Rapporteur
Philippe Vialletelle, Ingénieur, STMicroelectronics, Crolles	Examinateur
Farouk Yalaoui, Professeur, Université de Technologie de Troyes	Examinateur
Stéphane Dauzère-Pérès, Professeur, EMSE, Gardanne	Directeur de thèse
Claude Yugma, Chargé de Recherche, EMSE, Gardanne	Co-directeur
Jacques Pinaton, Ingénieur, STMicroelectronics, Rousset	Invité
Renaud Roussel, Ingénieur, STMicroelectronics, Crolles	Invité

Spécialités doctorales	Responsables :	Spécialités doctorales	Responsables
SCIENCES ET GENIE DES MATERIAUX	K. Wolski Directeur de recherche	MATHEMATIQUES APPLIQUEES	O. Roustant, Maître-assistant
MECANIQUE ET INGENIERIE	S. Drapier, professeur	INFORMATIQUE	O. Boissier, Professeur
GENIE DES PROCEDES	F. Gruy, Maître de recherche	IMAGE, VISION, SIGNAL	JC. Pinoli, Professeur
SCIENCES DE LA TERRE	B. Guy, Directeur de recherche	GENIE INDUSTRIEL	X. Delorme, Maître assistant
SCIENCES ET GENIE DE L'ENVIRONNEMENT	D. Graillot, Directeur de recherche	MICROELECTRONIQUE	Ph. Lalevée, Professeur

EMSE : Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'Etat ou d'une HDR)

ABSI	Nabil	CR	Génie industriel	CMP
AUGUSTO	Vincent	CR	Image, Vision, Signal	CIS
AVRIL	Stéphane	PR2	Mécanique et ingénierie	CIS
BADEL	Pierre	MA(MDC)	Mécanique et ingénierie	CIS
BALBO	Flavien	PR2	Informatique	FAYOL
BASSEREAU	Jean-François	PR	Sciences et génie des matériaux	SMS
BATTON-HUBERT	Mireille	PR2	Sciences et génie de l'environnement	FAYOL
BEIGBEDER	Michel	MA(MDC)	Informatique	FAYOL
BLAYAC	Sylvain	MA(MDC)	Microélectronique	CMP
BOISSIER	Olivier	PR1	Informatique	FAYOL
BONNEFOY	Olivier	MA(MDC)	Génie des Procédés	SPIN
BORBELY	Andras	MR(DR2)	Sciences et génie des matériaux	SMS
BOUCHER	Xavier	PR2	Génie Industriel	FAYOL
BRODHAG	Christian	DR	Sciences et génie de l'environnement	FAYOL
BRUCHON	Julien	MA(MDC)	Mécanique et ingénierie	SMS
BURLAT	Patrick	PR1	Génie Industriel	FAYOL
CHRISTIEN	Frédéric	PR	Science et génie des matériaux	SMS
DAUZERE-PERES	Stéphane	PR1	Génie Industriel	CMP
DEBAYLE	Johan	CR	Image Vision Signal	CIS
DELAFOSSE	David	PR0	Sciences et génie des matériaux	SMS
DELORME	Xavier	MA(MDC)	Génie industriel	FAYOL
DESRAYAUD	Christophe	PR1	Mécanique et ingénierie	SMS
DJENIZIAN	Thierry	PR	Science et génie des matériaux	CMP
DOUCE	Sandrine	PR2	Sciences de gestion	FAYOL
DRAPIER	Sylvain	PR1	Mécanique et ingénierie	SMS
FAVERGEON	Loïc	CR	Génie des Procédés	SPIN
FEILLET	Dominique	PR1	Génie Industriel	CMP
FOREST	Valérie	MA(MDC)	Génie des Procédés	CIS
FOURNIER	Jacques	Ingénieur chercheur CEA	Microélectronique	CMP
FRACZKIEWICZ	Anna	DR	Sciences et génie des matériaux	SMS
GARCIA	Daniel	MR(DR2)	Génie des Procédés	SPIN
GAVET	Yann	MA(MDC)	Image Vision Signal	CIS
GERINGER	Jean	MA(MDC)	Sciences et génie des matériaux	CIS
GOEURIOT	Dominique	DR	Sciences et génie des matériaux	SMS
GONDRAN	Natacha	MA(MDC)	Sciences et génie de l'environnement	FAYOL
GRAILLOT	Didier	DR	Sciences et génie de l'environnement	SPIN
GROSSEAU	Philippe	DR	Génie des Procédés	SPIN
GRUY	Frédéric	PR1	Génie des Procédés	SPIN
GUY	Bernard	DR	Sciences de la Terre	SPIN
HAN	Woo-Suck	MR	Mécanique et ingénierie	SMS
HERRI	Jean Michel	PR1	Génie des Procédés	SPIN
KERMOUCHE	Guillaume	PR2	Mécanique et Ingénierie	SMS
KLOCKER	Helmut	DR	Sciences et génie des matériaux	SMS
LAFOREST	Valérie	MR(DR2)	Sciences et génie de l'environnement	FAYOL
LERICHE	Rodolphe	CR	Mécanique et ingénierie	FAYOL
MALLIARAS	Georges	PR1	Microélectronique	CMP
MOLIMARD	Jérôme	PR2	Mécanique et ingénierie	CIS
MOUTTE	Jacques	CR	Génie des Procédés	SPIN
NIKOLOVSKI	Jean-Pierre	Ingénieur de recherche	Mécanique et ingénierie	CMP
NORTIER	Patrice	PR1		SPIN
OWENS	Rosin	MA(MDC)	Microélectronique	CMP
PERES	Véronique	MR	Génie des Procédés	SPIN
PICARD	Gauthier	MA(MDC)	Informatique	FAYOL
PIJOLAT	Christophe	PR0	Génie des Procédés	SPIN
PIJOLAT	Michèle	PR1	Génie des Procédés	SPIN
PINOLI	Jean Charles	PR0	Image Vision Signal	CIS
POURCHEZ	Jérémy	MR	Génie des Procédés	CIS
ROBISSON	Bruno	Ingénieur de recherche	Microélectronique	CMP
ROUSSY	Agnès	MA(MDC)	Génie industriel	CMP
ROUSTANT	Olivier	MA(MDC)	Mathématiques appliquées	FAYOL
STOLARZ	Jacques	CR	Sciences et génie des matériaux	SMS
TRIA	Assia	Ingénieur de recherche	Microélectronique	CMP
VALDIVIESO	François	PR2	Sciences et génie des matériaux	SMS
VIRICELLE	Jean Paul	DR	Génie des Procédés	SPIN
WOLSKI	Krzystof	DR	Sciences et génie des matériaux	SMS
XIE	Xiaolan	PR1	Génie industriel	CIS
YUGMA	Gallian	CR	Génie industriel	CMP

Mise à jour : 01/02/2016

Acknowledgements

Over the past few years, I had the chance to work on a very interesting topic in a pleasant environment. Within a fruitful cooperation between academia and industry, many supportive people helped me along the way. I wish to express my gratitude to my thesis adviser Stéphane Dauzère-Pérès for wisely guiding and supporting me throughout the thesis. With his vast experience in scheduling and semiconductor manufacturing, he supported me at the best over all these years. This work would not have been possible without the engineers of STMicroelectronics who took the time for providing me with invaluable information. In particular, I would like to thank Amélie Pianne and Renaud Roussel for many discussions and exchanges during our cooperation. I would also like to thank Phillippe Vialletelle and Jacques Pinaton for supporting this work and participating in the jury of the thesis. Of course, I also wish to thank Claude Yugma for co-advising the thesis.

I would like to thank Lars Mönch for the time he took for discussions during his stay in Gardanne, for inviting me to the illuminating Dagstuhl seminar on modeling and analysis of semiconductor supply chains, and of course for refereeing my thesis. As well, I would like to thank Christian Artigues for refereeing the thesis. I would also like to thank Christelle Jussien-Guéret, Scott J. Mason and Farouk Yalaoui for being part of the jury. Again, my gratitude goes to Scott Mason and Lars Mönch for providing test instances.

I want to thank all current and former members of the department of manufacturing sciences and logistics for the pleasant time and all the interesting discussions during the past years. Among many other things, I was very pleased about your patience when teaching me some French, in particular during my first year. I thank my officemate Abdoul Bitar for sharing his experience from scheduling in the photolithography area. Ich danke meiner Familie, meinen Freunden und besonders Elisabeth Zehendner, die mich alle jederzeit unterstützt haben. Thank you very much! Vielen Dank! Merci beaucoup!

Sebastian Knopp
Vienna, November 2016

Contents

General Introduction

The subject of this thesis is the scheduling of lots that have to be processed in a particular workcenter of a semiconductor manufacturing facility that imposes a rich set of constraints. We present suitable models and optimization methods which include the numerous constraints and objectives which are present in this work area. Though this is a specific application area, we will see that the underlying properties lead to very general scheduling problems.

This thesis was conducted in the context of the European project INTEGRATE. This European project aimed at developing optimization, information and control tools as well as manufacturing procedures to efficiently manage a high product and technology mix of heterogeneous lots. Within this context, the close cooperation with the industrial engineering departments of the semiconductor manufacturing facilities of STMicroelectronics in Crolles and Rousset (France) enabled us to gain a detailed understanding of the challenges one has to face when solving scheduling problems in a complex industrial setting. A software package consisting of a scheduler library at its core together with a graphical user interface and adapters for data import and export has been developed. Beyond the experimental results stated in this thesis, the scheduler also has been intensively evaluated by our industrial partner.

In chapter 1, a broad overview of the semiconductor manufacturing landscape is given. In particular, the variety of decision making challenges and solution methods in the context of semiconductor manufacturing is discussed in order to understand the role of scheduling in its context. A detailed specification of the diffusion and cleaning work area is given in chapter 2 that provides an interface between engineers from a fab and combinatorial optimization researchers. This specification is based on information obtained from the engineers working at the facilities of our industrial partner. The first goal is to provide a clear, textual, in-depth description verified by fab engineers. The second goal is to provide a comprehensive description that allows formal models for optimization methods to be developed.

The fundamental challenge that we face in this thesis is a complex job-shop scheduling problem which includes the principal characteristics of the diffusion and cleaning work area: The integration of batching machines within a job-shop environment. One of our main contributions, introduced in chapter 3, is a novel batch-oblivious disjunctive graph representation that uses edge weights to model batching decisions. This representation facilitates the modification of batching decisions and allows the development of an original “on the fly” batching algorithm. Complemented by a known move for integrated resequencing and reassignment of operations, we obtain an integrated neighborhood which is applied within a parallel GRASP based meta-heuristic approach.

In chapter 4, we increase the detail of our model for the diffusion and cleaning area by including internal components of machines. The combination within a job-shop setting leads to a job-shop scheduling model with extended resource and routing flexibility. An important contribution is the inclusion of resource acquisition constraints that help to model the

exclusive acquisition of a resource between two operations of the same job. This modeling offers an additional generality that e.g. allows to solve scheduling problems with auxiliary resources from the photolithography area.

Finally, temporal constraints that limit the maximum time between given pairs of operations are taken into account in the final chapter of this thesis. We consider maximum time lag constraints as soft constraints and include the severity of their violations in the objective function lexicographically.

Chapter 1

Introduction and Industrial Context

The production of a single wafer requires up to 700 processing steps and takes up to 3 months. Wafers are produced in the most expensive facilities that can be found throughout industry. This requires wise decisions making on different decision levels. The focus of this thesis is to improve scheduling decisions in the diffusion and cleaning work area. There, multifold constraints lead to general complex scheduling problems that also find applications in other industries.

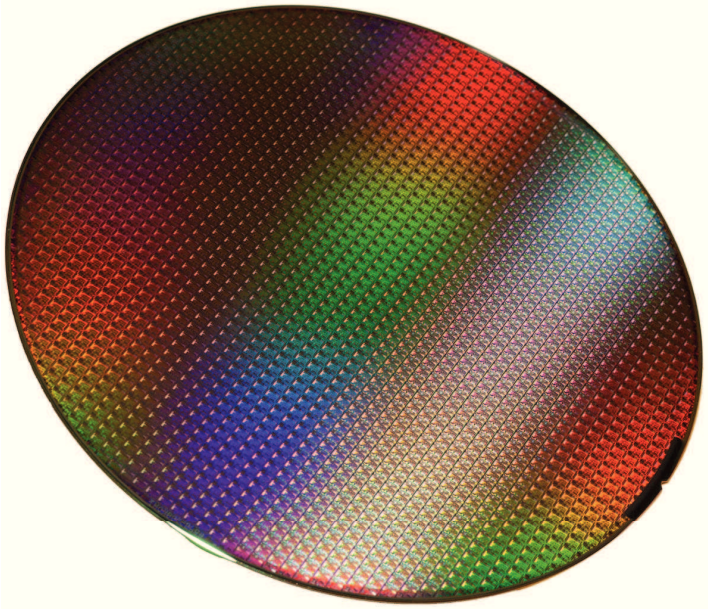


Image source: Flickr, Rob Bulmahn
http://www.flickr.com/photos/rbulmahn/ (CC License)

The subject of this thesis is the scheduling of lots that have to be processed in a particular work center of a semiconductor manufacturing facility that imposes a rich set of constraints. We present suitable models and optimization methods that include the numerous constraints and objectives that are present in this work area. Though this is a specific application area, we will see that the underlying properties lead to very general scheduling problems. This introduction starts with an explanation of the industrial context and summarizes the stages of the semiconductor manufacturing process in section 1.1. Then, section 1.2 describes the different planning levels and positions scheduling within this context. Section 1.3 provides a brief overview of scheduling challenges in the considered diffusion and cleaning work area. The literature review given in section 1.4 focuses on scheduling problems related to those considered in this work and their known resolution methods. Finally, section 1.5 provides an overview of the chapters in this thesis and highlights their main contributions.

Digital electronics showed an enormous influence and growth over the last 50 years. Since the predictive statement of Moore (1965), which declares that device integration density will double approximately every two years, applications have become ubiquitous: computers, sensors, data centers, automotive electronics, or wearable devices. Today, electronic devices are everywhere. Semiconductor manufacturing remains the core process that enables all these applications. Figure 1.1 shows the worldwide market billings provided by the Semiconductor Industry Association (SIA (2015)). Between 1995 and 2015, the overall worldwide sales increased from 50 to 337 billion U.S. dollars per year, which underlines the importance and size of this industry sector.

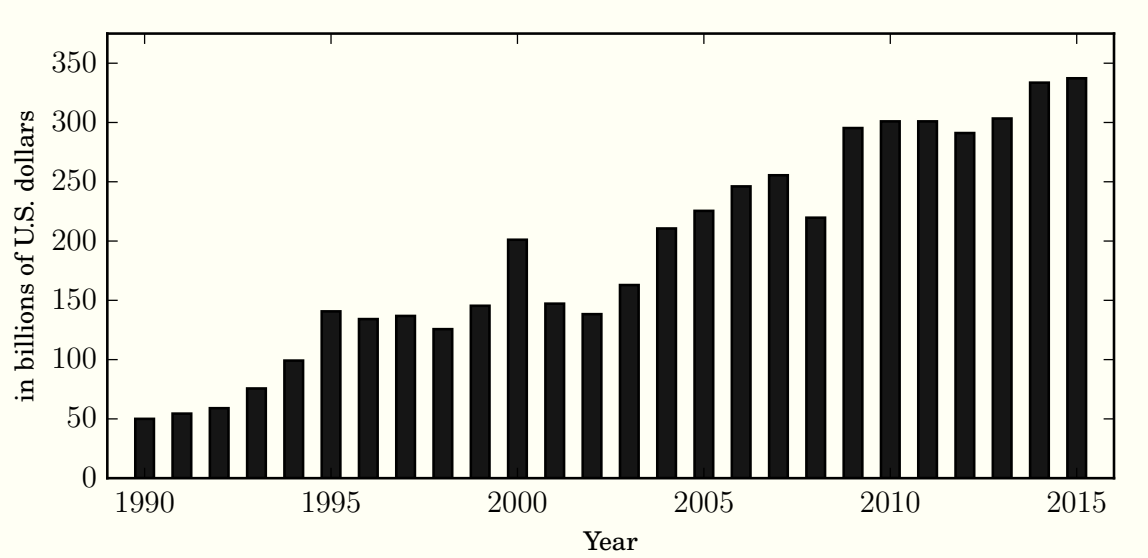


Figure 1.1 – *Worldwide Market Billings, Semiconductor Industry Association (SIA)*

The semiconductor manufacturing sector is a very capital intensive industry with a fierce competition. Beside the high velocity of this market in terms of product lifecycle times, the high cost of manufacturing is a key factor. Machines and cleanroom space are very expensive. Individual machines cost between 100 thousand and 40 million U.S. dollars (consider e.g.

the annual report of ASML (2016)). Several hundred machines can be found in a single semiconductor manufacturing facility (*fab*). Quirk and Serda (2001) report that around 75% of the investment for a single fab is spent on the machines. A single fab nowadays can cost up to 5 billions U.S. dollars. Therefore, a high fab utilization is crucial for success in this industry. Since many options for cost reductions, such as increased wafer sizes and automation, have already been exploited to a large extent, operational cost reductions through better decision systems are considered to be an important direction. From an academic point of view, this area also offers insightful challenges. The complexity observed in this industry is rarely found elsewhere and the transferability of results for operations research questions from this to other industries promises to be viable and fruitful.

1.1 Semiconductor Manufacturing: An Overview

This section summarizes the semiconductor manufacturing process in order to understand the background for the planning decisions that need to be taken at different decision levels. Semiconductor manufacturing and its underlying physical and chemical principles are described in the textbook of Quirk and Serda (2001). Descriptions of the production process that focus more on production planning, decision making and analysis are given by Uzsoy et al. (1992) and Mönch et al. (2013). Based on these sources, this section aims at providing the background information needed to understand and motivate the utility of scheduling in an individual work area. Since work area schedulers need to collaborate with neighboring information and decisions systems, it is important to understand the involved interrelations.

Raw wafers are the basic material that is used to produce integrated circuits (also known as chips, or dies). A wafer is a thin slice of semiconductor material obtained from a single crystal ingot. From a single wafer, hundreds or thousands of microelectronic chips can be produced, depending on the size of the chips and the diameter of the wafer. Usual wafer diameters increased over time to diameters of 200 mm or 300 mm which are used in most wafer fabs today. An integrated circuit consists of a 3-dimensional structure of conductors, semiconductors and insulators that are build upon a wafer. This structure is created by successively adding layers upon the wafer. Already for one of these layers, many processing steps are needed. Since up to 40 layers per chip are necessary, this leads to a large overall number of processing steps. Up to 700 steps can be required for a single wafer. Individual production steps can take between several minutes and many hours. Overall, the production of a single wafer can take up to 3 months.

Semiconductor manufacturing is divided into a *front-end* stage and a *back-end* stage. Figure 1.2 provides an overview of the principal phases of both stages. The core part of the front-end stage is the fabrication of wafers. This is the technologically most demanding part since this is where the structurally smallest parts of the chips are created. Wafer fabrication takes place in large buildings where strict cleanroom conditions have to be ensured in order to avoid wafer contamination. Cleanroom requirements might be less strict in modern fabs where wafers are kept in specialized containers that ensure a clean environment. The details

of this production phase are explained later in this section. At the end of the front-end stage, the probing phase electrically tests each individual chip on the wafer to identify the chips that are eligible for assembly. Front-end production sites are usually located in highly industrialized nations and the production steps performed there consume about 75% of the overall cycle time. During all phases of the front-end, all chips remain located on the same wafer in order to profit from common processing. In the back-end, wafers are then cut into separate chips and their individual assembly takes place. This includes wire bonding, where the chips are attached to its package or another chip, and molding, that encloses the chip and its connections in a protective casing. Also lid sealing, optical inspections, environmental testing and other steps can take place. Since at this stage the manufacturing of integrated circuits on the wafer is already completed, the back-end production requires less strict cleanroom conditions. Back-end phases are mostly located in countries with cheaper labor wages. The final test phase includes electrical tests and heat-stress tests which are performed in burn-in ovens.

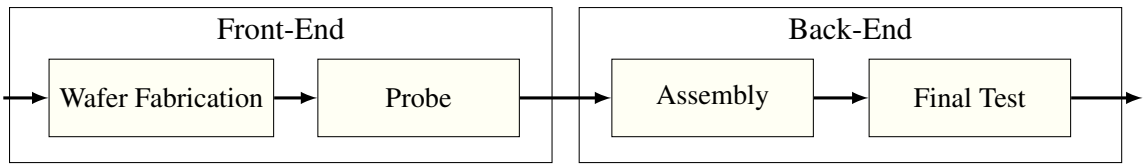


Figure 1.2 – *Stages of semiconductor manufacturing*

In the front-end stage, the structures on each wafer are built layer by layer. The different processing steps needed to produce a single layer are performed in specialized work areas that consist of machines with similar capabilities. Processing steps are executed multiple times to create all layers. This leads to reentrant flows where each wafer visits different work centers multiple times. The interaction between the work centers is shown in Figure 1.3. The following listing describes the work centers and classifies them according to Mönch et al. (2011):

Diffusion/Oxidation/Deposition The *diffusion* process disperses material on the surface of the wafer. The *oxidation* process grows a layer of oxide on the surface of a cleaned wafer. Such layers are modified by subsequent processing steps in order to develop connected semiconductor devices (e.g., transistors, resistors, or diodes) that make up the integrated circuit. Diffusion and oxidation steps can require very high processing durations of 12 hours or longer. Diffusion and oxidation are high-temperature processes performed on horizontal or vertical furnaces. These furnaces are batching machines that can process multiple lots of wafers at the same time. The *deposition* process places thin conductor or insulator films on the surface of a wafer. Deposited thin films serve different purposes: They either become part of the device structure or are used as an auxiliary layer which is removed later on. This work center also contains wet cleaning machines which decontaminate wafers by removing unwanted particles.

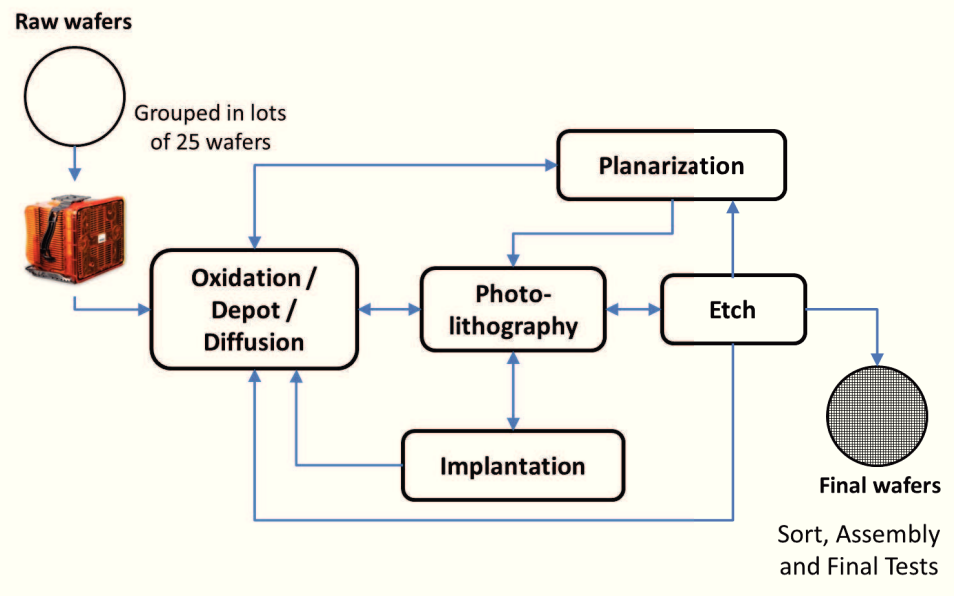


Figure 1.3 – *Processing steps within wafer fabrication (Mönch et al. (2011))*

Photolithography The *photolithography* process transfers device features and wiring patterns to the surface of a wafer by passing ultraviolet light through a mask. The resulting temporary patterns are then made permanent during subsequent etching or ion implantation steps. To perform a photolithography operation, in addition to the photolithography machine (stepper or scanner), a *mask* (or *reticle*) is required as an auxiliary resource. This work area contains the most expensive machines in a fab that can cost up to 40 million U.S. dollars and often constitutes a bottleneck area in the overall fab.

Etching The *etching* process removes unneeded material from the wafer surface. Etching can be patterned or unpatterned. Patterned etching removes a pattern that was brought onto the wafer during photolithography. Unpatterned etching reduces thickness and involves the entire area of the wafer. There are two types of etching: Dry etching exposes the wafer surface to a plasma, and wet etching removes material using chemical solutions.

Implantation The *implantation* process introduces dopants (i.e., desired impurities, ions) into the crystal structure of the semiconductor material in order to modify its conductivity. This step can follow a photolithography or etching step.

Planarization The *planarization* process uses Chemical Mechanical Polishing in order to level the surface of the wafer. Planarization reduces thickness differences over the wafer and is performed each time before a new layer is added. This technique avoids the accumulation of uneven topology over multiple layers and avoids therefore various nonplanarity related problems, such as lens focusing issues in photolithography.

The machines that can be found throughout the work areas show different characteristics: Some of them are capable of batching, others involve sequence-dependent setup times, and some show a pipelining behavior that allows overlapping operations. A set of identical machines in the same work center is also called a tool group. Usually, 25 wafers are grouped within a carrier that can either be a front opening unified pod (*FOUP*) in 300 mm fabs and modern 200 mm fabs, or a wafer carrying *cassette* in 200 mm fabs.

To ensure the quality of produced wafers, inspection and measuring procedures are performed between production steps on sampled lots. If an inspection operation detects a damaged wafer, that wafer can be reworked in rare cases and has to be scrapped in most cases. Damages can be caused by several reasons that include contamination or machine failures. Another cause which is becoming increasingly important with shrinking structural sizes of chips are violations of time constraints. Time constraints limit the duration between two processing steps and originate in the chemical or physical degradation that might appear during overly long waiting periods. In addition, virtual metrology techniques aim at monitoring and improving production quality by indirect means. Sensor data is collected from production machines and analyzed using statistical methods in order to deduce machine failures or quality problems. An important quality measure is the percentage of the chips on the wafer that is correctly produced. This percentage is called the *yield* of the wafer.

1.2 Decision Making for Production Planning in Semiconductor Manufacturing

Planning and decision making in semiconductor manufacturing comprise several decision levels with scopes that range from the entire supply chain to the internal dispatching decisions within cluster tools. The scopes of different decision levels differ in their time horizon, the level of modeled detail, and the granularity of decisions to be taken. A general overview of decision levels and their interrelation is given in the textbooks of Silver et al. (1998) or Stadtler and Kilger (2000). The positioning of scheduling within a matrix model for supply chains is provided in Rohde et al. (2000). In Chien et al. (2011), the arising challenges are discussed from a semiconductor manufacturing point of view. Mönch et al. (2013) distinguish decision levels on the enterprise level, the factory level, and the work center level. In the following, we provide an overview of the decisions that need to been taken at each of these levels. An overview of the described decision levels is depicted in Figure 1.4.

The *enterprise level* comprises long-term planning for a time horizon of several quarters or years. The decisions at this level are strategic and are taken for coarse time buckets that are typically weeks or months. The scope of this planning is the end-to-end customer supply chain or the full supply chain within the enterprise. It is based on anticipated demand (forecasts). The master planning determines production quantities to be completed within given time buckets and quantities to be released to each fab (or subcontractors). This defines the routing throughout the company's internal supply chain. Other long term planning problems encompass investment decisions for production facilities or equipment and product mix de-

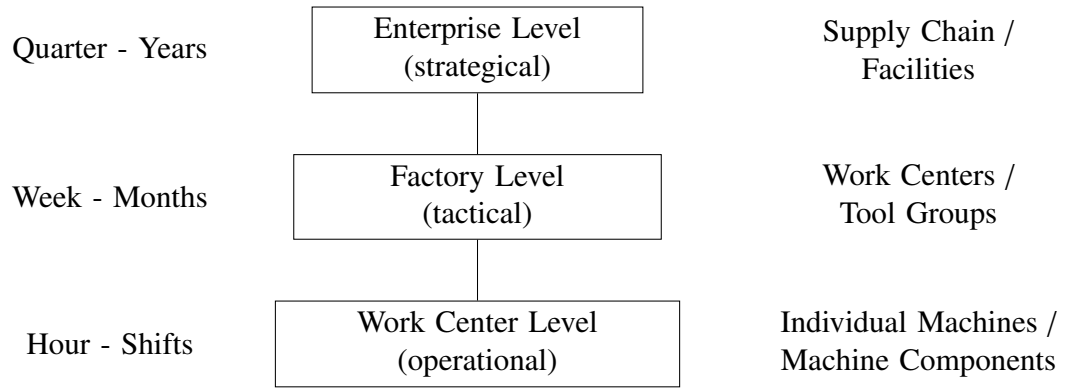


Figure 1.4 – *A characterization of decision levels in semiconductor manufacturing*

cisions for a long term horizon. Ponsignon and Mönch (2012) describe and resolve master planning problems that determine wafer quantities over products, facilities and time periods for given demand and capacity constraints.

The *factory level* comprises mid-term planning for a time horizon of several weeks or months. Its scope are activities within a fab and decisions are based on the current state of the fab and confirmed demand. The decisions at this level are tactical and define the production and inventory plan at the fab level. *Multiple orders per job* decisions, which are described in Mönch et al. (2011), arise since the wafers of some customer orders do not entirely fill up a carrier (FOUP or cassette). The objective is to group wafers of different orders in the same carrier to enable increased machine utilization. *Order release* decisions determine when the production of a lot is started to ensure its consistent and uniform flow through the fab. This can be enforced by different measures. One way is to prescribe job release dates and job due dates for each work center that a job is (repeatedly) traversing. This approach apportions single customer due dates into multiple work center related due dates. Another approach to achieve a uniform flow is the introduction of production targets. Each production step is assigned to a product family. Production targets prescribe for each work center the number of steps for each product family that need to be performed within a given period of time. Both kinds of order release decisions are considered at the work center level by taking the resulting due dates, production targets or priorities (weights) as an input for scheduling or dispatching based approaches. In this context, time constraints as described in Klemmt and Mönch (2012) can also be important since they might span over different work centers. *Qualification management* decisions are needed to manage machine capabilities. Every operation cannot be performed on all machines. Machines must be prepared to gain production abilities by performing preparatory setups and tests. This preparation is called qualification and can be very time-consuming. Therefore, qualification decisions can have an important impact as shown in Johnzén et al. (2011) and Rowshannahad et al. (2015).

The *work center level* comprises short-term planning over at most several hours or shifts. The decisions at this level are operational and are taken for detailed time buckets that can be seconds or minutes. Systems on this level must often react quickly in order to take near-term decisions in time. Crucial are *scheduling decisions* that decide which machine is used to process a lot, which lots are grouped in the same batch, and in which order lots are processed on their assigned machine. In practice, dispatching rules are still often used to take these kinds of decisions. These rules usually decide only on the very next operation to be processed. Many elaborated dispatching rules haven been developed and analyzed (see Mönch et al. (2013)). Scheduling takes a longer time horizon into account and creates a detailed production plan over several hours or shifts. In comparison to dispatching, scheduling is less myopic and usually uses optimization methods to calculate solutions that optimize a given objective function. The input of scheduling and dispatching systems comprises the current state of the fab and the lots to be released within the considered time horizon. Such problems can be described as complex job-shop scheduling problems where each lot corresponds to a job that has to follow a sequence of processing steps. The considered constraints include batching machines, sequence-dependent setup times and other properties, such as auxiliary resources, that depend on the work area that is considered. The problem could be modeled in this way already at the factory level. However, for known scheduling methods, the sizes of such instances seem to be intractable. *Transportation policy decisions* are required in fabs that use Automatized Material Handling Systems (AMHS) to transport lots between machines within the cleanroom. Policies for a material handling system, as described in Kiba et al. (2010), aim at avoiding machine starvation caused by transportation delays. *Tool risk management decisions* concern inspection and control procedures performed between production steps. For capacity reasons, only a subset of all processed wafers is selected for such measurements. As shown by Rodriguez Verjan et al. (2011), these selection decisions are crucial in order to avoid late detections of machine disturbances. Good inspection strategies avoid sampling lots that bring little information. Instead, only wafers that minimize the number of potentially defective wafers are measured.

The decomposition of the entire production planning into subproblems, such as those described before, yields manageable planning tasks that can be tackled independently. Decomposition seems indispensable at present since fully integrated approaches seem to involve a nearly untamable complexity. This separation requires well-thought-out vertical and horizontal interfaces between the involved systems. There is only little work in the literature that concentrates on this integration. The integration between the fab level and particular work areas is discussed in Bureau et al. (2007). They propose to feed information on global production objectives to the work center decisions systems by aggregating global information within input parameters of dispatchers or schedulers and to periodically update this information. The effects of local scheduling decisions at work center level to the overall fab level are studied in Mönch and Rose (2004). They use simulation to evaluate the impact of scheduling within a rolling horizon setting. From a scheduling perspective, several input parameters allow to steer scheduling decisions towards global objectives. Release dates, due dates, job weights or production targets can incorporate factory level objectives.

There are additional factors that complicate production planning and scheduling. One influencing factor is the *product mix* characteristic of the fab. We differentiate between *low-mix* and *high-mix* fabs. In low-mix fabs, high quantities of few product types are manufactured. In high-mix fabs, the situation is inverse: Many different product types are manufactured with potentially small quantities for each product type. High-mix fabs usually produce Application Specific Integrated Circuits (ASICs) which are customized chips for specialized applications. In this work, we concentrate on high-mix fabs. There, lots for different product types and at different production stages are competing for the same machines. In addition, priorities or due dates related to customer needs must be taken into account. Therefore, in high-mix fabs, deciders need to weigh up between a large number of competing objectives. Also, high-mix fabs require a high number of reticles in the photolithography area since a different reticle is needed for each pattern that is brought onto a chip. Thus, these auxiliary resources are more critical in high-mix fabs. Additional complexity is coming from the properties that we observe in individual work areas. Processing durations can vary immensely between production steps and are in the range between a few minutes and over 12 hours. Many properties of machines, such as batching, have a strong influence and require detailed models. The properties of machines in the diffusion and cleaning area are discussed in detail in section 1.3 and in chapter 2. Rapid technological advancements in semiconductor technologies and the high velocity of innovation in this business sector require permanent research and development efforts. This is reflected by the need to include engineering lots within the production process. They are needed for the development of future products and are processed on the same machines as regular production lots. Therefore, engineering lots also influence the capacity of a fab. Ziarnetzky and Mönch (2016) study the combined planning of engineering and production lots. Another aspect is that production steps do not always work out perfectly and can cause defects. This impacts the planning in several ways: Preventive maintenance works on machines have to be scheduled and unexpected machine breakdowns can disturb planning. An approach to consider machine reliability in a non-binary way is the assignment of a machine health indicator to each machine. Such indicators estimate the probability that a step on a machine introduces a defect on the wafer. Doleschal et al. (2015) study the consideration of such indicators in a simulation based evaluation of dispatching and optimization approaches. An integration within job-shop scheduling is presented in Kao et al. (2016).

Relevant performance indicators depend on the considered planning problem and its interplay with related components. Often, multiple objectives are of interest. On the fab level, a variety of performance indicators is important. In an industry survey presented by Pfund et al. (2006), companies consider the overall factory throughput as the most important objective. It is followed by on-time delivery, cycle time, wafer starts, equipment throughput and inventory levels in that order. To resolve, model and analyze the challenges described in this section, a range of tools and methods are available and in use. Some problems can be tackled using optimal or heuristic algorithms. *Simulation* (see Fowler and Rose (2004)) is beneficial to study the effects of decisions and policies with a high degree of modeling detail. Rule based *dispatching* systems are still widely used. An study on dispatching rules for wafer fabrication is provided in Sarin et al. (2011). In practice, such rules can be quite complex

since they evolved over longer time periods and take various company specific needs into account. However scheduling methods using optimization techniques obtain better results and offer the advantage that objectives can be specified explicitly. Queuing models can be used to predict aggregated system behavior while requiring only limited computational effort. To analyze load dependent cycle times, cycle time throughput curves (Fowler et al. (2001)) and clearing functions (Karmarkar (1989)) can be used.

1.3 Scheduling in the Diffusion and Cleaning Area

In the focus of this work are scheduling problems that originate from the diffusion and cleaning areas of the semiconductor manufacturing facilities of STMicroelectronics in Rousset (France) and Crolles (France). Despite the fact that this work area is called *Diffusion/Oxidation/Deposition* in the schema of work centers presented in section 1.1, we will stick to the term *diffusion and cleaning area* throughout this work. The rationale for this is to have a term that includes all properties of the work area specified in detail in chapter 2. The work area specified there includes a wider range of machines: It comprehends not only diffusion and oxidation furnaces, but also other machine types such as cleaning or wet bench machines. The impact of this area to the overall performance of the fab can be high since up to 30% of the WIP can be located in this area (Jung et al. (2013)). This section presents the main characteristics of the rich scheduling problem that is imposed by this work area and it provides the context for the literature review of section 1.4.

The fundamental problem beneath the considered application is a *complex job-shop* scheduling problem where batching machines are considered within a job-shop setting. For each job in a given set, a fixed sequence of operations (route) must be performed on given machines. Each operation belongs to a family which specifies the machines that are qualified for its processing. Processing durations depend on selected machines. Some machines are capable of batching: They can process multiple operations of the same family at the same time as long as machine capacity restrictions are respected (p-batching). In addition, sequence-dependent setup times have to be considered (s-batching) for some machines. Associated to each job is a ready date and, depending on the used policy for order releases, potentially also a due date. Initially, we are interested in mono-criteria regular objective functions such as total weighted tardiness or makespan. Additional industrial objectives described in section 2.3 require to consider multiple objective functions at the same time.

Modeling complex machines by one fixed processing duration neglects many properties of machines that can be found in a real fab. Let us illustrate this for the case of furnaces. They usually consist of two tubes, four boats (two per tube), and one load port. A tube is the place where processes are conducted and a boat is a movable carrier for wafers and necessary to run a process inside a tube. Boats are also utilized to load, unload and cool wafers. The load port is the place where wafers are loaded and unloaded. To process a set of wafers, a boat is used as follows: First, wafers are loaded from its carrier to the boat at the load port. Then, the boat is moved into the tube where the process is conducted. Afterwards, the

boat is removed from the tube and has to cool down before its wafers can be unloaded at the load port. Potentially, the boat has to wait in case the tube or the load port is occupied. Some operations, require more than one internal resource of the same machine at the same time. In the "simple" complex job-shop scheduling model, we would assume that each tube corresponds to one machine which is independently operating. As illustrated, this is not the case and we want to consider a more detailed modeling of machines, integrated within job-shop scheduling, in order to increase the accuracy of our modeling.

In practice, an additional property that is considered to be crucial are temporal constraints. Chemical and physical processes impose *maximum time lag* constraints that limit the time between given pairs of operations. In general, maximum time lags can occur for arbitrary pairs of operations of the same job. Thus, time lags can be adjacent or overlapping and, at the same operation, one maximum time lag can start and another one can end. This can lead to chained constraints that are also called *queue time constraints*. There are different types of time constraints. This reflects the fact that lots that violate a maximum time lag can be reworked only in some cases. In other cases, occurring defects are not reworkable and wafers have to be scrapped.

1.4 Related Work

This literature review is structured along the main chapters of this thesis. It starts with an overview of solution methods and a brief survey of classical and flexible job-shop scheduling. Then, the sections 1.4.1 (complex job-shop scheduling), 1.4.2 (route and resource flexibility) and 1.4.3 (maximum time lag constraints) are organized in parallel to the chapters in this work.

A tremendous amount of research on scheduling was conducted in the last decades. General introductions on scheduling are provided in the text books of Blazewicz et al. (2007), Brucker (2007) and Pinedo (2012). To classify scheduling problems, they provide updated versions of the classification scheme introduced in Graham et al. (1977). This thesis focuses on generalizations of job-shop scheduling problems to model real-world needs of semiconductor manufacturing facilities. An overview of the scheduling challenges in semiconductor manufacturing is provided in Mönch et al. (2011) and Mönch et al. (2013). Benefits of scheduling methods for the performance of semiconductor manufacturing facilities are known for years, consider for example Wein (1988). Known and successful algorithms for scheduling problems cover a wide range of solution methods known in combinatorial optimization. We provide a brief overview in the following.

Enumerative methods, such as dynamic programming or branch-and-bound, systematically explore the solution space. *Dynamic programming* relies on solving subproblems and storing their results. Stored solutions for subproblems are combined in order to obtain solutions of larger subproblems until the original problem is solved. *Branch-and-bound* methods search the solution space by exploring a search tree. A node in that tree represents a set of solutions and its child nodes correspond to a partitioning of this set. A search strategy

specifies the traversal ordering and bounds limit the search space that is explored. *Mixed Integer Linear Programming* (MILP) is a widely used approach that can solve optimization problems with real- and integer-valued decision variables. It is supported by powerful commercial software packages and is the basis for elaborated methods such as lagrangean relaxation or column generation. *Constraint Programming* is based on feasibility and the propagation constraints.

Heuristic methods can determine good solutions without guaranteeing their optimality. For NP-hard problems, they often provide reasonable trade-offs between quality and computational effort. Often, greedy heuristics are used to construct initial solutions. List scheduling is a heuristic method to create a schedule based on a dispatching rule. For NP-hard scheduling problems, *metaheuristic approaches* have proven to be successful. Kirkpatrick (1984) introduced the Simulated Annealing metaheuristic that escapes from local optima with a probability that decreases over time. Genetic algorithms (see Goldberg and Holland (1988)) maintain a population of solutions that evolves generation-wise using mutation, cross-over and selection operators. Tabu search (see Glover (1989)) avoids being stuck in local optima by (temporally) declaring certain moves as tabu. Greedy Randomized Adaptive Search Procedures (GRASP), introduced in Feo and Resende (1995), use a multi-start based approach which independently improves solutions constructed by a randomized greedy heuristic. Mladenović and Hansen (1997) introduce the concept of Variable Neighborhood Search (VNS). In this concept, different neighborhoods are used in an alternating way to escape form local minima. Recently, many new metaheuristics which use inventive analogies have been proposed. Sörensen (2015) critically highlights recent developments in this area and discusses important future directions.

Classical Job-Shop Scheduling The basic problem related to the one at hand is the classical job-shop scheduling problem. As indicated in Blazewicz et al. (2007), the earliest formulations of this problem stem from the late fifties (Bowman (1959), Wagner (1959), Manne (1960)) and it was proven to be NP-hard in Garey et al. (1976). Successful solutions methods are often based on the disjunctive graph representation that models dependencies between operations in a concise way. This representation was introduced by Roy and Sussmann (1964). Blazewicz et al. (2000) propose data structures for an efficient implementation of disjunctive graphs. An overview of solution methods in general is provided by Jain and Meeran (1998). An overview which concentrates on local search based methods is given by Vaessens et al. (1996). Carlier and Pinson (1989) propose a branch-and-bound approach that solved for the first time the 10×10 job-shop problem proposed by Muth and Thompson (1963). A well-known decomposition based solution approach is the shifting bottleneck heuristic of Adams et al. (1988). It subsequently identifies bottleneck machines and optimizes their sequence of jobs. This is done by a problem reduction that fixes operations on all machines except one and then solves the resulting single-machine scheduling problem using a suitable subproblem solution procedure. For this, the authors proposed to use the branch-and-bound approach of Carlier (1982). An improved version of this approach was proposed in Dauzère-Pérès and Lasserre (1993). Balas and Vazacopoulos (1998) propose an approach that embeds a guided

local search procedure within a shifting bottleneck heuristic. Heads and tails provide a measure for the time that is needed before and after an operation, including all machine and route dependencies in the solution. Their properties and adjustments are discussed in Carlier and Pinson (1994).

Simulated annealing was first used to tackle the job-shop scheduling problem by Van Laarhoven et al. (1992). There, a neighborhood induced by the swapping of critical arcs is used and the connectivity of this neighborhood is proven. Taillard (1994) propose parallel tabu search techniques for the job-shop scheduling problem. Nowicki and Smutnicki (1996) propose a tabu search method for which they introduce a neighborhood definition which employs blocks of jobs. Vaessens (1995) discusses deterministic and probabilistic local search approaches for the job-shop scheduling problem. An extension of the classical approaches which consider the makespan objective is the extension of the problem to other objective functions. A criterion is said to be regular if earlier completion dates of operations do not deteriorate the objective function value. Mati et al. (2011) propose an algorithm for regular criteria that utilizes an efficient method to evaluate moves.

Flexible Job-Shop Scheduling The flexible job-shop scheduling problem was first studied by Brucker and Schlie (1990). Brucker and Neyer (1998) reference to this as the multi-purpose-machine job-shop problem. An extensive study including lower bounds for several test instances is given in Jurisch (1992). Brandimarte (1993) as well as Hurink et al. (1994) present a tabu search method for the problem. A disjunctive graph model that allows the reassignment and resequencing of operations in an integrated way was introduced in Dauzère-Pérès and Paulli (1997). Mastrolilli and Gambardella (2000) improved results using a comparable tabu search approach. Chen et al. (1999) propose a genetic algorithm. Gao et al. (2008) propose a hybrid genetic algorithm. Yazdani et al. (2010) schedule flexible job-shops using a parallel variable neighborhood search algorithm. Recently, the problem was successfully tackled using constraint programming based approaches by Pacino and Van Hentenryck (2011) and Schutt et al. (2013). A summary of experimental results for the flexible job-shop scheduling problem is given in Behnke and Geiger (2012). Sobeyko and Mönch (2016) propose a heuristic to solve flexible job-shop scheduling problems with total weighted tardiness objective. García-León et al. (2015) propose a heuristic method that avoids the runtime cost of performing a move by evaluating its effects on the objective function without actually performing the move.

1.4.1 Complex Job-Shop Scheduling with Batching Machines

The surveys of Brucker et al. (1998) and Potts and Kovalyov (2000) provide an overview of scheduling problems that include batching machines. The term batching is used in the context of parallel processing (p-batching) and in that of sequential processing (s-batching). The term *p-batching* refers to the capability of machines to process more than one job at the same time. The term *s-batching* refers to the presence of sequence-dependent setup times. There, a batch is a series of sequentially processed operations with no (or small) setup dura-

tions in between. Potts and Kovalyov (2000) discuss publications dealing with two-machine scheduling problems in a flow-shop or open-shop environment, but batching machines within a job-shop environment are not mentioned. Most of the approaches discussed in their review consider single-machine scheduling problems. A literature review on scheduling problems with batching in the context of semiconductor manufacturing is given in Mathirajan and Sivakumar (2006). They classify literature depending on the problem configuration and the used scheduling methodology.

We observe that the scheduling of parallel batching machines and variants of the job-shop scheduling problem are well-studied problems whereas their combination is rarely considered. In practice, often dispatching rules are used. Glassey and Weng (1991) and Gurnani et al. (1992) study dispatching rules designed for batching machines. Cigolini et al. (2002) propose and study a dynamic look-ahead rule for batching machines in a setting with reentrant flows. An extensive overview of dispatching rules is provided in Mönch et al. (2003).

Starting with the work of Ovacik and Uzsoy (1997), several approaches for complex job-shop scheduling problems are based on the shifting bottleneck heuristic of Adams et al. (1988). For this setting, Ovacik and Uzsoy (1997) introduce a batch-aware disjunctive graph representation that represents batches using dedicated nodes. Mason and Oey (2003) propose a cycle elimination procedure that increases the number of feasible schedules that are found. This approach shows that the avoidance of cycles is an important complicacy in the batch-aware disjunctive graph representation. This representation was also used in Mason et al. (2005) and Mönch and Rose (2004), where the authors show that a modified shifting bottleneck heuristic outperforms classical dispatching rules. A distributed version of the shifting bottleneck heuristic is presented in Mönch and Drießel (2005). Similar approaches are proposed in Upasani et al. (2006) and Sourirajan and Uzsoy (2007). Results were improved in Mönch et al. (2007) by using a genetic algorithm in the subproblem solution procedure. Another approach is presented in Yugma et al. (2012) which relies on batch specific moves, e.g. moving one batch or swapping operations from different batches. Again, batches are represented using dedicated nodes. They optimize three different objectives related to throughput, machine utilization and waiting time. An initial solution is constructed using iterative sampling and improved by a simulated annealing metaheuristic. In distinction to these approaches, we will see in chapter 3 that our approach uses a less complex disjunctive graph model and a more holistic integration of batching decisions. A mixed integer linear programming (MILP) formulation for complex job-shops with total weighted tardiness objective is given in Mason et al. (2005). Bilyk et al. (2014) propose an improved method to solve parallel-machine scheduling problems which appear as subproblems of shifting bottleneck based approaches. A sequential decomposition approach for complex job-shops is presented in Guo et al. (2012) who apply an ant colony optimization heuristic. Two-machine scheduling problems with batching in a permutation flow-shop environment are considered by Skorin-Kapov and Vakharia (1993), Vakharia and Chang (1990), Sotskov et al. (1996), Danneberg et al. (1999), or Tan et al. (2014). Though also considering multiple operations per job, this problem is less general than the complex job-shop scheduling problem considered in this work.

In contrast to the job-shop setting, in single and parallel machine scheduling problems, there is no route for the job and only individual operations are scheduled. Mönch et al. (2005) introduce a method that uses dispatching heuristics in combination with a genetic algorithm. The used dispatching rules are based on the BATC (Batch Apparent Tardiness Cost) rule. They analyze two variants: Either they determine batches first, or they determine assignments to machines first. The results for such parallel machine scheduling problems with batching were improved by a memetic algorithm of Chiang et al. (2010). Almeder and Mönch (2011) present and compare different meta-heuristics for a scheduling problem with batching and parallel machines. In contrast to the previous approach, they do not consider release dates. An ant colony optimization method is introduced and a variable neighborhood search heuristic is compared with a genetic algorithm similar to the one introduced in Balasubramanian et al. (2004). The variable neighborhood search clearly outperforms the other approaches in this setting. Kashan et al. (2008) consider a parallel batching machine scheduling problem with makespan objective. They propose to start with a random solution that is made feasible by grouping long jobs. Such schedules are then used as the initial population for a genetic algorithm. They combine the genetic algorithm with two different improvement methods. Scheduling in the diffusion and cleaning area of semiconductor manufacturing with its particular constraints is also addressed by Yurtsever et al. (2009), Kim et al. (2010) and Jung et al. (2013). However, they do not consider a job-shop environment. Yurtsever et al. (2009) propose a MILP formulation and a heuristic algorithm. They provide practical insights from the deployment of a scheduling solution in a real fab. Fowler et al. (2002) discuss optimal batching in a wafer fabrication facility using a multiproduct model with batch processing.

A number of publications study scheduling problems for single batching machines that are motivated by semiconductor manufacturing. Mehta and Uzsoy (1998) present a dynamic programming algorithm and a heuristic approach. Wang and Uzsoy (2002) consider a single batch machine and suggest a dynamic programming algorithm that determines exact solutions of the minimum makespan problem for a fixed sequence of jobs. A heuristic dynamic programming algorithm to minimize maximum lateness is proposed as well. This is then used to initialize a genetic algorithm with a random number based crossover operator. Finally, they optimize maximum lateness using binary search. Perez et al. (2005) decompose the problem into independent batch formation and sequencing problems that are solved heuristically. Sobeyko and Mönch (2011) compare heuristics to solve scheduling problems for a single batch machine with unequal ready times of the jobs. Dauzère-Pérès and Mönch (2013) schedule jobs on a single batch processing machine with incompatible job families to minimize weighted number of tardy jobs.

For the back-end of semiconductor manufacturing, Lee et al. (1992) present algorithms for single and parallel machine scheduling problems related to burn-in operations. For the same application, Mathirajan et al. (2004) describe a simulated annealing algorithm with non-identical job sizes. Also for burn-in ovens, Kempf et al. (1998) include secondary resource constraints and propose a scheduling method to minimize makespan and total completion time. They decompose the problem and form batches for each family independently. The

best result out of four different heuristics is taken and a local search algorithm is used to improve results. Chandra et al. (2008) use a tabu search approach for scheduling a burn-in oven with non-identical job sizes and secondary resource constraints.

A real-world problem from the chemical industry which includes batching is tackled by Brucker and Hurink (2000). They propose a general-shop modeling that includes minimum time lags and present a tabu search metaheuristic approach. Kovalyov et al. (2004) discuss two-stage assembly scheduling problems with batching. They show properties of optimal schedules with batch availability and develop a polynomial heuristic algorithm.

The presence of sequence-dependent setup times is an important characteristic in complex job-shop scheduling problems. There are several works on job-shop scheduling problems that consider sequence-dependent setups without including batching machines. Ovacik and Uzsoy (1994) present a heuristic for a problem stemming from a semiconductor testing facility. Balas et al. (2008) adapt the shifting bottleneck heuristic to solve a job-shop problem with sequence-dependent setup times. Artigues and Feillet (2008) present an branch-and-bound method for a job-shop problem with sequence-dependent setup times. The method employs the solution of elementary shortest path problems with resource constraints solved using dynamic programming. Oddi et al. (2009, 2011) tackle the problem using an iterative sampling based approach. Good results for a lateness minimization problem were achieved by a tabu search based approach in González et al. (2013). Shen (2014) propose a tabu search algorithm for the job-shop problem with sequence-dependent setup times.

1.4.2 Routing and Resource Flexibility in Job-Shop Scheduling

As described in section 1.3, the machines in this work area imply complex behavior that motivates a closer look at their internal components. Complex internal behavior creates a relation to cluster tool scheduling problems and other approaches that consider properties of machines in detail. Lee (2008) provides an overview of the literature on cluster tool scheduling. A detailed description of furnace equipment is provided by Hasper et al. (1999). Kao et al. (2012) consider furnace tool allocation. Mauer and Schelasin (1993) evaluate integrated tool performance in semiconductor manufacturing by simulation. Kohn and Rose (2011) analyze and generate process time models for cluster tools in semiconductor manufacturing. The scheduling of a wet-etch station is considered in Geiger et al. (1997) and Ham (2012).

We aim at integrating the consideration of internal components within job-shop scheduling. We are not aware of other approaches that do this for machines in a semiconductor manufacturing facility. However, there are approaches that generalize job-shop scheduling in similar ways. The closest approach to our model presented in chapter 4 that we are aware of is that of Kis (2003). It describes processing alternatives as a directed graph. His approach considers also nonlinear routes given as partial orderings of operations (called *and-subgraphs*), which are not required in our case. Our approach is distinguished in particular from the one of Kis by the consideration of resource acquisitions. A generalization of precedence constraints by Möhring et al. (2004) allows a job to depend on further jobs which can be chosen from a set of alternatives. Beck and Fox (2000) present a modeling of alternative graphs

using xor and xand nodes. Čapek et al. (2012) describe a scheduling problem with alternative process plans motivated by the production of wire harnesses. Hutchinson and Pflughoeft (1994) consider general flexible process plans. Rossi et al. (2015) include routing and resource flexibility in a mixed-shop setting that combines flow-shop and open-shop problems. Barták and Čepek (2008) consider nested precedence networks with alternatives. Golmakani and Namazi (2012) describe an algorithm for a multiple-route job-shop scheduling problem. Brucker and Thiele (1996) present a branch-and-bound method for a general-shop scheduling problem. There, the assignment to machines is given and arbitrary precedence relations can occur. Ho and Moodie (1996) solve a cell formation problem in a manufacturing environment with flexible processing and routing capabilities. Ding et al. (2006) model specific behavior of machine components by introducing event graphs.

The detailed modeling approaches which we will present in chapter 4 considers individual components of machines as resources. This requires operations to demand multiple resources at the same time. The multi-processor job-shop problem was introduced as an extension of the flexible job-shop problem by Dauzère-Pérès et al. (1998). It is further generalized in Dauzère-Pérès and Pavageau (2003) by occupying the resources of a specific operation for varying periods of time. Brucker and Neyer (1998) present a tabu search algorithm for the multi-mode job-shop problem. There, a set of machines sets is assigned to each operation. For each operation, one of this set of sets has to be selected. Artigues and Roubellat (2000) present an insertion algorithm for multi-resource scheduling. Bürgy (2014) present scheduling approaches for complex job-shops with a focus on transportation which involves rails as additional resource. Kis and Pesch (2005) review exact solution methods for non-preemptive multiprocessor flow-shop problems.

1.4.3 Time Constraints in Complex Job-Shop Scheduling

The presence of time lags poses a substantial difficulty as shown by Wikum et al. (1994). Time lags render an otherwise polynomial one-machine scheduling problem into an NP-hard problem. Brucker et al. (1999) show the generality of such problems and propose a branch-and-bound approach. An overview and classification of different maximum time lag constraints that appear in semiconductor manufacturing is given in Klemmt and Mönch (2012). They also provide a MILP based decomposition heuristic for a job-shop setting without batching machines. The combination of non-adjacent maximum time lag constraints is often denoted by the term queue time constraints.

Yurtsever et al. (2009) describe a custom heuristic which has been deployed in an industrial setting and which includes maximum time lag constraints. Jung et al. (2013) consider maximum time lags and their violations in the diffusion and cleaning area of a semiconductor manufacturing facility. They propose to use a MILP based heuristic in a rolling horizon environment. They limit each planning horizon by allowing only a certain number of operations to be scheduled on each tool. Kohn et al. (2013) present a parallel batch machine scheduling problem that includes maximum time lag constraints. They propose to use a VNS based approach for a parallel batch machine scheduling problem in combination with simulation.

They study the correlation between different key performance indicators and discuss the incorporation of a minimum batch size constraint. Maximum time lags (denoted *time bounds* in that work) are considered as primary objectives in a lexicographical objective function. Scholl and Domaschke (2000) describe a simulation based approach that includes maximum time lag constraints. All approaches discussed before do not consider a job-shop setting and therefore can consider in their optimization methods only maximum time lag constraints that have already started. Yugma et al. (2012) present an approach for scheduling complex job-shops that considers maximum time lags between adjacent operations.

Cho et al. (2014) consider queue time constraints in the context of wafer fabrication using a higher abstraction level. They use gate-keeping decisions to specify whether the processing of an operation that initiates a maximum time lag can start. Sadeghi et al. (2015) (a) develop an approach to support the same type of decisions by estimating the probability of a job to satisfy its maximum time lag constraints. Sadeghi et al. (2015) (b) propose a simulation-based approach to control jobs that are within maximum time lag constraints. The goal of these two papers is not to determine optimized schedules but rather to control jobs in rule-based dispatching systems.

Only few works consider maximum time lags within job-shop scheduling problems. Lacomme et al. (2012) consider maximum time lags between arbitrary operations and propose a randomized heuristic which extends an approach of Deppner and Portmann (2006). Other approaches consider maximum time lags only between adjacent operations, e.g. González et al. (2015) present a scatter search approach which applies tabu search and path relinking to tackle a job-shop scheduling problem with time lags and sequence-dependent setup times. Artigues et al. (2011) present a heuristic approach using an insertion heuristic and resource constraint propagation. Grimes and Hebrard (2015) include time lags in a generic constraint programming based approach. Caumond et al. (2008) propose a memetic algorithm.

Botta-Genoulaz (2000) considers maximum time lags within a flow-shop scheduling problem. Fondrevelle et al. (2006) consider time lags between successive operations in permutation flow shops. Dhouib et al. (2013) propose a mathematical programming formulation and a simulated annealing heuristic approach for a permutation flow-shop scheduling problem with sequence-dependent setup times and maximum time lag constraints. Bartusch et al. (1988), Dorndorf et al. (2000), Nonobe and Ibaraki (2006) and Schwindt and Trautmann (2000) consider maximum time lags in the context of resource constraint project scheduling. Hurink and Keuchel (2001) consider maximum time lags in single-machine scheduling. Raaymakers and Hoogeveen (2000) study no-wait constraints. Rossi et al. (2002) consider soft time constraints in a more general setting. Zhang and van de Velde (2010) consider an online open-shop problem with time lags.

1.5 Overview and Main Contributions

We present an overview of the structure of this work and highlight the main contributions of the individual chapters:

Chapter 2, Industrial Problem Specification
This chapter provides a detailed specification of the diffusion and cleaning work area at the interface between engineers from a fab and combinatorial optimization researchers. It is based on information obtained from the engineers of two real-world fabs. The first goal is to provide a clear, textual, in-depth description verified by fab engineers. The second goal is to provide a comprehensive description which allows formal models for optimization methods to be developed.

Chapter 3, Complex Job-Shop Scheduling: A Batch-Oblivious Approach
This chapter considers complex job-shop scheduling problems which include the main characteristics of the diffusion and cleaning work area: The integration of batching machines within a job-shop environment. Our main contribution is a novel batch-oblivious disjunctive graph representation which uses edge weights to represent batching decisions. This representation facilitates the modification of batching decisions and allows the development of an original "on the fly" batching algorithm. Complemented by a known move for integrated resequencing and reassignment of operations, we obtain an integrated neighborhood which is applied within a parallel GRASP based meta-heuristic approach.

Chapter 4, Extended Route and Resource Flexibility in Job-Shop Scheduling
We increase the detail of our model for the diffusion and cleaning area by including internal components of machines. The combination within a job-shop setting leads to a job-shop scheduling model with extended resource and routing flexibility. A main contribution is the inclusion of resource acquisition constraints which help to model the exclusive acquisition of a resource between two operations of the same job. This modeling offers an additional generality that e.g. allows to solve scheduling problems with auxiliary resources from the photolithography area.

Chapter 5, Time Constraints in Complex Job-Shop Scheduling
Temporal constraints that limit the maximum time between given pairs of operations are taken into account in this chapter. We consider maximum time lag constraints as soft constraints and include the severity of their violations lexicographically in the objective function. The main contribution is the combination of time lags within a complex job-shop scheduling problem.

Chapter 2

Industrial Problem Specification

Optimization methods that are useful in practice require a thorough understanding of the real-world problem to be solved. An accurate textual specification helps to find and understand reasonable abstractions needed in formal models, and it can be easily verified by domain experts.

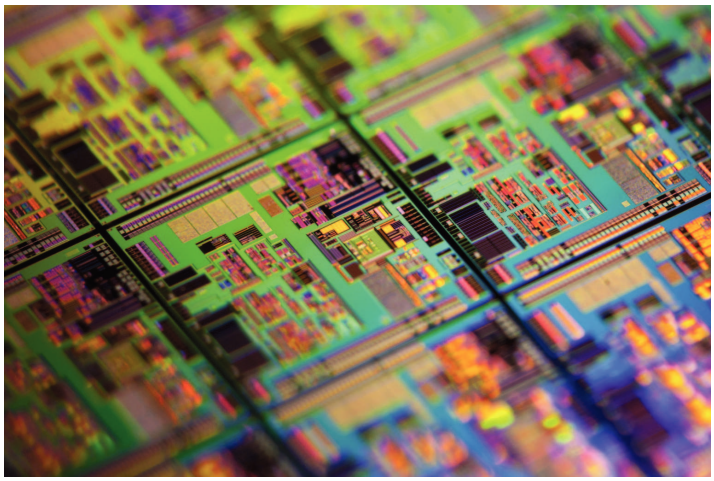


Image source: Flickr, Rob Bulmahn
http://www.flickr.com/photos/rbulmahn/ (CC License)

This chapter provides a detailed specification of the diffusion and cleaning work area at the interface between fab engineers and combinatorial optimization researchers. It was written in the context of the European project ENIAC INTEGRATE based on information from engineers of two real-world fabs and following numerous meetings and discussions. The specification was written by the author of this thesis and verified in an iterative proof reading process by fab engineers. The first goal of this chapter is to provide a comprehensive, textual, in-depth description of all equipment and constraints that should be observed by an automatic scheduling algorithm. Potential restrictions due to limitations of known or future scheduling algorithms are not in the scope of this specification. The second goal of this chapter is to provide a foundation to develop formal models for scheduling in this work area.

Section 2.1 describes the general properties of the shop floor in the diffusion and cleaning area without considering machine types in detail. This is done in section 2.2, where different types of machines are described and their behavior is modeled in detail. Section 2.3 describes the objectives that should be optimized when calculating schedules for this work area. Later chapters in this thesis provide formal models and solution methods for the most crucial aspects specified here.

2.1 Basic Model

The basic entity we consider in this specification is a *lot*: This is a set of wafers that need to be processed together. A *wafer* is a thin slice of semiconductor material that needs to undergo hundreds of processing steps during its manufacturing process. We are given a set of lots that need to be scheduled. For each lot, its number of contained wafers is given (up to 25). Depending on the fab, each lot is contained in a FOUP (Front Opening Unified Pod used in 300 mm fabs or modern 200 mm fabs) or a cassette (used in 200 mm fabs). We model this by using the term *container* to denotes a storage for one or more lots. A container may store lots for different products or customer orders (see multiple items per job, Mönch et al. (2011)). We assume the assignment of lots to containers to be given in advance. This assignment is fixed and not subject of our optimization efforts.

A given sequence of processing steps has to be performed for each lot that is processed in this work area. In the literature, a step is called an *operation*. In this chapter, we use the term *step* instead since, within STMicroelectronics, the term *operation* is used with a different meaning. The order of steps is fixed and this sequence is called the *route* of the lot. Each step has to be performed by a *machine* that needs to be selected from a given set. We denote the execution of a step also as *processing* of a lot. Each lot can be processed by only one machine at a time. Once a process is started, it cannot be interrupted (preemption is not allowed). For each step, a *recipe* is given that specifies properties of its processing. Different steps can share the same recipe. A step can be processed only on machines that are qualified for its recipe. For each machine, we are given a list of qualifications. A *qualification* defines that a specific machine is able to process a specific recipe and it provides corresponding processing parameters. We call a machine qualified for a step if it has a suitable qualification.

Some machines can process multiple lots at the same time. A set of lots that is simultaneously processed on the same machine is called a *batch*. To obtain a consistent notation, we define the term batch for all kinds of machines—even if they are incapable of batching. We cannot combine arbitrary lots in a batch. To specify feasible batches, we are given a *family* for each qualification. We call two qualifications *compatible* if they refer to the same machine and have identical families. Only steps that are processed with compatible qualifications can be combined in a batch. We are given *machine capacities* that restrict the number of containers and the number of wafers that can be processed within the same batch. Capacities depend on the qualification (recipe and machine) used to process the batch. They specify a maximum number of containers and a maximum number of wafers. Analogously, we can be given a *minimum batch size* that specifies the minimum number of wafers in a batch.

Machines process lots enclosed in containers. However, within one processing step, they do not necessarily process all lots of one container; some could be processed while others are waiting. Note that lots within the same container could require different recipes. This could prohibit to process the lots of a container together.

2.1.1 Time constraints

We represent each point in time as a natural number corresponding to seconds. For each step, its *processing duration* is determined by the qualification corresponding to the machine that is selected to process the step. Before or after processing, machine dependent processes such as loading, unloading, or cooling may take place. Section 2.2 details those processes and their durations for each machine type.

Containers of lots need to be transported between machines. So, we have to consider the time needed for material handling and transportation. To model this, we prescribe a *transport duration* between consecutive steps. This duration specifies for all lots the minimum duration between the end of unloading of a machine until the beginning of loading on the succeeding machine. It is dependent on the distances between machines and therefore given for all unordered pairs of distinct machines. We also have to consider the time to prepare a machine for a different recipe. This changeover duration imposes *sequence-dependent setup times*. Note that they are relevant only for some machines. For each affected machine, a setup duration is given for each pair of qualifications. It defines a minimum duration between the end of processing of a step and the beginning of processing of the following step on the same machine.

Physical and chemical processes may set up time constraints between different steps. For example, the time between some steps must be limited to avoid contamination and oxidation. So, we can be given *maximum time lag* constraints for all unordered pairs of distinct steps of each lot. Such a constraint specifies the maximum period of time between the beginning (or end) of processing of a step until the beginning (or end) of processing of the following step in the route of the lot. Analogously, *minimum time lag* constraints are given to constrain the minimum period of time between steps. Those constraints relate to the actual processing of lots on the machine, not to loading, unloading or other auxiliary activities. Figure 2.1

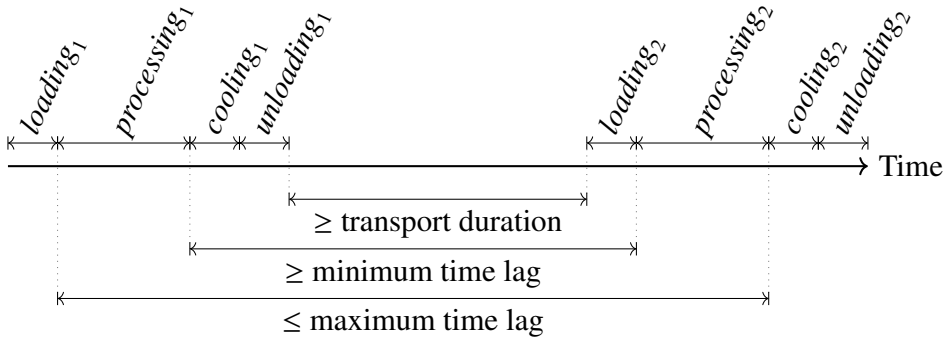


Figure 2.1 – *Time constraints for two consecutive steps of a lot*

illustrates an example for time constraints between two consecutive steps. Maximum time lag constraints can be chained in the sense that there can be steps where one maximum time lag constraint ends and another one starts. For resulting schedules, we always require that both constraints are observed. As a consequence, in some cases, starting the first step of the earlier constraint must be delayed because the later constraint can otherwise not be fulfilled.

Machines sometimes must be shut down for maintenance, repair or other work. Such planned down times are provided by given *machine availability* periods for each machine. Outside of these time windows, machines neither can perform processing steps nor can they be loaded or unloaded.

2.1.2 Control Runs

The quality of the manufacturing process must be permanently monitored. For this purpose, *control runs* are conducted: Specific monitoring containers are added within the course of manufacturing. They contain monitoring wafers that are used to measure the state of production quality and to detect defects in machines or wafers. We need to consider control runs only for specific types of machines where they take up room regarding the limited machine capacity.

Each control run needs to include an additional control run container. Thus, regarding the limited machine capacity, the execution of a control run reduces the number of payload containers that can be processed at the same time. Note that control runs cannot be affected without production containers. Consequently, we require that the container capacity is at least two for all machines that conduct control runs. To guarantee a certain amount of executed control runs, we introduce the term *control run quantity*. It specifies the maximum number of consecutive processing steps without a control run. *Control run quantities* are given per machine (defectivity) and per qualification (thickness).

2.1.3 Moves and Priorities

In order to count the number of performed processing steps, we introduce the term *move*. This is defined as the processing of a single wafer on a machine. The *number of moves of a batch* is the number of wafers contained in that batch. To be able to prefer specific lots, we introduce a *priority* for lots. This is a positive natural number that is given for each lot. Larger priority values indicate a greater importance. They can be used to prioritize lots over each other. For example, some manufactured wafers may have a larger business value than others.

We want to measure productivity while taking priorities into account. So, analogously to the term move, we define the term *weighted move*. One weighted move is defined as the processing of a lot with priority one that contains exactly one wafer. The *number of weighted moves of a batch* is defined as follows:

$$(\text{\#weighted moves of Batch } B) = \sum_{(\text{Lots } L \text{ in } B)} (\text{priority of } L) \cdot (\text{\#wafers of } L) \,.$$

Recall that wafer fabrication is a reentrant process: Each lot visits a work area multiple times. As mentioned in the discussion of order release decisions in section 1.2, in order to globally manage the flow of production in the whole fab, individual work areas must balance their number of moves depending on the production stages of their lots in progress. Therefore, a production target constraint is introduced in the following. Each step of each lot belongs to a *production target group*. For each production target group, we are given a *production target constraint*. It prescribes that the number of moves belonging to this production target group must stay between given minimum and maximum numbers during the considered planning horizon.

2.1.4 Interlacing Constraints

Certainly, we restrict our scope of optimization to a limited area and time period. In this section, we describe constraints that interlace the current scope with adjacent scopes. Those include past and future issues of this and other areas of the fab. We restrict our scope to a fixed period of time (e.g. 24 hours) that is called *planning horizon*. The system to be developed needs to make scheduling decision for this planning horizon.

Now, we regard the state of the diffusion area at the beginning of our planning horizon. There, machines can be occupied with *in-process lots*. These machines are not available until their respective in-process lots are finished and unloaded. Additionally, initial states for all machines and its relevant components must be given for the beginning of the planning horizon.

Consider that in-process lots may have triggered maximum time lag constraints before the current planning horizon. As well, a maximum time lag constraint might have been triggered before in a different area of the fab. Both are modeled as a *step due date* that prescribes the completion time for a step of a lot. It can be given independently for all steps of each lot. In

contrast to the maximum time lag constraint that defines relative time periods, it prescribes absolute points in time.

Containers that need to be processed in the diffusion area arrive from other areas of the fab. Not all of them are immediately available. So, for each container, an *initiation date* is given that specifies its arrival date in the diffusion area. The initiation date can reflect known arrival dates of containers in the past or projected future arrival dates. The *ready date* of a container specifies the earliest point in time the container can be scheduled. It takes the beginning of the planning horizon into account. Processing of the contained lots (including loading wafers to machines) cannot begin earlier than this point in time.

We do not want to set up time constraints for future planning periods that will not be satisfiable. To avoid this, we must take care of the maximum time lag constraint. If we schedule only some but not all steps of a lot, the remaining steps have to be scheduled in a future planning horizon. However, this partial scheduling may trigger time constraints that become step due dates in the next planning horizon. So, we want to ensure it remains possible to schedule them later. Due to this, we define a *lot completeness constraint*. This prescribes that it is not allowed to schedule only a portion of the steps of a lot. For each lot, all of its steps have to be scheduled.

2.2 Machine Types

In the following section, the modeling of different machine types is explained. We distinguish between five types of machines: *Serial single-wafer machines*, *parallel single-wafer machines*, *batch machines with a unique chamber*, *furnaces*, and *wet bench machines*. Beside the qualification dependent processing duration, other activities may be of importance. They depend on the machine characteristics described in the following.

2.2.1 Serial Single-Wafer Machines

A serial single-wafer machine sequentially processes wafers one after another. We observe an overlap between the processing periods of consecutive wafers: The processing of the current wafer can start before the preceding wafer is completed. This overlap is quantified by a minimum duration Δ between the start dates of two consecutive wafers. Moreover, we are given the processing duration p of a single wafer. Both durations are given per qualification. Together, the *processing duration* of a lot A is given by

$$p_A + \Delta_A \cdot (\#\text{wafers}_A - 1).$$

Analogously, an overlap is possible between wafers of different consecutive lots. In the following, we describe the computation of the start date of a lot B that directly follows a lot A on the same serial single-wafer machine. For simplicity, we assume the tool and both lots to be immediately available. We denote the completion time of lot A as C_A. We denote the

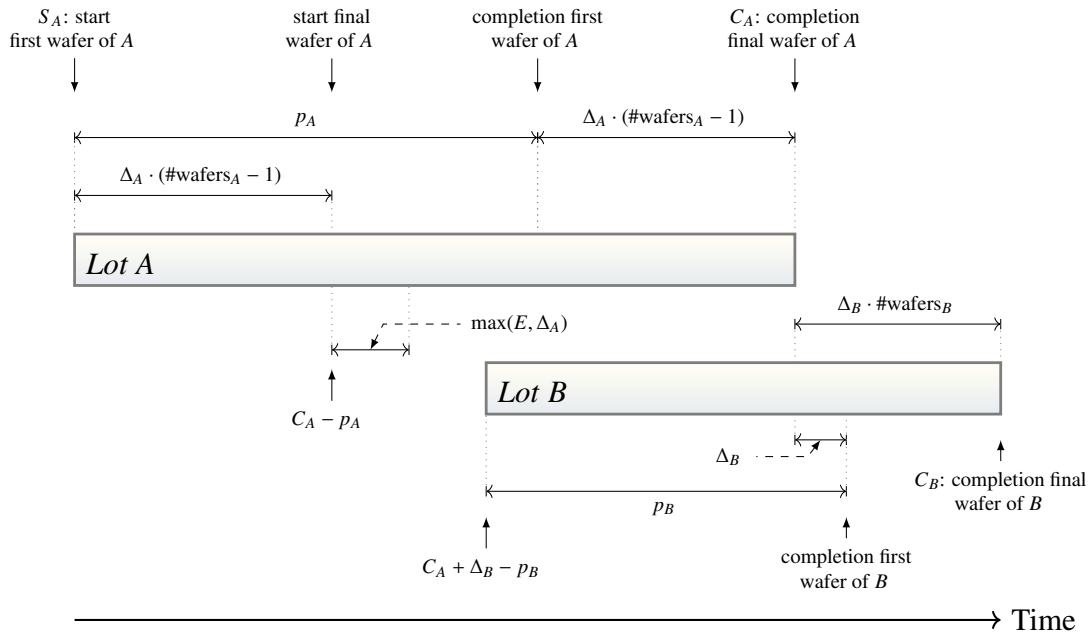


Figure 2.2 – *Overlapping behavior for two subsequent lots on a serial single-wafer machine*

sequence-dependent setup time between the lots as E. Then, the processing of lot B can start at the time

$$C_A + \max(\max(\Delta_A, E) - p_A, \Delta_B - p_B).$$

Figure 2.2 illustrates the computation of start times with an example that includes two consecutive lots. Note that the processing duration already contains the time needed for loading and unloading. So, we do not need to additionally take loading durations into account. We can neglect control runs for serial single-wafer machines since they do not affect any relevant parameters of our schedules. As well, capacities are not relevant for single-wafer machines. Load ports of serial single-wafer machines are not included in the model since we assume that they are not a critical resource. Note that in the literature (see Lee (2008); Lee and Lee (2010)), such serial single-wafer tools are also called pipeline tools, which emphasizes that operations can overlap.

2.2.2 Parallel Single-Wafer Machines

A parallel single-wafer machine behaves almost like a set of identical serial single-wafer machines. This set is given as a *number of parallel processing steps* for each parallel single-wafer machine. However, those parallel processing components are not entirely independent. They are linked by a *mutual family exclusion constraint*: Two lots cannot be processed at the same time on a parallel single-wafer machine if they are *compatible* (their qualifications share the same family). Note that the loading robot of parallel single-wafer machines is not considered in the model since we do not assume it to be a critical resource.

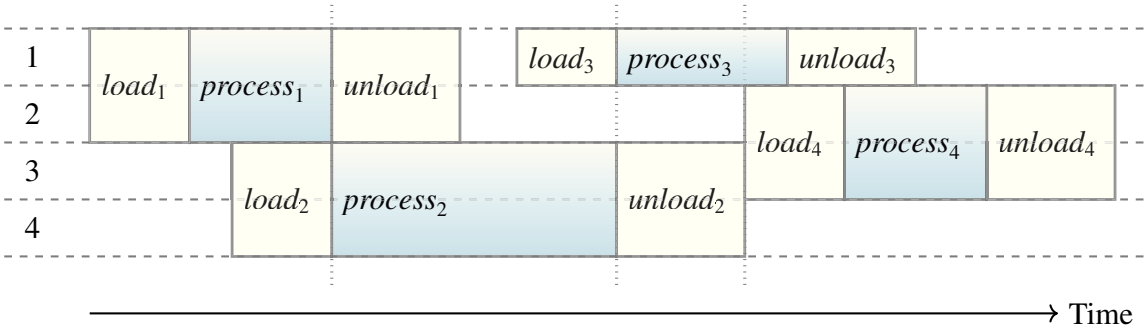


Figure 2.3 – *Four consecutive batches on a batch machine with a unique chamber*

2.2.3 Batch Machines with a Unique Chamber

A batch machine with a unique chamber consists of a *chamber* for processing and several *load ports*. It can process wafers from multiple lots commonly within one batch. The maximum number of wafers per batch is usually fixed to 50 and the maximum number of containers per batch is two. A *number of load ports* is given for each batch machine with a unique chamber. Load ports can be occasionally unavailable: We are given *load port availability* periods for each load port.

For each qualification, we are given a batch *processing duration* that is independent of the number of involved wafers and containers. Before processing, wafers must be loaded; after processing, they must be unloaded. So, for each batch machine with a unique chamber, we are given a *loading duration* and an *unloading duration* that describe durations per batch. As well, these durations are independent of the number of wafers and containers in the batch. Note that loading and unloading are crucial since their durations do not differ much from actual processing durations. Batch machines with a unique chamber do not take sequence-dependent setup times into account.

Each container involved in the processing of a batch occupies a load port from the beginning of loading until the end of unloading. Since the number of containers per batch is limited to two, either one or two load ports are occupied during that period of time. Loading, unloading, and processing can only take place if the number of available load ports is sufficient. All load ports have the same properties and are indistinguishable. We can neglect control runs for batch machines with a unique chamber.

The chamber is occupied during the actual processing of a batch, but not during loading and unloading. Only one batch at a time can be processed in the chamber and processing cannot be interrupted. Figure 2.3 provides an example that illustrates the usage of such a machine. In this example, batches 2 and 3 cannot start processing before the chamber is available. Batch 4 cannot load wafers as long as load port 3 is occupied.

2.2.4 Furnaces

A furnace can process a set of containers in parallel within a batch. To obtain a realistic model, we need to consider internal properties of furnaces. They consist of *tubes*, *boats* and a *robot*. A *tube* is the place where processes are conducted. A *boat* is a moveable carrier for wafers and necessary to run a process inside a tube. Boats are also utilized to load, unload and cool wafers. The *robot* is needed to load and unload containers. Essentially, a boat is used as follows: first, wafers are loaded from its containers to the boat using the robot. Then, the boat is moved into the tube where the process is conducted. Afterwards, the boat is removed from the tube and has to cool down before its wafers can be unloaded using the robot. It can happen that the boat has to wait in case the tube or the robot is occupied. All wafers that are part of the same batch stay together from the beginning of loading until the end of unloading. In the fabs we consider, there exist two types of furnaces. The simpler one consists of one tube and one boat. The more complex one consists of two tubes and four boats (two assigned to each tube). For both types of furnaces, we need to observe synchronization constraints. We describe them for the complex case first and continue with the simpler one.

For technical reasons, within each tube there must always be a process running. If there is no process conducted on wafers, a so-called *standby process* has to be performed instead. This standby process requires an empty boat. This means: If there is no boat performing a production process, then one boat must perform the standby process. As for real processes, the boat needs to cool down after the standby process. When one boat leaves the tube, another one has to enter the tube. This is called *boat swap*. Robots and tubes can be used by only one boat at the same time. A boat corresponds to a specific tube and cannot use the other tube.

We subsume these requirements as *boat synchronization* constraints. The described behavior of boats can be modeled as a state machine as given in Figure 2.4. It describes the interplay between the states *swap in*$_{\circ}$ (*si*$_{\circ}$), *standby*$_{\circ}$, *swap out*$_{\circ}$ (*so*$_{\circ}$), *cooling*$_{\circ}$, *waiting*$_{\circ}$, *loading*$_{\circledcirc}$, *unloading*$_{\circledcirc}$, *swap in*$_{\bullet}$ (*si*$_{\bullet}$), *processing*$_{\bullet}$, *swap out*$_{\bullet}$ (*so*$_{\bullet}$), *cooling*$_{\bullet}$, *waiting1*$_{\bullet}$ and *waiting2*$_{\bullet}$. The following visualization scheme is used in this and later figures: A darker (lighter) background color and straight (dotted) lines indicate the boat is (not) loaded. Sharp (round) edges indicate that the tube is (not) used by this boat. For each boat its own state is maintained. Examples are given in Figures 2.5 - 2.7. Figure 2.5 shows a case where both boats are used in an alternating way. Both of them are qualified for the wafers to be processed. In Figure 2.6, boat 2 does not process any wafer (e.g., because there are no lots boat 2 is qualified for). Therefore, it performs a standby process while boat 1 is preparing the next processing step. In Figure 2.7, there is no batch loaded on boat 2 when boat 1 finishes processing. So, boat 2 has to do the standby process. Then, wafers arrive that can be processed only with boat 2. Next, boat 1 runs the standby process while boat 2 can prepare processing.

The preceding paragraph considered the case of two boats per tube. We now adapt the previous constraints for the case that only a single boat is available. In this situation, we relax the constraint that requires the tube to be used all the time. Instead, we want the single available boat to use the tube as much as possible. So, we do not allow it to be in

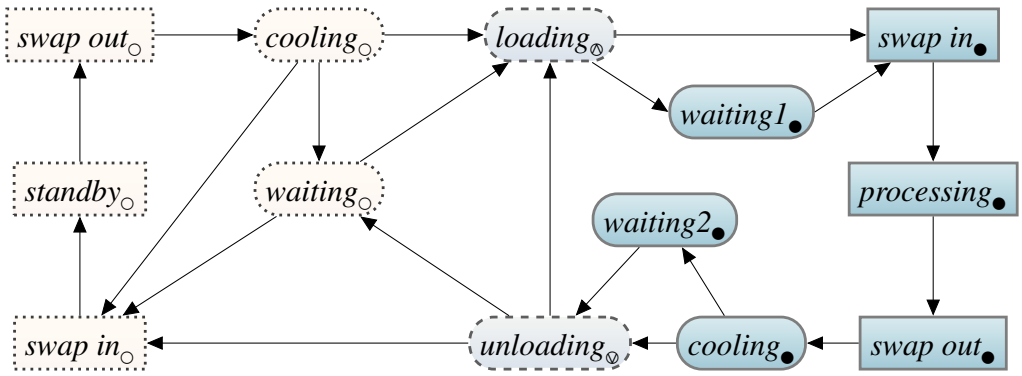


Figure 2.4 – *A state diagram describing boat behavior*

the state *waiting*$_{\bigcirc}$. Hence, after being in the state *unloading*$_{\circledcirc}$, it must either proceed to *swap in*$_{\bigcirc}$ or *loading*$_{\circledcirc}$. In the example shown in Figure 2.8, only a single boat is available. After it is unloaded, further containers are ready to be processed. So, the boat can load them immediately to continue processing. In the example shown in Figure 2.9, there also is only a single boat available. After it is unloaded, no containers are ready to be processed. So, the boat starts to perform the standby process.

We stated previously that qualifications are given for each machine. However, for machines with multiple boats, we need a more detailed modeling. In such cases, qualifications are given for each boat separately. So, one boat could be qualified for a certain recipe while a different boat of the same machine is not. The *processing duration* of a batch is given by the qualification of the used boat. It is independent of the number of wafers and containers to be processed. Some recipes prescribe the tube to stay empty for some time to cool down before the process starts. We assume that it is sufficient to model this by including this idle time as increased processing duration. A *cooling duration* is given per boat. It specifies the time needed for cooling after standby or production processes. A *minimum standby duration* is given for each boat and prescribes standby processes to take at least the given period of time. Also, the *loading duration* and the *unloading duration* are given per boat and are independent of the number of involved wafers and containers. A *boat swap duration* specifies the duration of a boat swap and is given per tube.

Boats as parts of furnaces have availability periods, too. These *boat availability* periods are given for each boat separately. So, one boat of a machine can be available while the other is not. In this case, the synchronization constraints for furnaces with a single boat apply. We require the state of a boat to be always *waiting*$_{\bigcirc}$ outside of its availability periods. We need to consider control runs for *furnaces*. Since we may have two tubes and four boats in one machine, control run counters are maintained per tube. Note that sequence-dependent setup times are not relevant for furnaces.

Furnaces include *internal buffers* to store containers. For each furnace, a *buffer capacity* specifies the number of containers that can be stored at the same time. Containers must be stored in an internal buffer while their wafers pass through the furnace. For each furnace, we

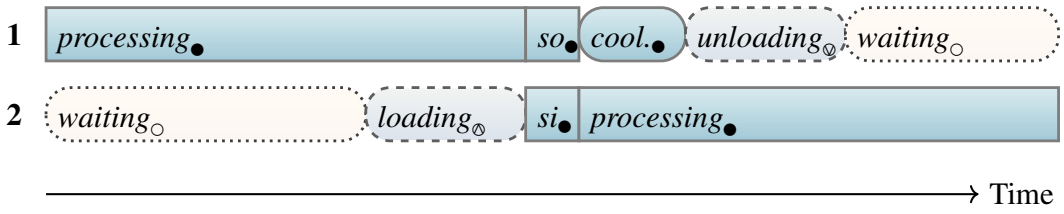


Figure 2.5 – *Both boats are used alternately*

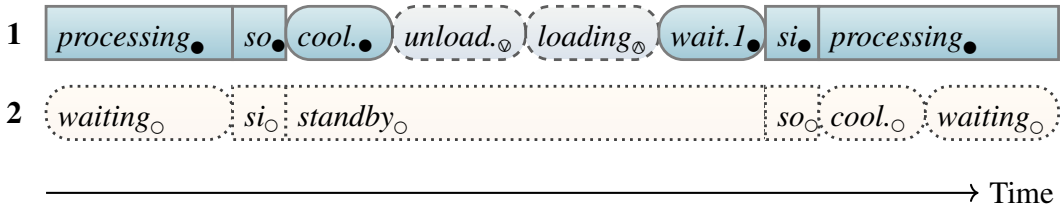


Figure 2.6 – *One standby process needs to be performed*

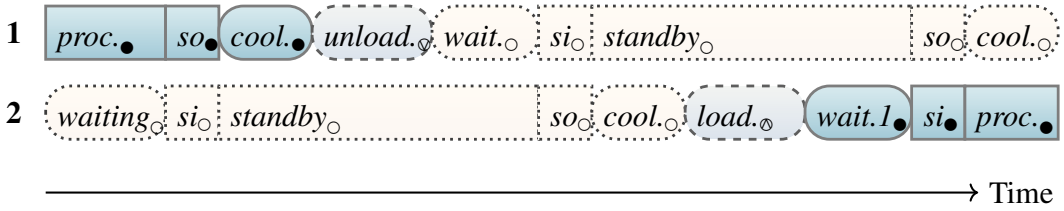


Figure 2.7 – *Two subsequent standby processes are necessary*

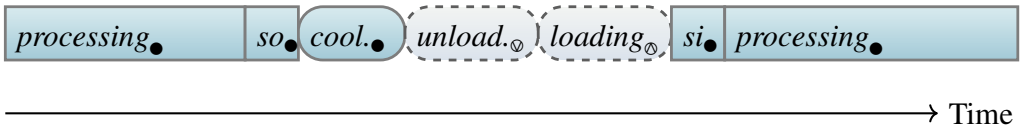


Figure 2.8 – *A single boat and immediate continuation of processing*

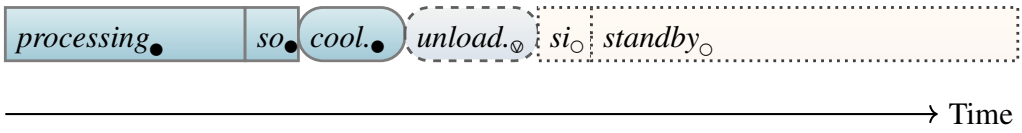


Figure 2.9 – *Processing followed by standby for a single boat*

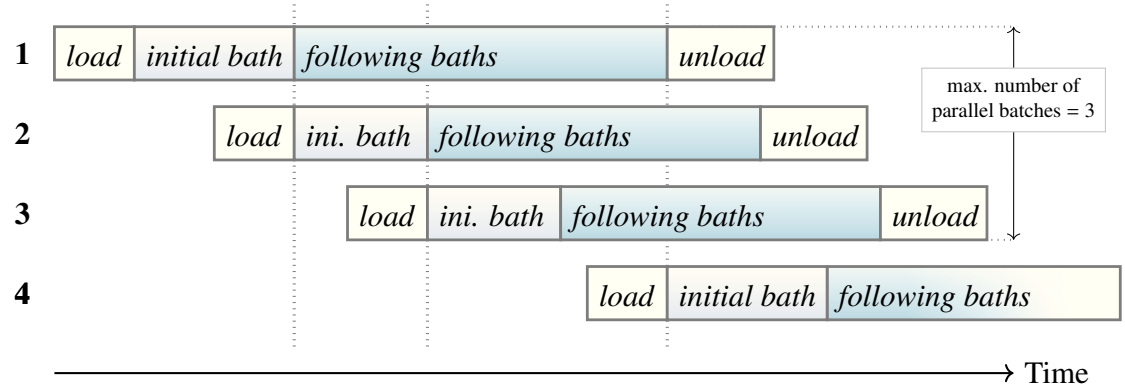


Figure 2.10 – *Four consecutive batches on a wet bench machine*

can constantly keep track of the current buffer usage as follows: Every time a container is loaded, the current buffer usage is increased by one. Analogously, if a container is unloaded, the current buffer usage is reduced by one. As long as the buffer capacity is sufficient, loading and unloading durations are not affected. This may not always be the case: A *buffer exceedance penalty duration* is applied if the buffer capacity is exceeded. It means that this penalty duration is added to related loading and unloading durations.

2.2.5 Wet Bench Machines

Wet bench machines consist of several *bath tanks* in a row. Wafers are processed by traversing a sequence of chemical baths in those tanks. Such machines are capable of batching, so multiple wafers from different lots can be processed at the same time. Moreover, the presence of multiple baths allows more than one batch to be processed at the same time. However, different batches cannot start at the same time; they are processed in a kind of pipeline. A new batch can start processing after the preceding has left the initial bath tank. So, we are given an *initial bath duration* that specifies the time spend in the first bath. For the overall duration, we are given a *processing duration* which includes the time of the initial bath. Processing cannot be interrupted during this period of time. Both durations depend on the recipe.

The number of batches that can be run at the same time on a wet bench machine is limited: For each wet bench machine, we are given a *maximum number of parallel batches*. We also need to take loading operations into account: For each wet bench machine, we are given a *loading duration* and an *unloading duration* that describe durations per batch. They are independent of the number of involved wafers and containers. Sequence-dependent setup times are not relevant for this kind of machines. Figure 2.10 provides an example for a sequence of four batches that are performed consecutively on a wet bench machine. Batches 2 and 3 cannot start before the initial bath tank is available. Batch 4 cannot start before the number of running batches is below three. We can neglect control runs for wet bench machines. This modeling is slightly simplified since it ignores the overtaking of lots that can happen in case the set of subsequently used bath tanks is different.

2.3 Objectives

In the following, we describe the objectives to be optimized. Note that we can state them independently of any means used to improve them. This is an advantage of scheduling systems over rule-based systems. An objective function is a mapping from the set Π of schedules for the given problem to an ordered set. There is a plethora of possibilities to set up objective functions. Hence, the scheduling system to be developed should offer a flexible way to exchange them. In the following, we describe different objectives and an initial approach to combine them. For this multicriteria setting, we suggest a combination of a lexicographical ordering and a linear combination.

2.3.1 Minimize Constraint Violations

Our first objective is to create a feasible schedule. This is not always possible since the given constraints could be too restrictive. To be able to return a schedule in any case, we turn some of the constraints into objectives. However, since constraints are not negotiable, we treat them as objectives of highest importance. Those objective functions stemming from constraints return zero in case their respective constraint are fulfilled. In case the constraints are violated, the associated objective function returns a positive value which indicates the severity of the constraint violation. Our goal is to minimize these functions. If all of them are equal to zero, we have found a feasible schedule. So, in the overall objective function, these violation related objective functions do not have any influence on the assessment of feasible solutions.

In the following, we describe the constraints to be converted to objective functions such that a schedule can be computed for any given input. For this, we replace the constraints for *maximum time lags*, *production targets* and *availability periods* by the following objectives. For each objective, we define its amount of violation in the following.

Total time lag violation (g_1) Lots that have not yet started the processing of a time lag triggering step can always be scheduled without maximum time lag violations because the start of the constraint can still be delayed. However, ongoing maximum time lags become fixed due dates for the final operation of the time lag. Since time lags might be nested or chained, lots with due dates induced by time lags can imply other unstarted time lags that might become unsatisfiable as a consequence. The scheduler has to deal with the possibility of such time lag violations and we must define appropriate violation costs. We distinguish *reworkable* time lags from non-reworkable time lags. Lots with violated reworkable time lags need to be reworked in case a time lag violation occurs. This imposes a *rework cost*. Lots with violated non-reworkable time lags have a defectivity risk that rises increasingly with the duration of the maximum time lag violation. Once a non-reworkable lot is defect, it must be scrapped which induces a *scrap cost*. We assume that scrapping is inevitable once a certain violation duration is surpassed. Lots that must be reworked or scrapped remain unscheduled since we know that processing them is pointless. Thus, for each maximum time lag, a *violation cost k*,

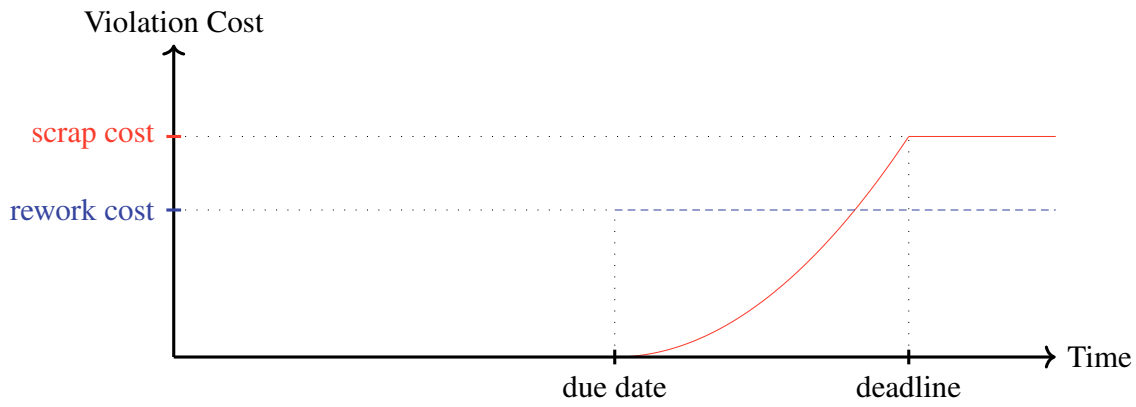


Figure 2.11 – *Time lag violation cost as a function of the completion date of a lot.*

a *maximum duration* d (a relative due date) and an *ultimate duration* γ (a relative deadline) is given. The ultimate duration must be equal to or greater than the maximum duration. For a completion time C of the considered operation, the violation severity of the time lag is specified as

$$\begin{cases} 0 & \text{if } C \leq d \\ k & \text{if } C > \gamma \\ k \cdot \frac{(C-d)^2}{(\gamma-d)^2} & \text{else} \end{cases}$$

Figure 2.11 illustrates the violation cost as a function of the completion time of a lot. Maximum and ultimate durations are denoted as due date and deadline, since Figure 2.11 assumes that the time constraint has already started. For non-reworkable lots, the ultimate duration is greater than the maximum duration. For reworkable lots, the time lag maximum and ultimate durations are equal. We want to minimize the sum of all time lag violations over the maximum time lags of all scheduled steps.

Total production target gap (g_2) We want to minimize the sum of the production target gaps over all production target groups defined in section 2.1.3. A production target gap for a production group is the difference between the scheduled and the minimum number of moves for this production group in the planning horizon. The gap is zero in case a sufficient number of moves is performed.

Amount of availability period violations (g_3) We want to minimize the sum over the durations of all periods during which operations are scheduled outside of availability periods.

2.3.2 Objectives for Feasible Schedules

In the following, we specify objectives to be optimized for feasible schedules. The objectives stem from experience of a preceding research project about scheduling in the diffusion area (Yugma et al. (2012)). In addition to those, we include minimizing the number of control runs as a new objective.

Weighted flow factor (h_1) To specify the first objective, we introduce the *theoretical route duration* of a lot. This determines the shortest possible manufacturing duration of a lot. In order to define this, we choose the fastest machine for each step of the lot. The fastest machine for a step is the one with the smallest sum of the durations for loading, processing and unloading (plus boat switching and cooling durations if a furnace is involved). This sum is what we call the *minimum duration of a step*. The theoretical route duration of a lot is the sum of the minimum durations of all steps of its route plus the minimum time lags between them.

The *actual route duration* of a lot is the time between the initiation date of the lot and the completion date of the lot (end of unloading after its final operation). The *flow factor* of a lot is its actual route duration divided by its theoretical route duration. Now, the *weighted flow factor* that we want to minimize is the weighted average of all flow factors. This criterion is also known as *x-factor*. We only take lots into account that are completed within the planning horizon. With L denoting the set of lots completed within the planning horizon, the objective can be written as

$$\frac{1}{\sum_{l \in L} (\text{priority of } l)} \cdot \sum_{l \in L} \frac{(\text{priority of } l) \cdot (\text{actual route duration of } l)}{(\text{theoretical route duration of } l)}.$$

Number of weighted moves (h_2) Our next objective is to maximize the *number of weighted moves* in the given planning horizon. The definition of this term was given in section 2.1.3. This objective is related to the throughput of the working area.

Batching coefficient (h_3) The *batching coefficient* is a performance indicator that describes the capacity usage of the operations performed on batching machines. It is calculated as the number of (unweighted) moves in the planning horizon divided by the sum of the number of batches performed on each machine, times the maximum capacity of that machine. The denominator is the number of wafers that would have been processed in case all machines have been loaded for every batch to their respective maximum capacities. With M denoting the set of machines and H denoting the planning horizon, the objective can be written as

$$\frac{(\#\text{moves in } H)}{\sum_{m \in M} (\#\text{batches performed on } m \text{ in } H) \cdot (\text{wafer capacity of } m)}.$$

Note that this quotient is equal to one if all batches are filled up to their maximum wafer capacity. We include the objective of maximizing the batching coefficient into our

objective function. This objective can be seen as a means to support the improvement of other goals. This objective also avoids the cost that is associated to each machine run.

Control runs (h_4) Each time a control run is performed, a monitoring container is used. Since such a usage induces a certain cost, we want to minimize the number of *control runs* within the planning horizon as an additional objective.

2.3.3 Combination of Objectives

As stated before, our first goal is to compute a schedule that is feasible. For feasible schedules, we want to optimize the objectives h_i given in the previous section. In summary, we distinguish two kinds of objectives: Objectives to minimize violations and objectives to improve feasible schedules. This suggests to link the two kinds of constraints in a lexicographical way. Objectives of the same kind can be combined in a scalar way as proposed by Yugma et al. (2012). Hence, we introduce parameters α_i and β_j to weight the presented objectives. We define our combined objective function f as follows:

$$f: \quad S \to \mathbb{R}^2, \quad x \mapsto \left(\sum_{i=1}^{3} \alpha_i \cdot g_i(x), \ \sum_{j=1}^{4} \beta_j \cdot h_j(x) \right).$$

Note that the different objectives functions are not comparable regarding their scale and units. So the given parameters α_i and β_j do not only provide a weighting; they provide a normalization as well. We discuss this issue in section 5.3.2 and tackle it by applying ideas known from multicriteria optimization.

2.3.4 A Discussion of Flow Factor Definitions

We have seen a definition of objective functions that fits the needs of a particular fab. Regarding the definition of flow factors, different definitions are possible that we want to discuss in this section. First, note that there are different kinds of flow factors. The *aggregated flow factor* for the overall schedule is our optimization objective and needs to be defined by combining refined flow factors. Refined flow factors definitions can refer to an overall lot or individual steps of a lot. Weighted flow factors take the priority of the lot into account.

In the following, we introduce some notation that will be needed in the discussion on different options to define flow factors. Note that we neglect the priorities of lots in this context. For each lot, we are given a sequence of processing steps $o_1, \ldots, o_n$. Assume that the first $k-1$ steps have already been processed. So, only the steps steps $o_k, \ldots, o_n$ remain to be processed and need to be scheduled. Let us denote the *theoretical step duration* of a step i as p_i. It is defined by the processing duration of a this step using the fastest qualified tool. Then, the sum $\sum_{i=1}^{n} p_i$ of the *theoretical durations* of all steps of the lot defines the *theoretical route duration*.

As described in section 2.1.4, we are given an *initiation date* for each lot. This corresponds to the earliest possible point in time where we could have started to process step o_k. Let us denote this point in time as step release date r_k. Let us denote the current time as t. So, operation o_k cannot be scheduled before $max(t, r_k)$. Now, corresponding to r_k, we assume that past values r_i with $i < k$ are known. Next, we assume that the completion dates of steps performed in the past are given as $C_1, \ldots, C_{k-1}$. Then, assume that we have computed a schedule that is to be evaluated. The schedule provides completion dates $C_k, \ldots, C_n$ for the remaining steps of the lot. Let us now compare different options for the definition of the flow factor of a lot L. The basic flow factor definition for a single step is $f_i = \frac{C_i - r_i}{p_i}$. This is a common measure in semiconductor manufacturing and the values expected by fab managers differ strongly depending on the considered tool group. It remains to define how the flow factor of a lot can be computed from this in our scheduling context. We could use the sum of step flow factors divided by the number of remaining steps ($n + 1 - k$):

$$f_L^A = \frac{1}{n+1-k} \cdot \sum_{i=k}^{n} \frac{C_i - r_i}{p_i}$$

We could use the flow factor definition specified in section 2.3.2. It uses the sum of the theoretical durations of all steps to be scheduled but ignores steps performed in the past:

$$f_L^S = \frac{C_n - r_k}{\sum_{i=k}^{n} p_i}$$

We could use a flow factor definition that does include steps performed in the past. In contrast to f_L^S, we start at step 1 instead of step k:

$$f_L^P = \frac{C_n - r_1}{\sum_{i=1}^{n} p_i}$$

In order to understand the impact of these definitions, let us consider the examples provided in Figure 2.12. The figure shows different options to schedule the same lot and presents its evaluations. Rectangles correspond to steps and the number in each rectangle gives the theoretical step duration. Assume that the dotted steps already have been scheduled. Assume that the gaps between operations originate from other parts of the schedule that are not shown here. The evaluations of the different flow factor definitions are listed on the right-hand side. Let us in the following point out some observations. We notice that the scheduling of short steps can have a strong influence on f_L^A and f_L^S (see Option 5). f_L^A might consider a later lot completion date to be better (Option 2 vs. Option 7) and therefore it is not a regular criterion. f_L^A cares about the distribution of waiting times (Option 2 vs. Option 3). We believe that the definition f_L^P is the preferable objective function since it promises a more stable behavior in a rolling horizon setting. Preceding decisions have a smaller influence for the evaluation of the following time horizon.

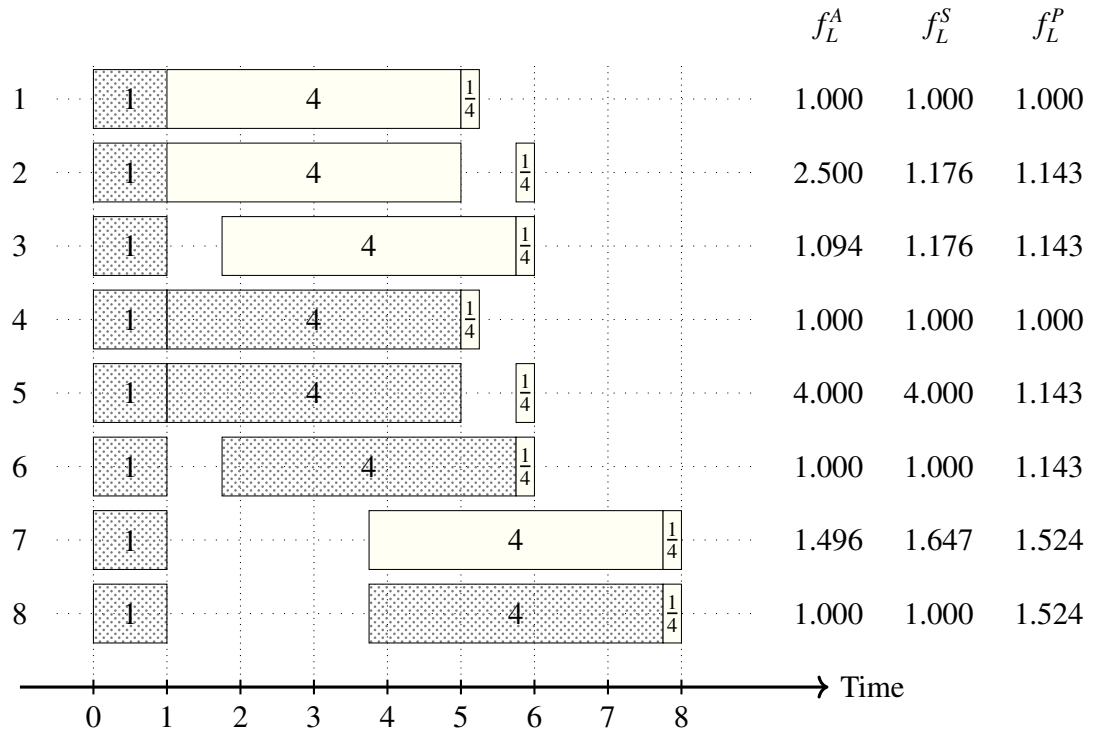


Figure 2.12 – *Comparison of flow factor definitions for eight options to schedule the same lot*

2.4 Conclusion

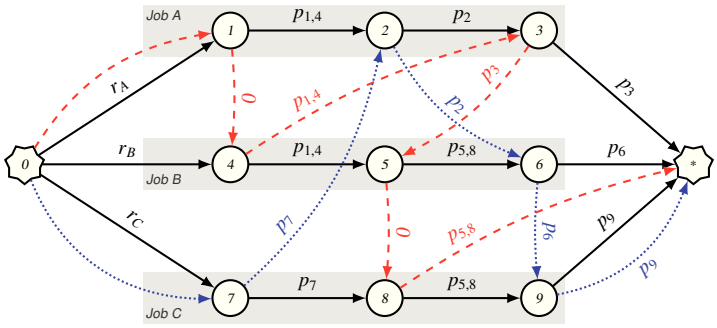
The specification presented in this chapter provides a comprehensive definition of a complex optimization problem. Only few works in the literature focus on this work area while considering substantial portions of the described constraints. Namely, we are only aware of Yurtsever et al. (2009), Yugma et al. (2012), Jung et al. (2013) and Kohn et al. (2013) to consider this specific work area in a comprehensive way. Among these approaches, only Yugma et al. (2012) tackle it as a generalization of the complex job-shop scheduling problem as defined in e.g. Mason et al. (2005). Due to the complexity and extent of the problem, we concentrate in this thesis on the core parts of the given specification. Our goal is to cover the basic properties in a way that facilitates the future integration of further constraints. In this thesis, we have integrated all fundamental properties of the specification and we can compute solutions that are applicable in practice. However, not all the details that were specified are covered in this thesis. Though a further increase in detail bears some potential, we believe that the omitted constraints do not fundamentally change the structure of solutions. Though this is a subjective statement, this point of view is supported by the prioritization of our industrial partner. An outlook towards an inclusion of further properties is given in chapter 6.

A problem as complex as ours needs to be broken down in manageable portions. Thus, each chapter in this thesis focuses on a particular aspect. The thesis is organized in a bottom-up manner, in the sense that each chapter extends the problem of its preceding chapter. The presented approaches are designed to be combinable and the resulting implementation is a single piece of software that comprises all methods presented in this thesis. The structure of this thesis presented in section1.5 overviews the properties that are discussed in the individual chapters.

Chapter 3

Complex Job-Shop Scheduling: A Batch-Oblivious Approach

The integration of batching machines within a job-shop scheduling problem is the core challenge of this thesis. An original batch-oblivious *approach is proposed which integrates the computation of earliest start dates and the creation of batches during the traversal of a conjunctive graph.*



This chapter tackles a fundamental version of the industrial scheduling problem specified in chapter 2 and presents an approach that has been published in Knopp et al. (2015a) and Knopp et al. (2015b). The considered industrial scheduling problem can be seen as a job-shop scheduling problem that features a wide range of additional constraints and properties. Since batching machines constitute the main characteristic of the problem, this chapter concentrates on scheduling batching machines within a job-shop environment. This, together with some additional constraints, leads to a *complex job-shop* scheduling problem that can be described as a flexible job-shop scheduling problem with p-batching, reentrant flows, sequence-dependent setup times and release dates. As noted in section 1.4.1, the *complex job-shop* scheduling problem has been considered in the literature before, in most cases also in the context of semiconductor manufacturing. We optimize a regular mono-criterion objective function. Depending on the context, we consider the weighted flow factor as defined in the industrial specification as well as other objectives from the scheduling literature such as makespan, total weighted tardiness, or maximum lateness. A formal definition of the problem is given in section 3.1.

The considered problem is NP-hard since it generalizes both the NP-hard classical job-shop scheduling problem as well as the NP-hard single-machine scheduling problem with total weighted tardiness objective (see Garey et al. (1976)). Recall that we want to solve industrial instances with about 100 machines and hundreds of jobs, each job consisting of up to ten operations. We develop a heuristic method since we want to solve large instances of an NP-hard problem in reasonable CPU times. The scheduling of parallel batching machines and variants of the job-shop scheduling problem are have already been studied whereas their combination is rarely considered. Most existing solution approaches for complex job-shop scheduling problems with batching machines rely on the disjunctive graph representation of Ovacik and Uzsoy (1997). This representation introduces dedicated nodes to represent batching decisions. We propose a novel batch-oblivious modeling which avoids additional batching nodes. Instead, we encode batching decisions in the weights of edges to reduce the structural complexity of the graph and to facilitate modifications. This modeling is explained in detail in section 3.2.

In order to take advantage of the batch-oblivious representation, we then introduce a novel integrated algorithm to compute start dates and to create batches. This representation allows our integrated algorithm to take batching decisions "on the fly" during graph traversal. This can be used to "fill up" underutilized batches by applying a combined resequencing and reassignment strategy. In addition, an integrated batch-oblivious move is proposed to relocate individual operations. The combination of the algorithm and the batch-oblivious move yields a neighborhood that implicitly comprises specific moves known from the literature such as the swapping of batches. This neighborhood also includes more moves generated by the interplay of the algorithm and the integrated batch-oblivious move. These building blocks for heuristic methods are detailed in section 3.3.

We apply the previously presented building blocks within a GRASP based approach (Feo and Resende (1995)). We randomize the construction of initial solutions by successively

inserting jobs using a randomly perturbed ordering of jobs. Solutions are improved using a Simulated Annealing heuristic. This GRASP based metaheuristic approach is presented in section 3.4. Computational experiments using a parallelized implementation yield very good results for various types of instances and show the generality and applicability of our approach. Numerical results are presented and discussed in section 3.5, where new public industrial benchmark instances are proposed and known benchmark instances from the literature are used. The approach is designed to facilitate extensions towards further constraints, in particular those given in the industrial problem specification. It is extended in the later chapters of this thesis.

3.1 Formal Problem Description

This section provides a formal definition of the considered flexible job-shop scheduling problems with p-batching, reentrant flows, sequence-dependent setup times and release dates (*complex job-shop scheduling problem*). We want to optimize regular objective functions which are formally defined later in this section. Using the $\alpha|\beta|\gamma$ notation of Graham et al. (1977), this class of scheduling problems can be denoted as $FJc|r_j, s_{i,j}, B, recr|reg$. We consider p-batching and its embedding in a job-shop environment to be the main characteristics of the problem.

We are given a set of *jobs* J which have to be processed using a given set of *machines* M. For each job $j \in J$, we are given a set of *operations* $O_j = \{o_{1,j}, o_{2,j}, \ldots, o_{|O_j|,j}\}$, and a *release date* $r_j \in \mathbb{Z}$. The disjoint union $O = O_1 \dot\cup O_2 \ldots \dot\cup O_{|J|}$ denotes all given operations. We are given a set of *batch families* F and a set of *setup families* $\widetilde{F}$. Each batch family $f \in F$ prescribes a machine $m_f \in M$, a *processing duration* $p_f \in \mathbb{N}_0$, a *setup family* $\sigma_f \in \widetilde{F}$, and a batching *capacity* $b_f \in \mathbb{N}_{>0}$. For each operation $o_{i,j} \in O$, a set of eligible batch families $F_{i,j} \subset F$ with $F_{i,j} \neq \emptyset$ is given. A given mapping $s : \widetilde{F} \times \widetilde{F} \to \mathbb{N}_0$ prescribes *sequence-dependent setup times* between operations that are scheduled on the same machine.

A *schedule* is completely characterized by selecting families $f_{i,j} \in F_{i,j}$ and *start dates* $S_{i,j} \in \mathbb{Z}$ for all given operations $o_{i,j} \in O$. To avoid using $f_{i,j}$ as an index, we denote the machines, processing durations, setup families, and batching capacities related to this selection as $m_{i,j}$, $p_{i,j}$, $\sigma_{i,j}$ and $b_{i,j}$, respectively. To describe a schedule that is *feasible*, selected families $f_{i,j}$ and start dates $S_{i,j}$ of operations $o_{i,j}$ have to respect several constraints that are detailed in the following. Preemption is not allowed: Once the processing of an operation has begun, it cannot be interrupted. Thus, the *completion time* of an operation $o_{i,j} \in O_j$ is given by $C_{i,j} = S_{i,j} + p_{i,j}$. Operations belonging to the same job have to be performed in the order given by the *route* of the job. So, $C_{i,j} \leq S_{i+1,j}$ has to be fulfilled for all $o_{i,j} \in O$ with $i < \left|O_j\right|$. The first operation $o_{1,j} \in O_j$ of each job cannot be processed before its release date, so $S_{1,j} \geq r_j$ must hold for all $j \in J$. Operations performed on the same machine must not overlap. Hence, for two operations $o_{i,j}, o_{k,l} \in O$ with $m_{i,j} = m_{k,l}$, either $S_{i,j} = S_{k,l}$ or $S_{i,j} \geq C_{k,l}$ or $C_{i,j} \leq S_{k,l}$ must hold. Only operations of the same family can be processed

at the same time on the same machine. So, for two operations $o_{i,j}, o_{k,l} \in O$ with $f_{i,j} \neq f_{k,l}$ and $m_{i,j} = m_{k,l}$, we require $S_{i,j} \neq S_{k,l}$. A subset $B \subset O$ of operations with $m_{i,j} = m_{k,l}$ and $S_{i,j} = S_{k,l}$ is called a *batch*. Batching capacities limit the number of operations per batch. Thus, we require $\left|\left\{o_{k,l} \in O \;\middle|\; m_{k,l} = m_{i,j} \wedge S_{k,l} = S_{i,j}\right\}\right| \leq b_{i,j}$ for all operations $o_{i,j} \in O$. To respect sequence-dependent setup times, for all operations $o_{i,j}, o_{k,l} \in O$ with $m_{i,j} = m_{k,l}$ and $S_{i,j} \neq S_{k,l}$, either $C_{i,j} + s(\sigma_{i,j}, \sigma_{k,l}) \leq S_{k,l}$ or $C_{k,l} + s(\sigma_{k,l}, \sigma_{i,j}) \leq S_{i,j}$ must hold.

Our goal is to determine schedules that optimize regular objective functions. An *objective function* is a function $f : \mathbb{R}^{|O|} \to \mathbb{R}$ that maps tuples of operation start dates to a real number. We call an objective function *regular* (Brucker (2007)) if, for any pair of tuples of start dates $(S_1, \ldots, S_{|O|}), (S'_1, \ldots, S'_{|O|}) \in \mathbb{R}^{|O|}$ with $S_1 \leq S'_1 \wedge \cdots \wedge S_{|O|} \leq S'_{|O|}$, it follows that $f(S_1, \ldots, S_{|O|}) \leq f(S'_1, \ldots, S'_{|O|})$. Intuitively speaking, the quality of a schedule cannot deteriorate by advancing the start date of some of its operations. Most objective functions considered in the scheduling literature (e.g., makespan, maximum lateness, total weighted completion time, or total weighted tardiness) are regular objective functions.

We have provided a concise formal definition of a complex job-shop scheduling problem which generalizes several scheduling problems defined in the literature: It reduces to a flexible job-shop scheduling problem if all batching capacities are equal to one, and to a scheduling problem with parallel batching machines if the routes of all jobs contain only a single operation. *Tool groups*, i.e. sets of identical tools, can be taken into account by considering each tool as an individual machine. Recurrent flows are comprised in the definition: No constraint forbids to reuse a machine for multiple operations of the same job. Note that some objective functions might depend on due dates $d_j \in \mathbb{Z}$ or weights $w_j \in \mathbb{R}$ which may be associated to each job $j \in J$. These parameters were not explicitly included in the formal definition above since they do not impose any hard constraint on schedules. Yet they can be integrated in the definition of an objective function.

Note that we distinguish two different types of families: One is related to batching and one is related to setups. Setups are needed between batches of operations, which implies that two operations with the same batching family must have the same setup family. However, two operations with the same setup family could have different batching families. An equivalent but more concise formulation of the same complex job-shop scheduling problem that uses only a single type of family is possible when defining setup times appropriately. Our motivation to distinguish batch and setup families is to obtain a uniform notation over the chapters of this thesis. The distinction between family types is meaningful in chapter 4 in order to differentiate setup families when multiple resources per operation are required. There, each resource usage can be assigned to a different setup family.

3.2 Disjunctive Graph Modeling

Disjunctive graphs, introduced by Roy and Sussmann (1964), allow combinatorial properties of schedules to be represented in a concise way and have been applied to solve a broad

range of scheduling problems. To tackle the inclusion of p-batching within job-shop environments, we introduce in this section a *batch-oblivious* disjunctive graph representation which is designed to facilitate decision-making on batches during graph traversals. First, we recall the disjunctive graph model for complex job-shops. Then, two alternative representations for batching decisions are described: In section 3.2.1, we recall and discuss an established representation which inserts dedicated batching nodes into the graph (see Ovacik and Uzsoy (1997)). In section 3.2.2, a novel, batch-oblivious representation is introduced which modifies edge weights instead of introducing auxiliary nodes. This batch-oblivious representation helps us to take batching decisions on the fly (during graph traversal). This idea provides the foundation for the scheduling approach proposed in this chapter.

Disjunctive graphs represent structural properties of schedules but do not encode assignment, sequencing or batching decisions. *Conjunctive graphs* encode all decisions to be taken and are the principal tool for our algorithms. Let us briefly recapitulate those graph types. In both cases, each node represents an operation and each edge represents a dependency induced by either the route of a job or the assignment and sequencing decisions for an operation on a machine. Disjunctive graphs model all possible assignments of operations to machines and sequences of operations on the machines using undirected edges. By replacing undirected by directed edges while satisfying some feasibility constraints, a conjunctive graph is constructed which corresponds to an assignment of operations to machines and an ordering of operations (sequencing) on the machines. Redundant edges are removed in the conjunctive graph. Next, we provide a definition of a basic conjunctive graph representation that still neglects the representation of batching decisions. This basic conjunctive graph representation corresponds to that of Dauzère-Pérès and Paulli (1997) for flexible job-shops.

A *conjunctive graph* $G = (V, E)$ is an acyclic directed graph with nodes $V = O \cup \{0, *\}$ that correspond to the given operations O plus an artificial start node 0 and an artificial end node $*$. For each job and each machine, the graph contains one path from the artificial start node 0 to the artificial end node $*$. The disjoint union of those paths yields all edges of the graph. Each node $v \in O$ is part of exactly two paths: One corresponding to the route of its job and one corresponding to the sequence of the machine it is assigned to. For a node $v \in O$, we denote its *route successor* by $r(v) \in V \setminus \{0\}$ and its *machine successor* by $m(v) \in V \setminus \{0\}$. Analogously, its predecessors are denoted by $r^{-1}(v) \in V \setminus \{*\}$ and $m^{-1}(v) \in V \setminus \{*\}$. The artificial start node 0 has $|J| + |M|$ outgoing edges and no incoming edges. Analogously, the artificial end node $*$ has $|J| + |M|$ incoming edges and no outgoing edges. Overall, the graph contains $|E| = 2|O| + |J| + |M|$ edges.

A conjunctive graph can be used to determine start dates S_v of operations $v \in O$. A weight $l_{u,v} \in \mathbb{N}_0$ is assigned to each edge $(u, v) \in E$ in order to ensure a minimum duration between the beginning of adjacent operations: $S_v \geq S_u + l_{u,v}$ for each edge $(u, v) \in E$. Having this, start dates of operations correspond to distances of longest paths from the artificial start node with respect to those edge weights. We denote the distance of a *longest path* from a node $v \in V$ to a node $w \in V$ by $L(v, w) \in \mathbb{N}_0$. For each operation $v \in O$, its start date is determined by $S_v = L(0, v)$. To reflect the given constraints, we define edge weights as follows.

For edges $(0, o_{1,j}) \in E$ connecting the artificial start node 0 with the initial operation $o_{1,j}$ of a job j, the edge weight is set to the release date r_j of job j. For edges $(0, o_m) \in E$ connecting the artificial start node 0 with the initial operation o_m scheduled on machine $m \in M$, the edge weight is set to zero. For route edges $(v, r(v)) \in E$ with $v \neq 0$, the edge weight is set to the processing duration p_v of operation v. For machine edges $(v, m(v)) \in E$ with $v \neq 0$ of non-batching machines, the edge weight is set to the sum $p_v + s(\sigma_v, \sigma_{m(v)})$ of the processing duration of operation v and the sequence-dependent setup time between operation v and operation $m(v)$.

Now, what remains is to provide a representation for batching machines. They can be either modeled by modifying the structure of the graph (*batch-aware*) or by adapting the weights of edges (*batch-oblivious*). The following two subsections present both alternatives. Recall that each batch has to respect the capacity of the machine as well as the equality of involved families. The adherence to those constraints has to be guaranteed for each schedule. The related checks are not detailed in this section in order to focus on the essential parts of both representations.

3.2.1 Batch-Aware Conjunctive Graphs

This section reviews a batch-aware conjunctive graph representation that was introduced by Ovacik and Uzsoy (1997). All solution approaches for complex job-shop scheduling problems that we are aware of make use of this type of representation (e.g., Mason et al. (2005), Mönch et al. (2003), or Yugma et al. (2012)).

Recall that a *batch* denotes a set of operations $B \subset O$ that is processed simultaneously on the same machine. In the batch-aware representation, for each batch, an additional node b is added to the graph. The start date of this batching node is taken as the common start date for all operations contained in the batch. A batch requires all of its operations to be ready before it can begin processing. To reflect this, each operation node $v \in B$ is connected to the batching node b via an edge (v, b) of weight zero. Then, operations following in the routes of involved jobs are connected as follows: For each operation $v \in B$ of the batch, an edge $(b, r(v))$ from the batching node to the route successor $r(v)$ of v is introduced. Those edges are given the processing duration of p_v as their weight. Two additional edges $(m^{-1}(b), b)$ and $(b, m(b))$ are introduced to order the batch in the sequence of operations on machine m_b. Analogously to the non-batching case, the weight of each machine edge (u, w) is defined by the sum $p_u + s(\sigma_u, \sigma_w)$. Each operation node $v \in B$ has exactly one incoming edge and one outgoing edge. The batching node b has $|B| + 1$ incoming edges and $|B| + 1$ outgoing edges.

Batch-aware conjunctive graphs represent dependencies stemming from batching decisions in a structural way. The number of nodes in those graphs depends on the number of batches. This structure renders modifications of batching decisions complicated to handle: The number of nodes in the graph must be adapted and several edges have to be manipulated while the acyclicity of the graph must be preserved.

3.2.2 Batch-Oblivious Conjunctive Graphs

In the following, we introduce a novel representation for batching decisions in conjunctive graphs which is non-intrusive regarding the structure of the graph. No dedicated batching nodes are introduced and the basic representation presented at the beginning of this section can remain as is. Our only means to represent batching decisions is to adapt the weights of machine edges $(v, m(v)) \in E$. The weight of a machine edge is set to zero if its adjacent operations should be processed in the same batch. Otherwise, the edge weight is set to $p_v + s(\sigma_v, \sigma_{m(v)})$, as in the non-batching case. Unfortunately, it is not that simple: $l_{v,m(v)} = 0$ only guarantees that $S_v \leq S_{m(v)}$ but not that $S_v = S_{m(v)}$. Since the start dates of all operations in a batch must be equal, setting edge weights to zero can lead to infeasible solutions. In the following, we develop a simple criterion that decides on the feasibility of zero weighted machine edges.

First, let us reconsider a general property of longest paths in directed acyclic graphs. The start date of a node $v \in V$ directly depends on the start dates of its predecessors as follows:

$$S_v = \max_{(u,v)\in E} (S_u + l_{u,v}). \tag{3.1}$$

Now, consider two operations $v \in O$ and $m(v) \in O$ that might be scheduled in the same batch. The node $m(v) \in V$ has two incoming edges coming from its machine predecessor v and its route predecessor $w = r^{-1}(m(v))$. We can apply equation (3.1) to obtain

$$S_{m(v)} = \max(S_v + l_{v,m(v)},\ S_w + l_{w,m(v)}). \tag{3.2}$$

If the length $l_{v,m(v)}$ of the machine edge $(v, m(v)) \in E$ is set to zero, we want to obtain $S_v = S_{m(v)}$ from a longest path computation. So, let us assume that $l_{v,m(v)} = 0$ and $S_v = S_{m(v)}$. With equation (3.2), we obtain

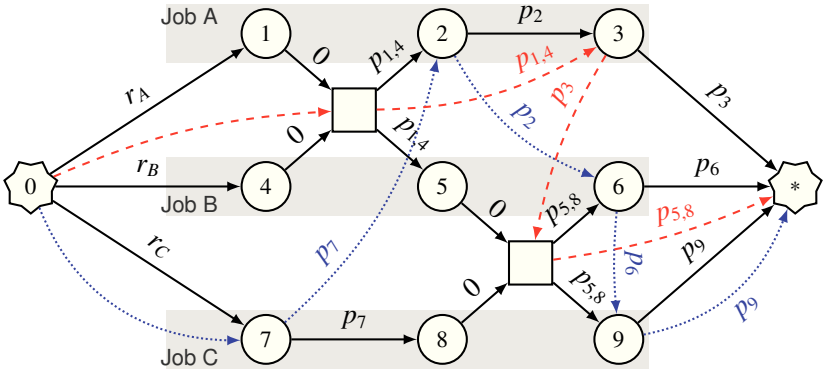
$$S_v = \max(S_v,\ S_w + l_{w,m(v)}) \iff S_v \geq S_w + l_{w,m(v)}. \tag{3.3}$$

This means (3.3) is a necessary condition to combine v and $m(v)$ in the same batch. Thus, in batch-oblivious disjunctive graphs, we require the *invariant*

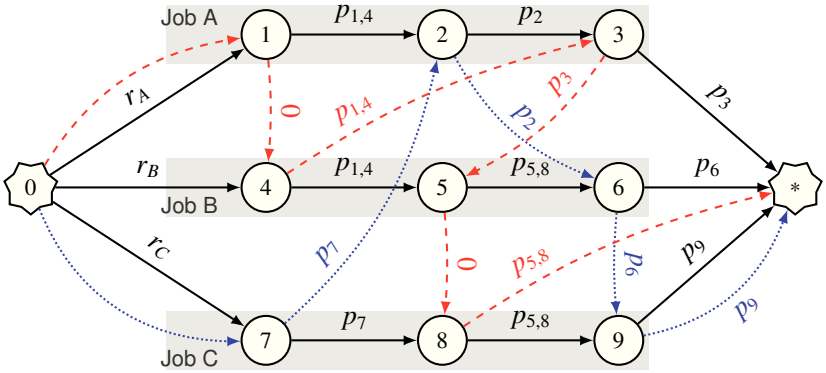
$$(l_{v,m(v)} = 0 \ \wedge\ S_v \geq S_w + l_{w,m(v)}) \ \vee\ (l_{v,m(v)} = p_v + s(\sigma_v, \sigma_{m(v)})) \tag{3.4}$$

to be fulfilled for all nodes $v \in O$ and $w \in V$ with $w = r^{-1}(m(v))$. It follows that, for each operation $v \in V$, a longest path computation schedules the machine successor operation $m(v)$ either at the same time as v or at a later point in time where processing durations and sequence-dependent setup times are satisfied. This property propagates in a natural way: Multiple operations belonging to the same batch are connected in a path of zero weighted machine edges. Next, we want to show that each optimal schedule can be represented using our batch-oblivious conjunctive graph representation.

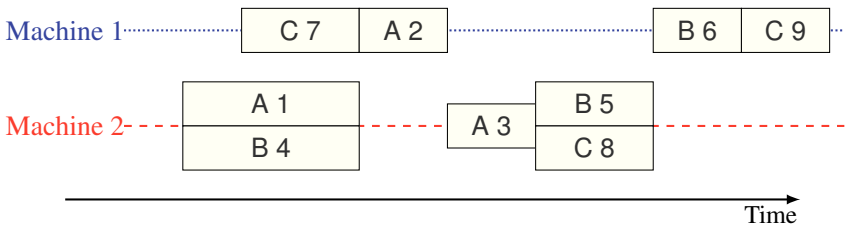
Theorem 3.1. *For any given regular criterion, there exists a batch-oblivious conjunctive graph G with edge weights $l : V \to \mathbb{N}_0$ such that longest paths in this graph represent an optimal schedule.*



(a) *Batch-Aware Conjunctive Graph*



(b) *Batch-Oblivious Conjunctive Graph*



(c) *Gantt Chart*

Figure 3.1 – *A comparison of alternative representations of the same schedule*

Proof. Consider a feasible schedule that is optimal with respect to the given regular criterion. We denote the operation start dates of this optimal schedule by S_v. Now, we construct a batch-oblivious conjunctive graph that defines the assignment and ordering of operations on the machines as follows:

a) The graph respects all machine assignment decisions of the optimal schedule.
b) Ordering decisions on the machines respect the start dates of the optimal schedule: If $S_v > S_w$ for $v \in V$ and $w \in V$, then v is ordered before w.
c) Nodes $v \in V$ and $w \in V$ that are part of the same batch (i.e. $S_v = S_w$) are ordered as follows: If $S_{r^{-1}(v)} + l_{r^{-1}(v),v} < S_{r^{-1}(w)} + l_{r^{-1}(w),w}$, then v is ordered before w.
d) For two nodes $v \in V$ and $w \in V$ of the same batch with $S_{r^{-1}(v)} + l_{r^{-1}(v),v} = S_{r^{-1}(w)} + l_{r^{-1}(w),w}$, their relative order can be arbitrarily decided as long as no cycle is introduced.

Since those rules are derived from a feasible schedule, this graph is constructed without any cycle. Edge weights are set according to the batching decisions in the given optimal schedule. Property c) guarantees that, for all adjacent nodes $v \in V$ and $m(v) \in V$ of the same batch, $S_{r^{-1}(m(v))} + l_{r^{-1}(m(v)),m(v)} \leq S_v$ holds. Thus, invariant (3.4) holds for all edges of the graph. □

Figure 3.1 shows an example that allows to compare batch-aware and batch-oblivious representations. It shows a schedule with three jobs A, B and C using two machines. We see two batches processed on machine 2, each consisting of two operations: One is composed of operation 1 plus operation 4, another one is composed of operation 5 plus operation 8. For brevity of notation, sequence-dependent setup times have been omitted and let us denote $p_{1,4} = p_1 = p_4$ and $p_{5,8} = p_5 = p_8$. Note that invariant (3.4) is not visualized in Figure 3.1 (b), so assume that $S_1 \geq r_B$ and $S_5 \geq S_7 + p_7$.

3.3 Building Blocks for Integrated Batching Decisions

This section develops the foundation of our heuristic approach. First, we describe in section 3.3.1 how start dates can be computed from a given batch-oblivious conjunctive graph. The graph will remain structurally unchanged in this first version. Then, we define a general move in section 3.3.2. It moves individual operations and is designed to be complemented by the interleaved start date computation and graph modification that we introduce in section 3.3.3. This interleaved computation advances suitable nodes to "fill up" incomplete batches. Overall, our method integrates adaptive batching decisions with one general move to resequence and reassign operations. It can be used as a building block for metaheuristic approaches.

3.3.1 Static Start Date Computation

Let us first describe how start dates of operations can be computed from a given batch-oblivious conjunctive graph. For this, batching decisions are taken dynamically ("on the fly") during a traversal of the graph by deciding the weights of edges. Thereby, it is im-

portant to preserve invariant (3.4) introduced in section 3.2. In contrast to the adaptive start date computation presented in section 3.3.3, this static algorithm does not modify the graph itself: It preserves all ordering and assignment decisions inherent in the given conjunctive graph. The ordering is relaxed in the sense that a directed edge $(u, v) \in E$ requires only that operation v must not be processed before operation u. So, u and v might start at the same time which means they are part of the same batch.

The computation is based on topological orderings. In an acyclic directed graph, a *topological ordering* is a linear ordering of the nodes of the graph. It can be defined as a relation $\prec \subseteq V \times V$ with $v \prec w \Rightarrow \nexists$ a path from w to v. Thus, traversing a conjunctive graph in topological order guarantees that, for each node $v \in O$, both predecessors $r^{-1}(v)$ and $m^{-1}(v)$ are visited before v. Hence, the inductive formula for S_v given in equation (3.1) can be applied.

Algorithm 3.1 A static batching algorithm for a given conjunctive graph G

computeStartDatesStatically (G)
$S_0 \leftarrow 0$
$\beta_v \leftarrow 1 \quad (\forall\, v \in V)$
for $v \in computeTopologicalOrdering\,(G \setminus \{0\})$
if $(S_{r^{-1}(v)} + p_{r^{-1}(v)} \leq S_{m^{-1}(v)}$ **and** $f_{m^{-1}(v)} = f_v$ **and** $\beta_{m^{-1}(v)} < b_v)$
$S_v \leftarrow S_{m^{-1}(v)}, \quad \beta_v \leftarrow \beta_{m^{-1}(v)} + 1$
else
$S_v \leftarrow \max\left(S_{r^{-1}(v)} + p_{r^{-1}(v)},\ S_{m^{-1}(v)} + p_{m^{-1}(v)} + s(\sigma_{m^{-1}(v)}, \sigma_v)\right)$

Algorithm 3.1 provides the pseudo-code for a static graph evaluation algorithm. It tracks the used capacity β for each node and checks if the families $f_v \in F$ and $f_{m^{-1}(v)} \in F$ of consecutive operations are equal. The algorithm greedily creates batches while preserving the invariant for batch-oblivious conjunctive graphs. This corresponds to a longest path computation with dynamically specified edge weights. The computation takes $O(|E|)$ time since a topological ordering in a conjunctive graph can be computed in $O(|E|)$ and each node is visited exactly once. Batching decisions are taken greedily and strongly depend on the structure of the given graph.

3.3.2 An Integrated Batch-Oblivious Move

In order to develop heuristic algorithms, we want to introduce moves which modify a given batch-oblivious conjunctive graph. Known heuristic approaches we are aware of employ specific knowledge about previous batching decisions. E.g. they explicitly displace, combine or split entire batches, or they exchange operations belonging to different batches (Bilyk et al. (2014); Yugma et al. (2012)). To keep it simpler, we follow a different strategy and maintain the batch-obliviousness of our graph also for our moves. An observation from section 3.2 is that, except of its edge weights, our conjunctive graph representation does

not differ from a conjunctive graph representation for flexible job-shop scheduling problems without batching. This allows us to apply the move introduced in Dauzère-Pérès and Paulli (1997) which integrates the resequencing and reassignment of operations. We include a detailed description of this move in this section not only for completeness, but also to adapt it to our notation and to show that it remains valid for redefined edge weights.

Assume that all batching decisions have been taken and thus edge weights and start dates have been fixed. An operation v is moved after another operation w as follows: First, the machine related conjunctive edges $(m^{-1}(v), v) \in E$ and $(v, m(v)) \in E$ of operation v are replaced by an edge $(m^{-1}(v), m(v))$. Then, operation v is reinserted after operation w by replacing the edge $(w, m(w)) \in E$ with two edges (w, v) and $(v, m(w))$. In the graph that is created by executing the move, we then have $m(w) = v$. Recall that m_w must be a machine for which v is qualified to be processed on. To be computed efficiently, the feasibility check of a move relies on start dates of operations as shown in the following. Let us denote by $l_v = \min(l_{v,m(v)}, l_{v,r(v)})$ the minimum weight of the outgoing edges of a node $v \in O$.

Theorem 3.2. *(Dauzère-Pérès and Paulli (1997)) An operation $v \in O$ can be moved between two operations w and $m(w)$ with $w \neq r(v)$ and $m(w) \neq r^{-1}(v)$ if $S_{r(v)} + l_{r(v)} > S_w$ and $S_{m(w)} + l_{m(w)} > S_{r^{-1}(v)}$.*

Proof. (Dauzère-Pérès and Paulli (1997)) When the node v is removed, the edges $(m^{-1}(v), v)$ and $(v, m(v))$ are replaced by an edge $(m^{-1}(v), m(v))$ which cannot introduce a cycle. When v is reinserted after w, there are only two possible ways to create a cycle:

a) There was a path from operation $r(v)$ to operation w before moving v. This implies $S_{r(v)} + l_{r(v)} \leq S_w$, which contradicts the first assumption.

b) There was a path from operation $m(w)$ to operation $r^{-1}(v)$ before moving v. This implies $S_{m(w)} + l_{m(w)} \leq S_{r^{-1}(v)}$, which contradicts the second assumption.

□

Note that the original theorem has been adapted to include redefined edge weights. This move has been successfully applied to solve flexible job-shop scheduling problems without batching and is not restricted to a particular objective function. However, in our case which includes batching, those moves might tear apart batches. This might lead to poor solutions containing unfavorable batches of only a single operation. Thus, to escape from a local optimum, a sequence of moves might need to strongly deteriorate a given solution before it can improve it again. The following subsection shows how we tackle this problem.

3.3.3 Adaptive Start Date Computation

To improve batching decisions, we interleave the computation of start dates with modifications of the batch-oblivious conjunctive graph. In particular, we want to improve schedules

created by the moves described in section 3.3.2 by "filling up" batches with remaining machine capacity: We advance suitable nodes by removing and reinserting them in the graph. In algorithm 3.1 of section 3.3.1, a topological ordering is computed first and then all nodes are traversed in this order. This is not viable anymore if we modify the graph while traversing it. Thus, we propose to interleave the computation of a topological ordering with a dynamic modification of the graph. We will see in the following that this idea can be described in terms of unidirectional cuts. During the course of our algorithm, unidirectional cuts distinguish unsettled nodes that still can be modified from settled nodes that are fixed. Finally, we propose in section 3.3.4 different strategies for such adaptive graph modifications.

Definitions and Notation For a given batch-oblivious conjunctive graph $G = (V, E)$, we consider a *cut* $V_s \subseteq V$ that partitions the graph into a subset of *settled* nodes V_s and a subset of *unsettled* nodes $V_u = V \setminus V_s$. A cut V_s is called *unidirectional* if there are no edges from an unsettled node to a settled node, i.e. $E \cap (V_u \times V_s) = \emptyset$. Let us denote by $G_s = (V_s, E_s)$ and $G_u = (V_u, E_u)$ the resulting subgraphs. The edges of each graph $G_\square \in \{G_s, G_u, G\}$ are given by $E_\square = E \cap (V_\square \times V_\square)$. Let us denote for a node $v \in V_\square$ its indegree in $G_\square$ by $deg^-_\square(v)$ and its outdegree in $G_\square$ by $deg^+_\square(v)$. A node $v \in V_\square$ without incoming edges (i.e. $deg^-_\square(v) = 0$) is called a *root node* of $G_\square$. A node $v \in V_\square$ without outgoing edges (i.e. $deg^+_\square(v) = 0$) is called a *leaf node* of $G_\square$.

Proposition 3.1. *In a conjunctive graph $G = (V, E)$, $V_s = \{0\}$ is a unidirectional cut.*

Proof. Since the only settled node $0 \in V_s$ is a root in G, no edges can end in a settled node. □

To *settle* a node $v \in V_u$ with $r^{-1}(v) \in V_s$ after a leaf node $w \in V_s$ of G_s, v is removed from G_u and appended to G_s. If $m^{-1}(v) = w$, the operation remains assigned to machine m_v sequenced after the same machine predecessor w. In this case, no edges need to be modified. Otherwise, if $m^{-1}(v) \neq w$, we modify the graph G as follows: The machine related conjunctive edges $(m^{-1}(v), v) \in E$ and $(v, m(v)) \in E$ of operation v are replaced by an edge $(m^{-1}(v), m(v))$ and the edge $(w, m(w)) \in E$ is replaced by two edges (w, v) and $(v, m(w))$. Settling a node does not change any route edges. If $m^{-1}(v) \neq w$ and $m_v = m_w$, then v is *resequenced*. If $m^{-1}(v) \neq w$ and $m_v \neq m_w$, then v is *reassigned*. Note that we require for a node $v \in V_u$ to be reassigned after a node $w \in V_s$ that $\exists\, q \in R_v$ with $m_q = m_w$.

Theorem 3.3. *Let $G = (V, E)$ be a conjunctive graph and let V_s be a unidirectional cut in G. When a node $v \in V_u$ with $r^{-1}(v) \in V_s$ is settled after a leaf node $w \in V_s$ of G_s, the modified graph $G' = (E', V')$ does not contain any cycle and $V'_s = V_s \cup \{v\}$ is a unidirectional cut in G'.*

Proof. When $v \in G_u$ is settled, three edges from $E \setminus E_s$ are deleted and the edges $(m^{-1}(v), m(v))$, (w, v) and $(v, m(w))$ are inserted. Edge deletions can neither introduce a cycle, nor invalidate any unidirectional cut. Since V_s is a unidirectional cut in G and $v \in V_u$, it follows that $m(v) \in V_u$ and $r(v) \in V_u$. With $r(v) \in V'_u$ and $m(w) \in V'_u$, we conclude that v is

a leaf in G'_s. Since the predecessors of v in G' are settled, i.e. $r^{-1}(v) \in V'_s$ and $w \in V'_s$, edges adjacent to v cannot invalidate the unidirectional cut V'_s. The only inserted edge that is not adjacent to v in G', $(m^{-1}(v), m(v))$, does not invalidate the unidirectional cut since $m(v) \in V_u$. Thus, V'_s is a unidirectional cut in G'.

It remains to show that no cycle is introduced in G'. Since $v \in V'_s$, the edge $(m^{-1}(v), m(v))$ is the only inserted edge that might be contained in the subgraph G'_u. It cannot introduce a cycle since it replaced the edges $(m^{-1}(v),\ v)$ and $(v,\ m(v))$. Thus, the subgraph G'_u is acyclic. Both edges $(r^{-1}(v),\ v) \in E'$ and $(w,\ v) \in E'$ that are added to G'_s end in the node v. Since v is a leaf in G'_s, this cannot introduce a cycle in G'_s. Thus, the subgraph G'_s is acyclic. Overall, since G'_u and G'_s are acyclic, a cycle in G' must include an edge from V'_u to V'_s. Such an edge cannot exist since V'_s is a unidirectional cut in G'. □

Algorithm 3.2 An adaptive batching algorithm for a given conjunctive graph G

computeStartDatesAdaptively (G)
$S_0 \leftarrow 0$
$V_s = \{0\}$
$\beta_v \leftarrow 1 \quad (\forall\ v \in V)$
while $V_s \neq V$
  $v,\ w \leftarrow$ select $(v \in V \setminus V_s,\ w \in V_s)$
  assert $(r^{-1}(v) \in V_s$ **and** $deg^+_s(w) = 0)$
  settle v after w
  if $(S_{r^{-1}(v)} + p_{r^{-1}(v)} \leq S_{m^{-1}(v)}$ **and** $f_{m^{-1}(v)} = f_v$ **and** $\beta_{m^{-1}(v)} < b_v)$
    $S_v \leftarrow S_{m^{-1}(v)}, \quad \beta_v \leftarrow \beta_{m^{-1}(v)} + 1$
  else
    $S_v \leftarrow \max\left(S_{r^{-1}(v)} + p_{r^{-1}(v)},\ S_{m^{-1}(v)} + p_{m^{-1}(v)} + s(\sigma_{m^{-1}(v)}, \sigma_v)\right)$
  $V_s \leftarrow V_s \cup \{v\}$

Algorithm 3.2 shows how the results on unidirectional cuts can be applied to interleave the computation of start dates with modifications of the graph. Initially, only the artificial start node 0 is considered to be settled. Then, nodes that meet the criteria of Theorem 3.3 can be successively settled without introducing any cycle. The start dates of settled nodes are calculated as proposed in Algorithm 3.1. The quality of the resulting schedule and the efficiency of the algorithm strongly depends on the selection of the nodes v and w. In the following, we propose and analyze three selection strategies.

3.3.4 Strategies for Selecting Nodes

A Static Selection Strategy A straightforward selection strategy chooses in each step a root node $v \in V_u$ of G_u and settles it after its machine predecessor $w = m^{-1}(v)$. This strategy does not modify the graph and iterates the nodes in topological order. Algorithm 3.2 with this static selection strategy is equivalent to Algorithm 3.1 presented in section 3.3.1. In order

to implement this strategy, we need to determine root nodes of G_u efficiently. This has been done by applying the approach of Kahn (1962). It maintains the indegree in G_u for each node of the graph G and a list containing all root nodes in G_u. When a node is settled, it is removed from the list of root nodes in G_u and, for each of its successor nodes, the number of incoming edges in G_u is decreased. These successor nodes $v \in V_u$ are added to the list of root nodes when their indegree in G_u becomes zero. Since these auxiliary data structures can be updated in constant time, the runtime of the algorithm is linear in the number of edges of G.

A Resequencing Selection Strategy The idea of this strategy is to "fill up" batches that underutilize the available batching capacity. This is done by advancing suitable operations on their assigned machines and can be implemented as follows: As in the static strategy, we first determine a root node $v \in V_u$ in G_u. If it can be included in the same batch as its machine predecessor $w = m^{-1}(v)$ or if no batching capacity is remaining for w, v is settled after w as in the static selection strategy. Otherwise, we iterate through the machine successors of v until we find an operation $u \in V_u$ with $r^{-1}(u) \in V_s$ and $q_u = q_w$ for which invariant (3.4) is fulfilled. If such an operation is found, u is settled after w, and is combined in a batch with operation w by Algorithm 3.2. If no such operation exists, we fall back to settling $v \in V_u$ after w. Again, the auxiliary data structures can be updated in constant time.

A Reassigning Selection Strategy We can enhance the resequencing selection strategy by extending the search for suitable "batch-filling" operations to other machines. If no resequenceable operation is found, we continue to search on other machines for suitable operations to be reassigned: We search in turn, starting from root nodes $y \in V_u$ in G_u with $m_y \neq m_v$. Again, we successively search machine successors of y until an operation $u \in V_u$ is found such that $r^{-1}(u) \in V_s$ and $\exists\, q \in R_u : q = q_w$ and which fulfills invariant (3.4). If such an operation is found, it is settled after w. Otherwise, we fall back to settling $v \in V_u$ after w.

Analysis We proposed three selection strategies which differ in their effort to "fill up" underutilized batches. These strategies offer a solid baseline to evaluate our algorithmic framework. However, finding improved strategies might be an interesting challenge for future research. In the worst case, both the resequencing and the reassignment strategies explore $O(|V|)$ operations to select a node. This increases the runtime bound of Algorithm 3.2 to $O(|E| \cdot |V|)$. However, in the average case, as observed in the numerical experiments of section 3.5, a much better behavior is obtained since only few batches are underutilized and only those will trigger a search.

An interesting property of our method is that it includes various classical moves. Consider as an example the swapping of adjacent batches of different families. First, an integrated move could displace a single operation of the second batch before the first batch. Then, the resequencing selection strategy fills up that newly created batch with all operations that had been part of the second batch. In the end, both batches are swapped. Note that this is only a simple example of possible interactions. We observe much more complex rearrangements in practice.

3.4 Heuristic Approaches

In this section, we apply the building blocks developed in Section 3.3 within different heuristics. Since our batch-oblivious methodology is not bound to one specific solution approach, we deploy it within classical heuristic frameworks in order to evaluate its performance. In the following, we describe a construction heuristic, a local search method, a Simulated Annealing metaheuristic and a Greedy Randomized Adaptive Search Procedure (GRASP) based metaheuristic.

First, we define a *construction heuristic* which adapts the methods presented in Yugma et al. (2012) and Knopp et al. (2014). If due dates and weights are given, jobs are initially sorted in decreasing order of their ratio $\frac{w_j}{d_j}$ (weight divided by due date). Otherwise, jobs are initially sorted in decreasing order of the sum of the shortest processing durations of their operations. The heuristic then iterates over the sorted list of jobs and successively inserts all operations of the current job. The operations of a job are greedily inserted, starting from the first operation, by selecting the best insertion position for each operation. The best insertion position is determined by the objective function value of the partial solution obtained by actually inserting the considered operation. The construction is completed when all operations of all jobs have been inserted.

In both local search and Simulated Annealing, we combine the batch-oblivious move from Section 3.3.2 with the adaptive start date computation from Section 3.3.3 as follows. After a batch-oblivious move is performed, an adaptive start date computation follows in order to determine start dates and batching decisions. The combined result of both modifications is considered as one single move. If such a move is rejected, all involved changes are collectively reverted. The *local search* starts with the solution found by the construction heuristic, and explores the neighborhood using steepest descent. All moves reachable from the current solution are evaluated and the one leading to the best solution is selected. The local search continues until no strictly better solution is found.

Our *Simulated Annealing* metaheuristic is based on the same integrated move and also starts with the solution found by the construction heuristic. In each step, one node is randomly chosen to be moved, its feasible insertion positions are computed, and one of them is randomly selected and performed. We use a geometric cooling schedule that maintains a temperature T which is multiplied by a cooling factor $P_c < 1$ after each iteration. At iteration n, the move is immediately accepted if the current value of the objective function f_n improves the previous objective function value f_{n-1}. Otherwise, the new solution is accepted with a probability of $exp(\frac{-\Delta}{T})$, where $\Delta = f_n - f_{n-1}$. If the new solution is not accepted, all changes related to the move are reversed. The search is stopped if the best solution does not improve during a specified number of iterations P_m. The initial temperature is determined by sampling a fixed number P_s of random moves. For each random move r, we evaluate its influence $\Delta = f_r - f_i$ on the objective function value f_i of the initial solution. Then, for a tuning parameter P_p, the P_p-th percentile of these values is selected as initial value for the temperature T.

In order to further diversify the search and to make use of the ever increasing parallelism of modern CPUs, we developed a heuristic approach based on the idea of *Greedy Randomized Adaptive Search Procedures* (GRASP) of Feo and Resende (1995). Our heuristic creates many different starting solutions by randomizing the construction heuristic. This is done by perturbing the sorted list of jobs used in the construction heuristic as follows. A tuning parameter $P_i \geq 1$ is introduced that steers the perturbation intensity. At each iteration of the construction heuristic, the next job to be inserted is determined by randomly selecting one of the first P_i elements in the sorted list of remaining jobs. The operations of the job are then greedily inserted as described earlier and the job is then removed from the list. Each solution is independently improved using the Simulated Annealing metaheuristic. The GRASP based approach is parallelized as follows. Each solution is constructed and improved independently and thus can be run in its own thread. Communication between threads is only needed to update the best overall solution once a thread has completed its computation. A fixed number of threads is used, and each thread restarts with a new initial solution once its Simulated Annealing metaheuristic has met the stopping criterion.

3.5 Numerical Results

The algorithms presented in this chapter were implemented in C++14 and compiled using the GCC MinGW-W64 compiler in version 4.9.1. All numerical experiments are conducted on an Intel Xeon E5-2620 2.1 GHz machine (6 cores) running Microsoft Windows 7. Extensive numerical experiments on different types of industrial and academic instances were performed. The generality of our approach allows to assess its performance using instances of different scheduling problems. Section 3.5.1 evaluates our algorithms on new complex job-shop benchmark instances that stem from a real-world semiconductor manufacturing facility. Section 3.5.2 compares our methods with results for the instances for parallel batch machines of Mönch et al. (2005). Section 3.5.3 compares our methods with results for the complex job-shop instances of Mason et al. (2005). Section 3.5.5 provides results for the flexible job-shop benchmark instances of Hurink et al. (1994). The sampling strategy of our simulated annealing implementation avoids the need to adapt parameters for individual instances. For all numerical experiments, we used the following identical parameter settings: A cooling factor of $P_c = 0.99999$, a number of samples $P_s = 100$, a maximum number of iterations $P_m = 100\,000$, a temperature percentile of $P_t = 5\,\%$, and a perturbation intensity of $P_i = 5$. All heuristics are run only once, and 6 parallel threads are used in all runs of the GRASP based approach.

3.5.1 Instances from the Diffusion and Cleaning Area

This sections presents results for new benchmark instances from the diffusion and cleaning area of a semiconductor manufacturing facility. We provide two types of instances. First, 15 industrial instances were provided by STMicroelectronics and modified to anonymize

confidential data. Second, 15 random instances that are close to the industrial instances were generated. The random instances include due dates which are not present in the industrial instances. All instances are published under github.com/sebastian-knopp/cjs-instances and its details are described in the following.

Industrial Instances We perform experiments on 15 industrial instances that were extracted from the Manufacturing Execution System (MES) of a semiconductor manufacturing facility over a period of one year. These instances represent various situations that actually appeared in production. Smaller instances with around 25 machines represent a subset of the actual area while larger instances with around 100 machines correspond to the full area. The number of jobs per instance is between 119 and 346. For each job, between one and seven operations have to be performed. Only some of the machines are capable to process multiple operations in the same batch. Sequence-dependent setup times are required only for some of the non-batching machines. Since no due dates are provided, the total weighted completion time is minimized.

Random Instances We perform experiments on 15 random instances that are close to the industrial instances for the diffusion and cleaning area. The instance generation method described below and its parameters are chosen to serve this purpose. As in the industrial instances, machines are partitioned into batching machines and non-batching machines. We assume that all batching machines have the same capacity. A family is processable either on a random subset of the batching machines or on a random subset of the regular machines. We generated 15 instances using all possible combinations for the numbers of jobs $|J| \in \{20, 40, 60, 100, 200\}$ and batching capacities $b \in \{2, 4, 6\}$. We consider $\frac{|J|}{20}$ batching machines, $\frac{|J|}{10}$ non-batching machines, $\frac{|J|}{10}$ batching families, and $\frac{|J|}{5}$ non-batching families. We denote a discrete uniform distribution over $[a, b]$ by $DU[a, b]$. For each job $j \in J$, a random number $\left|O_j\right| \sim DU[1, 7]$ of operations is chosen. Each operation is randomly assigned to a family. Sequence-dependent setup times $\sim DU[1, 10]$ are generated between all non-batching families. We use $w_j \sim DU[1, 10]$, $r_j \sim DU[0,\ 2 \cdot |J|]$, $d_j \sim r_j + DU[p_j,\ \frac{3}{2} p_j]$ for job weights, release dates and due dates, respectively (p_j denotes the minimum sum of the processing durations of all operations of the job). The number of recipes per family is selected according to $DU[1, 5]$. For the processing time of operations $o_{i,j} \in O$ we use $p_{i,j} \sim b \cdot DU[10, 20]$ with $b = 1$ for non-batching machines. The total weighted tardiness is minimized.

We performed numerical experiments for the described industrial (I) and random (R) instances allowing a maximum computation time of 5 minutes per instance. Table 3.2 provides the obtained objective function values for the Simulated Annealing and GRASP based approaches. Table 3.5 provides results in terms of the relative deviation from the best objective function value that has been found. We provide average ($\overline{I}$, $\overline{R}$) and median ($\widetilde{I}$, $\widetilde{R}$) values of these relative deviations over all instances. The column initial refers to the solution that is computed using the non-randomized version of the construction heuristic. We clearly see that the GRASP based approach outperforms all other approaches. The static selection strategy is outperformed by the resequencing and the reassigning strategies with a slight advantage for

				Simulated Annealing			GRASP		
	I	$\|J\|$	$\|M\|$	*static*	*reseq*	*reass*	*static*	*reseq*	*reass*
Industrial (total weighted completion time)	01	119	24	92973	93208	93189	92899	92803	**92532**
	02	148	22	250240	249400	243372	245939	242890	**239892**
	03	195	25	217863	203678	203485	208286	202801	**201790**
	04	209	24	271548	265801	268121	267669	**260286**	260931
	05	186	88	171229	169174	170373	167902	163345	**162884**
	06	268	26	341448	333429	333323	336012	331276	**329918**
	07	210	94	150271	150984	158954	**149680**	150115	150332
	08	310	17	465701	461574	460594	454252	451612	**447305**
	09	231	95	167754	167973	168011	167271	166798	**166643**
	10	245	94	202722	199565	204280	198112	197369	**195789**
	11	302	24	561202	562295	561767	554883	555655	**554670**
	12	302	24	350461	349444	371985	345590	**344109**	345673
	13	324	94	349147	337979	340014	346464	**334409**	334416
	14	315	101	475725	470249	505656	469239	**450909**	465354
	15	346	94	777426	726829	749775	736514	**698666**	702559
Random (total weighted tardiness)	01	20	3	10618	10613	10613	10598	**10011**	**10011**
	02	20	3	6030	6098	6354	5939	**5883**	**5883**
	03	20	3	7063	7074	7074	7036	**7006**	**7006**
	04	40	6	9201	9302	9256	**8801**	9083	9015
	05	40	6	**14208**	14246	14842	14573	14904	14674
	06	40	6	37152	33318	32629	32406	**32048**	32321
	07	60	9	17832	17522	17427	16126	**15165**	15668
	08	60	9	41609	**40514**	41537	42560	41094	42508
	09	60	9	35888	37155	37068	31984	32354	**30890**
	10	100	15	28503	30051	30209	28015	**27954**	28342
	11	100	15	34501	33315	36518	**32738**	33370	33161
	12	100	15	50505	44284	49300	**39565**	40851	43102
	13	200	30	28580	**27030**	47916	28259	27685	32242
	14	200	30	55075	53193	67950	52867	**51444**	58399
	15	200	30	66589	63042	66672	60993	**60494**	62464

Table 3.2 – *Detailed results for industrial and random instances*

				reass						reseq	static	
	#Threads:		1	1	2	3	4	5	6	1	1	1
	I	$\|O\|$	$\frac{\#m}{s}$	r	r	r	r	r	r	r	r	$\frac{ns}{\|O\|}$
Industrial	01	193	5810	1.00	2.28	3.39	4.54	5.67	6.72	1.38	1.40	637
	02	293	3809	1.00	2.39	3.53	4.74	5.75	6.93	1.39	1.38	647
	03	305	3215	1.00	2.34	3.50	4.59	5.76	6.67	1.48	1.54	664
	04	370	2768	1.00	2.29	3.42	4.59	5.72	6.79	1.43	1.50	649
	05	452	1740	1.00	2.28	3.36	4.50	5.56	6.56	1.76	1.81	700
	06	461	1960	1.00	2.28	3.40	4.43	5.67	6.61	1.57	1.58	698
	07	472	1300	1.00	2.06	3.08	4.35	5.12	6.24	2.16	2.23	731
	08	480	2134	1.00	1.99	2.95	3.69	4.91	5.91	1.31	1.38	705
	09	511	1502	1.00	1.88	2.77	3.76	4.73	5.69	1.77	1.77	736
	10	539	1441	1.00	1.92	2.96	3.99	4.93	5.92	1.70	1.76	733
	11	569	2070	1.00	1.97	2.92	3.88	4.77	5.74	1.17	1.18	718
	12	720	963	1.00	1.62	2.22	3.14	3.75	4.55	1.84	1.83	787
	13	725	654	1.00	2.07	3.05	3.98	5.09	6.14	2.64	2.72	776
	14	752	748	1.00	2.03	3.11	3.69	5.17	6.12	2.28	2.34	759
	15	835	609	1.00	2.17	3.23	3.93	5.01	5.96	2.42	2.44	804
Random	01	82	18605	1.00	1.93	2.92	3.84	4.80	5.54	1.02	1.02	640
	02	80	17831	1.00	1.98	2.97	3.76	4.86	5.65	1.04	1.09	644
	03	75	18635	1.00	1.97	2.97	3.97	4.94	5.83	1.05	1.11	645
	04	147	10053	1.00	1.98	2.94	3.93	4.78	5.67	1.02	1.04	652
	05	165	7732	1.00	2.06	3.09	3.60	5.15	6.12	1.23	1.22	641
	06	159	8749	1.00	2.00	2.93	3.75	4.72	5.21	1.09	1.16	618
	07	222	5589	1.00	2.00	3.00	4.00	4.97	5.90	1.20	1.26	640
	08	269	3959	1.00	2.06	3.10	3.51	5.14	6.08	1.37	1.47	638
	09	220	5682	1.00	1.96	2.92	3.90	4.77	5.63	1.19	1.23	649
	10	398	2485	1.00	2.06	3.04	4.00	5.07	6.01	1.41	1.51	668
	11	406	2167	1.00	1.95	3.03	3.62	4.90	5.84	1.64	1.70	670
	12	387	2507	1.00	1.81	2.68	3.49	4.46	5.29	1.44	1.49	692
	13	796	885	1.00	1.96	2.92	3.88	4.81	5.57	1.87	1.94	730
	14	796	743	1.00	1.98	2.90	3.84	4.83	5.96	2.14	2.23	758
	15	768	820	1.00	1.92	2.86	3.54	4.78	5.59	1.96	2.08	765

Table 3.4 – *Analysis of the number of moves performed per second*

	Initial	Local Search			Sim. Annealing			GRASP		
		static	*reseq*	*reass*	*static*	*reseq*	*reass*	*static*	*reseq*	*reass*
$\bar{I}$	12.2%	7.4%	6.4%	7.6%	3.9%	2.1%	3.8%	2.0%	**0.3%**	**0.3%**
$\widetilde{I}$	10.8%	6.7%	5.7%	6.7%	4.1%	1.6%	3.0%	1.8%	0.3%	**0.0%**
$\bar{R}$	52.9%	41.2%	40.2%	41.2%	8.3%	5.7%	15.2%	2.3%	**1.5%**	4.2%
$\widetilde{R}$	49.6%	39.9%	39.6%	36.5%	5.7%	4.0%	8.1%	1.1%	**0.0%**	2.4%

Table 3.5 – *Aggregated results for industrial and random instances.*

the resequencing strategy. One reason that could explain the performance of the resequencing strategy is that a larger number of moves can be performed in the same amount of time compared to the reassigning strategy, for which additional time is spent to search for nodes that can be settled.

To explain this further, the number of moves performed per second is analyzed in Table 3.4. In column $|O|$, the total number of operations for each instance is provided since we assume it is correlated to the number of moves which are performed per second. Different variants of our algorithms have been tested on the industrial and random instances using a computation time of one minute for each variant and instance. In order to determine the number of moves performed per second, the time for constructing solutions is not taken into account. The GRASP based heuristic is run using a large value for the maximum number of non-improving iterations P_m in order to avoid triggering runs of the construction algorithm. This setting means there is one parallel run of the Simulated Annealing algorithm for each thread that is used. The second row of Table 3.4 provides the number of used threads. For runs using the reassigning strategy with one thread, the absolute number of moves performed per second is given in column $\frac{\#\mathrm{m}}{\mathrm{s}}$. Based on that, all columns entitled by "r" provide the relative number of moves per second. The three rightmost columns provide results for the resequencing and static node selection strategies. The results show that the reassigning strategy requires more time per move than the other strategies. Column $\frac{\mathrm{ns}}{|O|}$ provides the average number of nanoseconds that is spent per node while performing a single move. Being almost constant, these values show that the static selection strategy yields an algorithm that is linear in the number of nodes. The slight increase that can be observed might be due to secondary reasons such an increasal of cache misses coming along with the increased memory consumption of larger instances; also, additional time might be needed for a larger number of jobs. The numbers of moves performed per second increases linearly with the numbers of threads, which shows that the parallel implementation of our algorithm scales with the number of threads.

3.5.2 Instances for Parallel Batch Machines of Mönch et al. (2005)

Mönch et al. (2005) consider a scheduling problem for the diffusion and cleaning area that models the machines in that area as parallel batch processors. This modeling does not include a job-shop environment, so the problem is much less general than ours. From the perspective of this chapter, their instances consist of jobs with exactly one operation without any sequence-dependent setup times. We compare our algorithms with results that are obtained by methods dedicated to this less general problem. Table 3.7 provides average values for total weighted tardiness. The best results for such instances that we are aware of are reported by Chiang et al. (2010). We put their results in brackets since they used a parameter-identical reimplementation instead of the original instances of Mönch et al. (2005). For this comparison, we scaled their reported results using the relative values given in their paper.

	Time	**#Machines**			**Batch size**	
	(s)	m=3	m=4	m=5	b=4	b=8
Mönch et al. (2005)	58	412	300	231	389	240
Chiang et al. (2010)	4	(370)	(272)	(209)	(347)	(220)
Yugma et al. (2012)	178	411	278	206	367	229
Initial	<1	630	455	356	577	383
Local Search static	14	607	434	334	553	363
Local Search reseq	30	539	399	316	487	348
Local Search reass	60	486	361	269	452	292
Sim. Annealing static	120	548	383	294	503	314
Sim. Annealing reseq	120	457	344	261	418	290
Sim. Annealing reass	120	411	303	210	382	234
GRASP static	120	502	356	266	469	280
GRASP reseq	120	429	318	244	399	262
GRASP reass	120	**382**	**274**	**197**	**356**	**212**

Table 3.7 – *Results for instances of Mönch et al. (2005) with total weighted tardiness objective*

Our best method (*GRASP reass*) outperforms the results of Mönch et al. (2005) and Yugma et al. (2012). It reaches a quality that is comparable to the dedicated method of Chiang et al. (2010). In contrast to the previous section, we see stronger differences between the different selection strategies. We assume that this is due to the fact that, in non-job-shop instances, a larger number of operations is available to be settled. We observe that the reassigning strategy strongly outperforms both the static and the resequencing selection strategies. Again, the GRASP metaheuristic approach yields a clear improvement over Simulated Annealing alone.

3.5.3 Complex Job-Shop Instances of Mason et al. (2005)

Mason et al. (2005) consider a complex job-shop scheduling problem from semiconductor manufacturing and provide results for instances based on the mini-fab model of El Adl et al. (1996). They minimize the total weighted tardiness instead of general regular criteria. There is a difference to our problem definition that concerns sequence-dependent setup times. Mason et al. (2005) do not allow the setup between operations o_a and o_b to begin before the route predecessor operation of o_b is completed. In our definition, this setup can begin as soon as o_a is completed. We consider this difference by modifying the instances as follows: We consider all setup durations to be zero and prolong operation processing durations instead. We extend each processing duration by adding the longest possible setup duration that might precede the operation. It is important to note that, in case setup durations are crucial, this modification might increase (but never decrease) the optimal total weighted tardiness.

Tables 3.8, 3.9, and 3.9 show results for the batching capacities $b = 2$, $b = 3$ and $b = 4$, respectively. Columns "Batch-Oblivious" show our results, all others are taken from Mason et al. (2005) for comparison. The column GRASP refers to results obtained using the GRASP based algorithm with the reassigning strategy for node selections. We allowed a computational time of 5 seconds per instance. Values represent normalized average total weighted tardiness and 1.000 represents the best solution found by Mason et al. (2005). For smaller instances, setup durations are crucial and our results are worse due to the assumptions for setups in the modified instances. For instances with more than 5 jobs, setup durations are negligible and our method outperforms the results of Mason et al. (2005). Initial solutions obtained by our construction heuristic are strongly improved by our GRASP based approach. We observe a similar behavior for all considered batching capacities.

3.5.4 Sequence-Dependent Setup Time Instances of Brucker and Thiele (1996)

This section presents numerical results for the instances of Brucker and Thiele (1996) for the job-shop scheduling problem with sequence-dependent setup times. The objective function is the makespan. Though no batching machines are considered, this allows a performance comparison which focuses on sequence-dependent setup times. Our GRASP based metaheuristic approach was run using a computational time of two minutes per instance. Results are provided in Table 3.12. Column *GRASP* reports makespan values of our GRASP based approach and column *Initial* reports makespan values obtained by our construction heuristic. Values are compared to results from the literature. The best known results we are aware of are reported in González et al. (2013) and Grimes and Hebrard (2010). González et al. (2013) propose a tabu search based algorithm. In their numerical results, they report computational times between one and four minutes which are comparable to the two minutes per instances that we allowed. Grimes and Hebrard (2010) present a constraint programming approach. They report computational times below one second for smaller instances and one hour for larger instances. For comparison, we also include results from other known publications on these instances. Both Brucker and Thiele (1996) and Artigues and Feillet (2008)

	SB	Dispatching			MIP	Batch-Oblivious	
$\|J\|$	best	ATCS	CR	EDD	6 hrs	Initial	GRASP
3	2.171	2.871	2.972	2.972	**1.000**	2.036	1.351
4	1.952	2.944	3.004	3.004	**1.000**	2.122	1.294
5	1.822	2.810	3.019	3.019	**1.000**	2.291	1.231
6	1.510	1.850	1.821	1.821	1.000	1.458	**0.937**
7	1.260	1.261	1.347	1.347	1.200	1.402	**0.706**
8	1.301	1.106	1.044	1.044	1.377	1.406	**0.656**
9	1.179	1.013	1.099	1.099	3.315	1.177	**0.639**
10	1.294	1.033	1.030	1.030	3.972	1.333	**0.708**

Table 3.8 – *Results for instances of Mason et al. (2005) with b = 2*

	SB	Dispatching			MIP	Batch-Oblivious	
$\|J\|$	best	ATCS	CR	EDD	6 hrs	Initial	GRASP
3	1.484	3.082	2.974	2.996	**1.000**	2.024	1.268
4	2.028	2.401	3.495	3.371	**1.000**	2.257	1.313
5	1.885	2.267	3.604	3.207	**1.000**	1.575	1.098
6	1.283	1.534	1.860	1.929	1.003	1.440	**0.876**
7	1.131	1.608	1.446	1.515	1.198	1.349	**0.735**
8	1.164	1.325	1.120	1.193	2.860	1.180	**0.713**
9	1.240	1.299	1.087	1.207	N/A	1.497	**0.806**
10	1.266	1.453	1.032	1.066	N/A	1.292	**0.768**

Table 3.9 – *Results for instances of Mason et al. (2005) with b = 3*

	SB	Dispatching			MIP	Batch-Oblivious	
$\|J\|$	best	ATCS	CR	EDD	6 hrs	Initial	GRASP
3	3.786	6.133	6.155	6.155	**1.000**	2.009	1.421
4	2.094	3.010	3.092	3.092	**1.000**	2.118	1.312
5	1.680	2.231	2.343	2.343	**1.000**	1.485	1.102
6	1.451	1.769	1.720	1.720	1.005	1.355	**0.831**
7	1.033	1.237	1.346	1.346	1.343	1.086	**0.716**
8	1.101	1.131	1.206	1.206	N/A	1.164	**0.701**
9	1.076	1.123	1.271	1.271	N/A	1.251	**0.770**
10	1.175	1.015	1.107	1.107	N/A	1.076	**0.718**

Table 3.10 – *Results for instances of Mason et al. (2005) with b = 4*

	Brucker and Thiele (1996)	Artigues and Feillet (2008)	Balas et al. (2008)	Grimes and Hebrard (2010)	González et al. (2013)	Initial	GRASP
t2-ps01	**798**	**798**	**798**	**798**	**798**	1059	**798**
t2-ps02	**784**	**784**	**784**	**784**	**784**	1069	**784**
t2-ps03	**749**	**749**	**749**	**749**	**749**	925	**749**
t2-ps04	**730**	**730**	**730**	**730**	**730**	999	**730**
t2-ps05	**691**	**691**	693	**691**	**691**	780	693
t2-ps06	1056	**1009**	1018	**1009**	1013	1274	1026
t2-ps07	1087	**970**	1003	**970**	**970**	1264	**970**
t2-ps08	1096	**963**	975	**963**	**963**	1184	965
t2-ps09	1119	1061	**1060**	**1060**	**1060**	1335	**1060**
t2-ps10	1058	**1018**	**1018**	**1018**	**1018**	1294	**1018**
t2-ps11	1658	1494	1470	**1443**	**1443**	2012	1455
t2-ps12	1528	1381	1305	**1269**	1274	1679	1299
t2-ps13	1549	1457	1439	**1415**	**1415**	2229	1430
t2-ps14	1592	1483	1485	**1452**	1492	1852	1492
t2-ps15	1744	1661	1527	1486	**1485**	1963	1518

Table 3.12 – *Comparison of makespan for instances of Brucker and Thiele (1996)*

present branch-and-bound based approaches. Balas et al. (2008) propose an approach based on the shifting bottleneck heuristic. The gap between our results and those obtained by dedicated approaches is small, which shows that our approach is also competitive for this type of instances.

3.5.5 Flexible Job-Shop Instances of Hurink et al. (1994)

Finally, we present results for the flexible job-shop instances of Hurink et al. (1994). These instances do not incorporate batching machines and have been widely used to assess the performance of several highly efficient dedicated methods. The instances are partitioned into edata, rdata, and vdata instances that include low, medium and high flexibility levels, respectively. We allow a computation time of 2 minutes per instance for our GRASP based approach. The comparison with best known results from literature refers to the best known solution that is obtained by combining the results of Jurisch (1992), Dauzère-Pérès and Paulli (1997), Mastrolilli and Gambardella (2000), Pacino and Van Hentenryck (2011), Behnke and Geiger (2012) and Schutt et al. (2013). Detailed results are presented in Table 3.13. Columns "Init." show the makespan obtained by our construction heuristic. Columns "GRASP" show the makespan obtained by our GRASP based metaheuristic approach. Columns "Publ." show the best known makespan published in the literature, indicating the first paper that reported the result. We observe an average gap to the best known solution of 0.8%, 0.6%, and 0.2% for the edata, rdata, and vdata instances, respectively. We observe a maximum gap to the best known instance of 3.8%, 2.9%, and 1.4% for the edata, rdata, and vdata instances, respectively. These results show that our non-dedicated method obtains good results even for this less general problem.

	edata			rdata			vdata		
	Init.	GRASP	Publ.	Init.	GRASP	Publ.	Init.	GRASP	Publ.
mt06	56	55	55 J	54	47	47 J	47	47	47 J
mt10	1202	878	**871 J**	831	686	686 D	932	655	655 J
mt20	1407	1092	**1088 J**	1219	1023	**1022 M**	1063	1023	**1022 J**
la01	840	609	609 J	662	573	**570 S**	700	572	**570 J**
la02	761	655	655 J	667	531	**529 S**	559	531	**529 J**
la03	730	550	550 J	619	479	**477 S**	526	478	**477 M**
la04	694	568	568 J	656	504	**502 J**	596	502	502 J
la05	659	503	503 J	580	458	**457 J**	557	458	**457 M**
la06	1102	833	833 J	857	800	**799 M**	1004	799	799 J
la07	887	768	**762 J**	818	751	**749 S**	834	750	**749 M**
la08	929	845	845 J	989	766	**765 M**	875	766	**765 M**
la09	1093	878	878 J	1017	853	853 M	955	854	**853 D**
la10	1019	866	866 J	860	805	**804 M**	915	804	804 J
la11	1220	1104	**1103 J**	1244	1072	**1071 J**	1206	1072	**1071 J**
la12	1215	960	960 J	1035	936	936 D	992	936	936 J
la13	1156	1053	1053 J	1178	1038	1038 J	1106	1038	1038 J
la14	1345	1123	1123 J	1276	1070	1070 J	1134	1070	1070 D
la15	1234	1111	1111 J	1285	1090	**1089 S**	1167	1090	**1089 D**
la16	1285	892	892 J	933	717	717 J	772	717	717 J
la17	807	707	707 J	872	646	646 J	646	646	646 J
la18	1088	842	842 J	826	669	**666 J**	783	663	663 J
la19	1058	799	**796 J**	848	703	**700 M**	728	617	617 J
la20	1033	857	857 J	1161	756	756 J	991	756	756 J
la21	1412	1039	**1009 P**	1143	854	**835 M**	1031	815	**804 B**
la22	1178	887	**880 P**	1103	780	**760 M**	946	741	**736 B**
la23	1333	950	950 M	1193	857	**842 M**	1200	818	**815 M**
la24	1314	916	**908 P**	1162	820	**808 M**	1079	779	**775 B**
la25	1189	955	**936 P**	1052	798	**791 M**	1071	762	**756 M**
la26	1468	1128	**1107 P**	1442	1073	**1061 M**	1250	1057	**1054 M**
la27	1594	1198	**1181 P**	1487	1094	**1091 M**	1282	1088	**1084 B**
la28	1766	1168	**1142 P**	1575	1084	**1080 M**	1363	1072	**1070 M**
la29	1516	1137	**1111 P**	1363	1006	**998 M**	1337	997	**994 M**
la30	1627	1241	**1195 P**	1531	1093	**1078 M**	1288	1072	**1069 M**
la31	2056	1559	**1538 S**	1988	1525	**1521 M**	1783	1523	**1520 M**
la32	2521	1698	1698 D	2080	1660	**1659 M**	1907	1659	**1658 J**
la33	2086	1547	1547 J	1806	1500	**1499 M**	1687	1499	**1497 M**
la34	2117	1645	**1599 M**	2001	1539	**1536 M**	1973	1536	**1535 M**
la35	2147	1736	1736 J	1920	1551	**1550 M**	1895	1552	**1549 M**
la36	1782	1171	**1160 P**	1415	1037	**1030 D**	1232	948	948 J
la37	1588	1397	1397 J	1579	1085	**1077 M**	1269	986	986 J
la38	1601	1178	**1141 S**	1214	976	**962 M**	1184	943	943 J
la39	1710	1208	**1184 J**	1430	1048	**1018 B**	1234	922	922 J
la40	1469	1168	**1144 P**	1268	978	**970 M**	1140	955	955 J

Table 3.13 – *Results for flexible job-shop instances of Hurink et al. (1994).*
Abbreviations: **B***: Behnke and Geiger (2012),* **D***: Dauzère-Pérès and Paulli (1997),* **J***: Jurisch (1992),* **M***: Mastrolilli and Gambardella (2000),* **P***: Pacino and Van Hentenryck (2011),* **S***: Schutt et al. (2013)*

3.6 Conclusion

In this chapter, we considered a complex job-shop scheduling problem with a focus on the integration of batching machines. We reduced the structural complexity of disjunctive graphs by introducing a novel batch-oblivious representation. This representation allows batching decisions to be taken during a traversal of the graph and enables the implementation of resequencing and reassignment strategies that adaptively "fill up" underutilized batches. Together with an integrated batch-oblivious move, we obtain a neighborhood that is applied in a GRASP based heuristic approach. The scheduling problem on parallel batching machines is often considered to be important because it is a subproblem in the shifting bottleneck heuristic. Our holistic way to modify schedules outperforms such approaches for the instances considered in our numerical experiments. Our batch-oblivious approach improves both solution quality and implementation complexity in comparison to decomposition based approaches.

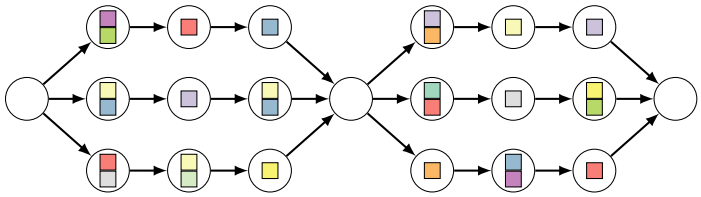
Avoiding the complexity of additional batching nodes simplifies the inclusion of further constraints. Regarding the diffusion and cleaning area, we want to model machines in more detail in order to account for the industrial problem specification presented in section 2. Chapter 4 extends the approach of this chapter by allowing complex routing structures and multiple resources per operation. This can be used to model machines in more detail. Also in this context, we want to include additional time constraints that limit the time between certain operations (see Klemmt and Mönch (2012); Sadeghi et al. (2015)). Chapter 5 will extend the approach in order to include such maximum time lag constraints.

Though we already obtain good numerical results, we still see opportunities for further improvements. Enhancing the node selection strategies proposed in section 3.3.4 seems promising. The batch-oblivious approach is independent of the proposed GRASP based metaheuristic and also can be applied within other heuristic or exact methods. GRASP has strong diversification and intensification mechanisms but lacks elements of mutual learning that can be found in path-relinking approaches or genetic algorithms. Speeding up individual moves by only partially updating the graph as proposed by Michel and Van Hentenryck (2003), Pearce and Kelly (2007) and Sobeyko and Mönch (2016) seems applicable and promising as well. García-León et al. (2015) obtain good results for flexible job-shops. They avoid the runtime cost of performing a move by evaluating its effects on the objective function without actually performing the move.

Chapter 4

Extended Route and Resource Flexibility in Job-Shop Scheduling

MACHINES in the diffusion and cleaning area show a very complex behavior. To increase the accuracy of our model, we manage internal resources of machines while taking scheduling decisions. This leads to a job-shop scheduling problem with extended route and resource flexibility.



As comprehensively discussed in the industrial specification in chapter 2, a wide range of different machines can be found in the diffusion and cleaning area. Each machine has its specific properties and often their internal components must be considered. in order to fully describe their behavior. We want to consider machines that are modeled in detail as part of a job-shop scheduling problem in order to reduce the gap between our model and the reality. Thus, to obtain practicable schedules, we consider several machines, in particular furnaces, in detail and include their components as resources in our model. This leads to a generalized version of the flexible job-shop problem that imposes additional constraints for resource utilization. A literature review on related work is presented in section 1.4.2. We have presented a preliminary version of this approach for scheduling job-shops with complex routing structures in Knopp et al. (2014), where the scheduling of internal machine components is integrated in a job-shop scheduling problem for the overall work area. We are not aware of other approaches that consider machines in detail while combining this with a job-shop scheduling problem. The closest approach we are aware of is that of Kis (2003). Note that this chapter generalizes the approach of chapter 3. The main extensions are the introduction of route graphs and the consideration of multiple resources per operation.

Let us take furnaces as an example and consider them in detail in order to illustrate the idea of the approach. Recall from section 2.2.4 that furnaces in the diffusion and cleaning area consist of tubes, boats and load ports. A *tube* is the place where processes are conducted. A *boat* is a movable carrier for wafers which is necessary to run a process inside a tube. Boats are also utilized to load, unload and cool wafers. A *load port* is the place where wafers are loaded and unloaded. Commonly, such machines consist of two tubes, four boats (two per tube), and one load port. To process a set of wafers, a boat is used as follows. First, wafers are loaded from its carrier to the boat at the load port. Then, the boat is moved into the tube where the process is conducted. Afterwards, the boat is removed from the tube and has to cool down before its wafers can be unloaded at the load port. Potentially, the boat has to wait in case the tube or the load port is occupied. Some operations, e.g. loading wafers, require more than one resource at the same time. The load port, tubes and boats are components of a furnace. Figure 4.1 provides a schematic illustration of such a machine. A more detailed description of this type of machines is given in section 2.2.4 of the industrial problem specification. We decompose one processing step of a furnace into separate operations using its internal resources as described before. Note that processing times of operations may depend on the specific internal resource that is used.

The properties described in the previous paragraph impose dependencies between resources used by consecutive operations. Consider the following two characteristic examples. Firstly, if a loading operation uses the load port of a machine, the corresponding unloading operation must use the load port of the same machine. Secondly, after a specific boat is loaded, it cannot be used elsewhere before being unloaded. So, a resource can be blocked even if no operation is currently using it.

We base our approach on the job-shop scheduling problem presented in the preceding chapter. There, a given set of jobs must be scheduled using given resources. For each job, an individual route of operations must be processed. It is a generalization of the flexible

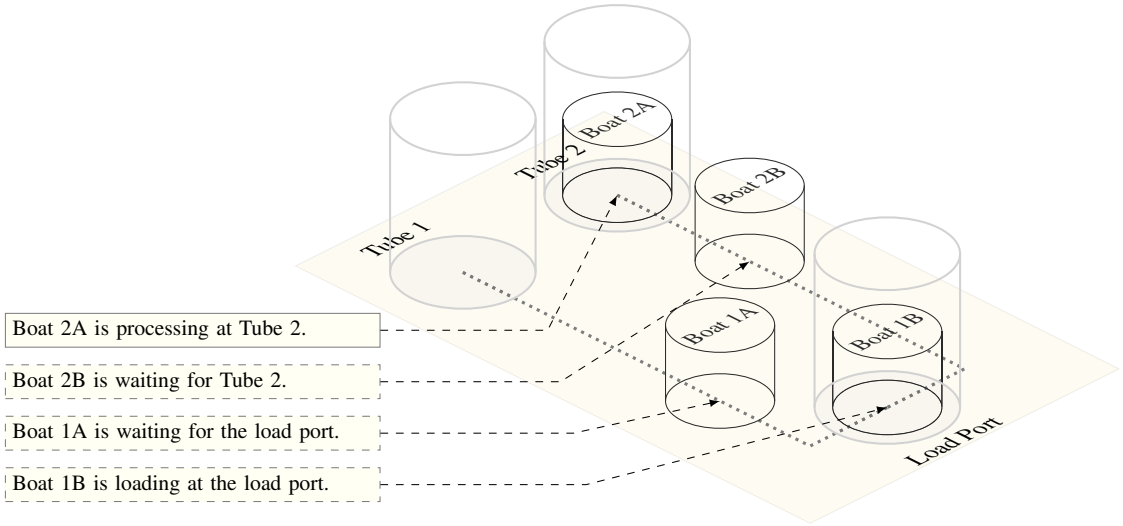


Figure 4.1 – *Schematic representation of a furnace in the diffusion and cleaning area*

job-shop scheduling problem that allows the resource used to process an operation to be chosen from its set of allocated resources. To cope with the given constraints of furnaces, we formulate an extension of the complex job-shop scheduling problem presented in chapter 3. The formal description is given in section 4.1. In this extension, resource dependencies are taken into account by defining multiple routes with fixed resource assignments: For each job, we statically assign resources to operations and resource flexibility is obtained by allowing different routes. Allowed routes are specified by *route graphs* which are introduced before the actual problem definition. The introduced model is not dependent on the objective function to be optimized and, as in the preceding chapter, regular criteria are optimized.

A suitable disjunctive graph representation is given in section 4.2 that is the basic structure needed in the following algorithms. Section 4.3 defines a neighborhood by reordering and resequencing operations that is based on the insertion of nodes into the conjunctive graph in order to extend the GRASP based meta-heuristic solution approach presented in the preceding chapter. In section 4.4, we present and discuss numerical results of our implementation.

4.1 Formal Problem Description

This section extends the scheduling problem that was formally described in section 3.1. The main ideas of the problem description given here have been introduced in Knopp et al. (2014). Before formally defining the problem, let us outline its main elements. We generalize the routes of jobs as follows: Instead of considering a fixed linear sequence of operations, feasible routes are defined by a route graph that is associated to each job. This allows dependencies between resources to be taken into account. Machine flexibility is achieved by choosing paths in these graphs. Thus, other than in section 3.1, the assignment of machines to op-

erations is fixed, but not all operations have to be scheduled. This increased generality in the modeling is emphasized by a change in the terminology: The term resource is used instead of the term machine (though both could be used interchangeably). Instead of exactly one resource, multiple resources per operation can now be required. Section 4.1.1 provides preliminary definitions and notation required for subsequent definitions. Then, the basic problem definition is given in section 4.1.2. It is refined in section 4.1.3 where resource acquisitions are introduced. Section 4.1.4 specifies the inclusion of batching machines into the modeling. Finally, section 4.1.5 discusses properties of the problem and provides application examples.

4.1.1 Preliminaries and Notation

Consider a graph $G = (V, E)$ with a set of nodes V and a set of edges E. For a node $v \in V$, we denote the set of incoming edges as $in(v) \subset E$ and the set of outgoing edges as $out(v) \subset E$. A *path* from $v_1 \in V$ to $v_k \in V$ is defined as a sequence of nodes $(v_1, v_2, \ldots, v_k)$ with $(v_i, v_{i+1}) \in E$ for all $1 \leq i < k$. Next, we introduce the term *two-terminal series parallel graph* according to the definition of Eppstein (1992) and repeat his definition here for completeness. Then, we use that definition to define the term route graph.

Definition 4.1. *Eppstein (1992) A directed graph G is* two-terminal series parallel*, with terminals α and ϕ, if it can be produced by a sequence of the following operations:*

1. Initialization*: Create a graph that consists of a single edge directed from α to ϕ.*
2. Parallel composition*: Given two two-terminal series parallel graphs X and Y, with terminals α_x, ϕ_x, α_y, and ϕ_y, form a new graph $G = P(X, Y)$ by identifying $\alpha = \alpha_x = \alpha_y$ and $\phi = \phi_x = \phi_y$.*
3. Serial composition*: Given two two-terminal series parallel graphs X and Y, with terminals α_x, ϕ_x, α_y, and ϕ_y, form a new graph $G = S(X, Y)$ by identifying $\alpha = \alpha_x$, $\phi_x = \alpha_y$, and $\phi_y = \phi$.*

Definition 4.2. *A two-terminal series parallel graph $G = (V, E, \alpha, \phi)$ is called a* route graph *if, for all $v \in V$, one of the following properties is true:*

1. *v is neither branching, nor merging:* $|\text{in}(v)| = |\text{out}(v)| = 1$
2. *All paths from α to ϕ include v; i.e., they have the form $(\alpha, \ldots, v, \ldots, \phi)$. Such a node v is called a* route separator.

A node of a route graph corresponds to an *operation*. A path from α to ϕ in a route graph is called a *route*. The nodes of a route define a sequence of operations. Note that the start node α and the terminal node ϕ are also *route separators*.

4.1.2 Basic Problem Description

This section provides the formal description of a complex job-shop scheduling problem with extended route and resource flexibility. We want to optimize regular objective functions. The definition given here is self-contained. Definitions known from section 3.1 are reintroduced for completeness while pursuing a consistent notation. This section does not yet include batching machines or resource acquisitions which are introduced in subsequent sections. The problem is formally defined in the following.

We are given a set of *jobs* J that have to be processed using a set of *resources* M. For each job $j \in J$, we are given a set of *operations* O_j and a *release date* $r_j \in \mathbb{Z}$. The disjoint union $O = O_1 \dot\cup O_2 \ldots \dot\cup O_{|J|}$ denotes all given operations. For each job $j \in J$, feasible sequences of operations are specified by a given *route graph* $G_j = (O_j, E_j, \alpha_j, \phi_j)$. For each operation $v \in O_j$, we are given a processing time $p_v \in \mathbb{N}_0$ and a set of resources $M_v \subset M$. For each resource $m \in M_v$ that is required by an operation v, a setup family $\sigma_{v,m} \in \widetilde{F}$ from a given set of *setup families* $\widetilde{F}$ is specified. A given mapping $s : \widetilde{F} \times \widetilde{F} \to \mathbb{N}_0$ prescribes *sequence-dependent setup times* between scheduled operations that use the same machine. Setup times must fulfill the triangle inequality: For all $(f_1, f_3, f_3) \in \widetilde{F} \times \widetilde{F} \times \widetilde{F}$ it must hold that $s(f_1, f_3) \leq s(f_1, f_2) + s(f_2, f_3)$.

A schedule is completely characterized by providing for each job $j \in J$ a *route selection* $R_j \subset O_j$ and *start dates* $S_{i,j} \in \mathbb{Z}$ for all *selected operations* $o_{i,j} \in R_j$. The route selection R_j must describe a path $(\alpha_j = o_{1,j}, \ldots, o_{|R_j|,j} = \phi_j)$ in the route graph G_j. We denote the resources and processing durations related to this selection as $M_{i,j}$ and $p_{i,j}$, respectively. The disjoint union $R = R_1 \dot\cup R_2 \ldots \dot\cup R_{|J|}$ denotes all selected operations. To describe a schedule that is *feasible*, selected routes R_j and start dates $S_{i,j}$ have to respect several constraints that are detailed in the following. Preemption is not allowed: Once the processing of an operation has begun, it cannot be interrupted. Thus, the *completion time* of an operation $o_{i,j} \in O_j$ is given by $C_{i,j} = S_{i,j} + p_{i,j}$. Operations belonging to the same job have to be performed in the order that is specified by the route selection. So, $C_{i,j} \leq S_{i+1,j}$ has to be fulfilled for all $o_{i,j} \in R$ with $i < \left|R_j\right|$. The first operation $o_{1,j} \in R_j$ of each job cannot be processed before its release date, so $S_{1,j} \geq r_j$ must hold for all $j \in J$. For two operations $o_{i,j}, o_{k,l} \in R$ with $M_{i,j} \cap M_{k,l} \neq \emptyset$, having $S_{i,j} = S_{k,l}$ in general means that a batch is created, which is only allowed in the cases specified in section 4.1.4. In all other cases, for all common resources $m \in M_{i,j} \cap M_{k,l}$ of two operations $o_{i,j}, o_{k,l} \in R$, either $S_{i,j} + p_{i,j} + s(\sigma_{i,j,m}, \sigma_{k,l,m}) \leq S_{k,l}$ or $S_{k,l} + p_{k,l} + s(\sigma_{k,l,m}, \sigma_{i,j,m}) \leq S_{i,j}$ must hold. We want to optimize regular objective functions as defined in section 3.1.

For separator operations $o_{i,j} \in O$, we assume without loss of generality that $M_{i,j} = \emptyset$ and $p_{i,j} = 0$. In particular, this assumption is interesting for the terminal nodes $\phi_j = o_{|J|,j}$ since it allows the identification of the completion time of a job $j \in J$ with the start date $C_{|J|,j} = S_{|J|,j}$ of its terminal node ϕ_j. This provides a notation consistent to that of Mati et al. (2011).

4.1.3 Resource Acquisitions

We extend the basic version of the problem description by an additional constraint. Consider two operations that are part of the same route and require a common resource. In some cases, we want to exclusively acquire the resource between such two operations; we prohibit other operations to use the resource in between. More formally, we are given a set of resource acquisitions as a subset $A_{i,j} \subset M_{i,j}$ for each operation $o_{i,j} \in O_j$. For all given acquisitions $m \in A_{i,j}$, their release must be uniquely defined; there must exist an operation $o_{k,j}$ with $m \in M_{k,j}$ such that there is a path $P_m = \left(o_{i,j}, \ldots, o_{k,j}\right)$ in G_j that does not contain a route separator. This path must be minimal in the sense that, for all operations $o_{l,j} \in P_m$ with $l \neq i$, $l \neq k$, we must have $m \notin M_{l,j}$. Note that a resource can be immediately reacquired: we allow $m \in A_{k,j}$. The resource acquisition constraint now imposes that, for an acquisition of a resource $m \in A_{i,j}$ at an *acquisition operation* $o_{i,j}$ with a corresponding *release operation* $o_{k,j}$, there must not be any other operation that uses m in the time between $S_{i,j}$ and $C_{k,j}$. So, for all operations $o_{x,y} \in O$ $(\neq o_{i,j}, \neq o_{k,j})$ with $m \in M_{x,y}$, either $S_{x,y} \geq C_{k,j}$ or $C_{x,y} \leq S_{i,j}$ must hold.

Figure 4.2 presents an example of a route graph including resource acquisitions. This route graph allows six different routes between α_j and ϕ_j: Three alternatives in the first section times two alternatives in the second section. The nodes represent operations and are labeled with the resources that are required to process them. Resource acquisitions are indicated by a superscript "A" and a dashed arc (which is not an edge of the route graph) from the acquisition operation to the release operation.

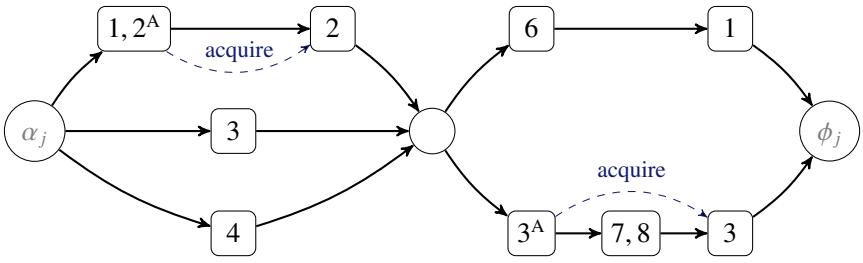


Figure 4.2 – *Example of a route graph of a job $j \in J$*

4.1.4 Batchable Resources

This section extends the formal problem description in order to include batching machines. Batching is restricted to individual operations and not considered for sequences of operations. It remains an interesting subject for future research to further generalize this part of the problem definition. Though batching is not in the focus of this chapter, including it in this definition shows that the problem considered in chapter 4 is a true generalization of the one considered in chapter 3.

Formally, we extend the problem as follows. We are given a set of *batch families* F. For each batch family $f \in F$, we are given a batching capacity $b_f \in \mathbb{N}_{>0}$. A batch family $f_v \in F$ is assigned to each operation $v \in O_j$. All operations $v \in O$ that are assigned to the same batch family $f \in F$ must have the same processing duration p_v, must require an identical set of resources M_v, and must have, for each resource $m \in M_v$, the same setup family $\sigma_{v,m}$. A path $(v_k, \ldots, v_{k+l})$ in a route graph G_j is called a *movable component* if it has a unique predecessor node v_{k-1} and a unique successor node v_{k+l+1} that both are separator nodes. We consider batching only for movable components that consist of a single node. More formally, for all operations $o \in O_j$ with $\exists\, v \in in(o) \cup out(o)$ such that v is not a separator node, we require $b_f = 1$ for $f = f_o$.

Only operations of the same family can be processed at the same time on the same machine. So, for two operations $o_{i,j}, o_{k,l} \in R$ with $f_{i,j} = f_{k,l}$ and $M_{i,j} \cap M_{k,l} \neq \emptyset$, we relax the resource sequencing constraints of section 4.1.2 and allow $S_{i,j} = S_{k,l}$. It $S_{i,j} \neq S_{k,l}$, the resource sequencing constraints from section 4.1.2 have to be fulfilled. A subset $B \subset R$ of operations with $M_{i,j} \cap M_{k,l} \neq \emptyset$, $f_{i,j} = f_{k,l}$, and $S_{i,j} = S_{k,l}$ is called a *batch* . Batching capacities limit the number of operations per batch. Thus, we require $\left|\left\{o_{k,l} \in R \;\middle|\; M_{k,l} \cap M_{i,j} \neq \emptyset \wedge S_{k,l} = S_{i,j}\right\}\right| \leq b_f$ for all operations $o_{i,j} \in R$ with $f = f_{i,j}$.

4.1.5 Discussion and Possible Applications

The problem specified in this section is a generalization of the problem specified in chapter 3. Since it is a generalization, this problem definition also includes all special cases mentioned in section 3.1, e.g., flexible job-shop scheduling problems or parallel batching machine scheduling problems. Again, additional properties such as due dates or job weights can be included in the objective function. Let us in the following show the generality of the problem description introduced in this section by illustrating possible applications. Note that the formal problem description can be extended to resource-dependent processing durations in the sense that the processing duration can be different for the route of the jobs as well as each individual resource. We informally provide the main ideas of the modeling without describing them in full detail.

Serial-single wafer machines are introduced in section 2.2.1 and show pipeline behavior: For two operations that are consecutively processed on the same machine, their processing intervals can overlap. This can be modeled by assigning different durations to the resource-dependent processing period and the processing duration of the operation. Thus, processing a subsequent operation can start while the preceding operation is still being processed.

Photolithography tools often constitute bottlenecks in semiconductor manufacturing facilities. The ability to consider multiple resources per operation, together with setup families that are given independently for each resource, allows us to tackle such problems. Two resources are required for each operation: The photolithography tool and an auxiliary resource, called reticle. Sequence-dependent times are required due to temperature

changes. Reticles need to be transported between different machines which involve transport durations that are given by a travel time matrix. Both can be modeled by defining appropriate resource-dependent setup families and sequence-dependent setup times.

Blocking Constraints Wet bench machines, described in section 2.2.5, involve a sequence of bath tanks that each operation has to consecutively traverse. Modeling this in detail introduces a blocking constraint: Since there is no storage between adjacent bath tanks, an operation can only enter a bath tank if the preceding operation has already left it, i.e., has started its subsequent operation. This can be modeled using resource acquisition constraints as follows. The first bath tank is acquired at an initial operation and released at the consecutive operation that requires two resources: The first bath tank and its subsequent bath tank. There, the first bath tank is released at the same time as the next one starts processing. This shows that combining different aspects of the modeling approach, in this case resource acquisitions and resource-dependent processing durations, offers valuable options.

Multiple orders per job Some lots may contain multiple wafers associated to different orders which might require varying routes. Being in the same physical container forbids different jobs to be processed at the same time on different machines. We believe this can be modeled by considering each order as a job that requires its container as an additional resource.

4.2 Generalized Batch-Oblivious Conjunctive Graphs

We tackle the generalized problem described in the preceding section by extending our conjunctive graph based heuristic approach. This section generalizes the batch-oblivious conjunctive graph representation of section 3.2 such that all properties of the scheduling problem described in section 4.1 are taken into account. We provide a self-contained description in order to introduce notation and specify the graph in detail. The absence of dedicated batching nodes and the absence of redundant edges in the sequencing of operations on resources remain core properties of the graph. We specify a disjunctive graph representation analog to the one presented in Dauzère-Pérès and Paulli (1997), Dauzère-Pérès et al. (1998), or Kis (2003). After specifying the graph in detail, we describe an adaptation of our start date and batching algorithm of section 3.3. The approach of section 3.3 remains applicable for the introduced generalized conjunctive graph with only few modifications. This section builds the foundation for section 4.3, where a solution approach based on efficient node insertions is presented.

4.2.1 Route-Graph-Aware Conjunctive Graphs

Let us now introduce a route-graph-aware batch-oblivious *conjunctive graph* representation. Schedules are represented as a directed acyclic graph $G = (V, E)$ with nodes $V = O \cup \{0, *\}$ that correspond to the given operations O plus an artificial start node 0 and an artificial end node $*$. Note that all operations can be considered to be part of the conjunctive graph, even if they are not scheduled. As in section 3.2, for each job and each resource, the graph contains one path from the artificial start node 0 to the artificial end node $*$. We denote the edges involved in the sequencing of operations on resource $m \in M$ by $E_m \subset E$. Analogously, edges involved in the sequencing of the operations of a job $j \in J$ are denoted by $E_j \subset E$. The disjoint union $E = \dot{\bigcup}_{j \in J} E_j \ \dot{\cup} \ \dot{\bigcup}_{m \in M} E_m$ of these paths yields all edges of the graph. Each scheduled node $v \in R$ is included in exactly $|M_v| + 1$ paths. One corresponds to the route of its job and all others correspond to the sequencing of its assigned resources. For a node $v \in R$, we denote its route successor by $r(v) \in V \setminus \{0\}$ and the set of its resource successors by $m(v) \subset V \setminus \{0\}$. Analogously, its predecessors are denoted by $r^{-1}(v) \in V \setminus \{*\}$ and $m^{-1}(v) \subset V \setminus \{*\}$. Each unscheduled operation $u \in O \setminus R$ corresponds to a disconnected node in the conjunctive graph, i.e. $in(u) = out(u) = \emptyset$. The artificial start node 0 has $|J| + |M|$ outgoing edges and no incoming edges. The artificial end node $*$ has $|J| + |M|$ incoming edges and no outgoing edges. Overall, the graph has $|E| = |J| + |M| + |R| + \sum_{v \in R} |M_v|$ edges.

Next, we include the resource acquisition constraint as follows. For each acquisition of a resource $m \in A_{i,j}$ at an operation $o_{i,j} \in R_j$ with a corresponding release operation $o_{k,j} \in R_j$, the following property must hold: For all operations $o_{x,y} \in R$ ($\neq o_{i,j}$, $\neq o_{k,j}$) with $m \in M_{x,y}$ there must not be any path in the disjunctive graph that has the form $(o_{i,j}, \ldots, o_{x,y}, \ldots, o_{k,j})$. Consequently, $o_{k,j}$ must directly follow $o_{i,j}$ in the sequencing of operations scheduled on resource m (if both operations are scheduled). Thus, there must be an edge $(o_{i,j}, o_{k,j}) \in E_m$. This property must be preserved if the graph is modified.

As described in section 3.2, start dates $S_v \in \mathbb{Z}$ of operations $v \in O$ are determined from conjunctive graphs using longest paths. Some adaptations are needed for the generalized problem. Again, a weight $l_{u,v}$ is associated to each edge $(u, v) \in E$ to ensure a minimum duration between the start dates of adjacent operations, i.e. $S_v \geq S_u + l_{u,v}$ is required for each edge $(u, v) \in E$. For each operation $v \in O$, its start date S_v is determined by the distance $L(0, v)$ of the longest path from the artificial start node 0 to the node v. To respect the constraints given in our scheduling problem, edge weights are defined as follows. For edges $(0, o_{1,j}) \in E_j$ connecting the artificial start node 0 with the initial operation $o_{1,j}$ of job $j \in J$, the edge weight is set to the release date r_j of job j. For edges $(0, o_m) \in E_m$ connecting the artificial start node 0 with the initial operation o_m scheduled on resource $m \in M$, the edge weight is set to zero. For route edges $(v, r(v)) \in E$ with $v \neq 0$, the edge weight is set to the processing duration p_v of operation v. For resource edges $(v, w) \in E_m$ with $v \neq 0$ and $w \in m(v)$, the edge weight is set either to zero when both operations are included in the same batch, or to the sum $p_v + s(\sigma_{v,m}, \sigma_{w,m})$ of the resource-dependent processing duration p_v of operation v on resource m and the sequence-dependent setup time $s(\sigma_{v,m}, \sigma_{w,m})$ between operation v and operation w on machine m. The cases in which operations can be combined into a batch are discussed in the following section.

4.2.2 Integrated Computation of Start Dates and Batches

In this section, the integrated algorithm for computing start dates and batching decisions of section 3.3 is adapted for the previously introduced route-graph-aware batch-oblivious conjunctive graph. The main difference with chapter 3 is that we now have to deal with operations that can require multiple resources at the same time. Though route graphs increase complexity, recall that (as described in section 4.1.4) batching is restricted to movable components consisting of one single node. So, at most one operation of the same job can be involved in each batch. Relaxing this constraint remains an interesting challenge for future research.

First, we provide a necessary condition for combining two operations in a common batch in the presence of multiple resources per operation. Recall that operations with the same batch family require identical sets of operations, i.e. for two operations $u, v \in O$ with $f_u = f_v$, it follows that $M_u = M_v$. In the following, a conjunctive graph as defined in the preceding section is considered.

Proposition 4.1. *For two operations $u, v \in O$ with $(u, v) \in E_m$ and $f_u \neq f_v$, it follows that $S_u + p_u + s(\sigma_{u,m}, \sigma_{v,m}) \leq S_v$.*

Proof. Since $f_u \neq f_v$, the operations u and v cannot be included in a common batch. Thus, the weight of the edge (u, v) has to be chosen as $l_{u,v} = p_u + s(\sigma_{u,m}, \sigma_{v,m})$ and the proposition follows. □

Theorem 4.1. *If, for two operations $u, w \in R$, we have $w \in m(u)$ and $|m(u)| \neq 1$, then $S_u + p_u + s(\sigma_{u,m}, \sigma_{w,m}) \leq S_w$.*

Proof. Let $u, w \in R$ be two operations with $w \in m(u)$ and $|m(u)| \neq 1$. If $f_u \neq f_w$, the inequality holds due to Proposition 4.1. Let us therefore assume $f_u = f_w$. With $|m(u)| \neq 1$, it follows that $\exists\ v \in m(u)$ with $v \neq w$. Let us consider a resource m corresponding to an edge $(u, v) \in E_m$ and a resource $m' \neq m$ corresponding to an edge $(u, w)) \in E_{m'}$. With $f_u = f_w$, it follows $M_u = M_w$, and thus we know that $m \in M_u \cap M_v \cap M_w$. In the sequencing of the operations on the resource m, w cannot be scheduled before v, because otherwise (since $(u, v) \in E_m$) w must be also scheduled before u, which would lead to a cycle in the graph (since $(u, w) \in E_{m'}$). Thus, operation v has to be scheduled before operation w on resource m.

Now, let us assume that $f_v = f_u$ and thus $m' \in M_u \cap M_v \cap M_w$. In the sequencing of m', v cannot be scheduled before w, because otherwise (since $(u, w) \in E_{m'}$) v must be also scheduled before u, which would lead to a cycle in the graph (since $(u, v) \in E_m$). Thus, w has to be scheduled before v on m'. This contradicts that v has to be scheduled before w on m since there would be a cycle in the graph. Thus, we conclude $f_v \neq f_u$ by contradiction.

Finally, since $f_v \neq f_u = f_w$ and because the considered operations are scheduled on the resource m in the order u before v and v before w, we conclude with Proposition 4.1 that $S_u + p_u + s(\sigma_{u,m}, \sigma_{v,m}) \leq S_v$ and $S_v + p_v + s(\sigma_{v,m}, \sigma_{w,m}) \leq S_w$. Since s satisfies the triangle inequality, then $S_u + p_u + s(\sigma_{u,m}, \sigma_{w,m}) \leq S_w$. □

Theorem 4.1 describes a necessary condition for two operations to be scheduled in the same batch. For a node $v \in R$ encountered during graph traversal, this is the first condition to be checked for deciding if v should extend an existing batch. If $\left|m^{-1}(v)\right| = 1$ is fulfilled, v has a unique resource predecessor and combining both in the same batch can be considered. Note that in the following, if $|m(v)| = 1$, by abuse of notation we identify a set $m(v)$ with its only element, e.g. if $m(v) = \{u\}$ then $S_{m(v)}$ refers to S_u. We include this condition by adapting invariant 3.4 of chapter 3 as follows: We require the invariant

$$\left(l_{v,u} = 0 \ \wedge \ |m(v)| = 1 \ \wedge \ S_v \geq S_{r^{-1}(u)} + p_{r^{-1}(u)}\right) \ \vee \ \left(l_{v,u} = p_v + s(\sigma_{v,m}, \sigma_{u,m})\right) \tag{4.1}$$

to be fulfilled for all edges $(u, v) \in E_m$ of each resource $m \in M$. Algorithm 4.1 provides the pseudo-code for an integrated start date and batching algorithm which considers multiple resources per operation by generalizing Algorithm 3.1.

Algorithm 4.1 Static batching and start date computation for a conjunctive graph G

computeStartDatesStatically (G)
$S_0 \leftarrow 0$
$\beta_v \leftarrow 1 \quad (\forall\ v \in V)$
for $v \in computeTopologicalOrdering\,(G \setminus \{0\})$
if $\left|m^{-1}(v)\right| = 1$ **and** $S_{r^{-1}(v)} + p_{r^{-1}(v)} \leq S_{m^{-1}(v)}$ **and** $f_{m^{-1}(v)} = f_v$ **and** $\beta_{m^{-1}(v)} < b_v$
$S_v \leftarrow S_{m^{-1}(v)}, \quad \beta_v \leftarrow \beta_{m^{-1}(v)} + 1$
else
$S_v \leftarrow \max\left(S_{r^{-1}(v)} + p_{r^{-1}(v)},\ \left(\max_{m \in M,\,(u,v) \in E_m} S_u + p_u + s(\sigma_{u,m}, \sigma_{v,m})\right)\right)$

Having provided a static algorithm, we are now interested in applying the adaptive algorithm of section 3.3.3 to the generalized problem. Recall that in the adaptive algorithm, schedules are dynamically improved "on the fly" by filling up batches with remaining machine capacity. This involves modifications of the graph while it is being traversed. Algorithm 4.2 shows the pseudo-code of an adapted version of Algorithm 3.2. The main difference with Algorithm 3.2 is that a check of the condition $|m(v)| = 1$ is added for a all traversed nodes $v \in V$ as in the static algorithm.

Note that the selection strategies for resequencing and reassigning operations that have been presented in section 3.3.4 are adapted. If a node $v \in V \setminus V_s$ cannot extend the underutilized batch of its resource predecessor, an alternative node to be settled can be determined. Since the resource predecessor of an operation $v \in O$ is not uniquely determined if $|m(v)| > 1$, an arbitrary operation $u \in m(v)$ is chosen in this case for starting the search for a node that can extend the batch. Beside that, no other changes have been made for the selection strategies.

Overall, we have seen that the batch-oblivious approach still can be used if multiple resources per operation are considered. Constraining the problem to movable components of one single node considerably simplifies the task. We continue to adapt the approach that is proposed in chapter 3 for the generalized scheduling problem in the following section.

Algorithm 4.2 Adaptive batching and start date computation for a conjunctive graph G

computeStartDatesAdaptively (G)
$S_0 \leftarrow 0$
$V_s = \{0\}$
$\beta_v \leftarrow 1 \quad (\forall\, v \in V)$
while $V_s \neq V$
$v,\, w \leftarrow$ select $(v \in V \setminus V_s,\, w \in V_s)$
assert $(r^{-1}(v) \in V_s$ **and** $deg_s^+(w) = 0)$
settle v after w
if $\left|m^{-1}(v)\right| = 1$ **and** $S_{r^{-1}(v)} + p_{r^{-1}(v)} \leq S_{m^{-1}(v)}$ **and** $f_{m^{-1}(v)} = f_v$ **and** $\beta_{m^{-1}(v)} < b_v$
$S_v \leftarrow S_{m^{-1}(v)}, \quad \beta_v \leftarrow \beta_{m^{-1}(v)} + 1$
else
$S_v \leftarrow \max\left(S_{r^{-1}(v)} + p_{r^{-1}(v)},\ (\max_{m \in M,\, (u,v) \in E_m} S_u + p_u + s(\sigma_{u,m}, \sigma_{v,m}))\right)$
$V_s \leftarrow V_s \cup \{v\}$

4.3 Solution Approach

As described in the preceding section, solutions of the considered scheduling problem with extended route and resource flexibility can be represented using generalized batch-oblivious conjunctive graphs. An important element of the approach presented in chapter 3 is the integrated move for resequencing and reassigning operations of Dauzère-Pérès and Paulli (1997). In our batch-oblivious approach of chapter 3, this move complements batching decisions and conjunctive graph modifications which are performed "on the fly" during graph traversals. When generalizing the approach, the integrated move of Dauzère-Pérès and Paulli (1997) has to be modified or replaced since, in its original form, it cannot cope with multiple resources per operation or resource acquisition constraints. Therefore, based on the work of Kis (2003), this section introduces a move that takes into account multiple resources per operation and resource acquisition constraints at the same time. A crucial aspect of the move introduced in this section is the insertion of individual nodes. For this, an algorithm is described in section 4.3.1. Subsequently, section 4.3.2 details the integration of route-graph-aware moves in our heuristic algorithms.

The integrated move of Dauzère-Pérès and Paulli (1997) for resequencing and reassigning operations has been generalized to multiple resource per operation in Dauzère-Pérès et al. (1998) and Dauzère-Pérès and Pavageau (2003). Both approaches propose moves that modify, for one operation, its sequencing or assignment on only one out of multiple resources while leaving all others untouched. This is not viable in the presence of resource acquisition constraints since both affected operations must be moved at the same time. Moreover, moves for changing route selections are needed which requires to modify entire sequences of operations at once.

In a preliminary work presented in Knopp et al. (2014), two kinds of moves are introduced for this problem: One reorders operations on a given machine and the other one modifies the selected path in the route graph. However, the former move misses to address a case described in Dauzère-Pérès et al. (1998) where an operation has the same direct predecessor for more than one resource. In this case, both predecessor relations have to be modified at the same time.

The route graph formulation of our problem given in section 4.1 is closely related to the job-shop scheduling problem with processing alternatives introduced in Kis (2003), where also multiple resources per operation are considered. The meta-heuristic solution approaches presented in Kis (2003) are based on an efficient algorithm for inserting nodes in the conjunctive graph, focusing on the makespan as the objective function. A similar insertion technique for nodes is proposed by Artigues and Roubellat (2002) in a setting that includes sequence-dependent setups in addition to multiple resources per operation. Both insertion methods avoid to enumerate dominated insertion positions. In our approach, we adapt the node insertion technique of Kis (2003) to the problem at hand by considering resource acquisition constraints and applying it for problems with regular objective functions.

The move introduced in this section is based on the movable components of a route graph which are defined in section 4.1.4. A move works in two phases: First, it removes all nodes belonging to a currently scheduled movable component from the conjunctive graph. Second, it inserts all nodes that belong to a movable component into the conjunctive graph. The latter movable component could either be the same that was removed before or it could be a parallel (i.e. alternative) movable component. Removing nodes from a conjunctive graph is a straightforward procedure where no decisions have to be taken. However, inserting nodes efficiently is challenging since a meaningful and feasible insertion position has to be found. An algorithm to determine insertion positions for nodes is described in the following section which will then be used in section 4.3.2 to define an integrated route-graph-aware move.

4.3.1 Efficient Node Insertions

Nodes insertions are essential for the moves of our meta-heuristic algorithms as well as for our construction algorithm. In this section, it is not necessary to distinguish between these applications. Let us consider a conjunctive graph where the nodes from one movable component either have been removed or have not been inserted yet. In this situation, we want to insert an unscheduled movable component (i.e., a sequence of adjacent operations of the same job). This section aims at finding meaningful and feasible insertion positions for one individual node. Partial solutions will occur in between since a movable component is inserted by performing a sequence of independent individual node insertions. Partial solutions are infeasible in the sense of the problem description since the route selection must consist of valid paths for all given route graphs. Note that since also operations linked by resource acquisition constraints are subsequently inserted, resource acquisition constraints may be unsatisfied in partial solutions. When the first operation of a resource acquisition constraint has been inserted, its insertion position strongly constrains the insertion position of the second operation.

Let us consider a conjunctive graph $G = (V, E)$ with a route selection $R' \subset V$. We want to insert a currently unscheduled node[1] $v \in O \setminus R$, i.e., a node that is disconnected in the sense that $|in(v)| = |out(v)| = 0$. After inserting v, a new route selection $R = R' \cup \{v\}$ is obtained. In the following, we apply and adapt results of Kis (2003) in order to obtain a suitable method for efficiently inserting nodes. Results of Kis (2003) are restated here for completeness and in order to adapt them to the problem at hand. We start by defining node insertion positions in detail.

Insertion Position In order to introduce node insertion positions, we first introduce a *resource insertion position* as a pair $((u, w), m) \in E \times M$ that specifies for a resource m that a node should be inserted between the nodes u and w. For notational convenience, let us assume that both the artificial start node $0 \in V$ and the artificial end node $* \in V$ require all resources, i.e. $M_0 = M_* = M$. This provides a consistent notation since both artificial nodes are involved in all resource paths of the conjunctive graph. Since one operation can require multiple resources, a node insertion position is composed of a tuple of resource insertion positions: For a node $v \in O_j$ with a set of resource requirements $M_v \subset M$, a *node insertion position* specifies a tuple $\pi \in (E \times M)^{|M_v|}$ of $|M_v|$ resource insertion positions. Each resource insertion position $((u_k, w_k), m_k) \in E \times M$ of π must refer to a resource edge of m_k, i.e., $m_k \in M_{u_k}$ and $m_k \in M_{w_k}$. Resource insertion positions must be specified for all resources of the node v, so we require that $\bigcup_{k=1}^{|M_v|} m_k = M_v$.

Node Insertion A node $o_{i,j} \in O_j$ is inserted at a node insertion position $\pi \in (E \times M)^{|M_{i,j}|}$ as follows: For each resource insertion position $((u, w), m)$ of π, the edge (u, w) in the resource path of m is replaced by two edges $(u, o_{i,j})$ and $(o_{i,j}, w)$. The *route predecessor node* $o_{i-1,j} \in O_j \cup \{0\}$ and the *route successor node* $o_{i+1,j} \in O_j \cup \{*\}$ of the node $o_{i,j}$ can be determined from the given route graph. The existing route edge $(o_{i-1,j}, o_{i+1,j}) \in E$ is replaced by two edges $(o_{i-1,j}, o_{i,j})$ and $(o_{i,j}, o_{i+1,j})$. A *node insertion* is called *acyclic* if it creates no cycles in the conjunctive graph. An acyclic node insertion is called *feasible* if no resource acquisition constraint is violated in the resulting conjunctive graph. Note that this creates a difference in notation between this work and Kis (2003): Our definition of "feasible" requires in addition that all resource acquisition constraints are taken into account. "Acyclic" in our notation corresponds to "feasible" in the notation of Kis (2003).

Determining Insertion Positions Having provided all necessary preliminaries, we now want to identify reasonable insertion positions for an unscheduled operation $v \in O \setminus R$. For a resource $m \in M$, let us denote by $R_m \subset R$ the set of all scheduled operations which require resource m. A node insertion position can be considered as selecting, for each $m \in M_v$, one node from R_m after which the node v should be inserted. Overall, including infeasible ones, there are $\prod_{m \in M_v} |R_m|$ different possibilities for choosing an insertion position. Clearly,

[1] Note that throughout this section, we denote the node to be inserted by either $v \in O$ or $o_{i,j} \in O$, depending on which notation is more convenient. We use $o_{i,j}$ if its route predecessor and successor nodes $o_{i-1,j}$ and $o_{i+1,j}$ need to be referred. Otherwise, denoting the node by v facilitates readability.

it would be too time consuming to enumerate all of them. Thus, we aim at reducing this set as much as possible without missing reasonable insertion positions. The approach presented subsequently directly follows the one given in Kis (2003) which allows the efficient identification of feasible node insertion positions while implicitly avoiding some dominated insertion positions (dominated in the sense of makespan). The approach of Kis (2003) is generalized here to consider resource acquisition constraints.

All operations that share resources with the operation v to be inserted are denoted by

$$\mathcal{Q}_v := \bigcup_{m \in M_v} R_m.$$

An insertion position for an operation v can be expressed as a partitioning $H \subset \mathcal{Q}_v$. After inserting v into the graph G at the insertion position H (i.e. directly after H), there exist paths from each node in H to v, and conversely, paths from v to each node in $\mathcal{Q}_v \setminus H$.

When inserting an operation $v = o_{i,j} \in O$, no cycles must be introduced and resource acquisition constraints must be respected. To include these constraints, we start by considering all nodes that must be directly adjacent to $o_{i,j}$ after its insertion. For this, Kis (2003) introduces the sets $pred(v) \subset V$ and $succ(v) \subset V$ which contain all route predecessors and route successors in so called *and-subgraphs*. And-subgraphs allow general precedence relations to be considered, e.g., as appearing in resource constrained project scheduling problems. Though and-subgraphs are not considered in our case, we also use the sets $pred(v)$ and $succ(v)$ for referring direct predecessor and successor nodes.

Apparently, the set $pred(v)$ contains the route predecessor $o_{i-1,j} \in V$ of $o_{i,j}$, and the set $succ(v)$ contains the route successor $o_{i+1,j} \in V$ of $o_{i,j}$. Beside these route predecessor and successor operations, additional nodes are added to $prec(v)$ and $succ(v)$ in order to cope with resource acquisition constraints. Recall that $v \in O \setminus R$ since v is not scheduled yet. Each resource acquisition constraint can be represented as a tuple $(u, w, m) \in O \times O \times M$ in the sense that resource m is acquired at operation u and released at operation w. Using this notation, nodes are added to $pred(v)$ as follows: For each resource acquisition constraint of the form (v, w, m) with $w \in R$, the resource predecessor $u \in m^{-1}(w)$ of the scheduled operation w on resource m is added to $pred(v)$. The intention of this is to schedule v directly after operation u on resource m. The insertion of operation v before w is taken into account by adding $o_{i+1,j}$ to $succ(v)$ since w is equal to or follows after $o_{i+1,j}$ in the route graph of their job. Thus, v will be scheduled between u and w on m. Conversely, for each resource acquisition constraint of the form (u, v, m) with $u \in R$, the resource successor $w \in m(u)$ of operation u on resource m is added to $succ(v)$. Denoting the set of all resource acquisition constraints by $A \subset O \times O \times M$, this leads to the following formal definitions:

$$pred(v) := \left\{o_{i-1,j}\right\} \cup \left\{u \in V \;\middle|\; \exists\, w \in R\ \exists\, m \in M \text{ with } (v, w, m) \in A, u \in m^{-1}(w), m \in M_u\right\}.$$

$$succ(v) := \left\{o_{i+1,j}\right\} \cup \left\{w \in V \mid \exists\, u \in R\ \exists\, m \in M \text{ with } (u, v, m) \in A, w \in m(u), m \in M_w\right\}.$$

To facilitate notation, a *reachability relation* $\prec \subset V \times V$ is defined which contains $(u, w) \in V \times V$ if and only if there exists a path from u to w in the conjunctive graph G. The relation is reflexive, i.e., $u \prec u$ holds $\forall\ u \in V$.

Using this definition, we introduce the sets

$$P_v = \{u \in \Omega_v \mid u \prec pred(v)\}$$

and

$$S_v = \{w \in \Omega_v \mid succ(v) \prec w\}.$$

Note that $P_v \cap S_v = \emptyset$ since G is acyclic. Based on the preceding definitions, let us recall the following results of Kis (2003) that can be applied to the problem at hand.

Definition 4.3. *Kis (2003) A subset H of Ω_v is called a* prefix *of $v \in V$ if and only if*

1. *$P_v \subseteq H$ and $H \cap S_v = \emptyset$, and*
2. *$x \in \Omega_v$, $y \in H$, $x \prec y \Rightarrow x \in H$.*

Lemma 4.1. *Kis (2003) $H \subset \Omega_v$, $v \in V$, H is an acyclic insertion $\Leftrightarrow$ H is a prefix of v.*

The proof of Lemma 4.1 given in Kis (2003) remains valid in the situation at hand since it solely brings forward arguments referring to the structure of the conjunctive graph without relying on edge weights or longest path distances (which are different in our case). In the following, we define maximum prefixes based on the work of Kis (2003). The definitions given there have been adapted to the problem at hand, in particular to take into account that outgoing edges of the same node now can have unequal edge weights. Let us beforehand introduce the following definitions. The maximum weight of all outgoing edges of a node $u \in V$ is denoted by $l_u = \max_{(u,w) \in E} l_{u,w}$ and the *length of a longest path* between two nodes $u, w \in V$ is denoted by $L(u, w)$.

Definition 4.4. *For a node $v \in V$, its* head *(or* earliest start date*) is denoted by*

$$h_v = L(0, v).$$

Definition 4.5. *The* headspan *of a set of operations $H \subset \Omega_v$ is defined as*

$$h(H) := \max\{h_u + l_u \mid u \in \Omega_v \text{ s.t. } \exists\, w \in H \text{ with } u \prec w\}.$$

Definition 4.6. *Kis (2003) A subset H of Ω_v is called a* maximal prefix *of $v \in V$ if and only if*

1. *$P_v \subseteq H$ and $H \cap S_v = \emptyset$, and*
2. *H contains all operations $x \in \Omega_v \setminus S_v$ with $h_x + l_x \leq h(H)$.*

Lemma 4.2. *Kis (2003) $H \subset \Omega_v$, $v \in V$, H is a maximal prefix $\Rightarrow$ H is a prefix of v.*

Proof. Let H be a maximal prefix and let $x \in \Omega_v, y \in H$ be two operations with $x \prec y$. It follows $h_x + l_x \leq h(H)$ by definition of $h(H)$ and thus $x \in H$. $P_v \subseteq H$ and $H \cap S_v = \emptyset$ hold by definition. □

Definition 4.5 has been adapted to the varying edge weights appearing in the problem at hand. Due to this modified definition, Lemma 4.2, proven by Kis (2003) in its original form, had to be proven again. Note that the conditions $P_v \subseteq H$ and $H \cap S_v = \emptyset$, mean that H lies between P_v and $\Omega_v \setminus S_v$ in the sense that $P_v \subseteq H \subset (\Omega_v \setminus S_v)$. Let us restate in the following two additional results of Kis (2003).

Lemma 4.3. *Kis (2003) Let H and H' be maximal prefixes. The following properties hold:*

1. *$H \subseteq H'$ if and only if $h(H) \leq h(H')$.*
2. *$H = H'$ if and only if $h(H) = h(H')$.*

Proof. The proof follows directly the one given in Kis (2003) (p_x is replaced by l_x).

1. First suppose $H \subseteq H'$. Then $h(H) \leq h(H')$ follows by definition. Conversely, suppose $h(H) \leq h(H')$. Let x be any operation in H. Since $h_x + l_x \leq h(H) \leq h(H')$, it holds that $x \in H'$.
2. Apply the first part to $h(H) \leq h(H')$ and to $h(H') \leq h(H)$, the statement follows.

□

Corollary 4.1. *Kis (2003) Maximal prefixes are nested, i.e. whenever $H \neq H'$ are two distinct maximal prefixes, either $H \subset H'$ or $H' \subset H$ holds.*

Maximum prefixes so far do not take resource acquisition constraints into account. Node insertion positions need to be chosen such that scheduled resource acquisition constraints remain satisfied. For this purpose, the following definition is introduced.

Definition 4.7. *A subset H of $\mathcal{Q}_v$ is called an* acquisition-aware maximal prefix *of $v \in V$ if and only if*

1. *H is a maximal prefix of $v \in V$, and*
2. *for each scheduled resource acquiring operation $v_a \in R$ that has a corresponding scheduled release operation $v_b \in R$, it holds that $v_a \in H \Leftrightarrow v_b \in H$.*

Note that acquisition-aware maximal prefixes determine feasible node insertions positions since they are maximal prefixes by definition. For constant edge weights used in Kis (2003), it can be shown (see Lemma 3 of Kis (2003)) that always an optimal insertion position can be found. An interesting research direction is to generalize these findings. Such a generalization would need to take into account that one node can have edge weights with unequal weights. In addition, edge weights might change after a node insertion since in our batch-oblivious approach edge weights can become zero if adjacent operations are processed in a common batch.

Algorithm Considering only acquisition-aware maximal prefixes for inserting nodes largely reduces the number of insertion positions to be explored. The number of insertion positions to be considered for a node is given by the number of acquisition-aware maximum prefixes

$$\left|\{h \in \mathbb{Z} \mid \exists \text{ maximum prefix } H : h(H) = h\}\right|.$$

Algorithm 4.3 Compute a set of insertion positions for a node $v \in G$

calculateInsertions(v)
  $I \leftarrow \emptyset$
  $P_v \leftarrow \mathcal{Q}_v \cap$ calculateReachableNodesUsingReverseBFS($pred(v)$)
  $S_v \leftarrow \mathcal{Q}_v \cap$ calculateReachableNodesUsingForwardBFS($succ(v)$)
  $H \leftarrow P_v$
  $L \leftarrow h(H)$
  while $\mathcal{Q}_v \setminus (S_v \cup H) \neq \emptyset$
    $A_0 \leftarrow \emptyset$
    do
      $H \leftarrow H \cup A_0$
      $L \leftarrow \max(L, h(H))$

      // restore max. prefix
      $H \leftarrow H \cup \{w \in \mathcal{Q}_v \setminus (S_v \cup H) \mid \exists\, u\ \mathcal{Q}_v$ with $u \succ w$ such that $h_u + l_u \leq L\}$

      // resource releases that with corresponding acquisitions in H
      $A_0 \leftarrow \{w \in \mathcal{Q}_v \setminus H \mid \exists\, u \in H, \exists\, m \in M$ s.t. $(u, w, m) \in A\}$
    while $A_0 \neq \emptyset$
    $I \leftarrow I \cup \{H\}$
    $L \leftarrow \min_{u \in \mathcal{Q}_v \setminus (S_v \cup H)} (h_u + l_u)$
  return I

This number is bounded by the number of nodes contained in the set $\mathcal{Q}_v \setminus (P_v \cup S_v)$. Kis (2003) shows that an algorithm can be provided for iterating all maximum prefixes with a runtime of $(|\mathcal{Q}_v \setminus (P_v \cup S_v)| \log_2(|M_v|))$.

In the following, we present an adapted version of the algorithm of Kis (2003) which returns all acquisition-aware maximal prefixes. It can be implemented by representing the current insertion position H as an array of $|M_v|$ node references. A node reference can be represented as an integer providing the index of the node. P_v and S_v can also be represented in this way. Each element of such an array corresponds to one resource $m \in M_v$ and refers to the final node in the sequencing R_m that belongs to H. These node references can be successively advanced during iteration. Initially, the insertion position is set to P_v and it is advanced until S_v is reached. Each step of the algorithm must make sure that H actually is an acquisition-aware maximal prefix: If an operation is added to H that acquires a resource, the corresponding release operation is also added in order to avoid inserting a node in between the two. The pseudo-code for this procedure is given in Algorithm 4.3. The complexity of iterating all maximum prefixes could remain $(|\mathcal{Q}_v \setminus (P_v \cup S_v)| \log_2(|M_v|))$, though we use an implementation with a complexity of $(|\mathcal{Q}_v \setminus (P_v \cup S_v)| |M_v|)$ since we assume the maximum number of resources per operation to be very small. A forward and a backward

breadth-first search (BFS) starting from v is needed to determine P_v and S_v, which dominates the complexity. This leads to an overall complexity of $O(|E|)$ for the algorithm. Note that Kis (2003) proposes further improvements to reduce this runtime by avoiding breadth-first searches which are not included in our work. We believe exploring their inclusion is also a promising direction for future research.

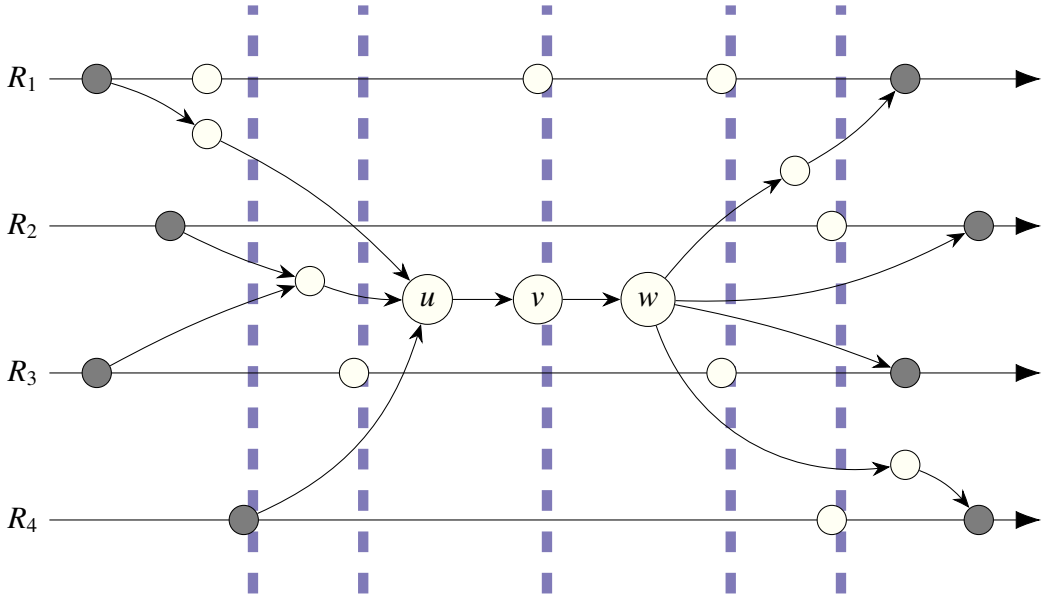


Figure 4.3 – *Insertion positions for a node v which requires 4 different resources*

Let us summarize again in the following why resource acquisition constraints are correctly handled by our approach. Resource acquisition aware node insertions need to take care of two things: First, all resource acquisition constraints in which the node to be inserted is directly involved must be respected. This was taken care of by adapting the sets $prec(v)$ and $succ(v)$ accordingly. Second, all correctly scheduled resource acquisition constraints must remain intact. This is taken care of by using acquisition-aware maximum prefixes defined in Definition 4.7 for insertion positions. This avoids node insertions between two operations that are linked by a resource acquisition constraint.

Figure 4.3 provides an example illustrating the search for insertion positions for a node $v \in O$ that requires four resources. The figure illustrates the elements of a conjunctive graph that are relevant for finding insertion positions—so, most parts of the conjunctive graph are omitted. R_1, R_2, R_3, and R_4 indicate the resource paths in the conjunctive graph of all resources required by the node to be inserted. The route predecessor of v is denoted by $u \in V$. The route successor of v is denoted by $w \in V$. The positioning of the nodes on the x-axis indicates the calculated headspan of each node. Thick dashed lines indicate the five insertion positions corresponding to maximal prefixes. Grey nodes indicate the sets P_v

and S_v which provide the boundaries within which all feasible insertion positions are located. The intuition is that, H is scanning this range linearly from left to right, starting behind P_v and ending before S_v.

Finally, let us summarize the contents of this section. We have presented an algorithm for inserting nodes into conjunctive graphs by adapting a method which is proposed in Kis (2003). It is shown that the proposed insertion method can be extended to take resource acquisitions into account.

4.3.2 Heuristic Methods

In this section, the elements introduced so far for extended route and resource flexibility in job-shop scheduling are combined in order to apply and adapt the heuristic approach presented in section 3.4. Based on the generalized conjunctive graph representation presented in section 4.2 and the method for efficiently determining node insertion positions presented in the preceding section 4.3.1, we introduce a neighborhood that can be applied in meta-heuristic solution approaches. We describe in the following the adaptation of our GRASP based approach.

Route-Graph-Aware Moves

The move introduced in this section is based on the movable components of a route graph which are defined in section 4.1.4. The move works in two phases: First, all nodes belonging to a currently scheduled movable component are removed. Second, all nodes belonging to a movable component are inserted. The movable component that is inserted has to be an alternative for (or to be identical to) the movable component that was removed. Formally, a *route-graph-aware move* is a tuple (O^-, O^+, Π) composed of a sequence of nodes $O^- = \left(o_{i,j}, \ldots, o_{k,j}\right)$ to be removed, a sequence of nodes $O^+ = \left(o_{f,j}, \ldots, o_{g,j}\right)$ to be inserted, and a sequence of insertion positions $\Pi = \left(\pi_f, \ldots, \pi_g\right)$. Initially, only the nodes to be removed have to be scheduled, so $O^- \subset R$ is required. Depending on whether the move describes a resequencing or a reassignment, either $O^- = O^+$ or $O^+ \subset V \setminus R$ has to be fulfilled. Both node sequences have to be proper movable components. Thus, $o_{i-1,j}$, $o_{k+1,j}$, $o_{f-1,j}$, and $o_{g+1,j}$ have to be separator nodes. Neither O^- nor O^+ must contain a separator node. We require that both sets O^- and O^+ are connected subgraphs (i.e. paths) of the route graph G_j.

A move is performed by first removing all nodes in O^- and then inserting the nodes O^+ (as described in section 4.3.1) at their corresponding insertion positions in Π. The nodes in O^- are removed by replacing their adjacent edges by edges which bride corresponding predecessor and successor operations. Each node to be inserted has a corresponding insertion position. Node that the sequencing of insertions is relevant; Each insertion position of a move refers to the graph obtained by executing all preceding node insertions.

Note that we allow $O^- = O^+$ to obtain a unified move which includes both resequencing moves (where the route selection remains unchanged) and reassignment moves (where a different path in the route graph is chosen, usually requiring different resources). It is also

allowed that O^- is empty. This allows the construction heuristic to be expressed in terms of route-graph-aware moves. Recall that, during the execution of route-graph-aware moves, infeasible schedules appear as long as modifications of route selection are incomplete. Thus, in the sense of a neighborhood, all modifications which are described by a single move are considered as one elementary change.

For given sets of nodes $O^- \subset R$ and $O^+ \subset O$ to be removed and inserted, the set of feasible insertion positions is determined as follows. First, all nodes in $O^- \subset R$ are removed. Then, all possible insertion positions for the nodes in O^+ are determined by repeatedly applying Algorithm 4.3. This is implemented using a variant of an algorithm for calculating the cartesian product of a given list of sets. If, as in our case, movable components contain only very few nodes, this simplistic approach is practicable. Note that the first node insertion can strongly reduce the number of possible insertion positions for the remaining operations if they share common resources or if resource acquisition constraints are involved. The combined insertion of a fixed sequence of operations has already been considered in the literature by Kis and Hertz (2003), Gröflin and Klinkert (2007) and Gröflin et al. (2008). It is an interesting direction of future research to incorporate ideas from these papers in order to consider multiple insertions at the same time in a more efficient manner.

Heuristic Algorithms

In this section, we adapt the solution approach presented in section 3.4. The approach is still based on the idea of Greedy Randomized Adaptive Search Procedures (*GRASP*) of Feo and Resende (1995): Many different starting solutions are created by randomizing a construction algorithm. Each solution is then independently improved using Simulated Annealing. Solutions are represented using generalized conjunctive graphs as presented in section 4.2. Route-graph-aware moves based on the method for inserting nodes presented in section 4.3.1 are used for defining a neighborhood to be used within Simulated Annealing.

In the *construction heuristic*, again, jobs are first sorted using an objective function dependent sorting criterion as in section 3.4. The heuristic then iterates the sorted list of jobs and successively inserts the operations of the current job by probing the best insertion position. To randomize this greedy construction method, the next job to be inserted is determined by randomly selecting one of the first P_i elements in the list of unscheduled jobs for a tuning parameter $P_i \geq 1$ that steers perturbation intensity.

Operations are inserted as follows. For each job, we iterate the sections between separator nodes—starting at the start node of the route graph. We determine the best insertion by probing all possible insertion positions for all movable components between the current separator nodes. Insertion positions are determined using the method presented in section 4.3.1. The best insertion is executed. Then, we continue with the section between the following pair of separator nodes. To evaluate the best insertion, we rate corresponding partially computed schedules by determining their objective functions.

Analogously to chapter 3, the *Simulated Annealing* improvement heuristic uses an integrated neighborhood which is obtained by combining route-graph-aware moves with the

adaptive algorithm for computing start dates and batching decisions. The adaptions that were made for both elements are taken into account as described in the preceding part of this section and in section 4.2. As in chapter 3, the combined result of both modifications is considered as one single move. If such a move is rejected, all involved changes are reverted collectively.

To determine a route-graph-aware move in the course of Simulated Annealing, a movable component of the current conjunctive graph is randomly chosen. If the movable component is scheduled, its nodes are removed from the graph. Otherwise, all nodes of its scheduled alternative movable component are removed from the graph. Then, the nodes of the movable component that had been chosen randomly are inserted. Therefore, all insertion positions for this movable component are calculated as mentioned before and one of them is chosen randomly. The Simulated Annealing parameters are the same as in section 3.4. A geometric cooling schedule is used: A temperature T is multiplied by a cooling factor $P_c < 1$ after each step. Moves are immediately accepted if their objective function value improves the previous objective function value. Otherwise, schedules are accepted with a probability of $exp(\frac{-f_n - f_{n-1}}{T})$. The search is stopped if the best solution did not improve during the preceding P_m iterations. The initial temperature is determined by sampling P_s random moves.

4.4 Numerical Experiments

This section presents numerical experiments to analyze the modeling presented in this chapter and to evaluate the implementation of the solution approach. One objective of this section is to justify the additional complexity introduced by the route graph based modeling. This is done by comparing, on a real-world instance, results obtained with our model to those obtained with a less complex model. In addition, we aim at evaluating the performance of our approach on instances for a scheduling problem in the photolithography work area for parallel machines with sequence-dependent setup times and auxiliary resources. The algorithms were run using the same environment as in the numerical experiments presented in section 3.5, i.e. an Intel Xeon E5-2620 2.1 GHz machine with 6 cores running Microsoft Windows 7.

4.4.1 Model Complexity

The degree of detail in our modeling induces a considerable complexity. We illustrate potential benefits of the more complex modeling using an instance from a real-world fab of our industrial partner STMicroelectronics. We compare two schedules: The first one is obtained using the job-shop scheduling model presented in this chapter, with extended route and resource flexibility. The second schedule is computed using a simpler model based on a flexible job-shop with multiple resources per operation. In the simpler model, a consistent resource usage is guaranteed neither for cooling operations including boats, nor for loading and unloading operations at load ports since only the sequencing of operations in tubes are

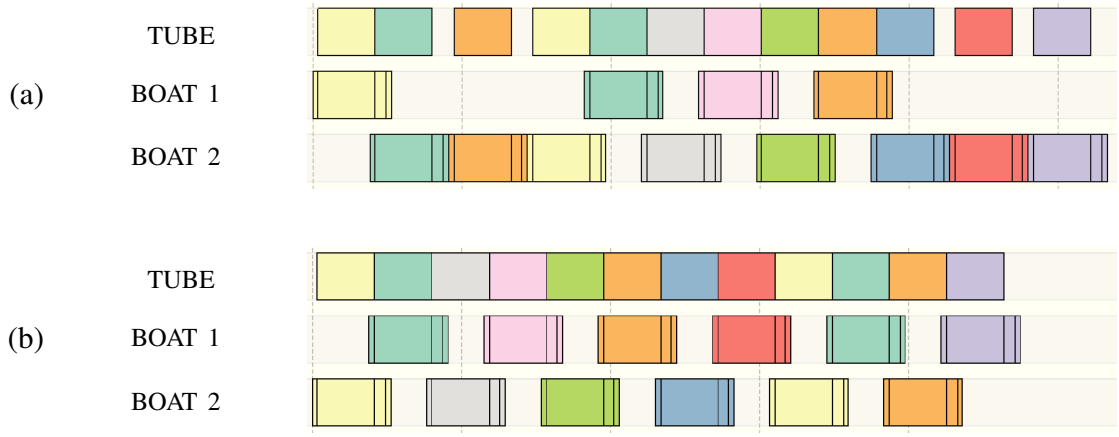


Figure 4.4 – *Comparison of schedules obtained by different models*

determined and all other resources are neglected. Consequently, the computed start times of operations may differ from those obtained by the first model. We simulate the performance of the sequencing determined by the simpler model in the extended model by using the extended model to compute a schedule with the same sequencing of operations—including loading, cooling and unloading. Note that batching is not considered in this evaluation.

Figure 4.4 compares the schedules obtained by the two approaches. This extract of a schedule visualizes the jobs scheduled on a machine with one tube and two boats. Schedule (a) is obtained by the simpler model, Schedule (b) by the model with extended route flexibility. The makespan of the schedule corresponding to the simpler model is 74 hours which is significantly larger than the makespan of 66 hours obtained with the extended model. This difference stems from idle periods in the usage of the tube in the simpler model. Those idle periods are caused by the inefficient assignment of loading, cooling, and unloading operations to boats. This is because the simpler model neglects those operations.

Note that here the alternating usage of boats is most efficient. However, we do not see how this observation could lead to a simpler model. One reason is that a job could be processable by only one of two boats. This example illustrates the relevance of considering internal components of complex machines for scheduling operations in the diffusion and cleaning area of a semiconductor manufacturing facility.

4.4.2 Photolithography Instances

Photolithography tools often are the most expensive machines in semiconductor manufacturing facilities and therefore constitute bottlenecks in the fab. For processing an operation on such a machine, masks are required as auxiliary operations. Transport durations given by a travel time matrix have to be considered since masks must be transported between machines. Additionally, photolithography tools involve sequence-dependent times due to temperature changes. The abilities to consider multiple resources per operation and sequence-dependent

setup families that are given independently for each resource allow us to tackle such problems. Two resources are required for each operation: The photolithography tool and an auxiliary resource which is called mask or reticle. Both types of resources can be modeled by defining appropriate sequence-dependent setup times. Setup families reflect reticle locations (i.e. photolithography tools) and recipes used at individual tools.

The scheduling problem with auxiliary resources as presented in Bitar et al. (2016) can be solved by our approach. Bitar et al. (2016) present a memetic algorithm for this problem. A comparison of numerical results is given in Table 4.1. Columns "$\overline{\sum_{j\in J} w_j C_j}$" provide average total weighted tardiness values over 10 instances and columns "Time (s)" provide the average runtime for individual instances. We use a runtime of 2 minutes per instance for all instances. The results show that we obtain very competitive results which improve the ones given in Bitar et al. (2016). This shows that the algorithm for node insertions based on the results of Kis (2003) can efficiently handle multiple resources per operation and the presented generalizations of our approach work well.

	Bitar et al. (2016)		GRASP	
$\|J\|$	$\overline{\sum_{j\in J} w_j C_j}$	Time (s)	$\overline{\sum_{j\in J} w_j C_j}$	Time (s)
10	**156**	16	**156**	120
20	526	29	**524**	120
30	746	60	**730**	120
40	1385	130	**1308**	120
50	**1840**	389	1857	120
70	2462	490	**2387**	120
90	**3743**	532	3750	120
100	4134	845	**3822**	120
120	5861	417	**5300**	120
150	7125	850	**6416**	120
200	8755	855	**7379**	120

Table 4.1 – *Results for the instance of Bitar et al. (2016)*

4.5 Conclusion

In this chapter, a job-shop scheduling problem with extended route flexibility was introduced to take internal components of machines in semiconductor manufacturing facilities and their properties into account. We use the concept of extended route flexibility to be able to ensure that different subsequent operations use the same resource. This is extended by introducing

resource acquisition constraints that can exclusively reserve a resource between two operations. We are not aware of any other solution approaches that model resource acquisition constraints. In addition, the approach can handle multiple resources per operation which is tackled only by few approaches in the literature. Our approach obtains better results on industrial instances than simpler modeling approaches. An interesting direction for future research is the application of the approach to a wider range of instances and areas beyond semiconductor manufacturing. Resource acquisition constraints appear, for example, in the scheduling of railway maintenance operations (Ramond et al. (2006)).

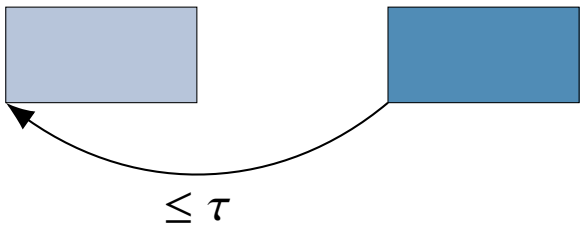
We have presented a GRASP based meta-heuristic based on a neighborhood which is induced by an integrated route-graph-aware move. We expect that our results can be improved by incorporating properties that are already exploited by established methods for the flexible job-shop scheduling problem. Also, studying other meta-heuristics such as tabu search, variable neighborhood search or genetic algorithms in combination with the main building blocks of our approach seems to be interesting.

In this chapter, we evaluated instances for which the makespan or total weighted completion time were minimized. However, many other objectives are more important in semiconductor manufacturing. So, it would be useful to consider them as well and optimize multiple criteria. A very important property of machines used in the diffusion area is their batching capability: They can process multiple lots of wafers at the same time. This can only be taken into account in the approach if the length of a movable component in the route graph has exactly length one. A very interesting generalization of the problem can be obtained by relaxing this limitation. Modeling and solving this is a challenging direction for future research. Additionally, temporal constraints, in particular maximum time lags, play an important role in the diffusion and cleaning area. The approach is extended in the following chapter in order to take them into account.

Chapter 5

Time Constraints in Complex Job-Shop Scheduling

MAXIMUM time lag constraints limit the total waiting and processing time between two operations of the same job. With the reduction in semiconductor technology scale, "the ability to handle this complexity efficiently becomes more important to overall fab performance." (Jung et al. (2013))



The preceding parts of this thesis concentrated on the modeling of various machine properties integrated within a job-shop environment. Chapter 3 presents a complex job-shop scheduling problem including batching machines and sequence-dependent setup times. In order to model machines in more detail, chapter 4 generalizes the problem by adding flexibility for the route of a job while allowing multiple resources per operation. This chapter aims to further extend the problem by including maximum time lag constraints. Maximum time lag constraints are an important aspect in the diffusion and cleaning area and crucial for schedules to be applicable in practice. In particular, interleaved maximum time lag constraints need to be observed. This is becoming increasingly important with the shrinking structural sizes of semiconducting devices. The particular importance of this in the diffusion area is highlighted in Jung et al. (2013), where the authors claim that with the reduction in semiconductor technology scale, "the ability to handle this complexity efficiently becomes more important to overall fab performance."

A review of the literature on maximum time lag constraints in related scheduling problems is presented in section 1.4.3. Properties of the considered maximum time lags in the industrial context are described in the problem specification of chapter 2. In particular, section 2.3.1 proposes to include time lags as soft constraints. We continue the discussion on the modeling of maximum time lags in section 5.1 and extend our formal problem description in section 5.2. We adapt our solution approach in section 5.3 and present related numerical experiments in section 5.4. A brief version of the contents of this chapter is presented in Knopp et al. (2016).

5.1 Modeling Time Constraints

Chemical and physical processes in the diffusion and cleaning work area impose *maximum time lag* constraints that limit the waiting and processing time between two operations of the same job. Often, time lags start after cleaning processes where the chemical conditions on the wafer surface deteriorate over time. Maximum time lags can be adjacent or overlapping which implies that starting a time lag might implicitly trigger another one. This interleaving of time lags is also called "queue time constraints" in the literature or "time constraint tunnels" in industrial terminology. Regarding the classification of maximum time lag constraints presented in Klemmt and Mönch (2012), the maximum time lags that we consider in this chapter belong to the most general class that allows overlapping of time lags as well as time lags between non-adjacent operations.

As defined in the industrial specification in chapter 2, we distinguish between *reworkable* and *non-reworkable* maximum time lags. In case a *reworkable* maximum time lag is violated, the lot needs to be reworked. We want to avoid reworking as much as possible, since it increases the cycle time of the concerned lot and requires additional machine capacity. In case a *non-reworkable* maximum time lag is violated, the risk that wafers contained in this lot are defective increases with the duration of the time lag violation. Additional measurements need to be performed after violations of non-reworkable maximum time lags in order

to evaluate wafer quality. Depending on the results of these measurements, wafers might be scrapped. Scrapping wafers is very expensive, not only due to the loss of the involved material, but also due to the uselessness of the afore conducted processing steps and delayed product completion times.

In practice, the scheduler is used in a rolling horizon setting. So, when a schedule is computed at a certain point in time, several operations of a job might have already been completed, one could be currently running, and the remaining ones still have to be scheduled. Now, consider a maximum time lag whose beginning refers to an operation that has started in the past and whose ending refers to an operation to be scheduled. Let us call such maximum time lags *initiated time lags*. Jobs without initiated time lags can always be scheduled without any time lag violation since its operations can be arbitrarily postponed. The canonical schedules introduced in Caumond et al. (2008) suggest a trivial method to schedule non-initiated maximum time lags in a feasible way. An initiated time lag however imposes a fixed due date for its end operation. Since maximum time lags can be adjacent or overlapping, it is possible that also the operations of some non-initiated time lags cannot be indefinitely delayed. Hence, we cannot guarantee that maximum time lag constraints related to ongoing jobs can always be satisfied. Though in practice, scheduling decisions are always needed—even if maximum time lag constraints cannot be satisfied. Therefore, we include maximum time lags as soft constraints in the sense that we minimize maximum time lag violations as our primary objective in a lexicographical objective function.

Since we want to minimize time lag violations, we need to quantify the time lag violations for a given schedule. An initial discussion is given in the industrial specification of the objective function in section 2.3. For each time lag, a violation severity is introduced. This is a real number that is zero in case the time lag is satisfied, and greater than zero in case the time lag is violated. The *total maximum time lag violation severity* for a schedule is the sum of the violation severities over all time lags. This sum then makes up the first component of our lexicographical objective function. We have found a feasible schedule if the total maximum time lag violation severity is zero.

If a maximum time lag constraint is violated, the corresponding violation severity depends on the reworkability of the time lag and the duration of the delay. Since a lot needs to be reworked once a reworkable maximum time lag is violated, independently of the duration of its delay, a constant rework cost is assigned to violated reworkable maximum time lags. In contrast, the duration of the delay is important for non-reworkable maximum time lags: The probability that wafers must be scrapped increases with the duration of the maximum time lag violation. Thus, long delays should be avoided at all times, while smaller ones can rather be tolerated. Though they propose a linear violation severity, Kohn et al. (2013) describe the same reasoning in an example with two lots, where they argue that two medium delays are favored over one small and one large delay. We propose to penalize delays quadratically, which is a straightforward way to include the cases above. This approach is similar to the method of least squares which is a standard approach in regression analysis and dates back to at least Legendre (1805). However, once a maximum time lag delay gets too large, all wafers contained in the lot most certainly have to be scrapped. Therefore, we bound the violation

cost imposed by a single time lag by introducing a scrap cost for a lot. This is the maximum cost that is applied once the violation duration is larger than the given limit. Specifying the maximum time lag violation severity in this way can be viewed as an estimation of the induced yield loss. Note that it would not make sense to impose a scrap cost more than once for the same lot. However, we have omitted this in our definition of the time lag violation severity for the sake of simplicity. In section 5.2, we provide a generalized formulation that defines the violation severity for both types of maximum time lags in a uniform way.

An additional observation is the following. For schedules with violated time lags, we know beforehand that some lots have to be reworked or scrapped. Knowing this, it makes no sense to further continue their processing and it could be reasonable to consider this directly during the calculation of schedules. The removal of supposedly scrapped lots would free capacities on related machines and thus could also affect the evaluation of other objective functions. Some of these effects are partially taken into account in our approach. Since the delay cost for a single time lag constraint is always bound to a maximum, violated operations can be delayed to the very end in order to fulfill other maximum time lag constraints. In practice, directly proposing to scrap valuable lots might hamper the acceptance of a decision making tool.

In addition to the discussed maximum time lag constraints, *minimum time lag constraints* can be given that impose a minimum duration between two operations. They can be included in a much simpler way and do not introduce infeasible solutions. In our case, we restrict minimum time lags to adjacent operations. Therefore, they can already be handled by adding an additional operation between the two concerned operations that has the minimum time lag duration as its processing duration and does not require any resources.

5.2 An Extension of the Formal Problem Description

This formal problem description extends that of section 4.1 and uses the same notation given there. Since the problem described in section 4.1 generalizes that of section 3.1, the extended problem given here is the most general scheduling problem considered in this thesis. This extension introduces an additional objective function but does not modify any constraints.

Recall that the set of operations for a job $j \in J$ is denoted as $O_j = \left\{o_{1,j}, \ldots, o_{|O_j|,j}\right\}$. In addition, let us consider a given set of maximum time lag constraints $T \subset J \times \mathbb{Z}^4 \times \mathbb{R}_{>0}$. The components of a time lag $(j, k, l, d, \gamma, c) \in T$ have the following meaning: $j \in J$ identifies the job; $k, l \in \mathbb{N}$ with $1 \leq k < l \leq \left|O_j\right|$ identify separator operations $o_{k,j}$ and $o_{l,j} \in O_j$; $d \in \mathbb{N}_{\geq 0}$ identifies a maximum time lag duration; $\gamma \in \mathbb{N}_{\geq 0}$ with $\gamma \geq d$ identifies an ultimate time lag duration; and $c \in \mathbb{R}_{>0}$ identifies the cost of a time lag violation. Now consider a feasible schedule with start dates $S_{i,j} \in \mathbb{N}$ given for all scheduled operations $o_{i,j} \in O$. For each time lag $\tau = (j, k, l, d, \gamma, c) \in T$, with a delay $L_\tau = S_{l,j} - S_{k,j}$, its *violation severity* is defined as

$$V_\tau = \begin{cases} 0 & \text{if } L_\tau \leq d, \\ c & \text{if } L_\tau > \gamma, \\ \frac{(L_\tau - d)^2}{(\gamma - d)^2} \cdot c & \text{else.} \end{cases}$$

Note that, in cases where the initial operation of a time lag refers to an operation that has started its processing in the past, we allow $k \leq 0$ and assume for notational consistency that (though the operation is not part of the considered scheduling problem) its start date is still given by $S_{k,j}$. Overall, the *total maximum time lag violation severity* to minimize is defined as

$$V = \sum_{\tau \in T} V_\tau.$$

This definition comprises the cases described in section 5.1. For reworkable lots we have $d = \gamma$, which applies a constant violation cost regardless of the duration of the delay. For non-reworkable lots we have $d < \gamma$, which applies a cost that increases quadratically with the duration of the delay as long as the delay does not exceed the ultimate duration which reflects the certain scrapping of the lot. Note that this objective function is not regular, since advancing an operation that starts a maximum time lag could increase the total maximum time lag violation severity. Figure 5.1 illustrates the time lag violation cost for both cases.

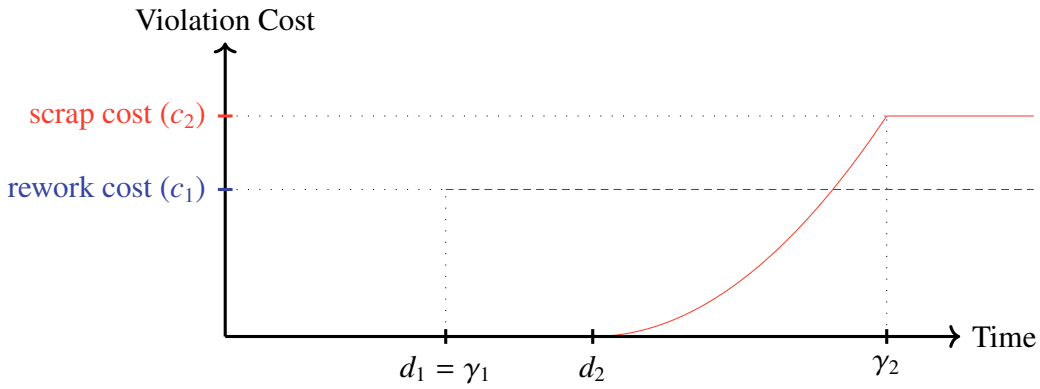


Figure 5.1 – *Time lag violation cost for a reworkable time lag τ_1 and a non-workable time lag τ_2*

Note that we require time lags to refer to separator operations (which are defined in section 4.1.1) as part of the route graph modeling. Since separator operations must be scheduled in every feasible schedule, we avoid referring to unscheduled operations. This restriction comes with a caveat. Time lags starting at a separator operation constrain the start date of its succeeding operation, whereas time lags ending at a separator operation constraint the start date of its preceding operation. Thus, referencing separator operations might couple time lags that are actually independent. To tackle this, we can introduce adjacent separator operations. One is meant to represent the end of its preceding operation, i.e. its start date is the completion date of its route predecessor operation. The other one is meant to represent the beginning of its following operation, i.e. its start date is the start date of its route successor operation. This representation can handle the conflict without introducing additional modeling complexity. Figure 5.2 shows an example for both cases. The first graph includes a separator node c where the time lag τ_1 ends and the time lag τ_2 starts. In the second graph, both time lags are independent due to the introduction of two separator nodes c and c'.

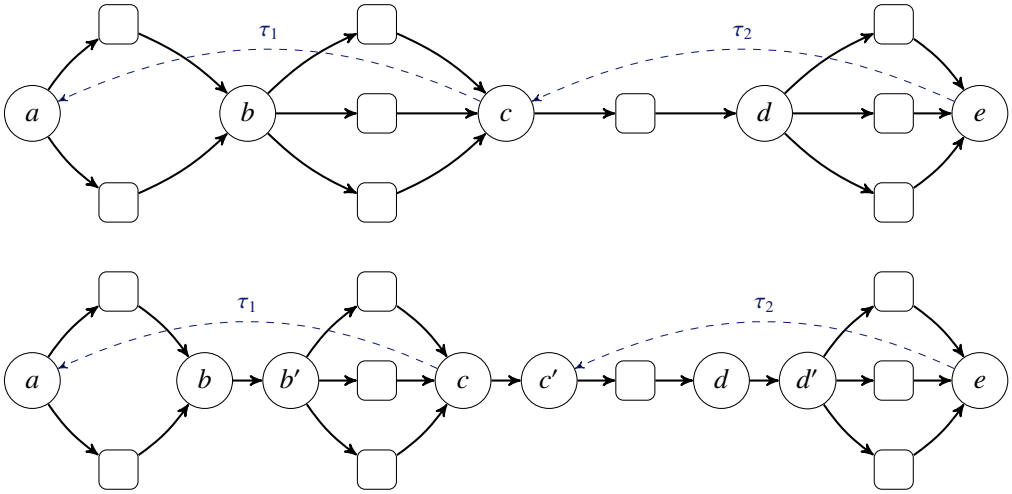


Figure 5.2 – *Two route graphs with and without duplicated separator nodes*

5.3 Solution Approach

In this section, we extend the solution approach of the preceding chapters in order to include maximum time lags as soft constraints in the way it was motivated and defined in the previous sections. This requires to compute maximum time lag violations and to optimize a lexicographical objective function instead of a single criterion. We tackle the problem by extending the batch-oblivious metaheuristic algorithm introduced in chapter 3 and extended in chapter 4. The proposed extension of the approach computes maximum time lag violations based on latest start dates of operations. In order to consider lexicographical objective functions we adopt an aggregation function known from multicriteria optimization into our approach.

We determine maximum time lag violations based on a computation of latest start dates that does not take maximum time lags into account. This allows the computational performance of the approach to be nearly maintained in terms of the number of moves explored per second. Maintaining the computational performance is necessary since, in the industrial instances at hand, only a few minutes are available for computing schedules. Admittedly, we cannot expect this approach to work out well for all possible instances, in particular ones with tight maximum time lags. However, we believe it is it worthwhile to explore this approach since it offers an apparent extension of the batch-oblivious approach and a baseline for more elaborate algorithms to be developed. The usage of latest start dates is additionally motivated by the fact that, in the considered industrial instances, most time lags are not tight and many do end at the final operation of a job. In comparison to earliest start dates, latest start dates minimize the time between the first operation and the final operation of each job. In addition, machine utilization is supposed to be very high in the industrial setting. Thus, “deliberate”

waiting times introduced to observe maximum time lags would decrease machine utilization and therefore would generate undesirable solutions. Still, latest start dates also introduce "deliberate" waiting times, although they are introduced in a way that avoids to deteriorate regular objective functions that depend on the completion times of jobs. For these reasons, we base the calculation of violation severities on latest start dates of operations. We see this as a promising starting point that can be further explored by future research. Section 5.3.1 demonstrates that the calculation of latest start dates in a batch-oblivious conjunctive graph cannot be directly based on longest path distances. An adapted algorithm is provided to calculate latest start dates for batch-oblivious conjunctive graphs.

We decided to adapt the GRASP based approach by shifting from a mono-criterion objective function to a lexicographical objective function. Recall that the insertion based construction heuristic described in section 3.4 requires only that (partial) solutions can be compared pairwise. Since the considered lexicographical orderings are also total orderings, pairwise comparisons remain possible and no changes need to be made to the construction heuristic. The shift is more difficult for Simulated Annealing since differences between objective function values need to be calculated. We tackle this by introducing an aggregation function known from multicriteria optimization. Section 5.3.2 details the inclusion of a lexicographical objective function into our approach. Finally, section 5.3.3 introduces a supplementary objective function that aims at guiding the search towards solutions with less severe maximum time lag violations.

5.3.1 Latest Start Dates in Batch-Oblivious Conjunctive Graphs

The computation of latest start dates presented here is based on the earliest start dates and batching decisions obtained by the integrated algorithm presented in chapter 3. There, each operation starts as early as possible while respecting the sequencing and assignment decisions inherent in the given conjunctive graph. Edge weights indicating batching decisions are a result of this procedure. Earliest start dates might incorporate *slack*, i.e. some operations could be postponed without negatively impacting the given regular objective function. Latest start dates delay operations in order to remove slack. In the following, we describe how the computation of latest start dates can be adapted to batch-oblivious conjunctive graphs.

Algorithm 5.1 Compute latest start dates $\overline{S}$ for a given graph G and earliest start dates S

computeLatestStartDates $(G = (V = O \cup \{0, *\},\ E),\ S)$
  $\overline{S}_v \leftarrow \infty \quad (\forall\ v \in V)$
  for $v \in \overline{O} \cup \{*\}$
    $\overline{S}_v \leftarrow S_v$
  for $v \in computeReverseTopologicalOrdering\,(G \setminus (\overline{O} \cup \{*\}))$
    for $(v, w) \in \mathrm{out}(v)$
      $\overline{S}_v \leftarrow \min\left(\overline{S}_v,\ \overline{S}_w - l_{v,w}\right)$

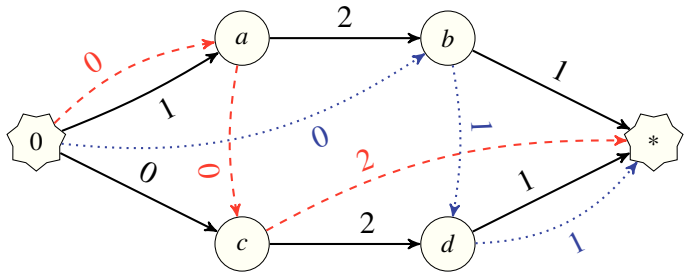


Figure 5.3 – *Conjunctive graph for two jobs with two operations per job scheduled on two machines*

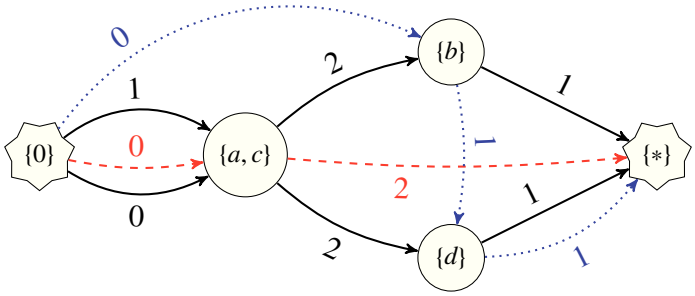


Figure 5.4 – *Quotient graph of the conjunctive graph in Figure 5.3*

Let us introduce the following notation. Recall that the *earliest start date* S_v of a scheduled operation $v \in O$ can be obtained by determining the length $L(0, v)$ of a longest path from the artificial start node 0 to node v (see section 3.2). Let us now denote the *latest start date* of an operation $v \in O$ by $\overline{S}_v$ and the set of the final operations of all jobs as $\overline{O} = \{o \in O \mid \exists\, j \in J \text{ with } o = o_{|O_j|,j}\}$. For all final operations of jobs $v \in \overline{O}$, we maintain $S_v = \overline{S}_v$. In the absence of batching machines, longest paths in the conjunctive graph allow latest start dates to be calculated for all operations $o_{i,j} \in O_j$ as $\overline{S}_{i,j} = S_{|O_j|,\,j} - L(o_{i,\,j},\ o_{|O_j|,j})$. Algorithm 5.1 provides the pseudocode of an algorithm that computes latest start dates in linear time for the conjunctive graph of a scheduling problem without batching machines.

The computation of latest start dates in Algorithm 5.1 is based on longest paths and requires all dependencies to be included in the graph. The invariant introduced in section 3.2.2 is only designed for the computation of earliest start dates and not encoded in the structure of the graph. Thus, in the presence of batching machines, Algorithm 5.1 cannot be directly used to compute latest start dates in batch-oblivious conjunctive graphs. A counterexample is provided in Figure 5.3. There, a batch-oblivious conjunctive graph is given that represents a schedule with two jobs and two machines. Job 1 that has release date 1 consists of the operations a and b, and job 2 that has release date 0 consists of the operations c and d. The processing durations are equal to 2 for operations a and c, and equal to 1 for operations b and d. Operations a and c are processed in a batch on the first machine (red

dashed arcs), thus the corresponding edge weight is zero. For earliest start dates, we have $S_a = S_c = L(0, a) = L(0, c) = 1$. The final operation of job 1 can start at $S_b = L(0, b) = 3$. The final operation of job 2 can start at $S_d = L(0, d) = 4$. However, longest paths in this graph cannot be used to compute latest start dates: Although a and c are supposed to start at the same time, we have $\overline{S}_a = S_b - L(a, b) = 1$, but $\overline{S}_c \neq S_d - L(c, d) = 2$.

Algorithm 5.1 requires that all dependencies are encoded in the structure of the graph. As shown before, this is not the case for batch-oblivious conjunctive graphs. A batch-aware conjunctive graph, as presented in section 3.2, would be suitable. However, its additional nodes would consume additional memory and computation time. To obtain an algorithm that can run directly on a batch-oblivious conjunctive graph, we describe in the following the quotient graph of a batch-oblivious conjunctive graph and its application to the situation at hand. The idea is to find a topological ordering of the batch-oblivious conjunctive graph that respects all dependencies of the quotient graph. An algorithm that computes latest start dates during a graph traversal of the batch-oblivious conjunctive graph can then be developed.

A *quotient graph* is a known concept from graph theory. Definitions can be found in e.g., Gustin (1963), or Schulz (2013). A quotient graph $G_q = (V_q, E_q)$ is defined for a partitioning of the nodes of a graph. In our case, let $O = B_1 \cup B_2 \cup \cdots \cup B_b$ be a partitioning of the operations such that each subset $B_i \subseteq O$ corresponds to all operations that are processed in a common batch on the same machine. The nodes of the quotient graph then are given by $V_q = (\bigcup_{i=1}^{b} \{B_i\}) \cup \{\{0\}, \{*\}\} \subseteq 2^V$. There exists an edge between two nodes B_i and B_j in the quotient graph if and only if there exists a corresponding edge in the original conjunctive graph, i.e. $(B_i, B_j) \in E_q \Leftrightarrow \exists\, (u, v) \in E$ s.t. $u \in B_i, v \in B_j$. The definition of edge weights is consistent between both graphs since batches are composed of operations that have equal processing times and setup families. The edges that are removed in the quotient graph are edges between operations of the same batch. These edges have an edge weight of zero. Figure 5.4 shows the quotient graph of the conjunctive graph in Figure 5.3.

Since all operations of the same batch are represented in the same node, no inconsistencies between their start dates can arise in the quotient graph. All edges that represent relevant route and resource dependencies are maintained. Thus, longest path distances in the quotient graph can be used to determine latest start dates. This is an interesting conceptual insight, but does not directly lead to an efficient implementation. We use this concept to calculate latest start dates directly on the existing batch-oblivious conjunctive graph: *A graph traversal on the batch-oblivious conjunctive graph is performed that emulates the topology of the corresponding quotient graph.* To obtain a node ordering that is suitable for such a traversal, we define *topological orderings* that are *consistent* between the original graph and the quotient graph. Intuitively speaking, this means that the topological ordering on G also respects all dependencies imposed by the quotient graph G_q.

In the following, we are given a graph $G = (V, E)$, a partitioning $\bigcup_{i=1}^{b} B_i = V$ of its nodes into batches, and the resulting quotient graph $G_q = (V_q, E_q)$. Additionally, we assume that the processing durations of all operations processed on batching machines are greater than zero. This means that, for each edge $(v, w) \in E$ such that $v \in B_i$, we have $l_{v,w} = 0 \Rightarrow w \in B_i \vee |B_i| = 1$. We denote by the *topological rank* of a node its position within a given topological ordering.

Definition 5.1. *A topological ordering* $\prec \subset (V \times V)$ *of* G *is called* consistent between G and G_q *if and only if* $\forall\, B_i \subset V,\ \forall\, u, v \in B_i, \forall\, x \in V \setminus B_i$ *such that* $(v, x) \in E$, *then* $u \prec x$.

Looking at the example in Figure 5.3, $(0, a, b, c, d, *)$ is a topological ordering that is not consistent between the given graph and its quotient graph (Figure 5.4). The reason is that b is positioned before c, although a and c are in the same batch and there is an edge (a, b). This means that the node b is visited before the batch $\{a, c\}$ has been entirely visited. In contrast, the topological ordering $(0, a, c, b, d, *)$ is consistent between both graphs.

Lemma 5.1. *Let* $\prec \subset V \times V$ *be a topological ordering of* G *with* $\forall\, u, v \in V$ *such that* $u \prec v$ *it follows* $L(0, u) \leq L(0, v)$. *Then,* $\prec$ *is consistent between* G *and* G_q.

Proof. Assume that $\prec$ is not consistent between G and G_q. Thus, there exist $B_i \subset V,\ u, v \in B_i$, $x \in V \setminus B_i$ such that $(v, x) \in E$ and $x \prec u$. Since $\prec$ is a topological ordering, we have $v \neq u$ and thus $l_{v,x} > 0$ since the processing duration of a batch cannot be equal to zero. Because $(v, x) \in E$ and $x \notin B_i$, then $L(0, x) > L(0, v)$ since the processing duration of v is greater than zero. With $u \succ x$, we have $L(0, u) \geq L(0, x)$. It follows that $L(0, u) > L(0, v)$, which contradicts $u, v \in B_i$. □

Lemma 5.2. *For a given topological ordering* $\prec \subset (V \times V)$, *the total ordering* $\sqsubset \subset (V \times V)$ *that is obtained by sorting* V *lexicographically, first by earliest start date (i.e.* $L(0, v)$*), second by topological rank based on* $\prec$, *is a topological ordering that is consistent between* G *and* G_q.

Proof. First, we show that $\sqsubset$ is a topological ordering of G. Let $u, v \in V$ be nodes such that $u \sqsubset v$. This implies $L(0, u) \leq L(0, v)$ by definition of $\sqsubset$. There cannot exist a path from v to u with $L(v, u) > 0$ since this would imply $L(0, u) > L(0, v)$. Thus, if there exists a path from u to v, it must be of length zero and thus $L(0, u) = L(0, v)$. In this case, no path from v to u can exist since $u \prec v$ due to the second lexicographical sorting criterion. Thus, the ordering is topological. From the definition of $\sqsubset$ it follows directly that if $u, v \in V$ such that $u \prec v$, then $L(0, u) \leq L(0, v)$. Thus, with Lemma 5.1, $\sqsubset$ is a topological ordering that is consistent between G and G_q. □

Lemma 5.2 allows to traverse a batch-oblivious conjunctive graph as if it were its quotient graph. This is applied in Algorithm 5.2. Instead of iterating over all outgoing edges of a single node, Algorithm 5.2 iterates over all outgoing edges of all nodes belonging to the same batch. The ordering according to Lemma 5.2 (which is reversed here) guarantees that the latest start dates of successor operations have been calculated before. When the first node of a batch is visited, the latest start date of its batch is calculated and stored for each node that belongs to the batch. Note that the subset $B_v \subset V$ is introduced in the pseudocode only for notational clarity. In the actual implementation, since the traversal uses a reverse topological ordering of G, the nodes of B_v can be obtained by following resource predecessor edges until a non-zero weighted edge is reached.

The algorithm includes the computation of a topological ordering which can be done in $O(|E|)$. In an actual implementation, the topological ordering obtained during the integrated start date and batching algorithm can be reused. Since the algorithm visits every edge

Algorithm 5.2 Compute latest start dates $\overline{S}$ for a given batch-oblivious conjunctive graph

computeLatestStartDates $(G = (V = O \cup \{0, *\},\ E),\ S)$
  $\overline{S}_v \leftarrow \infty \quad (\forall\ v \in V)$
  for $v \in \overline{O} \cup \{*\}$
    $\overline{S}_v \leftarrow S_v$
  $\pi \leftarrow computeReverseTopologicalOrdering(G)$
  $\pi \leftarrow reverse\_stable\_sort(\pi,\ L(0, v))$ // Stable sort, descending by earliest start date
  for $v \in \pi$ with $\overline{S}_v \neq \infty$
    $B_v \leftarrow$ Set of operations in the same batch as v
    for $u \in B_v$
      for $(u, w) \in \text{out}(u)$
        $\overline{S}_v \leftarrow \min\left(\overline{S}_v,\ \overline{S}_u,\ \overline{S}_w - l_{u,w}\right)$
    for $u \in B_v$
      $\overline{S}_u \leftarrow \overline{S}_v$

once, this caching does not reduce the complexity of the algorithm. Sorting nodes by earliest start date requires $O(|V| \cdot log(|V|))$. The overall runtime is $O(|E| + |V| \cdot log(|V|))$. So, a drawback of this method is the additional factor of $log(|V|)$ due to the sorting of the nodes. However, the magnitude of the number of nodes is at most 10 000 in all practical instances that we consider. Since sorting such an array stored in contiguous memory is fast and the construction of a duplicated quotient or batch-aware graph can involve larger constant factors, we use Algorithm 5.2 in our method. In addition, not constructing any auxiliary graph avoids additional implementation complexity.

Evaluating the total maximum time lag severity as defined in section 5.2 for given latest start dates is straightforward. As proposed in the literature, backward edges are added between the nodes involved in a maximum time lag. These edges can be stored in a separate list that is iterated once. Maximum time lag constraints that have already started can be treated as regular maximum time lag constraints by modeling them as edges that end at the artificial start node of the conjunctive graph. Note that we consider these additional edges as supplementary data—the original batch-oblivious graph remains untouched.

5.3.2 Lexicographical Objective Functions

The GRASP based solution approach presented in the preceding chapters considers a mono-criterion objective function. There, for the set of all feasible solutions Π (including partially constructed solutions), the objective function f assigns to a solution $s \in \Pi$ an objective function value $f(s) \in \mathbb{R}$. As motivated earlier, in order to include the total maximum time lag violation severity, we now want to lexicographically consider a multi-criteria objective function $f : \Pi \to \mathbb{R}^n$ instead. Since also additional objective functions are of interest, we

consider the general case $f : \Pi \rightarrow \mathbb{R}^n$ instead of restricting the approach to the particular case $f : \Pi \rightarrow \mathbb{R}^2$. Other objective functions are proposed in the industrial specification presented in section 2.3. An additional objective function to guide the search is introduced in section 5.3.3. In the following, we describe the adaptation of our GRASP based approach in order to include lexicographical objective functions.

A general overview of multi-criteria optimization for scheduling problems is given in T'kindt and Billaut (2001). To extend our method, we apply ideas from the multi-criteria approach of Bitar (2015) for a parallel machine scheduling problem with auxiliary resources arising in the photolithography area of semiconductor manufacturing facilities. Pfund et al. (2008) propose a multi-criteria approach that takes makespan, average cycle time, and total weighted tardiness into account. They use a desirability function to combine objectives. This function is based on two values selected for each criterion: The first value specifies a maximum allowable value while the second value specifies a goal value. Their solution approach is based on the modified shifting bottleneck procedure of Mason et al. (2002). The idea to compare allowable and goal values shares similarities with the usage of the nadir and utopian points used in the approach of Bitar (2015). The augmented weighted Tchebycheff norm used in Bitar (2015) was introduced in the context of multicriteria optimization by Steuer and Choo (1983). Dächert et al. (2010) propose an adaptive augmented weighted Tchebycheff method to determine problem dependent parameters for this metric. We use the augmented weighted Tchebycheff norm in this work while closely following the approach proposed in Bitar (2015). Several adjustments for the problem at hand are discussed in this section.

Formally, a relation $\preceq \subseteq \mathbb{R}^n \times \mathbb{R}^n$ is called a *lexicographical ordering* if $A \preceq B \Leftrightarrow A \prec B \vee A = B$ with $(a_1, \ldots, a_n) \prec (b_1, \ldots, b_n) \Leftrightarrow \exists\, m \leq n : \forall\, i < m : a_i = b_i \wedge a_m < b_m$. Since such relations meet the criteria for antisymmetry, transitivity and totality, these lexicographical orderings are also total orderings. This allows pairwise comparisons of objective function values. Thus, no changes are required for the construction algorithm (described in section 3.4) where the objective function is only involved in pairwise comparisons of (partial) solutions. Within Simulated Annealing, during iteration k, the objective functions values $f(s_{k-1})$ and $f(s_k)$ of two solutions $s_{k-1} \in \Pi$ and $s_k \in \Pi$ are compared. Depending on the result of the comparison, the move leading from solution s_{k-1} to solution s_k is accepted or not. If $f(s_k) \preceq f(s_{k-1})$, then the move is improving or constant and gets immediately accepted. Again, no changes are needed in the algorithm. However, if the move is not immediately accepted, it is accepted with a probability of $exp(\frac{-\Delta}{T})$, where $\Delta = f(s_k) - f(s_{k-1})$ measures the difference of the objective function values and T is the current temperature value. We need to rethink this situation, since Δ, that was a scalar in the mono-criterion case, now becomes a vector. To cope with this, as proposed in Bitar (2015), we introduce an *aggregation function* $a : \mathbb{R}^n \rightarrow \mathbb{R}$ to calculate the difference between the objective function values of two solutions as $\Delta = a(f(s_k)) - a(f(s_{k-1})) \in \mathbb{R}$.

In the following, we discuss the definition of a suitable aggregation function. We start by providing definitions known from multi-criteria optimization that can be found in the literature, e.g. T'kindt and Billaut (2001). We slightly adapt the notation for consistency.

Let us denote by $f_i : \Pi \to \mathbb{R}$ the projection of the objective function f to its i-th component. Note that we assume w.l.o.g. that we minimize the given objectives.

Definition 5.2. *A point* $(x_1, \ldots, x_n) \in \mathbb{R}^n$ dominates *a point* $(y_1, \ldots, y_n) \in \mathbb{R}^n$ *if and only if* $(\forall\ i \in \{1, \ldots, n\} : x_i \leq y_i) \wedge (\exists\ i \in \{1, \ldots, n\} : x_i < y_i)$.

Definition 5.3. *A solution* $x \in \Pi$ *is called a* weak Pareto optimum *(or* weakly efficient*) if and only if* $\nexists\ s \in \Pi$ *such that* $f_i(s) < f_i(x)\ \forall i \in \{1, \ldots, n\}$.

Definition 5.4. *A solution* $x \in \Pi$ *is called a* strict Pareto optimum *(or* strictly efficient*) if and only if* $\nexists\ s \in \Pi$ *such that* $f(s)$ *dominates* $f(x)$.

Definition 5.5. *A point* $(r_1^I, \ldots, r_n^I) \in \mathbb{R}^n$ *is called* ideal *if and only if for each* $i \in \{1, \ldots, n\}$ *it holds* $r_i^I = \min_{s \in \Pi}\ f_i(s)$.

Definition 5.6. *A point* $(r_1^U, \ldots, r_n^U) \in \mathbb{R}^n$ *is called an* utopian point *if and only if it dominates the ideal point.*

Definition 5.7. *Let* $P \subset \Pi$ *be the set of all strict Pareto optimal solutions. A point* $(r_1^N, \ldots, r_n^N)$ *is called* nadir *(or* anti-ideal point*) if and only if* $r_i^N = \max_{s \in P}\ f_i(s)\ \ \forall\ i \in \{1, \ldots, n\}$.

A known aggregation function in the context of multi-criteria optimization is the weighted Tchebycheff metric. It determines the weighted distance to a *reference point* $r \in \mathbb{R}^n$ using weights $\lambda \in \mathbb{R}^n$ and the maximum norm defined as $L_\infty : \mathbb{R}^n \to \mathbb{R},\ (x_1, \ldots, x_n) \mapsto \max_{i=1}^n |x_i|$. The *weighted Tchebycheff metric* is defined as

$$a_{\lambda,r} :\ \mathbb{R}^n \to \mathbb{R},$$

$$a_{\lambda,r}(x_1, \ldots, x_n)\ \ \mapsto\ \ \| \lambda_1(x_1 - r_1), \ldots, \lambda_n(x_n - r_n) \|_\infty .$$

As pointed out in Bitar (2015), Wierzbicki (1986) describes the following property of the *weighted Tchebycheff metric* which is based on results of Dinkelbach (1971) and Bowman (1976): For a reference point $r \in \mathbb{R}^n$ that dominates or is equal to the ideal point, there exist weights $\lambda \in \mathbb{R}^n$ for all Pareto optimal solutions $x \in \mathbb{R}^n$ such that x is an optimal solution of a minimization problem that has $a_{\lambda,r}$ as its objective function. This is an advantage over aggregation functions that are solely using a weighted sum since, used as an objective function, aggregation functions cannot generate all Pareto optimal solutions.

A drawback of the weighted Tchebycheff metric is that it does not break ties between solutions that are equal regarding the objective that dominates the maximum norm L_∞ but different regarding other objectives. This is tackled by the *weighted augmented Tchebycheff metric* which is defined as

$$\overline{a}_{\lambda,r,\rho} :\ \mathbb{R}^n \to \mathbb{R},$$

$$\overline{a}_{\lambda,r,\rho}(x_1, \ldots, x_n)\ \ \mapsto\ \ \| \lambda_1 (x_1 - r_1), \ldots, \lambda_n (x_n - r_n) \|_\infty\ +\ \rho \sum_{k=1}^{n} \lambda_k |x_k - r_k| .$$

This metric was introduced by Steuer and Choo (1983) and aims at avoiding solutions that are non-strict Pareto optima. In order to apply this metric within the Simulated Annealing

metaheuristic (as a part of the considered GRASP based metaheuristic approach), the parameters of this metric need be determined. In order to do this, a preprocessing step is performed once before the GRASP based algorithm is started. The following paragraphs describe how the reference point, the weights and the parameter ρ are determined in order to parametrize the metric.

Determination of the reference point $r \in \mathbb{R}^n$ As pointed out in Wierzbicki (1986), utopian points can be used as reference points in the weighted augmented Tchebycheff metric. The preprocessing procedure determines a (not necessarily feasible) *utopian solution* that has objective function values which yield an utopian point. This utopian point consists of lower bounds for the individual objective functions. In order to determine an utopian solution (i.e. lower bounds), we relax all resource constraints in the conjunctive graph. So, this conjunctive graph consists only of edges related to the routes of the job. To select routes, in the sense of the route graphs defined in section 4.1, for each job, the shortest path from the start separator operation of the job to the end separator operation of the job is chosen. In other words, each time the machine with the fastest processing time is selected. Note that the release dates of all jobs are still modeled in this graph. Then, computing earliest start dates in this graph yields a schedule without waiting periods. A computation of the objective function values for such schedules leads to lower bounds for all regular objective functions. The construction of this graph also leads to lower bounds for the objectives that were derived from maximum time lag constraints. Thus, we obtain objective function values that either represent an ideal point or an utopian point. They are then used as the reference point within the weighted augmented Tchebycheff metric.

Determination of the weights $\lambda \in \mathbb{R}^n$ The objective functions that we consider do not use the same unit of measurement. In the industrial case at hand, the total maximum time lag violation severity and weighted flow factors cannot be directly compared. Typical regular criteria cannot be directly compared as well. This observation leads to the insight that the weights of the metric essentially should normalize the values to be compared. For this, we follow again the approach of Bitar (2015) who proposes to cope with this by estimating the nadir (see Definition 5.7). For the scheduling problem at hand, which is different from that considered in Bitar (2015), we estimate the nadir as follows. During preprocessing, we determine a set of sample solutions $S \subset \Pi$ by performing random moves on a starting solution. Note that the same set is also used to calibrate Simulated Annealing (it is $|S| = P_s$, for the parameter P_s introduced in section 3.4). Then, the set $Q \subseteq S$ is defined to determine the sampled solutions that are best with respect to at least one considered objective function. This is given by

$$Q = \bigcup_{i=1}^{n} \operatorname*{argmin}_{x \in S} f_i(x).$$

An estimation of the nadir is then determined by the point $r^N \in \mathbb{R}^n$ which is defined by

$$r_i^N = \max_{s \in Q} f_i(s) \quad \forall\, i \in \{1, \ldots, n\}\,.$$

In addition, we are given weights $c \in \mathbb{R}^n_{>0}$ that allow to manually weight the objective functions. In most cases, these manual weights can be set to one. They allow the search to be manually enforced towards certain objectives, e.g. for instances where the estimations of the utopian point and the nadir yield weak results. Based on this, the weights $\lambda \in \mathbb{R}^n$, used in the weighted augmented Tchebycheff metric, are determined as

$$\lambda_i = \frac{c_i}{r_i^N - r_i} \qquad (\forall\ i \in \{1, \ldots, n\}),$$

where $r^N \in \mathbb{R}^n$ is the estimation of the nadir and $r \in \mathbb{R}^n$ is the reference point. This formula slightly generalizes the definition used in Bitar (2015) which focuses on the case $n = 2$. This choice of $\lambda \in \mathbb{R}^n$ normalizes the different objective function values.

Determination of the parameter ρ We apply ideas presented in Bitar (2015) also for determining the parameter ρ. In the weighted augmented Tchebycheff metric, the L_∞ norm should always be the dominating term to guarantee that the weighted sum is only relevant for breaking ties between solutions that are equally rated by the L_∞ norm. Thus, the parameter ρ must be chosen sufficiently small. With

$$\textstyle\sum_{i=1}^n \lambda_i |x_i - r_i| \quad \leq \quad n \cdot \max_{i=1}^n \lambda_i |x_i - r_i| \quad < \quad 2 \cdot n \cdot \max_{i=1}^n \lambda_i |x_i - r_i|$$

(for each $x \in f(\Pi)$ except the ideal point for which $\overline{a}_{\lambda, r, \rho}(r^I)$ might be equal to zero), we conclude that fixing the parameter to $\rho = \frac{1}{2 \cdot n}$ does always satisfy the desired property, i.e. the L_∞ norm is always the dominating term in the weighted augmented Tchebycheff metric.

5.3.3 A Guiding Objective Function

In this section, we introduce an additional objective function to guide the search towards solutions with less severe maximum time lag violations. This objective function is intended to be used as the third component in a lexicographic objective function. As earlier, the first component is given by the total maximum time lag violation severity and the second component is given by the original objective function (which is the weighted flow factor in our industrial setting).

This guiding objective function measures *total maximum time lag exceedance* and is defined as follows. Consider a feasible schedule with start dates $S_{i,j} \in \mathbb{N}$ given for all scheduled operations $o_{i,j} \in O$. As earlier, we obtain for each time lag $\tau = (j, k, l, d, \gamma, c) \in T$ its delay $L = S_{l,j} - S_{k,j}$. We define the *exceedance* of a maximum time lag as $E_\tau = \max(0, L - \gamma)^2$. Overall, the total maximum time lag exceedance of a schedule is defined as

$$E = \sum_{\tau \in T} E_\tau.$$

In contrast to the actual objective function that acts as a soft constraint, this function is unbounded regarding the delay. Therefore, lots with violated maximum time lag constraints

tend to be moved forward which might help to obtain more quickly solutions with less violated constraints. The motivation for making it quadratic is analogous to that given for the maximum time lag violation severity.

5.4 Numerical Results

The algorithms were run using the same environment as in the numerical experiments presented in section 3.5, i.e. an Intel Xeon E5-2620 2.1 GHz machine with 6 cores running Microsoft Windows 7. Section 5.4.1 presents results for industrial instances from a real-world semiconductor manufacturing facility. Section 5.4.2 presents results for the job-shop scheduling instances with maximum time lags of Caumond et al. (2008). For experiments on the academic instances, we used the same parameter settings as in section 3.5: A cooling factor of $P_c = 0.99999$, a number of samples $P_s = 100$, a maximum number of iterations $P_m = 100\,000$, a temperature percentile of $P_t = 5\,\%$, and a perturbation intensity of $P_i = 5$. For the industrial instances, the only difference in the parameter settings is the cooling factor, which is decreased to $P_c = 0.9995$.

5.4.1 Industrial Instances with Maximum Time Lags

In the following, we present numerical results for instances extracted from the Manufacturing Execution System (*MES*) of a real-world semiconductor manufacturing facility over the course of two months. Each instance reflects the state of the diffusion and cleaning area at a given moment in time. The objective function considered here is the *Weighted Flow Fac-*

	LB		**Initial**		**O1**		**O2**		**O3**	
	WFF	TVS	WFF	TVS	WFF	TVS	WFF	TVS	WFF	TVS
1	13.07	0.00	16.80	10.73	14.84	30.21	15.85	**0.00**	15.62	**0.00**
2	11.95	0.00	13.22	6.81	13.01	18.34	13.02	**0.00**	13.03	**0.00**
3	16.73	5.00	18.41	60.04	17.93	99.94	18.03	**5.00**	17.99	**5.00**
4	13.44	0.15	15.16	15.04	14.59	16.56	14.64	**0.15**	14.63	**0.15**
5	16.24	10.14	18.09	21.32	17.45	52.47	17.60	**10.14**	17.64	**10.14**
6	28.57	0.00	31.45	9.35	31.17	32.79	31.39	**0.00**	31.56	**0.00**
7	24.84	0.00	27.30	0.15	26.06	42.88	26.10	**0.00**	26.12	**0.00**
8	13.91	16.47	16.91	86.63	15.73	107.59	16.35	**25.52**	17.00	25.73
9	12.21	13.87	14.71	58.19	13.71	124.05	15.58	20.19	14.75	**20.12**
10	21.43	22.50	23.76	22.99	22.84	103.11	23.99	23.24	23.34	**22.60**

Table 5.1 – *Objective function values for industrial instances*

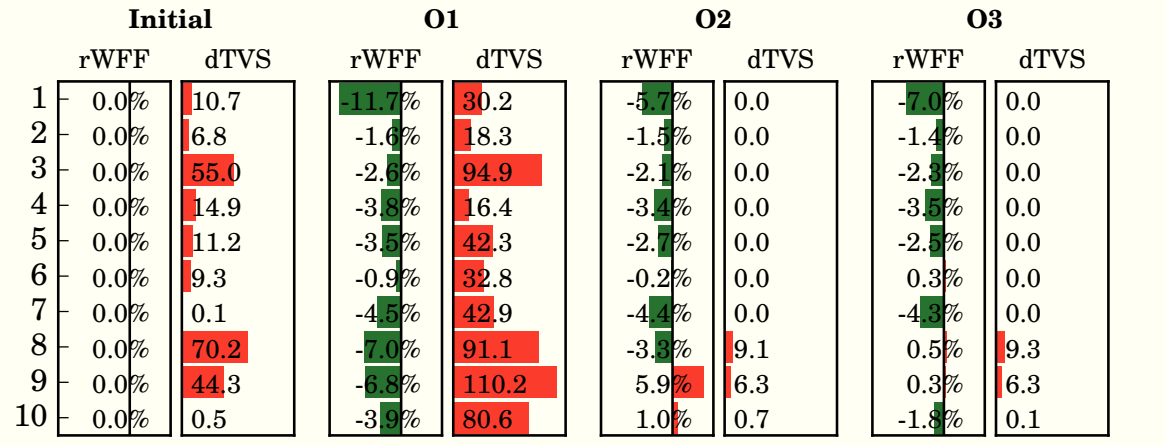


Figure 5.5 – *Comparison of three objective functions for 10 industrial instances*

tor (*WFF*) defined in the industrial problem specification in section 2.3.2. As described in this chapter, we also consider the *Total time lag Violation Severity* (*TVS*) defined in section 5.2 and the *Total time lag Violation Exceedance* (*TVE*) defined in section 5.3.3. We consider three different combined objective functions. The first, denoted as *O1* (WFF), refers to a mono-criterion objective function that minimizes the weighted flow factor alone, ignoring maximum time lag constraint violations. The second, denoted as *O2* (TVS, WFF), refers to a lexicographic objective function that minimizes the total time lag violation severity before minimizing the weighted flow factor. The third, denoted as *O3* (TVS, WFF, TVE), is an extension of *O2* that minimizes the total time lag violation exceedance as the third component of a lexicographical objective function in order to guide the search.

Detailed results are provided in table 5.1. Column *LB* refers to the lower bound obtained during the computation of the utopian solution (see section 5.3.2). Column *Initial* refers to a solution computed by the construction heuristic (see section 3.4) using *O3* (TVS, WFF, TVE) as objective function. The remaining columns refer to solutions obtained by running the GRASP based metaheuristic for the three objective functions *O1*, *O2*, and *O3*, allowing a computation time of 5 minutes per instance. The sub-columns *WFF* and *TVS* provide values for the weighted flow factor and the total maximum time lag violation severity of the best solutions, respectively.

An alternative representation of the same results is given in Figure 5.5. It aims at illustrating the relation between the weighted flow factor and the total maximum time lag violation severity. The weighted flow factors of optimized solutions are compared to the weighted flow factors of the initial solution. Using initial solution as a baseline is motivated by the assumption that our construction heuristic obtains solutions with objective functions values that are close to the results obtained by dispatching rules (see section 3.5.3). So, the columns *rWFF* (*relative weighted flow factor*) show the deviation in percent from the weighted flow factor obtained by the construction heuristic. The columns *dTVS* show the total maximum time lag violation severity minus the corresponding lower bound. Thus, if *dTVS* is equal to

zero, then the solution is optimal regarding the total maximum time lag violation severity.

Note that the results obtained by the construction heuristic are strongly improved. Optimizing the original objective function (*O1*) alone yields solutions that are up to 11.7% better. However, these solutions strongly violate the maximum time lag constraints. The results for *O2* show that the original objective function value can still be strongly improved if maximum time lag constraints are taken into account. Note that, when manually looking at the solutions for instances 8 and 9, we can see that, for the total maximum time lag violation severity, the gap to the lower bound is entirely caused by weaknesses in the lower bounds. The results for *O3* show that using the guiding objective function can improve solutions. Since maximum time lag constraints are satisfied as much as possible while obtaining high quality solutions, we conclude that the approach is practical and applicable for the considered industrial setting.

	O3		**E-O3**		**E-Initial**	
	WFF	TVS	WFF	TVS	WFF	TVS
1	15.62	**0.00**	17.03	8.05	16.86	27.18
2	13.03	**0.00**	13.10	**0.00**	13.29	6.84
3	17.99	**5.00**	18.12	60.45	18.53	110.00
4	14.63	**0.15**	14.99	32.44	15.25	52.85
5	17.64	**10.14**	17.93	20.94	18.17	52.18
6	31.56	**0.00**	31.98	17.12	31.49	68.17
7	26.12	**0.00**	26.48	0.80	26.87	38.59
8	17.00	**25.73**	18.44	125.67	16.44	206.73
9	14.75	**20.12**	14.25	82.08	14.14	155.17
10	23.34	**22.60**	23.56	74.22	23.28	96.07

Table 5.2 – *Comparison using earliest and latest start dates*

The results presented so far are based on latest start dates. To see if the effort of computing latest start dates is really necessary, it is interesting to compare results based on earliest start dates with those based on latest start dates. This comparison can be found in Table 5.2. Columns *O3* refer again to the objective function (TVS, WFF) using the GRASP based metaheuristic approach with latest start dates. Columns *E-O3* refer to the same objective function and solution approach, but time lag violation severities are calculated based on earliest start dates. Columns *E-Initial* refer to results obtained by our construction heuristic. The results clearly show that latest start dates yield better results. The earliest start date based algorithm never obtains better results for the ten industrial instances. Moreover, in many cases, the Total time lag Violation Severity (TVS) is much worse with earliest start dates. Although it is not as significant, the Weighted Flow Factor (WFF) is also worse. We believe that this

is because the approach with earliest start dates, which first tries to minimize TVS (as the primary criterion), is not very successful on this criterion. Thus, it does not have enough time to explore solutions that improve WFF or solutions with good WFF are discarded since their TVS is poor.

5.4.2 Job-Shop Scheduling Instances with Maximum Time Lags

A job-shop scheduling problem that considers maximum time lags between adjacent operations of the same job is investigated by Caumond et al. (2008). For their computational evaluation, they extend the job-shop instances of Lawrence (1984) (which optimize the makespan), by adding maximum time lag constraints. A *tightness factor* is fixed in order to generate maximum time lag constraints with durations derived from the processing durations of the original instances. We refer to these job-shop scheduling instances with maximum time lags as *JSTL* instances in the following. In contrast to our approach, maximum time lags are treated as hard constraints in the JSTL instances. However, if we obtain solutions with a total maximum time lag violation severity of zero, the results for the makespan can be compared. In our numerical experiments, we focus on the instances la06, la07, la08, because they are the only ones for which Caumond et al. (2008) provide numerical results for all tightness factors between 0.5 (very tight) and 10 (relaxed). Recall that these instances do not consider batching machines and the assignment to a machine is fixed for each operation (i.e. flexibility is not considered).

In order to solve these instances, we introduce maximum time lag constraints (see section 5.2) with a violation cost of one, a maximum duration equal to the duration in the JSTL instances, and an ultimate duration that is equal to the duration in the JSTL instances as well. Thus, the total maximum time lag violation severity counts the number of violated maximum time lags. We consider two different objective functions. The first objective function, denoted by *O4* (TVS, M), lexicographically optimizes first the total maximum time lag violation severity, and second, the Makespan. The second objective function, denoted by *O5* (TVS, TVE, M), optimizes a lexicographical objective function with three components: (1) the total maximum time lag violation severity, (2) the total maximum time lag violation exceedance, (3) the makespan. All weights of these objectives (w.r.t. the aggregation function in section 5.3.2) are equal to one, except for the total maximum time lag violation exceedance in the second objective function, which is weighted by two.

Table 5.3 provides numerical results for these instances. We provide results for our construction heuristic run using both objective functions (*C-O4* and *C-O5*). For our GRASP based metaheuristic approach, we allow a computation time of two minutes per instance and also provide results for both objective functions (*G-O4* and *G-O5*). We compare these results to the literature: Columns *MA* refer to a memetic algorithm presented in Caumond et al. (2008). Columns *BB* refer to a branch-and-bound approach presented in Artigues et al. (2011). Columns *SSPR* refer to a scatter search and path-relinking approach presented in González et al. (2015). Column *Opt* shows the known optimum solutions provided in González et al. (2015). For the results from the literature, the makespan (sub-columns *M*)

		Opt	BB		MA		SSPR		C-O4		C-O5		G-O4		G-O5	
		M	M	t (s)	M	t (s)	M	t (s)	M	V	M	V	M	V	M	V
la06	0.5	1003	1471	526	1153	545	**1006**	14	1182	15	1162	15	1342	2	1780	0
la06	1	926	1391	524	1086	1117	**926**	18	1163	15	1197	9	1154	1	1184	0
la06	3	926	1391	524	1101	405	**926**	7	1095	8	1126	7	**926**	0	**926**	0
la06	10	926	927	707	**926**	14	**926**	4	933	1	933	1	**926**	0	**926**	0
la07	0.5	953	1430	529	1132	573	**982**	12	1145	12	1328	13	1159	2	1604	0
la07	1	896	1065	754	1009	532	**906**	16	1046	12	1085	12	984	0	1040	0
la07	3	890	1079	659	975	477	**890**	6	1056	7	1082	7	**890**	0	**890**	0
la07	10	890	1123	518	**890**	39	**890**	6	1059	4	1003	4	**890**	0	**890**	0
la08	0.5	984	1454	529	1124	1199	**1014**	16	1265	13	1278	14	1526	1	1234	0
la08	1	892	1052	587	1013	541	**897**	13	1054	15	1303	10	1017	1	1143	0
la08	3	863	1052	587	1013	544	**863**	6	1003	9	1129	12	**863**	0	**863**	0
la08	10	863	**863**	260	**863**	17	**863**	5	1097	2	1060	3	**863**	0	**863**	0

Table 5.3 – *Results for instances of Caumond et al. (2008)*

and the runtime in seconds (sub-columns *t(s)*) is given. For the results obtained by our approaches, the makespan (sub-columns *M*) and the number of maximum time lag violations (sub-columns *V*) is given. Only the GRASP based approach that includes the guiding objective function *O5* is able to find feasible results for all instances. We obtain good results for instances that do not include very tight maximum time lags. For the tightness factors 3 and 10, optimal solutions are found. For instances with tighter maximum time lags, results are not as good. We believe that, for tight maximum time lags, computing latest start dates as performed in our approaches is not sufficient and a more elaborate method for computing start dates is needed. The results of this comparison to dedicated solution approaches support the usage of latest start dates in our approach, where we want to deal with complex but not extremely tight maximum time lag constraints for the industrial use case.

5.5 Conclusion

Maximum time lag constraints, which are crucial for the industrial application at hand, can be included in our approach as soft constraints. We assign individual violation costs to each maximum time lag constraint and distinguish between reworkable and non-reworkable maximum time lags. This chapter shows the extensibility of our scheduling approach: Additional objective functions which quantify maximum time lag constraints are included and lexicographically ordered multi-criteria objective functions can be optimized.

The lexicographic approach described in section 5.3.2 is based on a suitable aggregation function. We believe that it can be extended towards a true multi-criterial approach: Instead of maintaining exactly one lexicographically best solution, we could maintain a Pareto frontier containing a set of solutions.

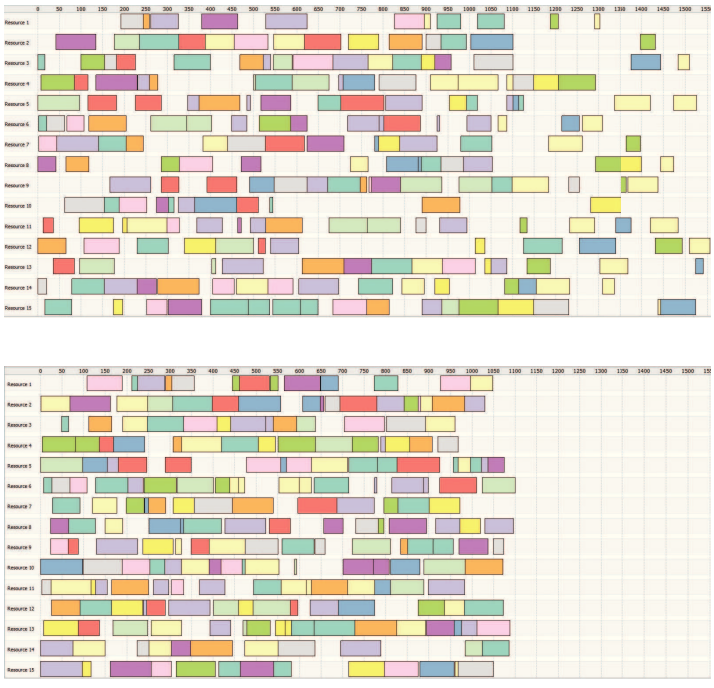
We have seen that the presented approach works well for the considered industrial instances and yields results that are applicable in practice. However, the approach needs to be improved if instances with tight maximum time lag constraints should be considered. For this, maximum time lag constraints should be somehow considered during the computation of start dates. In the literature, usually negative weighted backward edges are introduced and taken into account during the calculation of longest paths.

We have shown how latest start dates can be directly computed for batch-oblivious conjunctive graphs. Though the topology of the batch-oblivious conjunctive graph does not reflect all dependencies related to batching decisions, no auxiliary graph needs to be constructed. In addition, it would be interesting to see numerical results based on earliest start dates. Note in this context that, in the considered industrial setting, start dates often are only important to determine performance indicators. In practice, only the obtained sequencing decisions are provided to the Manufacturing Execution System (MES). Actual start dates might then differ from the start dates calculated by a scheduler due to inaccuracies in the data and simplifications in the modeling. Thus, the practical influence of the computational method to determine start dates might be small in practice—at least as long as no artificial waiting periods for lots available in front of machines must be prescribed.

Chapter 6

Conclusion and Perspectives

In this thesis, an optimization approach for scheduling jobs in the diffusion and cleaning area of a semiconductor manufacturing facility has been developed which is applicable in practice. The approach can handle a broad range of scheduling problems and can be further generalized.



Two Gantt Charts, showing an initial solution and an optimized solution of the instance rdata-la37 from Hurink et al. (1994).

In this thesis, models and optimization methods are presented for the rich set of constraints and objectives that we observe in the diffusion and cleaning area of a semiconductor manufacturing facility. We have seen that the underlying properties of this specific application area lead to very general scheduling problems. Meta-heuristic solution approaches based on a novel batch-oblivious conjunctive graph representation are proposed in this work. Section 6.1 summarizes the main contributions of this thesis and section 6.2 provides future research directions.

6.1 Conclusion

The complex industrial scheduling problem, which is specified in detail in chapter 2, imposes a rich set of constraints that can barely be tackled all at once. Therefore, this thesis starts by considering complex job-shop scheduling problems which cover the core characteristics of our industrial scheduling problem. The main property is the presence of batching machines in a job-shop scheduling environment. As preceding approaches known from the literature, we tackle the problem using a heuristic approach based on conjunctive graphs. We introduce a novel batch-oblivious representation which helps to reduce the structural complexity of batch-aware conjunctive graphs. Together with an integrated batch-oblivious move, an integrated computation of start dates and batching decisions during a traversal of the graph is proposed to adaptively "fill up" underutilized batches. The batch-oblivious representation enables an integrated neighborhood where batching, sequencing and assignment decisions are interleaved. We believe that the batch-oblivious representation facilitates the generalization of the approach towards additional constraints. The presented representation and neighborhood is independent of the meta-heuristic which is using it. We apply these building blocks in a GRASP based heuristic approach which can be straightforwardly parallelized—an important property to exploit the continuously increasing parallelism of modern CPUs. Good numerical results for a wide range of different benchmark instances validate the performance of our approach.

In order to increase the detail of our scheduling model, we integrate the management of internal machine components by extending the route and resource flexibility of the complex job-shop scheduling problem. Finding an appropriate conjunctive graph representation for this generalization is facilitated by our batch-oblivious approach. To model furnaces in detail, a resource acquisition constraint is introduced that exclusively reserves a resource between two operations of the same job. We show that an efficient approach for inserting nodes presented by Kis (2003) can be further generalized to include these constraints. We show that the batch-oblivious approach also works in case multiple resource per operation are required. Good numerical results are obtained for benchmark instances from the photolithography area.

Maximum time lag constraints, which are crucial for the industrial application at hand, can be included in our approach as soft constraints. In order to obtain a modeling that is applicable in practice, individual violation costs are assigned to each maximum time lag constraint while distinguishing between reworkable and non-reworkable maximum time lags.

Additional objective functions which quantify maximum time lag constraints are included. Lexicographically ordered multi-criteria objective functions are optimized in order to cope with infeasible schedules.

Overall, we have demonstrated that our batch-oblivious approach is extensible by implementing various generalizations. All presented constraints can be used in arbitrary combinations, which allows a rich set of scheduling problems to be tackled. We believe that further extensions are practicable. Therefore, we indicate future extensions in the following section for improving the performance of our approach and taking additional constraints into account.

6.2 Perspectives

Though we already obtain good numerical results, we believe that a huge potential remains for further improving the performance of our approach. Approaches for speeding up the computation of start dates by only partially updating start dates that were previously computed are proposed by Michel and Van Hentenryck (2003), Pearce and Kelly (2007) and Sobeyko and Mönch (2016). The idea is to recalculate, after each move, only those start dates that might have changed instead of updating the start dates of every scheduled node in the conjunctive graph. We assume that a major gain in the number of moves performed per second can be achieved by combining this ideas with our approach. This is in particular promising for large problem instances since the expected gain grows with the number of nodes in the graph. Another idea to improve our approach is a combination with the ideas presented in Dauzère-Pérès and Paulli (1997) and García-León et al. (2015) for flexible job-shops. Dauzère-Pérès and Paulli (1997) and García-León et al. (2015) avoid the runtime cost of performing a move by evaluating its effects on the objective function without actually performing the move. Regarding our batch-oblivious approach, it seems promising to further explore more advanced node selection strategies in order to improve the ones proposed in section 3.3.4. For scheduling instances with large movable components, exploring improvements of the exhaustive search proposed in section 4.3.2 in order to insert entire movable components more efficiently could be worthwhile. The construction heuristic which is used in the GRASP based approach probes a large number of node insertion positions. Since this can be slow for large instances, replacing it by a faster approach seems promising. An idea would be to use a randomized version of a dispatching rule (e.g., BATC (Mönch et al. (2013))) in order to accelerate the construction phase and thus to leave more time for the improvement phase (i.e. Simulated Annealing). Though we have shown that our approach works well for industrial instances and yields results that are applicable in practice, we observed difficulties for instances with tight maximum time lag constraints. It would be interesting to explore how maximum time lag constraints can be taken directly into account during the calculation of start dates.

Beside the concrete improvement ideas for our current approach presented in the preceding paragraph, exploring a wider range of ideas might be interesting as well. Our most

important contribution, the batch-oblivious approach, is not bound to a GRASP based meta-heuristic. GRASP has strong diversification and intensification mechanisms but lacks elements of mutual learning that can be found in path-relinking approaches or genetic algorithms. Thus, it would be interesting to explore the performance of our batch-oblivious method in combination with other meta-heuristic approaches. Let us emphasize that the current performance of our approach is already suitable for real-world instances in the diffusion and cleaning area. However, a significant improvement in performance would enable us to tackle larger instances and new application areas: We could get closer to the ability of optimizing the scheduling of lots through multiple work areas or even considering an entire fab at once, although the latter might not be relevant for multiple reasons, in particular because the planning horizon to be considered would be too long.

A comprehensive industrial problem specification is presented in chapter 2. Though the most crucial properties of this specification are included in the models and solution methods proposed in this thesis, there remains a feature gap between the industrial specification and our solution methods. It would be valuable to close that gap and take further modeling details into account. Several properties of the industrial problem can be tackled by the extended route and resource flexibility described in chapter 4. It would be very interesting to further apply and analyze the approach by performing extended numerical experiments. Also, an experimental study for analyzing a reasonable modeling granularity would be very interesting. Many properties can be taken into account by combining the available constraints as proposed in section 4.1.5. In order to model furnaces in detail, it would be necessary to combine the models presented in chapters 3 and 4 in a more general way. Since the proposed route graph modeling only takes batching machines into account if the length of the movable component is one, a relaxation of this limitation is required to consider movable components of arbitrary lengths. The obligatory boat standby durations in furnace machines could be tackled by an adaptation of edge weights. However, this adaptation seems to be non-trivial. A possible approach could (again) be to dynamically adapt edge weights during graph traversals. This would require to maintain a state for each furnace machine that is updated whenever a concerned operation is traversed. Utilizing the number of moves performed in the planning horizon as an additional objective function would be useful. Though, this might be difficult to combine with the previously proposed partial graph updates since then the number of moves in the planning horizon must be partially updated as well.

An interesting long-term perspective could be to integrate scheduling approaches with related decisions making systems which have been described in chapter 1. Integrating tool risk management decisions, concerning inspection and control procedures, with scheduling might help to choose machines with less risk already during scheduling. Similarly, the integration of a machine health indicator into the scheduling approach can reduce risk by avoiding "risky" machines for important lots. A machine health indicator is an index that represents the state of the machine at a given point in time. Note that such approaches already have been proposed in the literature (Doleschal et al. (2015), Kao et al. (2016)). It might be also interesting to consider the scheduler as a tool that could be used in order to make or verify qualification management decisions following the ideas described in Johnzén et al. (2008). Qualifica-

tion management decisions are needed to manage machine capabilities. They determine the machines that are prepared to gain production abilities by performing time-consuming preparatory setups and tests.

Applying the approach presented in this thesis to other work areas of a fab seems to be practicable and relevant. We already have shown that scheduling problems in the photolithography area can be solved efficiently by our method. Since the implantation area contains equipment with similar characteristics it might be promising and practicable to be applied there as well. Another interesting future direction is to analyze how the approach can be applied to other areas beyond semiconductor manufacturing. Resource acquisition constraints appear, for example, in the scheduling of railway maintenance operations (see Ramond et al. (2006)). Another important extension for such cases would be the consideration of time window constraints for machines. This is generally not necessary in semiconductor manufacturing since a fab operates 24 hours per day. However, in many other industries this is not the case. Time windows can also be used for modeling other use cases, e.g. a machine down time due to a planned maintenance activity. In many applications more than one optimization criterion is of interest. Since the lexicographic approach described in section 5.3.2 is based on an general aggregation function, we believe that this approach allows a straightforward extension towards a true multi-criteria approach: Instead of maintaining exactly one lexicographically best solution, we could maintain a Pareto frontier containing a set of solutions.

Appendix A

Résumé en français

A.1 Introduction

Le sujet de cette thèse est l'ordonnancement dans une usine de fabrication de semi-conducteurs ; plus précisément dans un atelier spécifique qui impose un ensemble de contraintes variées. Nous présentons des modèles et des méthodes d'optimisation qui prennent en compte les nombreux contraintes et objectifs qui sont présents dans cette zone de travail. Bien que ce soit un domaine d'application spécifique, nous allons voir que les propriétés sous-jacentes conduisent à des problèmes d'ordonnancement généraux.

L'électronique numérique a connu une énorme croissance au cours des 50 dernières années. Depuis la prédiction de Moore (1965), qui déclare que le nombre de transistors des microprocesseurs sur une puce de silicium double tous les deux ans, les applications sont devenues omniprésentes : les ordinateurs, les capteurs, les centres de données, l'électronique automobile et les dispositifs portables. Aujourd'hui, les appareils électroniques sont partout. La fabrication de semi-conducteurs reste le processus de base qui permet toutes ces applications. La Figure A.1 montre le chiffre d'affaires du marché mondial de semi-conducteurs fournis par la Semiconductor Industry Association (SIA (2015)). De 1995 à 2015, les ventes mondiales ont augmentées de 50 à 337 milliards de dollars américains par an, soulignant l'importance et la taille de ce secteur.

La fabrication de semi-conducteurs est un secteur à forte intensité capitalistique et avec une concurrence féroce. Des temps de production très longs, des durées de vie des produits courtes et des coûts de fabrication élevés sont les facteurs clé de ce secteur. Les machines et l'espace en salle blanche sont très chers. Une seule machine coûte entre 100 000 et 40 millions de dollars américains (voir par exemple le rapport annuel d' ASML (2016)). Des centaines de machines peuvent être présentes dans une seule usine de semi-conducteurs (*fab*). Quirk and Serda (2001) rapportent que près de 75% de l'investissement d'une fab est dépensé sur les machines. Une seule fab peut coûter jusqu'à 5 milliards de dollars américains. Par conséquent, une utilisation élevée de l'équipement est un facteur de succès important dans ce secteur. Des nombreuses stratégies pour réduire les coûts de production ont déjà été exploitées. De nos jours, la réduction des coûts opérationnels par une amélioration des systèmes de décision est considérée comme une possibilité d'amélioration prometteuse. D'un point de vue académique, ce secteur offre également des défis intéressants. La complexité observée

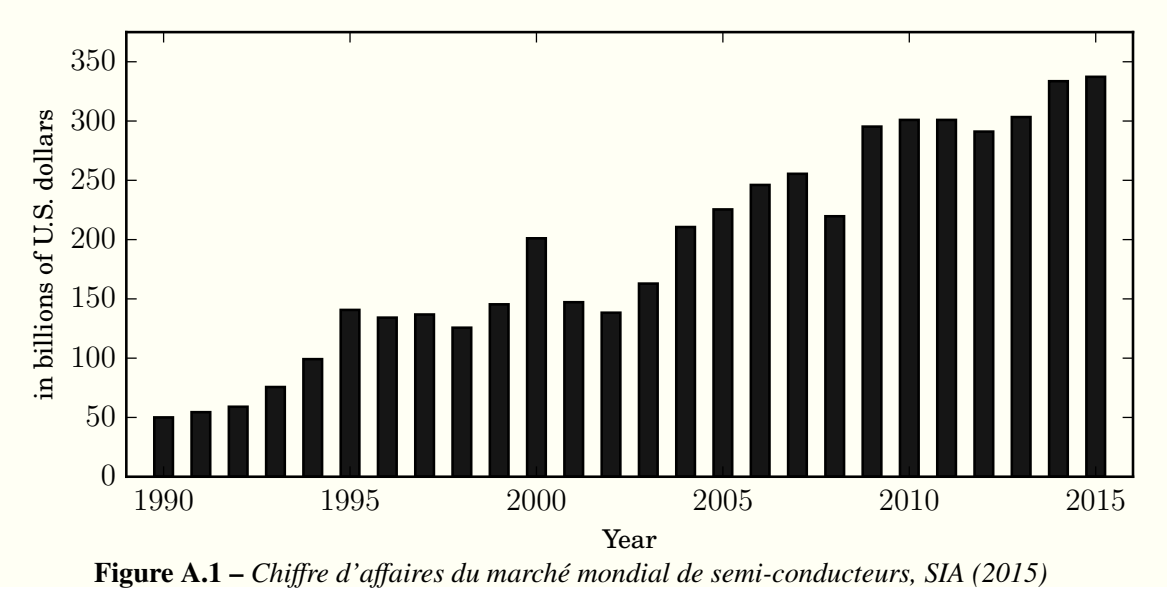


Figure A.1 – *Chiffre d'affaires du marché mondial de semi-conducteurs, SIA (2015)*

dans la fabrication de semi-conducteurs est unique. De surcroît, les résultats obtenus peuvent être transférés sur d'autres domaines d'applications de la recherche opérationnelle.

La Fabrication de Semi-Conducteurs

Cette section décrit le processus de fabrication de semi-conducteurs pour permettre au lecteur de mieux comprendre les décisions de planification à différents niveaux de décision. La fabrication de semi-conducteurs et ses principes physiques et chimiques sous-jacents sont décrits dans Quirk and Serda (2001). Des descriptions du processus de production en se concentrant sur la planification de la production et la prise de décision sont données par Uzsoy et al. (1992) et Mönch et al. (2013). En se basant sur ces sources, cette section résume le processus de planification et démontre l'utilité de l'ordonnancement dans un atelier. Comme les planificateurs d'un atelier doivent prendre en compte l'information et les décisions des ateliers voisins, il est important de comprendre les relations impliquées entre différents ateliers.

Les plaquettes brutes sont le matériau de base qui est utilisé pour produire des circuits intégrés (également connus sous le nom de puces). Une plaquette est une tranche mince du matériau semi-conducteur. Elle est obtenue à partir d'un lingot monocristallin. A partir d'une seule plaquette, des centaines ou des milliers de puces microélectroniques peuvent être produites. Ceci dépend de la taille des puces et du diamètre de la plaquette. Le diamètre des plaquettes standards a augmenté au fil du temps. Des diamètres de 200 mm ou 300 mm sont les plus utilisés actuellement. Un circuit intégré est constitué d'une plaquette revêtue d'une structure en 3 dimensions : des conducteurs, des semi-conducteurs et des isolants. Cette structure est créée en ajoutant successivement des couches sur la plaquette. Pour chacune de ces couches, plusieurs étapes de traitement sont nécessaires. Une seule puce peut contenir jusqu'à 40 couches. Un très grand nombre d'étapes de traitement est donc nécessaire, jusqu'à

700 étapes peuvent être nécessaires pour une seule plaquette. Une seule étape de production peut prendre entre quelques minutes et plusieurs heures. La production d'une seule plaquette peut prendre jusqu'à 3 mois.

La fabrication de semi-conducteurs est divisée en une étape *front-end* et une étape *back-end*. La Figure A.2 fournit un aperçu des principales phases de ces deux étapes. La partie centrale du front-end est la fabrication des plaquettes. Ceci est la partie la plus exigeante technologiquement car ici les plus petites structures des puces sont créées. La fabrication des plaquettes a lieu dans des grands bâtiments à l'intérieur des salles blanches afin d'éviter la contamination de la plaquette. Les exigences par rapport à la concentration des particules dans la salle blanche, peuvent être moins strictes dans les fabs modernes où les plaquettes sont conservées dans des conteneurs spécialisés qui assurent un environnement propre. Les détails de cette phase de production sont expliqués plus loin dans cette section. A la fin du front-end, la phase de détection à sonde teste électriquement chaque puce sur la plaquette afin d'identifier les puces qui sont éligibles pour le montage. Les sites de production du front-end sont généralement situés dans les pays hautement industrialisés. Les étapes de production du front-end représentent environ 75% de la durée de production entière. Pendant toutes les phases du front-end, les puces restent unies sur la même plaquette afin de bénéficier d'un traitement commun.

Dans l'étape du back-end, les plaquettes sont ensuite découpées en puces distinctes et l'assemblage des puces individuelles y a lieu. Ceci comprend la liaison, qui attache les puces à leurs emballages ou à d'autres puces, et le moulage, qui entoure la puce avec ses connexions dans un carter de protection. En plus, la pose des couvercles étanches, des inspections optiques, des essais environnementaux et d'autres étapes peuvent y avoir lieu. A ce stade la fabrication des circuits intégrés sur la plaquette est déjà terminée et les conditions de salle blanche sont moins strictes. Les usines du back-end sont principalement situées dans les pays à faible salaire. La phase de test final comprend des tests électriques et des tests de stress thermique qui sont effectués dans des fours.

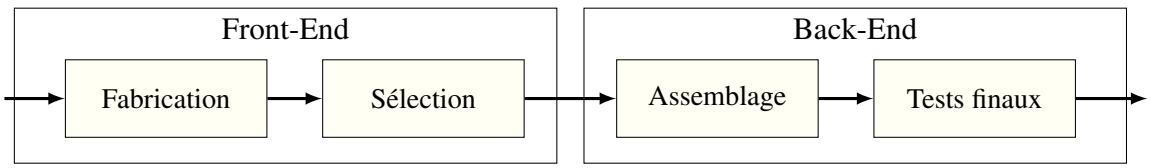


Figure A.2 – *Les étapes de fabrication de semi-conducteurs*

Dans le front-end, les structures de chaque plaquette sont construites couche par couche. Les différentes étapes de traitement nécessaires pour produire une seule couche sont effectuées dans des ateliers spécialisées. Ces ateliers se composent de machines avec des aptitudes similaires. Les différentes étapes de traitement sont exécutées plusieurs fois afin de créer toutes les couches. Ceci génère des flux rentrants car chaque plaquette visite les différentes zones de travail plusieurs fois. L'interaction entre les zones de travail est représentée dans la

Figure A.3. La liste suivante décrit les zones de travail en utilisant le classement de Mönch et al. (2011) :

Diffusion/Oxydation/Déposition Le processus de *diffusion* disperse le matériau sur la surface de la plaquette. Le processus d'*oxydation* fait développer une couche d'oxyde sur la surface d'une plaquette nettoyée. Ces couches sont ensuite modifiées par les étapes de traitement ultérieures dans le but de développer des semi-conducteurs connectés (par exemple, des transistors, des résistances ou des diodes) qui constituent le circuit intégré. Les étapes de diffusion et d'oxydation peuvent avoir des durées de traitement élevées de 12 heures ou plus. Ces étapes sont exécutées à haute température dans des fours horizontaux ou verticaux. Ces fours traitent les plaquettes par lot et peuvent traiter plusieurs lots en parallèle. Le processus de *dépôt* pose des couches conductrices ou isolantes sur la surface d'une plaquette. Ces couches servent à des fins différentes : soit elles deviennent une partie de la structure soit elles sont utilisées comme des couches auxiliaires qui sont retirées par la suite. Cet atelier contient également des machines de nettoyage, qui décontaminent les plaquettes en éliminant les particules indésirables.

Photolithographie Le processus de *photolithographie* transfère le schéma de câblage sur la surface d'une plaquette en faisant passer une lumière ultraviolette à travers un masque. Les motifs résultants sont d'abord temporaires et sont ensuite rendus permanents lors des étapes ultérieures de gravure ou d'implantation ionique. Pour la photolithographie, la machine de photolithographie ainsi qu'un *masque* (ou *réticule*) sont nécessaires. Cet atelier contient les machines les plus chères dans une fab (une machine peut coûter jusqu'à 40 millions de dollars américains) et représente souvent un goulot d'étranglement dans le processus de fabrication.

Gravure Le processus de *gravure* enlève les matières superflues de la surface de la plaquette. La gravure peut être avec ou sans masque. La gravure avec masque enlève le masque qui a été posé sur la plaquette pendant la photolithographie. La gravure sans masque réduit l'épaisseur de la plaquette et traite toute sa surface. Ils existent deux types de gravure : la gravure à sec expose la surface de la plaquette à un plasma ; la gravure humide enlève de la matière par des solutions chimiques.

Implantation Le processus d'*implantation* introduit des dopants (par exemple, des impuretés désirées ou des ions) dans la structure cristalline du matériau semi-conducteur afin de modifier sa conductivité. Cette étape peut suivre l'étape de photolithographie ou de gravure.

Aplanissement Le processus d'*aplanissement* utilise un polissage mécano-chimique afin d'aplatir la surface de la plaquette. L'aplanissement réduit les différences d'épaisseur sur une plaquette. Il est effectué avant chaque ajout d'une nouvelle couche. Cette étape évite que l'inégalité de l'épaisseur s'accumule sur plusieurs couches. Elle évite donc divers problèmes liés à l'aspérité, tels que des problèmes de focalisation de la lentille en photolithographie.

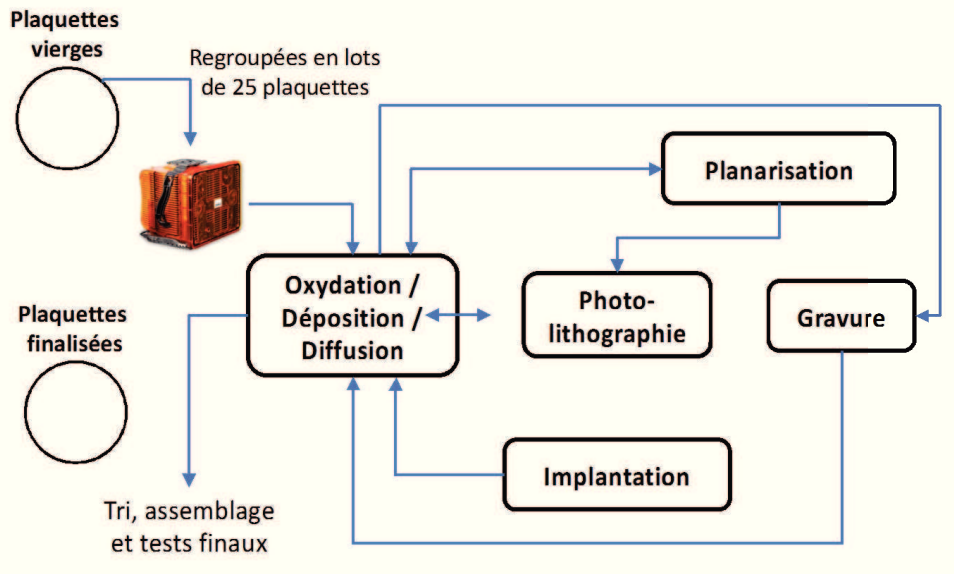


Figure A.3 – *Aperçu des étapes de fabrication de plaquettes (Mönch et al. (2011))*

Les machines des différentes zones de travail montrent des caractéristiques différentes : certaines fonctionnent avec un traitement par lot, d'autres imposent des temps de réglage, et encore d'autres permettent un chevauchement des opérations. Un ensemble de machines identiques dans un même atelier est appelé un groupe d'outils. Normalement, 25 plaquettes sont regroupées au sein d'un support qui peut soit être un contenant à ouverture frontale (Front Opening Unified Pod, *FOUP*) dans les fabs de 300 mm et les fabs modernes de 200 mm.

Pour assurer la qualité des plaquettes produites, des procédures d'inspection et de mesure sont effectuées entre les différentes étapes de production sur des lots échantillonnés. Si le contrôle détecte une plaquette endommagée, cette plaquette peut être retravaillée dans de rares cas et doit être détruite dans la plupart des cas. Les dégâts peuvent avoir différentes causes telles que la contamination, les défauts des machines ou la violation des contraintes de temps. Les contraintes de temps limitent la durée entre deux étapes de traitement et proviennent des dégradations chimiques ou physiques. Les contraintes de temps gagnent en importance avec la réduction des dimensions des structures. Les techniques de métrologie virtuelle visent à surveiller et à améliorer la qualité de la production par des moyens indirects. Les données sont collectées par des capteurs à partir des machines de production et ensuite analysées à l'aide de méthodes statistiques afin de déduire des défaillances des machines ou des problèmes de qualité. Une mesure de qualité importante est le pourcentage de puces sur une plaquette qui a été réalisé correctement. Ce pourcentage est appelé le rendement de la plaquette.

La Planification et la Prise de Décision

La planification et la prise de décision dans la fabrication de semi-conducteurs comportent plusieurs niveaux de décision : de la chaîne logistique globale à l'ordonnancement sur les postes de travail. Les champs d'application des différents niveaux de décision diffèrent par rapport à l'horizon temporel, au détail de modélisation et à la granularité des décisions à prendre. Un aperçu des niveaux de décision et des relations entre eux est donné dans Silver et al. (1998) et Stadtler and Kilger (2000). Le positionnement de l'ordonnancement dans les chaînes logistique est fourni par Rohde et al. (2000). Dans Chien et al. (2011), les défis de planification sont discutés d'un point de vue de la fabrication de semi-conducteurs. Mönch et al. (2013) distinguent les décisions au niveau de l'entreprise, au niveau de l'usine, et au niveau du poste de travail. Dans la suite, nous présentons un aperçu des décisions qui doivent être prises à chacun de ces niveaux.

La planification au *niveau de l'entreprise* comprend la planification à long terme pour un horizon de temps de plusieurs trimestres ou années. Les décisions à ce niveau sont stratégiques et sont prises pour des grandes mailles de temps qui sont habituellement des semaines ou des mois. L'envergure de cette planification est la chaîne logistique du fournisseur au client ou la chaîne logistique globale au sein de l'entreprise. Elle est basée sur la demande anticipée (les prévisions). Le plan directeur de production détermine les quantités à produire par maille de temps et par fab (ou sous-traitant). D'autres problèmes de planification à long terme englobent les décisions d'investissement pour les sites de production ou l'équipement ainsi que la gamme de produits sur le long terme. Ponsignon and Mönch (2012) décrivent et proposent un plan directeur de production qui détermine le nombre de plaquettes à produire par produit, par fab et par maille pour une demande et une capacité donnée.

La planification au *niveau de l'usine* comprend la planification à moyen terme pour un horizon de temps de quelques semaines ou mois. Son champ d'application sont les activités au sein d'une fab et les décisions sont basées sur l'état actuel de la fab et la demande confirmée. Les décisions à ce niveau sont tactiques et définissent le plan de production et la planification des stocks au niveau de la fab. Mönch et al. (2011) traitent le cas avec *plusieurs commandes par lot* où les plaquettes d'une commande ne remplissent pas entièrement le support (FOUP ou cassette). L'objectif est de regrouper des plaquettes de commandes différents dans un même support pour augmenter l'utilisation des machines. Les décisions de *lancement* déterminent le moment quand la production d'un lot est lancée afin d'assurer son flux constant et uniforme à travers de la fab. Ce but peut être obtenu par différentes mesures. Une façon est d'imposer pour chaque ordre de fabrication les dates de lancement et les échéances par atelier traversée (à plusieurs reprises). Cette approche définit à partir d'un délai client globale les échéances par atelier pour chaque ordre de fabrication. Une autre approche pour obtenir un écoulement uniforme est l'introduction d'objectifs de production. Chaque étape de production est affectée à une famille de produits. Les objectifs de production imposent pour chaque zone de travail, le nombre d'étapes par famille à exécuter dans une période de temps donnée. Les deux approches considèrent la fab de manière globale ; les délais, les objectifs ou les priorités (poids) qui en découlent sont ensuite utilisés comme données d'entrée

pour le dispatching ou l'ordonnancement. Dans ce contexte, les contraintes de temps telles que décrites dans Klemmt and Mönch (2012) s'avèrent important car ils peuvent s'étendre sur différentes zones de travail. Les décisions de *gestion de qualifications* sont nécessaires pour gérer les capacités des machines. Une opération ne peut pas être effectuée sur toutes les machines. Les machines doivent être préparées en effectuant des réglages et des essais préparatoires pour acquérir de la capacité de production. Cette préparation est appelée qualification et peut être très longue. Par conséquent, les décisions de qualification peuvent avoir un impact important comme indiqué dans Johnzén et al. (2011) et Rowshannahad et al. (2015).

La planification au *niveau de l'atelier* comprend la planification à court terme sur un horizon de temps de quelques heures. Les décisions à ce niveau sont opérationnelles et sont prises pour des mailles de temps très courtes de quelques secondes ou minutes. Les systèmes de décisions doivent réagir rapidement afin de prendre des décisions à court terme rapidement. Les décisions d'ordonnancement décident quelle machine traite un lot, quels lots sont regroupés dans un même lot plus grand, et dans quel ordre les lots sont traités sur la machine qui leur a été affectée. En pratique, les règles de dispatching sont encore souvent utilisées pour prendre ce genre de décisions. Ces règles considère souvent uniquement la prochaine opération à traiter. Beaucoup de règles de dispatching élaborées ont été développées et analysées (voir Mönch et al. (2013)). Par contraste, l'ordonnancement prend en compte un horizon de temps plus long et crée un plan de production détaillée sur plusieurs heures. Contrairement au dispatching, l'ordonnancement est moins myope et utilise généralement des méthodes d'optimisation qui proposent des solutions qui optimisent une fonction objective donnée. Les données d'entrée représentent l'état actuel de la fab et les lots à produire dans l'horizon de temps donné. Ces problèmes peuvent être décrits comme des problèmes d'ordonnancement de job-shop complexe où chaque lot correspond à un job qui doit suivre une séquence d'étapes de traitement donnée. Les contraintes considérées incluent des machines de regroupement par lot, les temps de préparation et d'autres propriétés, telles que les ressources auxiliaires. Même au niveau de l'usine, le problème d'ordonnancement pourra être modélisé de cette manière. Toutefois, pour les méthodes de planification connues, les tailles des instances d'une usine entière semblent intraitables. Les *décisions de transport* sont nécessaires dans les fabs qui utilisent des Automatized Material Handling Systems (AMHS) pour transporter les lots entre les machines à l'intérieur de la salle blanche. Les stratégies de transport visent à éviter l'arrêt des machines causés par les retards de transport (Kiba et al. (2010)). Les *outils de gestion des risques* adressent les procédures d'inspection et de contrôle effectuées entre les étapes de production. Pour des raisons de capacité, uniquement un sous-ensemble de toutes les plaquettes traitées est sélectionné pour être contrôlé. Comme le montre Rodriguez Verjan et al. (2011), ces décisions de sélection sont cruciales pour éviter de détecter les perturbations des machines trop tard. Des bonnes stratégies d'inspection évitent de faire beaucoup d'échantillonnage qui apporte peu d'informations. Uniquement les plaquettes qui minimisent le nombre de plaquettes potentiellement défectueuses sont mesurées.

La décomposition de la planification de production en sous-parties, telles que décrites précédemment, génère des tâches de planification gérables qui peuvent être traitées indépendamment. De nos jours, cette décomposition semble indispensable car les approches intégrées impliquent une complexité presque indomptable. En contrepartie, cette séparation impose des interfaces verticales et horizontales bien pensées entre les systèmes concernés. Peu de travaux considèrent cette intégration. Notamment l'intégration entre le niveau fab et les zones de travail est discutée dans Bureau et al. (2007). Ils proposent d'introduire l'information des objectifs globaux de production dans les systèmes de décision des zones de travail et de mettre à jour ces informations périodiquement. Pour ceci, ils regroupent l'information globale dans les paramètres d'entrée des répartiteurs et des ordonnanceurs. Les effets des décisions de planification locales au niveau de la zone de travail sur le niveau de la fab globale sont étudiés dans Mönch and Rose (2004). Ils utilisent la simulation pour évaluer l'impact de l'ordonnancement dans le cadre d'un horizon glissant. Plusieurs paramètres d'entrée de la planification permettent d'orienter les décisions d'ordonnancement vers les objectifs globaux. Des paramètres tels que la date de sortie, la date d'échéance, la priorité ou l'objectif de production peuvent intégrer les objectifs au niveau de l'usine.

Ils existent d'autres facteurs qui compliquent la planification de la production et l'ordonnancement. Un facteur très important est le *product mix* (gammes de produits). Nous distinguons les fabs *low-mix* et *high-mix*. Les fabs low-mix fabriquent des quantités élevées de quelques types de produits. Les fabs high-mix fabriquent de nombreux types de produits différents avec potentiellement des petites quantités pour chaque type de produit. Normalement, les fabs high-mix produisent les Application Specific Integrated Circuits (ASICs) qui sont des puces personnalisées pour des applications spécialisées. Dans ce travail, nous nous concentrons sur les fabs high-mix. Dans ce cas, beaucoup de types de produits différents à différentes étapes de production sont en concurrence pour les mêmes machines. En outre, les priorités et les dates d'échéance liées au besoin client doivent être prises en compte. Par conséquent, les décideurs doivent peser entre un grand nombre d'objectifs concurrents. Par ailleurs, les fabs high-mix nécessitent un nombre élevé de masques pour la photolithographie car un masque différent est nécessaire pour chaque motif qui est transféré sur la puce. Ainsi, ces ressources auxiliaires sont plus critiques dans les unités de production high-mix. Une complexité supplémentaire provient des propriétés spécifiques aux différentes zones de travail. Les temps de traitement varient énormément entre les différentes étapes de production et peuvent durer de quelques minutes jusqu'à plus de 12 heures. Les nombreuses propriétés des machines, telles que le traitement par lot, ont un grand impact sur la production et demandent une modélisation détaillée. L'accélération du progrès technologique dans la fabrication de semi-conducteurs ainsi que la rapidité de l'innovation dans ce secteur imposent des efforts de recherche et de développement (R&D) permanents. Cela se traduit par le besoin d'inclure les ordres de fabrication du R&D dans la planification de production. Les plaquettes du R&D sont utilisées pour le développement de produits futurs et sont traitées sur les mêmes machines que les plaquettes régulières. Par conséquent, l'activité du R&D influence fortement la capacité de fabrication de l'usine. Ziarnetzky and Mönch (2016) étudient la planification combinée des plaquettes du R&D et de la production. En outre, les étapes de production ne sont pas toujours exécutées correctement et peuvent ainsi causer des défauts

sur les plaquettes. Cela impacte la planification de différentes manières : la maintenance préventive des machines doit être planifiée et les pannes inattendues peuvent perturber le plan établi. Une approche pour représenter la fiabilité des machines d'une manière non-binaire est l'attribution d'un indicateur de santé à chaque machine. Cet indicateur estime la probabilité qu'une machine génère un défaut sur une plaquette. Doleschal et al. (2015) étudient l'évolution de ces indicateurs pour différentes règles de dispatching et d'approches d'optimisation à l'aide de la simulation. L'application de ces indicateurs dans un environnement d'ordonnacement de type job-shop est présentée dans Kao et al. (2016).

Ils existent différents indicateurs de performance. Les indicateurs à prendre en compte dépendent du problème de planification considéré et de son interaction avec les composants associés. Souvent une variété d'indicateurs de performance sont à prendre en compte. Selon un sondage présenté par Pfund et al. (2006), les entreprises considèrent le débit global de l'usine comme objectif primordial. D'autres indicateurs importants dans l'ordre décroissant sont le taux de service, le temps de cycle, le nombre de plaquettes commencées, le débit de l'équipement et le niveau de stocks . Pour analyser et résoudre les défis décrits dans cette section, une gamme d'outils et de méthodes sont disponibles et utilisées. Certains problèmes peuvent être résolus en utilisant des méthodes exactes ou des heuristiques. La *simulation* (voir Fowler and Rose (2004)) permet d'étudier les effets des décisions prises de manière très précise. Les systèmes basés sur des règles de *dispatching* sont encore largement utilisés. Une étude sur les règles de dispatching pour la fabrication de plaquettes est présentée dans Sarin et al. (2011). En pratique, ces règles peuvent être assez complexes car elles ont évolué dans le temps et prennent en compte les divers besoins spécifiques de l'entreprise. Cependant, les méthodes de planification qui utilisent des *techniques d'optimisation* obtiennent des meilleurs résultats et ont l'avantage que les objectifs peuvent être spécifiés explicitement. La *théorie des files d'attente* peut être utilisée pour prédire le comportement du système agrégé avec un effort de calcul limité. Pour analyser les temps de cycle en fonction de la charge, les courbes de débit (Fowler et al. (2001)) et les fonctions de compensation (Karmarkar (1989)) peuvent être utilisées.

Ordonnancement des Ateliers de Diffusion et de Nettoyage

Cette thèse adresse des problèmes d'ordonnancement qui proviennent des ateliers de diffusion et de nettoyage des usines de fabrication de semi-conducteurs de STMicroelectronics à Rousset (France) et à Crolles (France). La zone de travail que nous considérons comprend une grande variété de machines. Elle comprend non seulement la diffusion et les fours d'oxydation, mais aussi d'autres types de machines telles que le nettoyage ou les bancs humides. Cette zone de travail a un grand impact sur la performance globale de la fab puisque jusqu'à 30% de l'encours peuvent se retrouver dans cette zone (Jung et al. (2013)). Nous présentons d'abord les principales caractéristiques du problème d'ordonnancement qui sont imposées par cette zone de travail.

Le problème de base est un problème d'ordonnancement de type *complex job-shop* où les machines sont considérées dans un cadre de type job-shop. Pour chaque ordre de fab-

rication, une séquence fixe d'opérations (une gamme) doit être executée sur un ensemble de machines données. Chaque opération appartient à une famille qui précise les machines qui sont qualifiées pour traiter l'opération. La durée du traitement dépend de la machine sélectionnée. Certaines machines sont capables de faire un traitement par lot. Ils peuvent traiter plusieurs opérations de la même famille en parallèle en respectant la capacité de la machine (p-batching). Les temps de réglage de certaines machines dépendent de la séquence des opérations sur cette machine (s-batching). A chaque ordre de fabrication est associé une date de début au plus tôt et potentiellement aussi une date d'échéance. Dans un premier temps, nous nous intéressons à des fonctions objectives régulières avec un seul critère tel que le retard ou la durée totale de production pondérée.

Les machines complexes sont souvent modélisées par une durée de traitement fixe qui néglige de nombreuses contraintes présentes dans une vraie fab. Nous illustrons ceci pour le cas des fours. Un four est généralement constitué de deux tubes, de quatre nacelles (deux par tube) et d'un port de chargement. Le tube est l'endroit où le processus est exécuté. La nacelle est un support mobile pour les plaquettes et les accompagne à l'intérieur du tube. Les nacelles sont également utilisés pour charger, décharger et refroidir les plaquettes. Le port de chargement sert à charger et décharger les plaquettes de la machine. Le fonctionnement détaillé est le suivant. Tout d'abord, les plaquettes sont chargées de leur support sur la nacelle au port de chargement. Puis, la nacelle est déplacé dans le tube où se déroule le processus. Ensuite, la nacelle est retiré du tube et doit refroidir avant que les plaquettes peuvent être déchargées au port de chargement. Si le port de chargement est occupé, la nacelle doit attendre qu'il se libère. Certaines opérations exigent plus d'une ressource interne en parallèle. Le modèle simple d'ordonnancement de type complex job-shop suppose que chaque tube correspond à une machine qui fonctionne de façon indépendante. Nous avons vu que cela n'est pas le cas et nous allons proposer une modélisation plus détaillée des machines qui peut être intégrée dans l'ordonnancement de type job-shop et ainsi augmenter la précision du modèle.

En pratique les contraintes de temps sont cruciales. Les processus chimiques et physiques imposent des *délais maximaux* qui limitent le temps passé entre deux étapes de production. Les contraintes de temps peuvent avoir lieu entre différentes opérations d'un même ordre de fabrication. Les délais peuvent s'appliquer sur des opérations adjacentes, peuvent se chevaucher et une même opération peut apparaître dans plusieurs contraintes. Cela peut causer des contraintes enchaînées. Ils existent deux types de contraintes. Dans certains cas, les plaquettes qui ne respectent pas le délai maximal peuvent être retravaillées. Dans d'autres cas, les défauts ne sont pas réparables et les plaquettes deviennent du rebut si les délais ne sont pas respectés. Nous allons inclure les contraintes de temps dans notre modèle pour ainsi augmenter sa précision.

A.2 Ordonnancement pour les problèmes de type Complex Job-Shop

Ce chapitre aborde une version industrielle du problème d'ordonnancement et présente notre approche qui a été publiée dans Knopp et al. (2015a) et Knopp et al. (2015b). Le problème d'ordonnancement industriel que nous traitons peut être considéré comme un problème d'ordonnancement de type job-shop avec un large éventail de contraintes et de propriétés. Car les machines avec traitement par lot constituent la principale caractéristique du problème, ce chapitre se concentre sur ces machines dans un environnement de type job-shop. Le traitement par lot en combinaison avec des contraintes supplémentaires, conduit à une problématique d'ordonnancement de type complex job-shop. Ceci peut aussi être décrit comme un problème d'ordonnancement de type flexible job-shop avec p-batching, des flux rentrants, des temps de réglage qui dépendent de la séquence des opérations et des dates de sortie. Le problème d'ordonnancement de type complex job-shop est très présent dans la littérature et aussi appliqué dans le cadre de la fabrication de semi-conducteurs. Nous étudions le problème avec une fonction objective régulière avec un seul critère. En fonction du contexte, nous considérons le facteur d'écoulement pondéré (tel que défini dans la spécification industrielle) ainsi que d'autres objectifs de la littérature académique tels que la durée totale de production, le retard total pondéré ou le retard maximum.

Le problème considéré est NP-difficile, car il généralise d'autres problèmes NP-difficiles tels que le problème d'ordonnancement de type job-shop classique et le problème d'ordonnancement avec une seule machine et le retard pondéré comme fonction objective (voir Garey et al. (1976)). Notre objectif est de résoudre des instances industrielles avec environ 100 machines et des centaines d'ordres de fabrication dont chacun peut comprendre jusqu'à dix opérations. Nous développons une méthode heuristique puisque nous voulons résoudre des grandes instances d'un problème NP-difficile dans un temps de calcul raisonnable. L'ordonnancement des machines avec traitement par lot (p-batching) et des variantes du problème d'ordonnancement de type job-shop sont déjà étudiés séparément alors que leur combinaison est rarement considérée. La plupart des approches existantes pour les problèmes de type complex job-shop ou le p-batching sont basées sur la représentation du problème de Ovacik and Uzsoy (1997) et utilisent un graphe disjonctif. Cette représentation utilise des nœuds dédiés pour représenter les décisions de regroupement par lot. Nous proposons une nouvelle modélisation du regroupement par lot qui n'as pas besoin de nœuds spécifiques pour représenter ce regroupement. Nous représentons les décisions de regroupement par les poids des arêtes afin de réduire la complexité de la structure du graphe et de faciliter les modifications du graphe.

Pour profiter pleinement de cette nouvelle représentation du regroupement par lot, nous présentons un nouveau algorithme qui calcule les dates de début des opérations ainsi que le regroupement par lot. Grâce à cette nouvelle représentation, notre algorithme peut prendre des décisions de regroupement à la volée quand il parcourt le graphe. L'algorithme utilise une stratégie combinée de reséquencement et de réaffectation des opérations aux lots ainsi

que le déplacement d'opérations individuelles. Ceci permet de remplir des lots non-remplis. La combinaison du séquencement et des décisions de regroupement représente implicitement différents mouvements spécifiques décrits dans la littérature tels que l'échange de lots. Cette approche intégrée permet d'évaluer trés rapidement trés nombreux mouvements.

Nous appliquons notre algorithme au sein d'une approche basée sur le GRASP (Feo and Resende (1995)). Nous créons des solutions initiales en insérant successivement les ordres de fabrications dans un ordre aléatoire. Les solutions sont ensuite améliorées en utilisant une heuristique de type recuit simulé. Notre algorithme est parallélisé et peut utiliser tous les cœurs de la machine. Nous obtenons de très bons résultats avec notre approche pour différents types d'instances. Ceci montre la généralité et l'utilité de notre approche. Notre approche est conçue pour pouvoir intégrer d'autres contraintes facilement. Nous pensons en particulier aux contraintes figurant dans le cahier des charges du problème industriel. Quelques extensions de l'approche sont décrites dans les chapitres suivants.

Modélisation par Graphe Disjonctif

Les graphes disjonctifs, introduits par Roy and Sussmann (1964), permettent de représenter la séquence et les dates des opérations sur les machines de manière concise. Ils ont été appliqués pour résoudre un large éventail de problèmes d'ordonnancement. Pour pouvoir inclure le regroupement par lot dans un environnement de type job-shop, nous introduisons un graphe disjonctif qui représente le regroupement sans introduire de nouveaux nœuds (batch-oblivious). Ceci permet de définir le regroupement par lot lors de la traversée du graphe.

Nous présentons d'abord l'utilisation du graphe disjonctif pour les problèmes d'ordonnacement du type complex job-shop. Puis, deux modélisations alternatives pour représenter les décisions de regroupement sont décrites. Nous discutons d'abord une représentation établie qui insère des nœuds de regroupement dédiés dans le graphe (voir Ovacik and Uzsoy (1997)). Ensuite, nous introduisons une nouvelle représentation où le regroupement est défini par le poids des arêtes et aucun nœud supplémentaire est introduit. Cette nouvelle représentation permet de prendre les décisions de regroupement à la volée pendant la traversée du graphe. Cette idée est le point fondamental de l'approche d'ordonnancement proposée dans ce chapitre.

Les graphes disjonctifs représentent toutes les possibilités de production mais ne représentent pas une solution spécifique (ni une affectation à la machine, ni une séquence sur la machine, ni un regroupement par lot). Les graphes conjonctifs représentent des solution spécifiques avec toutes les décisions à prendre. Ils sont l'outil principal de nos algorithmes. Nous commençons par décrire ces deux graphes brièvement. Dans les deux types de graphe, chaque nœud représente une opération et chaque arête représente une dépendance induite soit par la gamme soit par la séquence des opérations sur une même machine.

Un graphe conjonctif est construit à partir du graphe disjonctif en remplaçant les arêtes non-orientées par des arêtes orientées en respectant les contraintes de faisabilité. Le graphe conjonctif représente l'affectation des opérations aux machines et la séquence des opérations sur une machine. Les arêtes redondantes sont supprimées du graphe conjonctif. Maintenant,

nous définissons la représentation du graphe conjonctif que nous utilisons dans la suite. Cette représentation du graphe conjonctif correspond à celle proposée par Dauzère-Pérès and Paulli (1997). Elle ne représente pas encore les décisions de regroupement qui seront introduites plus tard.

Un graphe conjonctif $G = (V, E)$ est un graphe orienté acyclique avec des nœuds $V = O \cup \{0, *\}$ qui correspondent aux opérations O, un nœud de départ artificiel 0 et un nœud de fin artificiel $*$. Pour chaque ordre de fabrication et pour chaque machine, le graphe contient un chemin à partir du nœud de départ artificiel 0 au nœud de fin artificiel $*$. L'union disjointe de ces chemins donne toutes les arêtes du graphe. Chaque nœud $v \in O$ fait partie d'exactement deux chemins : l'un représente la gamme de son ordre de fabrication et l'autre la séquence des opérations sur la machine à laquelle il est affecté. Pour un nœud $v \in O$, on définit son *successeur de gamme* par $r(v) \in V \setminus \{0\}$ et son *successeur de machine* par $m(v) \in V \setminus \{0\}$. De la même manière, ses prédécesseurs sont désignés par $r^{-1}(v) \in V \setminus \{*\}$ et $m^{-1}(v) \in V \setminus \{*\}$. Le nœud de départ artificiel 0 a $|J| + |M|$ arêtes sortantes et pas d'arêtes entrantes. De la même manière, le nœud de fin artificiel $*$ a $|J| + |M|$ arêtes entrantes et pas d'arêtes sortantes. En total, le graphe contient $|E| = 2|O| + |J| + |M|$ arêtes.

Le graphe conjonctif peut être utilisé pour déterminer les dates de début S_v des opérations $v \in O$. Un poids $l_{u,v} \in \mathbb{N}_0$ est affecté à chaque arête $(u, v) \in E$. Ce poids impose une durée minimale entre le début de deux opérations adjacentes : $S_u + l_{u,v} \leq S_v$ pour chaque arrête $(u, v) \in E$. Avec le poids des arêtes, les dates de début des opérations correspondent au plus long chemin entre le nœud de départ artificiel et le nœud de l'opération.

Soit $L(v, w) \in \mathbb{N}_0$ la distance d'un *plus long chemin* d'un nœud $v \in V$ vers un nœud $w \in V$. Pour chaque opération $v \in O$, sa date de début est déterminée par $S_v = L(0, v)$. Pour tenir compte des contraintes données, on définit le poids des arêtes de la manière suivante. Pour les arêtes $(0, o_{1,j}) \in E$ qui relient le nœud de départ artificiel 0 avec la première opération $o_{1,j}$ de l'ordre de fabrication j, le poids de l'arête est fixé à la date de sortie r_j de l'ordre de fabrication j. Pour les arêtes $(0, o_m) \in E$ qui relient le nœud de départ artificiel 0 avec la première opération o_m prévue sur la machine $m \in M$, le poids de l'arête est fixé à zéro. Pour les arêtes des gammes $(v, r(v)) \in E$ avec $v \neq 0$, le poids de l'arête correspond à la durée du traitement p_v de l'opération v. Pour les arêtes de séquence machine $(v, m(v)) \in E$ avec $v \neq 0$, le poids de l'arête correspond à la somme $p_v + s(\sigma_v, \sigma_{m(v)})$ qui représente la durée du traitement de l'opération v et le temps de réglage qui dépend de l'opération v et de son successeur machine $m(v)$. Ceci représente les machines sans traitement par lot.

Ils nous reste à proposer une représentation pour les machines avec regroupement par lot. Ces machines peuvent être modélisées en modifiant la structure du graphe ou en adaptant les poids des arêtes. Les deux paragraphes suivants présentent les deux alternatives. Nous rappelons que chaque regroupement doit respecter la capacité de la machine et doit se faire uniquement par famille. Ces deux contraintes doivent toujours être satisfaites. Nous ne détaillons pas les contrôles connexes pour vérifier la faisabilité d'une solution car nous souhaitons nous concentrer sur les parties essentielles des deux représentations.

Graphes Conjonctifs de type Batch-Aware

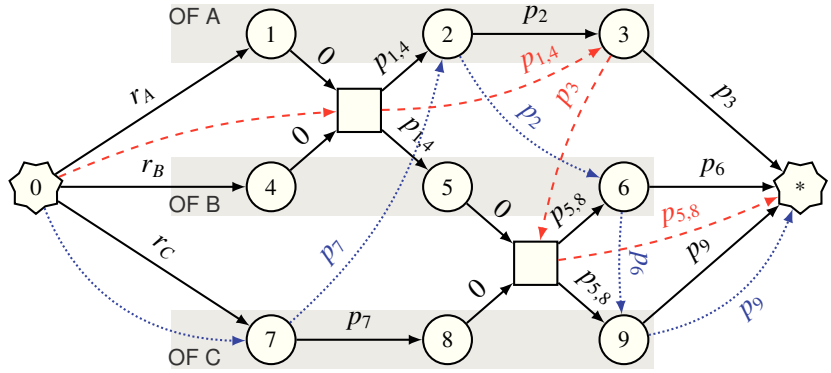
Cette section examine une représentation du regroupement par lot via un graphe conjonctif de type batch-aware qui a été présentée par Ovacik and Uzsoy (1997). Toutes les approches pour résoudre les problèmes d'ordonnancement de type complex job-shop dont nous nous sommes conscientes utilisent ce type de représentation (p. ex., Mason et al. (2005), Mönch et al. (2003) ou Yugma et al. (2012)).

Un *lot* est un ensemble d'opérations $B \subset O$ qui sont traitées simultanément sur la même machine. Dans une représentation de type batch-aware, un nœud supplémentaire b est ajouté au graphe pour chaque lot. La date de début d'un nœud de regroupement représente la date de début commune à toutes les opérations d'un même lot. Le traitement du lot ne peut commencer que quand toutes ses opérations sont prêtes. Pour ceci, chaque nœud d'une opération $v \in B$ est relié au nœud de regroupement b via une arête (v, b) avec un poids égal à zéro. Les opérations suivantes des gammes sont reliées de la manière suivante : pour chaque opération $v \in B$ du lot, une arête $(b, r(v))$ est introduite qui part du nœud de regroupement b vers le successeur de gamme $r(v)$ de v. Ces arêtes ont la durée de traitement p_v comme poids. Deux autres arêtes $(m^{-1}(b), b)$ et $(b, m(b))$ sont introduites pour placer le lot dans la séquence de sa machine M_B. Comme dans le cas sans regroupement, le poids de chaque arrête de séquence machine (u, w) est défini par la somme $p_u + s(\sigma_u, \sigma_w)$. Chaque nœud d'une opération $v \in B$ a exactement une arête d'entrée et une arête de sortie. Le nœud de regroupement b a $|B| + 1$ arrêtes entrantes et $|B| + 1$ arrêtes sortantes.

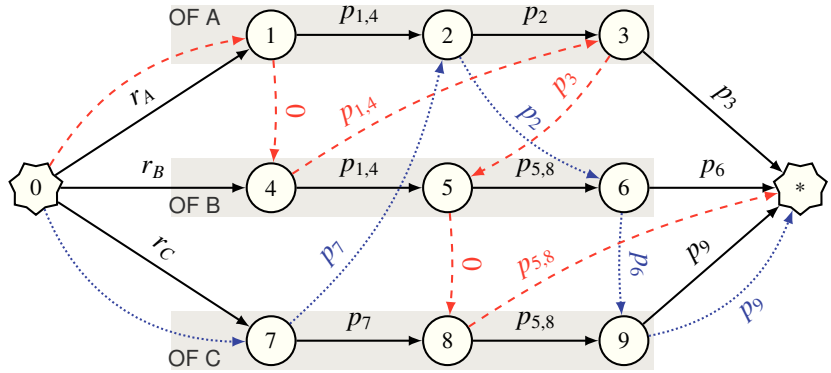
Les graphes conjonctifs de type batch-aware représentent les dépendances qui résultent des décisions de regroupement par des nœuds supplémentaires. Ceci implique une modification de la structure du graphe. Le nombre de nœuds dans ces graphes dépend du nombre de lots. Avec cette modélisation, une modification des décisions de regroupement est difficile à gérer : le nombre de nœuds dans le graphe doit être adapté et plusieurs arêtes doivent être manipulées en s'assurant que l'acyclicité du graphe est préservée.

Graphes Conjonctifs de type Batch-Oblivious

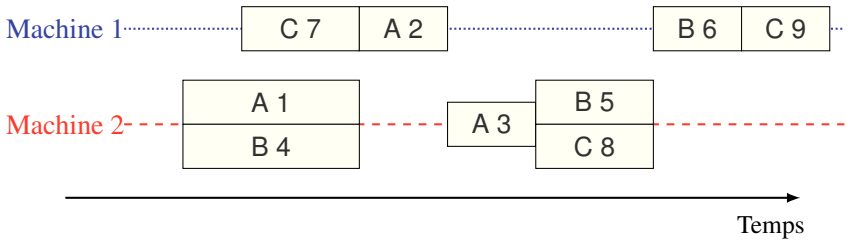
Nous introduisons une nouvelle représentation pour les décisions de regroupement par lot dans un graphe conjonctif qui est non-intrusive car elle ne modifie pas la structure du graphe. Nous n'introduisons aucun nœud de regroupement dédié et la représentation de base présentée au début de cette section reste valable. Nous représentons les décisions de regroupement en adaptant les poids des arêtes machines $(v, m(v)) \in E$. Le poids d'une arête machine est égale à zéro, si les opérations adjacentes doivent être traitées dans un même lot. Sinon, le poids de l'arête est égale à $p_v + s(\sigma_v, \sigma_{m(v)})$, comme dans le cas sans regroupement par lot. Malheureusement, ceci n'est pas aussi simple : $L_{v,m(v)} = 0$ ne garantit uniquement que $S_v \leq S_{m(v)}$ mais pas que $S_v = S_{m(v)}$. Nous pouvons ainsi obtenir des solutions infaisables car toutes les opérations d'un lot doivent être égales. Pour ceci, nous avons ajouté un critère simple qui vérifie la faisabilité de la solution.



(a) *Graphe conjonctif de type batch-aware*



(b) *Graphe conjonctif de type batch-oblivious*



(c) *Gantt Chart*

Figure A.4 – *Comparaison des représentations d'un même ordonnancement*

La Figure A.4 montre un exemple qui compare les représentations des regroupements par lot d'un graphe de type batch-aware et d'un graphe de type batch-oblivious. Elle décrit un plan de production avec trois ordres de fabrications A, B et C qui utilisent deux machines. La machine 2 (trait rouge) traite deux lots. Chacun est composé de deux opérations : le premier de l'opération 1 et 4 et le deuxième de l'opération 5 et 8. Pour des raisons de lisibilité, les temps de changement ont été omis et nous obtenons $p_{1,4} = p_1 = p_4$ et $p_{5,8} = p_5 = p_8$.

A.3 Modélisation des Ressources et du Routage

Lors de l'introduction, nous avons vu que beaucoup de machines différentes sont utilisées dans la zone de diffusion et de nettoyage. Chaque machine a ses propriétés spécifiques et le comportement global de la machine est souvent déterminé par le fonctionnement de ses composants internes. Nous proposons une modélisation détaillée de ces machines pour les problèmes d'ordonnancement de type job-shop afin de réduire l'écart entre le modèle et la réalité. Pour obtenir un ordonnancement réaliste, nous représentons les différents composants des machines (des fours en particulier) comme ressources individuelles dans notre modèle. Nous obtenons ainsi une version généralisée du problème d'ordonnancement de type complex job-shop avec des contraintes supplémentaires qui représentent l'utilisation des ressources internes des machines.

Nous illustrons l'idée et l'utilité de notre approche au travers des fours. Les fours de la zone de diffusion et de nettoyage se composent généralement de deux tubes, de quatre nacelles (deux par tube) et d'un port de chargement. Le tube est l'endroit où le processus est exécuté. La nacelle est un support mobile pour les plaquettes et les accompagne à l'intérieur du tube. Les nacelles sont également utilisés pour charger, décharger et refroidir les plaquettes. Le port de chargement sert à charger et décharger les plaquettes de la machine. Le fonctionnement détaillé est le suivant. Tout d'abord, les plaquettes sont chargées de leur support dans la nacelle au port de chargement. Puis, la nacelle est déplacée dans le tube où se déroule le processus. Ensuite, la nacelle est retirée du tube et doit refroidir avant que les plaquettes soient déchargées sur le port de chargement. Si le port de chargement est occupé, la nacelle doit attendre qu'il se libère. La Figure A.5 montre une représentation schématique d'un four. Certaines opérations, tel que le chargement des plaquettes, exigent plus d'une ressource interne. Chaque étape de traitement du four est décomposée dans ses opérations distinctes qui utilisent les différentes ressources internes telles que décrites ci-dessus. Par conséquent, le temps de traitement sur une machine peut dépendre du choix de ses ressources internes.

Les propriétés des machines décrites ci-dessus font que les ressources internes choisies pour une opération imposent l'utilisation de ressources internes données dans la suite. Nous illustrons ceci par deux exemples. Premièrement, si une opération de chargement utilise le port de chargement d'une machine, l'opération de déchargement doit utiliser le port de chargement de la même machine. Deuxièmement, si une nacelle est chargée, elle ne peut pas être utilisée ailleurs avant d'être déchargée. Ainsi, une ressource peut être bloquée même si aucune opération ne l'utilise actuellement.

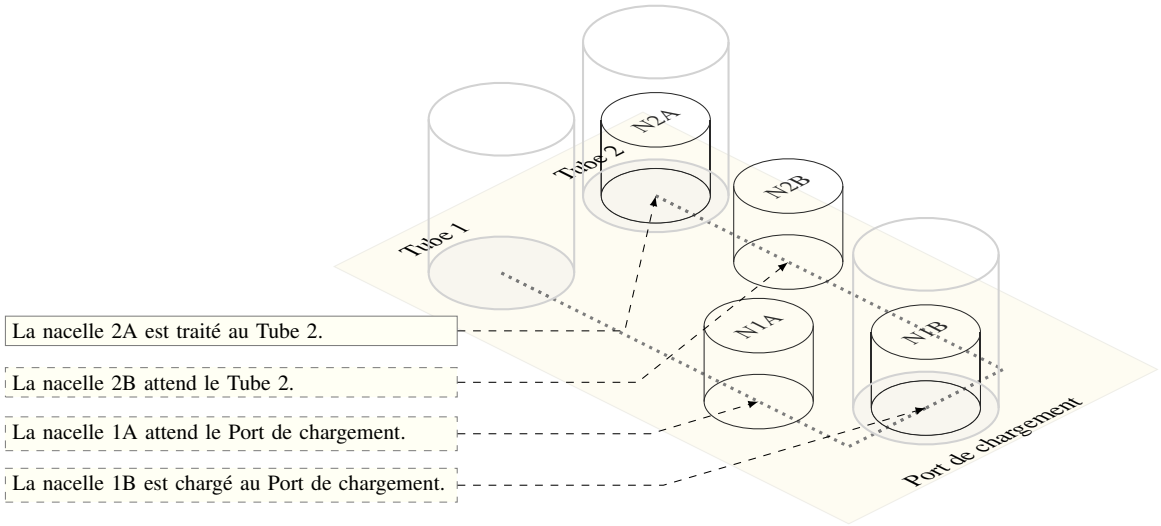


Figure A.5 – *Représentation schématique d'un four de la zone de diffusion et de nettoyage*

Notre approche se base sur le problème d'ordonnancement de type flexible job-shop. Dans ce cas, un ensemble d'ordres de fabrications doit être planifié sur un ensemble de ressources données. Pour chaque ordre de fabrication, une gamme spécifique d'opérations doit être exécutée. Elle permet de choisir une ressource à utiliser parmi un ensemble de ressources alloué à chaque opération. Pour inclure les contraintes représentant le fonctionnement des fours, nous proposons une extension du problème et une représentation comme problème d'ordonnancement de type complex job-shop. Cette extension représente les dépendances des ressources internes en définissant toutes les routes possibles qui résultent des différentes affectations possibles de ressources internes. Pour ceci, nous introduisons le concept d'un graphe de routage. Pour chaque ordre de fabrication, le graphe de routage affecte les ressources aux opérations de manière statique. La flexibilité du choix des ressources est obtenue par la représentations de toutes les routes possibles.

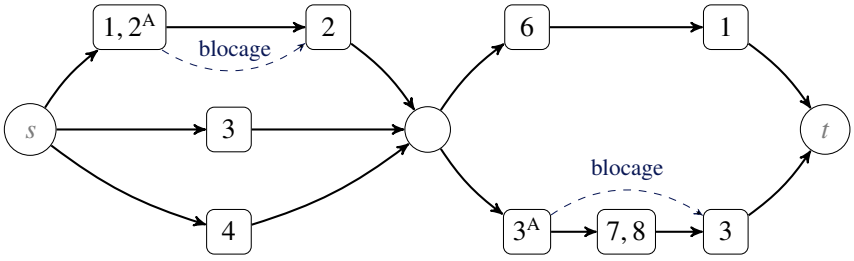


Figure A.6 – *Exemple d'un graphe de routage pour un ordre de fabrication*

La Figure A.6 présente l'exemple d'un graphe de routage pour un ordre de fabrication avec blocage des ressources. Ce graphe permet six routes différentes entre s et t : trois alternatives pour la première partie et deux alternatives pour la deuxième partie. Les nœuds représentent les opérations et les noms des ressources requises. Le blocage des ressources est indiqué par l'exposant "A" et les arcs pointillés de l'opération d'occupation jusqu'à l'opération de libération. Ces arcs servent uniquement pour l'illustration et ne font pas partie du graphe de routage.

Les itinéraires autorisés sont spécifiés par graphes d'itinéraire. Cette modélisation est indépendante de la fonction objective et peut être combinée avec tous les critères réguliers. Nos algorithmes d'ordonnancement se basent sur la représentation du problème comme graphe disjonctif. L'ordonnancement des opérations se fait via l'insertion des nœuds dans ce graphe. Pour ceci, nous utilisons une approche méta-heuristique de type GRASP.

A.4 Modélisation des Contraintes de Temps

Les chapitres précédents ont présenté différentes extensions du problème d'ordonnancement de type complex job-shop pour prendre en compte les spécificités des machines. Nous avons d'abord intégré le traitement par lot et les temps de réglage. Ensuite, le modèle a été généralisé pour représenter les machines avec leurs ressources internes : décisions de routage et utilisation de plusieurs ressources par opération. Ce chapitre vise à étendre davantage le champ d'application en incluant les contraintes de temps. Cette extension introduit une fonction objective supplémentaire mais ne modifie pas les contraintes. Les contraintes de temps sont un aspect important dans l'atelier de diffusion et de nettoyage. Elle représentent un délai maximal entre l'exécution de deux opérations. Elles sont sont cruciales et définissent si le plan de production est applicable en pratique. En particulier, les contraintes de temps enlacées doivent être prises en compte. L'importance des contraintes de temps augmente avec la miniaturisation des dimensions structurelles des dispositifs semi-conducteurs. L'importance particulière de ces contraintes dans la zone de diffusion est mise en évidence par Jung et al. (2013).

Les processus chimiques et physiques dans la zone de diffusion et de nettoyage imposent des contraintes de *temps* qui représentent le délai maximal (temps d'attente et temps de traitement) entre deux opérations d'un même ordre de fabrication. Ces délais s'appliquent souvent après le processus de nettoyage car les conditions chimiques sur la surface de la plaquette se détériorent au fil du temps. Les contraintes de temps peuvent être adjacentes ou se chevaucher. Ainsi, un délai maximal peut implicitement déclencher un autre. Cet entrelacement des délais est également appelé "tunnels de contraintes de temps" dans la terminologie industrielle. Par rapport à la classification des contraintes de temps présentée dans Klemmt and Mönch (2012), nous considérons les contraintes les plus générales : contraintes de temps avec chevauchement et contraintes de temps entre opérations non-adjacentes.

Nous distinguons les délais *récupérables* et les délais *non-récupérables*. Si une contrainte de temps *récupérable* est violée, le lot doit être retravaillé. Les opérations de reprise sont à

éviter car ils augmentent le temps de cycle et demandent de la capacité supplémentaire sur les machines. Si une contrainte de temps *non-récupérable* est violée, la probabilité que les plaquettes soient défectueuses augmente avec la durée du retard. Des mesures supplémentaires doivent être effectuées après avoir atteint le délai maximal des contraintes de temps non-récupérables pour évaluer la qualité des plaquettes. Selon le résultat des mesures, les plaquettes sont du rebut ou non. La destruction des plaquettes coûte très cher : non seulement à cause de la perte du matériel, mais aussi en raison de l'inutilité des étapes exécutées jusque là.

En pratique, l'ordonnancement se fait sur un horizon glissant. Au moment où un ordonnancement est calculé, plusieurs opérations d'un ordre de fabrication peuvent déjà être réalisées ou être en cours d'exécution. Uniquement les opérations non-commencées peuvent être planifiées dans le temps. Nous appelons *délai initié* le délai maximal entre une opération qui a commencé dans le passé et une opération du même ordre de fabrication qui n'a pas encore commencé. Les ordre de fabrications sans délai initié peuvent toujours être planifiés en respectant les contraintes de temps car toutes les opérations peuvent être décalées dans le temps. Mais une contrainte de temps avec délai initié impose toujours une date de fin fixe pour son opération de fin. Comme les contraintes de temps peuvent être adjacentes ou se chevaucher, il est aussi possible que certaines opérations sans délai initié ne peuvent pas être retardées indéfiniment. Par conséquent, nous ne pouvons pas garantir que toutes les contraintes de temps d'un ordre de fabrication commencé soient satisfaites. De plus, l'industriel a besoin d'un ordonnancement même si les contraintes de temps ne peuvent pas toutes être satisfaites. Pour ceci, nous utilisons des contraintes de temps souples. C'est à dire que nous utilisons une fonction objective lexicographique où la minimisation des contraintes de temps est l'objectif principal. L'objectif secondaire est un critère régulier qui mesure la performance de l'ordonnancement hormis les contraintes de temps.

Puisque nous tentons de minimiser les violations des contraintes de temps, nous avons besoin de quantifier les violations des délais pour un ordonnancement donné. Pour chaque contrainte de temps nous mesurons l'envergure du retard. L'envergure est un nombre réel qui est égal à zéro si la contrainte est satisfaite et supérieur à zéro si elle ne l'est pas. L'*envergure globale* pour un ordonnancement donné est la somme des envergures de toutes les contraintes de temps. Cette somme constitue la première composante de notre fonction objective lexicographique. Si l'envergure globale est nulle, nous avons trouvé un ordonnancement faisable qui respecte toutes les contraintes de temps.

Si une contrainte de temps n'est pas satisfaite, son envergure dépend du type de la contrainte (récupérables / non-récupérables) et de la durée du retard. Pour les contraintes récupérables, un coût de reprise constant s'applique si la contrainte n'est pas satisfaite. Ceci représente le fait, que les operations de reprise à faire ne dépendent pas de la durée du retard. Pour les contraintes non-récupérables, la durée du retard est importante. La probabilité que les plaquettes soient mises au rebut augmente avec la durée du retard. Par conséquent, les gros retards doivent être évités tandis que les petits retards peuvent être tolérés. Nous proposons de pénaliser les retards quadratiquement, car c'est une façon simple d'inclure les cas décrits ci-dessus. Cette approche est similaire à la méthode des moindres carrés qui est

une approche standard dans l'analyse de régression et remonte au moins à Legendre (1805). Cependant, si le retard devient trop grand, toutes les plaquettes doivent être mis au rebut. Par conséquent, nous introduisons un coût maximal qui s'applique si le retard est supérieur au retard maximal toléré. De cette manière, l'envergure peut être considérée comme une estimation de la perte du rendement des plaquettes. Veuillez notez qu'il n'est pas logique d'appliquer un coût de rebut plusieurs fois pour une même plaquette. Cependant, nous avons omis cela dans notre définition de l'envergure pour des raisons de simplicité. Nous fournissons une formulation généralisée qui définit l'envergure du retard pour les deux types de contraintes de temps d'une manière uniforme.

Figure A.7 illustre l'envergure d'une contrainte de temps dans les deux cas. Nous définissons $d \in \mathbb{N}_{\geq 0}$ comme délai maximal entre deux opérations, $\gamma \in \mathbb{N}_{\geq 0}$ avec $\gamma \geq d$ comme retard maximal toléré et $c \in \mathbb{R}_{>0}$ comme coût de non-respect d'une contrainte de temps. Pour les contraintes récupérables, nous avons $d = \gamma$ et un coût de violation constant qui est appliqué si le délai maximal n'est pas respecté indépendamment de la durée du retard. Pour les contraintes non-récupérables, nous avons $d < \gamma$ et le coût de violation augmente quadratiquement avec la durée du retard entre d et γ. Si le retard maximal toléré est atteint, toutes les plaquettes sont mis au rebut et le coût n'augmente plus. Veuillez notez que cette fonction objective n'est pas régulière. Si une opération de départ d'une contrainte de temps est avancée ceci peut augmenter l'envergure de retard.

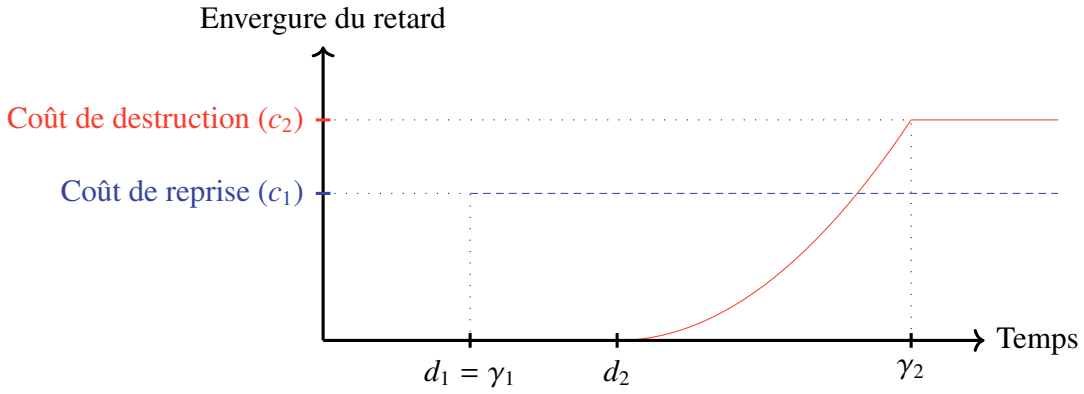


Figure A.7 – *Exemple de l'envergure pour une contrainte récupérable τ_1 et une non-récupérable τ_2*

A.5 Conclusion et Perspectives

Dans cette thèse, nous avons présenté des modèles et des méthodes d'optimisation qui prennent en compte les contraintes et les objectifs variés de la zone de diffusion et de nettoyage d'une usine de fabrication de semi-conducteurs. Nous avons vu que les propriétés sous-jacentes de ce domaine d'application spécifique génèrent des problèmes d'ordonnancement

très généraux. Nous avons proposé une approche méta-heuristique pour résoudre ces problèmes d'ordonnancement. L'approche est basée sur une représentation du problème avec ses différentes contraintes comme un graphe conjonctif qui intègre les décisions de regroupement de manière implicite.

Conclusion

Le problème d'ordonnancement industriel est très complexe et impose un riche ensemble de contraintes qui peut difficilement être abordé dans sa totalité. Par conséquent, cette thèse adresse d'abord le problème d'ordonnancement de type complex job-shop qui couvre les caractéristiques fondamentales de notre problème d'ordonnancement industriel. Le point capital est la présence de machines avec traitement par lot dans un environnement d'ordonnancement de type job-shop. Comme les approches existantes dans la littérature, nous abordons le problème en utilisant une approche heuristique basée sur les graphes conjonctifs. Nous introduisons une nouvelle représentation qui permet de représenter les décisions de regroupement de manière implicite (batch-oblivious). Ceci permet de réduire la complexité structurelle du graphe conjonctif par rapport aux graphes qui représentent les décisions de regroupement de manière explicite (batch-aware). Cette nouvelle représentation permet de calculer les dates de début, de prendre les décisions de regroupement et de remplir des lots sous-utilisés lors de la traversée du graphe. La représentation implicite des décisions de regroupement permet d'adresser les problèmes de regroupement, d'affectation et d'ordonnancement en parallèle. La représentation implicite des décisions de regroupement facilite la généralisation de l'approche en permettant d'intégrer des contraintes supplémentaires. La représentation est indépendant de la méta-heuristique utilisée. Nous appliquons les blocs de construction dans une approche heuristique de type GRASP parallélisée. Notre implémentation permet d'exploiter le nombre croissant de processeurs. De bons résultats numériques pour un large éventail de types d'instances de référence valident la performance de notre approche.

Afin d'augmenter le détail de notre modèle d'ordonnancement, nous intégrons la gestion des composants internes des machines. Pour ceci nous introduisons une flexibilité par rapport au choix des ressources internes et des routes dans un problème d'ordonnancement de type complex job-shop. Notre approche de représentation implicite des décisions de regroupement facilite la généralisation du graphe conjonctif pour intégerer ces nouveaux éléments. Une contrainte d'acquisition de ressource est introduite pour modéliser les fours de manière détaillée. Cette contrainte d'acquisition bloque une ressource entre deux opérations d'un même ordre de fabrication. Nous montrons que l'approche de Kis (2003) pour insérer des nœuds efficacement peut être généralisée pour inclure ces contraintes. L'approche de solution présentée pour les décisions implicites de regroupement peut aussi être appliquée en cas de besoin de plusieurs ressources par opération. Nous obtenons des bons résultats numériques pour les instances de référence de la zone de photolithographie.

Les contraintes de temps sont un point essentiel pour une application industrielle. Ces contraintes peuvent être inclues dans notre approche comme contraintes souples. Afin d'obtenir une modélisation qui soit applicable en pratique, des coûts de non-respect sont attribués

à chaque contrainte de temps. Nous distinguons entre les contraintes de temps récupérables et non-récupérables. Une fonction objective supplémentaire quantifie l'envergure du retard. Une fonction objective lexicographique avec plusieurs critères est optimisée afin d'éviter de générer un ordonnancement infaisable.

Pour résumer, nous avons démontré que notre approche de modélisation implicite des décisions de regroupement est extensible en mettant en œuvre diverses généralisations. Toutes les contraintes présentées peuvent être utilisées dans des combinaisons arbitraires. Ceci permet d'aborder un riche ensemble de problèmes d'ordonnancement. Nous pensons que d'autres extensions peuvent être intégrées. Nous discutons les possibilité d'améliorer la performance de notre approche et des extensions prometteuses dans la suite.

Perspectives

Bien que nous obtenions déjà de bons résultats numériques, nous sommes convaincu qu'un énorme potentiel reste pour améliorer la performance de notre approche. Ils existent des approches qui accélèrent le calcul des dates de début en mettant à jour les dates de début calculées précédemment que partiellement (Michel and Van Hentenryck (2003), Pearce and Kelly (2007) et Sobeyko and Mönch (2016)). L'idée est de recalculer, après chaque itération, uniquement les dates de début qui sont successibles d'avoir changés au lieu de mettre à jour les dates de début de chaque nœud du graphe conjonctif. Nous supposons que ceci permet d'augmenter considérablement le nombre de mouvements que notre heuristique peut effectué par seconde. Ceci est particulièrement prometteur pour les grandes instances puisque le gain attendu croît avec le nombre de nœuds du graphe. Une autre idée pour améliorer notre approche est d'inclure les idées présentées dans Dauzère-Pérès and Paulli (1997) et García-León et al. (2015) pour des problèmes d'ordonnacement de type flexible job-shop. Ils estiment le coût d'exécution d'un mouvement en évaluant ses effets sur la fonction objective sans réellement effectuer le déplacement. En outre, il semble prometteur d'explorer davantage les stratégies avancées de sélection de nœud. L'heuristique de construction que nous utilisons dans notre approche de type GRASP évalue un grand nombre de positions d'insertion pour chaque nœud. Puisque cela peut être lent pour les grandes instances, il semble prometteur d'appliquer une approche plus rapide. Une idée est d'utiliser une version randomisée d'une règle de répartition (par exemple, BATC (Mönch et al. (2013))) pour accélérer la phase de construction et laisser ainsi plus de temps pour la phase d'amélioration (par recuit simulé). Bien que nous ayons montré que notre approche fonctionne bien pour les instances industrielles et donne des résultats applicables en pratique, nous avons observé des difficultés pour les instances avec des contraintes de temps serrées. Il serait intéressant d'étudier comment les contraintes de temps peuvent être prises en compte directement lors du calcul des dates de début.

En plus de ces idées concrètes d'amélioration de la performance, d'autres pistes d'amélioration et de généralisation peuvent être poursuivies. Notre contribution la plus importante, la représentation implicite des décisions de regroupement, n'est pas liée à une méta-heuristique spécifique. Le GRASP a des mécanismes de diversification et d'intensification forts mais

ils lui manquent des éléments d'apprentissage mutuel qui peuvent être trouvés dans les approches path-relinking ou les algorithmes génétiques. Il serait donc intéressant d'explorer la performance de notre modélisation avec d'autres approches méta-heuristique. La performance actuelle de notre approche est déjà adaptée aux instances du monde réel d'une zone de diffusion et de nettoyage. Cependant, une amélioration significative de la performance nous permettrait de traiter des instances plus grandes et nouvelles domaines d'application. Nous pourrions ainsi optimiser l'ordonnancement des ordre de fabrications de plusieurs ateliers ou même de toute la fab de manière intégrée. Ce dernier pourrait ne pas être pertinent pour différentes raisons, mais notamment parce que l'horizon d'ordonnancement à considérer serait trop long.

Nous avons présente une spécification complète du problème industriel. Bien que les propriétés les plus importantes de cette spécification soient incluses dans nos modèles et nos méthodes de solution, il reste un écart entre la spécification industrielle et nos méthodes de solution. Il serait utile de combler cet écart et d'inclure d'autres aspects dans notre modélisation. Plusieurs caractéristiques du problème industriel peuvent être abordées en utilisant notre représentation détaillée des ressources et du routage. Il serait aussi très intéressant d'appliquer et d'analyser notre approche de manière plus détaillée en effectuant des expériences numériques étendues. En outre, une étude expérimentale pour analyser la bonne granularité du modèle serait très intéressante. De nombreuses caractéristiques peuvent être modélisées en combinant les contraintes disponibles. Afin de modéliser les fours de manière détaillée, une combinaison plus générale de nos modèles est nécessaire. Notre représentation graphique de la route proposée ne prend en compte uniquement les machines avec traitement par lot si la composant ne contient qu'une seule opération. Un assouplissement de cette limitation est nécessaire afin de pouvoir examiner les éléments mobiles de longueurs arbitraires. Les durées obligatoires de la nacelle de secours dans les fours pourraient être abordées par une adaptation du poids des arêtes. Cependant, cette adaptation ne semble par être triviale. Une approche possible est (encore une fois) d'adapter dynamiquement le poids des arêtes pendant la traversée du graphe. Cela nécessiterait de maintenir un état pour chaque ressource interne du four qui est mis à jour chaque fois qu'une opération concernée est traversée. Il serait utile d'utiliser le nombre de mouvements effectués dans l'horizon d'ordonnancement comme fonction objective supplémentaire. Cela est probablement difficile à combiner avec les mises à jour partielles du graphe proposées ci-dessus car le nombre de mouvements dans l'horizon d'ordonnancement doit être partiellement mis à jour.

Une perspective à long terme intéressante est la combinaison des approches d'ordonnancement avec des décisions connexes du système. L'intégration des décisions de gestion des risques des outils, provenant des procédures d'inspection et de contrôle, avec l'ordonnancement pourrait aider à choisir les machines au moindre risque lors de l'ordonnancement. De même, l'intégration des indicateurs de santé des machines dans l'ordonnancement peut réduire le risque en évitant les machines à haut risque pour les lots importants. Un indicateur de santé d'une machine représente l'état de la machine à un moment donné. Ces approches ont déjà été proposées dans la littérature (Doleschal et al. (2015), Kao et al. (2016)). Il pourrait également être intéressant de considérer l'ordonnancement comme un outil qui peut

être utilisé pour déterminer ou vérifier les décisions de gestion de qualification selon les idées décrites dans Johnzén et al. (2008). Les décisions de gestion de qualification sont nécessaires pour gérer les capacités des machines. La gestion de qualification détermine les réglages et les essais préparatoires qui déterminent si une machine est prête à recevoir une opération spécifique.

Il semble possible et pertinent d'appliquer l'approche présentée dans cette thèse dans d'autres domaines de la fab. Nous avons déjà montré que les problèmes d'ordonnancement dans la zone de photo-lithographie peuvent être résolus efficacement par notre approche. Etant donné que la zone d'implantation contient un équipement avec des caractéristiques similaires, il semble prometteur d'y appliquer notre approche. Une autre perspective intéressante est d'analyser la façon dont l'approche peut être appliquée à d'autres domaines au-delà de la fabrication de semi-conducteurs. Les contraintes d'acquisition de ressources apparaissent dans d'autres domaines, par exemple dans l'ordonnancement des opérations de maintenance des trains (voir Ramond et al. (2006)). Une extension importante pour une application dans d'autres domaines est la prise en compte des calendriers des machines. Ceci n'est pas nécessaire dans la fabrication de semi-conducteurs où l'usine tourne généralement 24 heures sur 24. Mais ceci n'est pas le cas dans d'autres industries. Les calendriers peuvent également être utilisés pour la modélisation d'autres cas d'utilisation, tel que le temps d'arrêt d'une machine en raison d'une activité de maintenance planifiée. Dans de nombreuses applications, plus d'un critère d'optimisation doit être considéré. Puisque notre approche lexicographique est basée sur une fonction générale d'agrégation, nous croyons que notre approche permet une extension directe vers une véritable approche multi-critères : au lieu de maintenir exactement la meilleure solution lexicographique, nous pourrions maintenir un ensemble de solutions du front de Pareto.

Bibliography

Adams, J., E. Balas, and D. Zawack (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science 34*(3), 391–401.

Almeder, C. and L. Mönch (2011). Metaheuristics for scheduling jobs with incompatible families on parallel batching machines. *Journal of the Operational Research Society 62*(12), 2083–2096.

Artigues, C. and D. Feillet (2008). A branch and bound method for the job-shop problem with sequence-dependent setup times. *Annals of Operations Research 159*(1), 135–159.

Artigues, C., M.-J. Huguet, and P. Lopez (2011). Generalized disjunctive constraint propagation for solving the job shop problem with time lags. *Engineering Applications of Artificial Intelligence 24*(2), 220–231.

Artigues, C. and F. Roubellat (2000). A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. *European Journal of Operational Research 127*(2), 297 – 316.

Artigues, C. and F. Roubellat (2002). An efficient algorithm for operation insertion in a multi-resource job-shop schedule with sequence-dependent setup times. *Production Planning & Control 13*(2), 175–186.

ASML (2016, February). Asml, 2015 annual report. `https://staticwww.asml.com/doclib/investor/annual_reports/2015/DownloadCenter/reports/Annual-Report-Form20F.pdf`.

Balas, E., N. Simonetti, and A. Vazacopoulos (2008). Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling 11*(4), 253–262.

Balas, E. and A. Vazacopoulos (1998). Guided local search with shifting bottleneck for job shop scheduling. *Management Science 44*(2), 262–275.

Balasubramanian, H., L. Mönch, J. Fowler, and M. Pfund (2004). Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness. *International Journal of Production Research 42*(8), 1621–1638.

Barták, R. and O. Čepek (2008). Nested precedence networks with alternatives: Recognition, tractability, and models. In *Artificial Intelligence: Methodology, Systems, and Applications*, pp. 235–246. Springer.

Bartusch, M., R. H. Möhring, and F. J. Radermacher (1988). Scheduling project networks with resource constraints and time windows. *Annals of Operations Research 16*(1), 199–240.

Beck, J. C. and M. S. Fox (2000). Constraint-directed techniques for scheduling alternative activities. *Artificial Intelligence 121*(1), 211–250.

Behnke, D. and M. J. Geiger (2012). Test instances for the flexible job shop scheduling problem with work centers. Technical report, Helmut-Schmidt-Universität, Universität der Bundeswehr Hamburg.

Bilyk, A., L. Mönch, and C. Almeder (2014). Scheduling jobs with ready times and precedence constraints on parallel batch machines using metaheuristics. *Computers & Industrial Engineering 23*(5), 1621–1635.

Bitar, A. (2015). *Ordonnancement sur machines parallèles appliqué à la fabrication de semi-conducteurs : Atelier de photolithographie*. Ph. D. thesis, École des Mines de Saint-Étienne.

Bitar, A., S. Dauzère-Pérès, C. Yugma, and R. Roussel (2016). A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing. *Journal of Scheduling 19*(4), 367–376.

Blazewicz, J., K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz (2007). *Handbook on Scheduling - From Theory to Applications*. Springer Berlin Heidelberg.

Blazewicz, J., E. Pesch, and M. Sterna (2000). The disjunctive graph machine representation of the job shop and scheduling problem. *European Journal of Operational Research 127*, 317–331.

Botta-Genoulaz, V. (2000). Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. *International Journal of Production Economics 64*(1), 101–111.

Bowman, E. H. (1959). The schedule-sequencing problem. *Operations Research 7*(5), 621–624.

Bowman, V. J. J. (1976). On the relationship of the tchebycheff norm and the efficient frontier of multiple-criteria objectives. In *Multiple criteria decision making*, pp. 76–86. Springer.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research 41*(3), 157–183.

Brucker, P. (2007). *Scheduling algorithms*. Springer.

Brucker, P., A. Gladky, H. Hoogeveen, M. Y. Kovalyov, C. Potts, T. Tautenhahn, and S. Van De Velde (1998). Scheduling a batching machine. *Journal of Scheduling 1*(1), 31–54.

Brucker, P., T. Hilbig, and J. Hurink (1999). A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics 94*(1), 77–99.

Brucker, P. and J. Hurink (2000). Solving a chemical batch scheduling problem by local search. *Annals of Operations Research 96*(1-4), 17–38.

Brucker, P. and J. Neyer (1998). Tabu-search for the multi-mode job-shop problem. *Operations-Research-Spektrum 20*(1), 21–28.

Brucker, P. and R. Schlie (1990). Job-shop scheduling with multi-purpose machines. *Computing 45*(4), 369–375.

Brucker, P. and O. Thiele (1996). A branch & bound method for the general-shop problem with sequence dependent setup-times. *Operations-Research-Spektrum 18*(3), 145–161.

Bureau, M., S. Dauzère-Pérès, C. Yugma, L. Vermariën, and J.-B. Maria (2007). Simulation results and formalism for global-local scheduling in semiconductor manufacturing facilities. In *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come*, pp. 1768–1773. IEEE Press.

Bürgy, R. (2014). *Complex Job Shop Scheduling: A General Model and Method*. Ph. D. thesis, Université de Fribourg.

Čapek, R., P. Šůcha, and Z. Hanzálek (2012). Production scheduling with alternative process plans. *European Journal of Operational Research 217*(2), 300–311.

Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research 11*(1), 42–47.

Carlier, J. and E. Pinson (1989). An algorithm for solving the job-shop problem. *Management Science 35*(2), 164–176.

Carlier, J. and E. Pinson (1994). Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research 78*(2), 146–161.

Caumond, A., P. Lacomme, and N. Tchernev (2008). A memetic algorithm for the job-shop with time-lags. *Computers & Operations Research 35*(7), 2331–2356.

Chandra, S. M., M. Mathirajan, R. Gopinath, and A. Sivakumar (2008). Tabu search methods for scheduling a burn-in oven with non-identical job sizes and secondary resource constraints. *International Journal of Operational Research 3*(1), 119–139.

Chen, H., J. Ihlow, and C. Lehmann (1999). A genetic algorithm for flexible job-shop scheduling. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, Volume 2, pp. 1120–1125. IEEE.

Chiang, T.-C., H.-C. Cheng, and L.-C. Fu (2010). A memetic algorithm for minimizing total weighted tardiness on parallel batch machines with incompatible job families and dynamic job arrival. *Computers & Operations Research 37*(12), 2257–2269.

Chien, C.-F., S. Dauzère-Pérès, H. Ehm, J. W. Fowler, Z. Jiang, S. Krishnaswamy, T.-E. Lee, L. Moench, and R. Uzsoy (2011). Modelling and analysis of semiconductor manufacturing in a shrinking world: challenges and successes. *European Journal of Industrial Engineering 4 5*(3), 254–271.

Cho, L., H. M. Park, J. K. Ryan, T. C. Sharkey, C. Jung, and D. Pabst (2014). Production scheduling with queue-time constraints: Alternative formulations. *Proceedings of the 2014 Industrial and Systems Engineering Research Conference*.

Cigolini, R., M. Perona, A. Portioli, and T. Zambelli (2002). A new dynamic look-ahead scheduling procedure for batching machines. *Journal of scheduling 5*(2), 185–204.

Dächert, K., J. Gorski, and K. Klamroth (2010). An adaptive augmented weighted tchebycheff method to solve discrete, integer-valued bicriteria optimization problems. Technical report, Technical Report BUWAMNA-OPAP 10/06, University of Wuppertal, FB Mathematik und Naturwissenschaften.

Danneberg, D., T. Tautenhahn, and F. Werner (1999). A comparison of heuristic algorithms for flow shop scheduling problems with setup times and limited batch size. *Mathematical and Computer Modelling 29*(9), 101–126.

Dauzère-Pérès, S. and J.-B. Lasserre (1993). A modified shifting bottleneck procedure for job-shop scheduling. *The International Journal of Production Research 31*(4), 923–932.

Dauzère-Pérès, S. and L. Mönch (2013). Scheduling jobs on a single batch processing machine with incompatible job families and weighted number of tardy jobs objective. *Computers & Operations Research 40*(5), 1224–1233.

Dauzère-Pérès, S. and J. Paulli (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research 70*, 281–306.

Dauzère-Pérès, S. and C. Pavageau (2003). Extensions of an integrated approach for multi-resource shop scheduling. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 33*(2), 207–213.

Dauzère-Pérès, S., W. Roux, and J. Lasserre (1998). Multi-resource shop scheduling with resource flexibility. *European Journal of Operational Research 107*(2), 289–305.

Deppner, F. and M.-C. Portmann (2006). A hybrid decomposition approach using increasing clusters for solving scheduling problems with minimal and maximal time lags. In *Tenth International Workshop on Project Management and Scheduling (PMS 2006)*, pp. 4–pages.

Dhouib, E., J. Teghem, and T. Loukil (2013). Minimizing the number of tardy jobs in a permutation flowshop scheduling problem with setup times and time lags constraints. *Journal of Mathematical Modelling and Algorithms in Operations Research 12*(1), 85–99.

Ding, S., J. Yi, and M. T. Zhang (2006). Multicluster tools scheduling: An integrated event graph and network model approach. *Semiconductor Manufacturing, IEEE Transactions on 19*(3), 339–351.

Dinkelbach, W. (1971). über einen lösungsansatz zum vektormaximumproblem. In *Unternehmensforschung Heute*, pp. 1–13. Springer.

Doleschal, D., G. Weigert, and A. Klemmt (2015). Yield integrated scheduling using machine condition parameter. In *Proceedings of the 2015 Winter Simulation Conference*, pp. 2953–2963. IEEE Press.

Dorndorf, U., E. Pesch, and T. Phan-Huy (2000). A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. *Management Science 46*(10), 1365–1384.

El Adl, M., A. A. Rodriguez, and K. S. Tsakalis (1996). Hierarchical modeling and control of re-entrant semiconductor manufacturing facilities. In *Proceedings of the 35th IEEE Conference on Decision and Control*, Volume 2, pp. 1736 – 1742.

Eppstein, D. (1992). Parallel recognition of series-parallel graphs. *Information and Computation 98*(1), 41–55.

Feo, T. A. and M. G. Resende (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization 6*(2), 109–133.

Fondrevelle, J., A. Oulamara, and M.-C. Portmann (2006). Permutation flowshop scheduling problems with maximal and minimal time lags. *Computers & Operations Research 33*(6), 1540–1556.

Fowler, J. W., G. L. Hogg, and S. J. Mason (2002). Workload control in the semiconductor industry. *Production Planning & Control 13*(7), 568–578.

Fowler, J. W., S. Park, G. T. MacKulak, and D. L. Shunk (2001). Efficient cycle time-throughput curve generation using a fixed sample size procedure. *International Journal of Production Research 39*(12), 2595–2613.

Fowler, J. W. and O. Rose (2004). Grand challenges in modeling and simulation of complex manufacturing systems. *Simulation 80*(9), 469–476.

Gao, J., L. Sun, and M. Gen (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research 35*(9), 2892–2907.

García-León, A., S. Dauzère-Pérès, and Y. Mati (2015). Minimizing regular criteria in the flexible job-shop scheduling problem. In *7th Multidisciplinary International Scheduling Conference : Theory & Applications, Prague*.

Garey, M. R., D. S. Johnson, and R. Sethi (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research 1*(2), 117–129.

Geiger, C. D., K. G. Kempf, and R. Uzsoy (1997). A tabu search approach to scheduling an automated wet etch station. *Journal of Manufacturing Systems 16*(2), 102–116.

Glassey, C. R. and W. W. Weng (1991). Dynamic batching heuristic for simultaneous processing. *Semiconductor Manufacturing, IEEE Transactions on 4*(2), 77–82.

Glover, F. (1989). Tabu search-part i. *ORSA Journal on computing 1*(3), 190–206.

Goldberg, D. E. and J. H. Holland (1988). Genetic algorithms and machine learning. *Machine learning 3*(2), 95–99.

Golmakani, H. R. and A. Namazi (2012). An artificial immune algorithm for multiple-route job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology 63*(1-4), 77–86.

González, M. A., A. Oddi, R. Rasconi, and R. Varela (2015). Scatter search with path relinking for the job shop with time lags and setup times. *Computers & Operations Research 60*, 37–54.

González, M. A., C. R. Vela, I. González-Rodríguez, and R. Varela (2013). Lateness minimization with tabu search for job shop scheduling problem with sequence dependent setup times. *Journal of Intelligent Manufacturing 24*(4), 741–754.

Graham, R. L., E. L. Lawler, J. K. Lenstra, and A. Rinnooy Kan (1977). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics 5*, 287–326.

Grimes, D. and E. Hebrard (2010). Job shop scheduling with setup times and maximal time-lags: A simple constraint programming approach. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 147–161. Springer.

Grimes, D. and E. Hebrard (2015). Solving variants of the job shop scheduling problem through conflict-directed search. *INFORMS Journal on Computing 27*(2), 268–284.

Gröflin, H. and A. Klinkert (2007). Feasible insertions in job shop scheduling, short cycles and stable sets. *European Journal of Operational Research 177*(2), 763–785.

Gröflin, H., A. Klinkert, and N. P. Dinh (2008). Feasible job insertions in the multi-processor-task job shop. *European Journal of Operational Research 185*(3), 1308–1318.

Guo, C., J. Zhibin, H. Zhang, and N. Li (2012). Decomposition-based classified ant colony optimization algorithm for scheduling semiconductor wafer fabrication system. *Computers & Industrial Engineering 62*(1), 141–151.

Gurnani, H., R. Anupindi, and R. Akella (1992). Control of batch processing systems in semiconductor wafer fabrication facilities. *Semiconductor Manufacturing, IEEE Transactions on 5*(4), 319–328.

Gustin, W. (1963). Orientable embedding of cayley graphs. *Bulletin of the American Mathematical Society 69*(2), 272–275.

Ham, M. (2012). Integer programming-based real-time dispatching (i-rtd) heuristic for wet-etch station at wafer fabrication. *International Journal of Production Research 50*(10), 2809–2822.

Hasper, A., E. Oosterlaken, F. Huussen, and T. Claasen-Vujcic (1999). Advanced manufacturing equipment: a vertical batch furnace for 300-mm wafer processing. *IEEE Micro 19*(5), 34–43.

Ho, Y.-C. and C. Moodie (1996). Solving cell formation problems in a manufacturing environment with flexible processing and routeing capabilities. *International Journal of Production Research 34*(10), 2901–2923.

Hurink, J., B. Jurisch, and M. Thole (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum 15*(4), 205–215.

Hurink, J. and J. Keuchel (2001). Local search algorithms for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics 112*(1), 179–197.

Hutchinson, G. and K. Pflughoeft (1994). Flexible process plans: their value in flexible automation systems. *The International Journal of Production Research 32*(3), 707–719.

Jain, A. S. and S. Meeran (1998). A state-of-the-art review of job-shop scheduling techniques. Technical report, Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland.

Johnzén, C., S. Dauzère-Pérès, and P. Vialletelle (2011). Flexibility measures for qualification management in wafer fabs. *Production Planning and Control 22*(1), 81–90.

Johnzén, C., P. Vialletelle, S. Dauzère-Pérès, C. Yugma, and A. Derreumaux (2008). Impact of qualification management on scheduling in semiconductor manufacturing. In *Proceedings of the 40th Conference on Winter Simulation*, pp. 2059–2066. Winter Simulation Conference.

Jung, C., D. Pabst, M. Ham, M. Stehli, and M. Rothe (2013). An effective problem decomposition method for scheduling of diffusion processes based on mixed integer linear programming. In *Advanced Semiconductor Manufacturing Conference (ASMC), 2013 24th Annual SEMI*, pp. 35–40. IEEE.

Jurisch, B. (1992). *Scheduling jobs in shops with multi-purpose machines*. Ph. D. thesis, Fachbereich Mathematik/Informatik, Universität Osnabrück.

Kahn, A. B. (1962). Topological sorting of large networks. *Communications of the ACM 5*(11), 558–562.

Kao, Y.-T., S. Dauzère-Pérès, and J. Blue (2016). Integrating equipment health in job shop scheduling for semiconductor fabrication. In *International Conference on Project Management and Scheduling*, pp. 223.

Kao, Y.-T., S.-C. Zhan, and S.-C. Chang (2012). Efficient and waiting time violation-free furnace tool allocation via integration of sequencing constraints. In *e-Manufacturing & Design Collaboration Symposium (eMDC), 2012*, pp. 1–2. IEEE.

Karmarkar, U. S. (1989). Capacity loading and release planning with work-in-progress (wip) and leadtimes. *Journal of Manufacturing and Operations Management 2*, 105–123.

Kashan, A. H., B. Karimi, and M. Jenabi (2008). A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. *Computers & Operations Research 35*(4), 1084 – 1098.

Kempf, K. G., R. Uzsoy, and C.-S. Wang (1998). Scheduling a single batch processing machine with secondary resource constraints. *Journal of Manufacturing Systems 17*(1), 37–51.

Kiba, J., S. Dauzère-Pérès, C. Yugma, S. G. Charpak, and G. Lamiable (2010). Comparing transport policies in a full-scale 300mm wafer manufacturing facility. *Proceedings of the 11th International Material Handling Research Colloquium*.

Kim, Y.-D., B.-J. Joo, and S.-Y. Choi (2010). Scheduling wafer lots on diffusion machines in a semiconductor wafer fabrication facility. *Semiconductor Manufacturing, IEEE Transactions on 23*(2), 246–254.

Kirkpatrick, S. (1984). Optimization by simulated annealing: Quantitative studies. *Journal of statistical physics 34*(5-6), 975–986.

Kis, T. (2003). Job-shop scheduling with processing alternatives. *European Journal of Operational Research 151*(2), 307 – 332. Meta-heuristics in combinatorial optimization.

Kis, T. and A. Hertz (2003). A lower bound for the job insertion problem. *Discrete Applied Mathematics 128*(2), 395–419.

Kis, T. and E. Pesch (2005). A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. *European Journal of Operational Research 164*(3), 592–608.

Klemmt, A. and L. Mönch (2012). Scheduling jobs with time constraints between consecutive process steps in semiconductor manufacturing. In *Proceedings of the Winter Simulation Conference*, WSC '12, pp. 194:1–194:10. Winter Simulation Conference.

Knopp, S., S. Dauzère-Pérès, and C. Yugma (2014). Flexible Job-Shop Scheduling with Extended Route Flexibility for Semiconductor Manufacturing. In *Proceedings of the 2014 Winter Simulation Conference (WSC)*, pp. 2478–2489. IEEE Press.

Knopp, S., S. Dauzère-Pérès, and C. Yugma (2015a). A Batch-Oblivious Approach for Complex Job-Shop Scheduling Problems. Working Paper EMSE CMP-SFL 2015/2.

Knopp, S., S. Dauzère-Pérès, and C. Yugma (2015b). Scheduling Complex Job-Shops using Batch Oblivious Disjunctive Graphs. In *7th Multidisciplinary International Conference on Scheduling: Theory and Applications*, pp. 788–793.

Knopp, S., S. Dauzère-Pérès, and C. Yugma (2016). Modeling Maximum Time Lags in Complex Job-Shops with Batching in Semiconductor Manufacturing. In *15th International Conference on Project Management and Scheduling*, pp. 227–230.

Kohn, R. and O. Rose (2011). Automated generation of analytical process time models for cluster tools in semiconductor manufacturing. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pp. 1803–1815. IEEE.

Kohn, R., O. Rose, and C. Laroque (2013). Study on multi-objective optimization for parallel batch machine scheduling using variable neighbourhood search. In *Proceedings of the 2013 Winter Simulation Conference (WSC)*, pp. 3654–3670. IEEE.

Kovalyov, M. Y., C. N. Potts, and V. A. Strusevich (2004). Batching decisions for assembly production systems. *European Journal of Operational Research 157*(3), 620–642.

Lacomme, P., , N. Tchernev, and M.-J. Huguet (2012). Job-shop with generic time-lags: A heuristic based approach. In *9th International Conference of Modeling, Optimization and Simulation-MOSIM*, Volume 12, pp. 06–08.

Lawrence, S. (1984). Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). *Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania*.

Lee, C.-Y., R. Uzsoy, and L. A. Martin-Vega (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research 40*(4), 764–775.

Lee, J.-H. and T.-E. Lee (2010). An open scheduling architecture for cluster tools. In *Automation Science and Engineering (CASE), 2010 IEEE Conference on*, pp. 420–425. IEEE.

Lee, T.-E. (2008). A review of scheduling theory and methods for semiconductor manufacturing cluster tools. In *Proceedings of the 2008 Winter Simulation Conference (WSC)*, pp. 2127–2135. IEEE.

Legendre, A. M. (1805). *Nouvelles méthodes pour la détermination des orbites des comètes*. Number 1. F. Didot.

Manne, A. S. (1960). On the job-shop scheduling problem. *Operations Research 8*(2), 219–223.

Mason, S., J. Fowler, W. Carlyle, and D. Montgomery (2005, May). Heuristics for minimizing total weighted tardiness in complex job shops. *International Journal of Production Research 43*(10), 1943–1963.

Mason, S. J., J. W. Fowler, and W. Matthew Carlyle (2002). A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops. *Journal of Scheduling 5*(3), 247–262.

Mason, S. J. and K. Oey (2003). Scheduling complex job shops using disjunctive graphs: a cycle elimination procedure. *International Journal of Production Research 41*(5), 981–994.

Mastrolilli, M. and L. M. Gambardella (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling 3*(1), 3–20.

Mathirajan, M. and A. Sivakumar (2006). A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *The International Journal of Advanced Manufacturing Technology 29*(9-10), 990–1001.

Mathirajan, M., A. Sivakumar, and P. Kalaivani (2004). An efficient simulated annealing algorithm for scheduling burn-in oven with non-identical job sizes. *International Journal of Applied Management and Technology 2*(2), 117–138.

Mati, Y., S. Dauzère-Pérès, and C. Lahlou (2011). A general approach for optimizing regular criteria in the job-shop scheduling problem. *European Journal of Operational Research 212*(1), 33–42.

Mauer, J. L. and R. E. Schelasin (1993). The simulation of integrated tool performance in semiconductor manufacturing. In *Proceedings of the 25th conference on Winter simulation*, pp. 814–818. ACM.

Mehta, S. V. and R. Uzsoy (1998). Minimizing total tardiness on a batch processing machine with incompatible job families. *IIE transactions 30*(2), 165–178.

Michel, L. and P. Van Hentenryck (2003). Maintaining longest paths incrementally. In *Principles and Practice of Constraint Programming–CP 2003*, pp. 540–554. Springer.

Mladenović, N. and P. Hansen (1997). Variable neighborhood search. *Computers & Operations Research 24*(11), 1097–1100.

Möhring, R. H., M. Skutella, and F. Stork (2004). Scheduling with and/or precedence constraints. *SIAM Journal on Computing 33*(2), 393–415.

Mönch, L., H. Balasubramanian, J. W. Fowler, and M. E. Pfund (2005). Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Computers & Operations Research 32*(11), 2731 – 2750.

Mönch, L. and R. Drießel (2005). A distributed shifting bottleneck heuristic for complex job shops. *Computers & Industrial Engineering 49*(3), 363–380.

Mönch, L., J. W. Fowler, S. Dauzère-Pérès, S. J. Mason, and O. Rose (2011). A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of Scheduling 14*(6), 583–599.

Mönch, L., J. W. Fowler, and S. J. Mason (2013). *Production Planning and Control for Semiconductor Wafer Fabrication Facilities*, Volume 52. Springer New York.

Mönch, L. and O. Rose (2004). Shifting-Bottleneck-Heuristik für komplexe Produktionssysteme: Softwaretechnische Realisierung und Leistungsbewertung. *Quantitative Methoden in ERP und SCM, DSOR Beiträge zur Wirtschaftsinformatik 2*, 145–159.

Mönch, L., O. Rose, and R. Sturm (2003). A simulation framework for the performance assessment of shop-floor control systems. *Simulation 79*(3), 163–170.

Mönch, L., R. Schabacker, D. Pabst, and J. W. Fowler (2007). Genetic algorithm-based subproblem solution procedures for a modified shifting bottleneck heuristic for complex job shops. *European Journal of Operational Research 177*(3), 2100–2118.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics 38*(8), 114–117.

Muth, J. F. and G. L. Thompson (1963). *Industrial scheduling*. Prentice-Hall.

Nonobe, K. and T. Ibaraki (2006). A metaheuristic approach to the resource constrained project scheduling with variable activity durations and convex cost functions. In *Perspectives in Modern Project Scheduling*, pp. 225–248. Springer.

Nowicki, E. and C. Smutnicki (1996). A fast taboo search algorithm for the job shop problem. *Management Science 42*(6), 797–813.

Oddi, A., R. Rasconi, A. Cesta, and S. Smith (2009). Iterative-sampling search for job shop scheduling with setup times. In *COPLAS-09. Proc. of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems at ICAPS*. Citeseer.

Oddi, A., R. Rasconi, A. Cesta, and S. F. Smith (2011). Solving job shop scheduling with setup times through constraint-based iterative sampling: an experimental analysis. *Annals of Mathematics and Artificial Intelligence 62*(3-4), 371–402.

Ovacik, I. M. and R. Uzsoy (1994). Exploiting shop floor status information to schedule complex job shops. *Journal of manufacturing systems 13*(2), 73–84.

Ovacik, I. M. and R. Uzsoy (1997). *Decomposition methods for complex factory scheduling problems*. Kluwer Academic Publishers Boston.

Pacino, D. and P. Van Hentenryck (2011). Large neighborhood search and adaptive randomized decompositions for flexible jobshop scheduling. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pp. 1997–2002. AAAI Press.

Pearce, D. J. and P. H. Kelly (2007). A dynamic topological sort algorithm for directed acyclic graphs. *Journal of Experimental Algorithmics (JEA) 11*, 1–7.

Perez, I. C., J. W. Fowler, and W. Carlyle (2005). Minimizing total weighted tardiness on a single batch process machine with incompatible job families. *Computers & Operations Research 32*(2), 327 – 341.

Pfund, M. E., H. Balasubramanian, J. W. Fowler, S. J. Mason, and O. Rose (2008). A multi-criteria approach for scheduling semiconductor wafer fabrication facilities. *Journal of Scheduling 11*(1), 29–47.

Pfund, M. E., S. J. Mason, and J. W. Fowler (2006). Semiconductor manufacturing scheduling and dispatching. In *Handbook of Production Scheduling*, pp. 213–241. Springer.

Pinedo, M. (2012). *Scheduling: theory, algorithms, and systems*. Springer.

Ponsignon, T. and L. Mönch (2012). Heuristic approaches for master planning in semiconductor manufacturing. *Computers & Operations Research 39*(3), 479–491.

Potts, C. N. and M. Y. Kovalyov (2000). Scheduling with batching: a review. *European Journal of Operational Research 120*(2), 228–249.

Quirk, M. and J. Serda (2001). *Semiconductor manufacturing technology*, Volume 1. Prentice Hall Upper Saddle River, NJ.

Raaymakers, W. H. and J. Hoogeveen (2000). Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing. *European Journal of Operational Research 126*(1), 131–151.

Ramond, F., D. de Almeida, and S. Dauzère-Pérès (2006). Enhanced operation scheduling within railcar maintenance centers. In *7th World Congress on Railway Research, Montréal, Canada.*

Rodriguez Verjan, G. L., S. Dauzére-Pérès, and J. Pinaton (2011). Impact of control plan design on tool risk management: a simulation study in semiconductor manufacturing. In *Proceedings of the Winter Simulation Conference*, pp. 1918–1925. Winter Simulation Conference.

Rohde, J., H. Meyr, M. Wagner, et al. (2000). Die supply chain planning matrix. *PPS-Management*.

Rossi, A., S. Soldani, and M. Lanzetta (2015). Hybrid stage shop scheduling. *Expert Systems with Applications 42*(8), 4105–4119.

Rossi, F., A. Sperduti, K. B. Venable, L. Khatib, P. Morris, and R. Morris (2002). Learning and solving soft temporal constraints: An experimental study. In *Principles and Practice of Constraint Programming-CP 2002*, pp. 249–264. Springer.

Rowshannahad, M., S. Dauzère-Pérès, and B. Cassini (2015). Capacitated qualification management in semiconductor manufacturing. *Omega 54*, 50–59.

Roy, B. and B. Sussmann (1964). Les problemes d'ordonnancement avec contraintes disjonctives. *Note ds 9*.

Sadeghi, R., S. Dauzère-Pérès, C. Yugma, and G. Lepelletier (2015). Production control in semiconductor manufacturing with time constraints. In *Advanced Semiconductor Manufacturing Conference (ASMC), 2015 26th Annual SEMI*, pp. 29–33. IEEE.

Sadeghi, R., S. Dauzère-Pérès, C. Yugma, and L. Vermarien (2015). Consistency between global and local scheduling decisions in semiconductor manufacturing: an application to time constraint management. In *International Symposium on Semiconductor Manufacturing Intelligence (ISMI), 2015*, pp. 5 pages.

Sarin, S. C., A. Varadarajan, and L. Wang (2011). A survey of dispatching rules for operational control in wafer fabrication. *Production Planning & Control 22*(1), 4–24.

Scholl, W. and J. Domaschke (2000). Implementation of modeling and simulation in semiconductor wafer fabrication with time constraints between wet etch and furnace operations. *Semiconductor Manufacturing, IEEE Transactions on 13*(3), 273–277.

Schulz, C. (2013). *High Quality Graph Partitioning*. epubli.

Schutt, A., T. Feydy, and P. J. Stuckey (2013). Scheduling optional tasks with explanation. In *Principles and Practice of Constraint Programming*, pp. 628–644. Springer.

Schwindt, C. and N. Trautmann (2000). Batch scheduling in process industries: an application of resource–constrained project scheduling. *OR-Spektrum 22*(4), 501–524.

Shen, L. (2014). A tabu search algorithm for the job shop problem with sequence dependent setup times. *Computers & Industrial Engineering 78*, 95–106.

SIA (2015, December). Global billings report history (3-month moving average), 1976 - december 2015. http://www.semiconductors.org/industry_statistics/global_sales_report/.

Silver, E. A., D. F. Pyke, R. Peterson, et al. (1998). *Inventory management and production planning and scheduling*, Volume 3. Wiley New York.

Skorin-Kapov, J. and A. Vakharia (1993). Scheduling a flow-line manufacturing cell: a tabu search approach. *The International Journal of Production Research 31*(7), 1721–1734.

Sobeyko, O. and L. Mönch (2011). A comparison of heuristics to solve a single machine batching problem with unequal ready times of the jobs. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pp. 2006–2016. IEEE.

Sobeyko, O. and L. Mönch (2016). Heuristic approaches for scheduling jobs in large-scale flexible job shops. *Computers & Operations Research 68*, 97–109.

Sörensen, K. (2015). Metaheuristics - the metaphor exposed. *International Transactions in Operational Research 22*(1), 3–18.

Sotskov, Y. N., T. Tautenhahn, and F. Werner (1996). Heuristics for permutation flow shop scheduling with batch setup times. *Operations-Research-Spektrum 18*(2), 67–80.

Sourirajan, K. and R. Uzsoy (2007). Hybrid decomposition heuristics for solving large-scale scheduling problems in semiconductor wafer fabrication. *Journal of Scheduling 10*(1), 41–65.

Stadtler, H. and C. Kilger (2000). *Supply chain management and advanced planning*. Springer.

Steuer, R. E. and E.-U. Choo (1983). An interactive weighted tchebycheff procedure for multiple objective programming. *Mathematical programming 26*(3), 326–344.

Taillard, E. D. (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing 6*(2), 108–117.

Tan, Y., L. Mönch, and J. W. Fowler (2014). A decomposition heuristic for a two-machine flow shop with batch processing. In *Proceedings of the 2014 Winter Simulation Conference*, pp. 2490–2501. IEEE Press.

T'kindt, V. and J.-C. Billaut (2001). Multicriteria scheduling problems: a survey. *RAIRO-Operations Research 35*(02), 143–163.

Upasani, A. A., R. Uzsoy, and K. Sourirajan (2006). A problem reduction approach for scheduling semiconductor wafer fabrication facilities. *Semiconductor Manufacturing, IEEE Transactions on 19*(2), 216–225.

Uzsoy, R., C.-Y. Lee, and L. A. Martin-Vega (1992). A review of production planning and scheduling models in the semiconductor industry part i: system characteristics, performance evaluation and production planning. *IIE transactions 24*(4), 47–60.

Vaessens, R. J., E. H. Aarts, and J. K. Lenstra (1996). Job shop scheduling by local search. *INFORMS Journal on Computing 8*(3), 302–317.

Vaessens, R. J. M. (1995). *Generalized Job Shop Scheduling: Complexity and Local Search*. Ph. D. thesis, Eindhoven University of Technology.

Vakharia, A. J. and Y.-L. Chang (1990). A simulated annealing approach to scheduling a manufacturing cell. *Naval Research Logistics (NRL) 37*(4), 559–577.

Van Laarhoven, P. J., E. H. Aarts, and J. K. Lenstra (1992). Job shop scheduling by simulated annealing. *Operations Research 40*(1), 113–125.

Wagner, H. M. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly 6*(2), 131–140.

Wang, C.-S. and R. Uzsoy (2002). A genetic algorithm to minimize maximum lateness on a batch processing machine. *Computers & Operations Research 29*(12), 1621–1640.

Wein, L. M. (1988). Scheduling semiconductor wafer fabrication. *Semiconductor Manufacturing, IEEE Transactions on 1*(3), 115–130.

Wierzbicki, A. P. (1986). On the completeness and constructiveness of parametric characterizations to vector optimization problems. *Operations-Research-Spektrum 8*(2), 73–87.

Wikum, E. D., D. C. Llewellyn, and G. L. Nemhauser (1994). One-machine generalized precedence constrained scheduling problems. *Operations Research Letters 16*(2), 87–99.

Yazdani, M., M. Amiri, and M. Zandieh (2010). Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications 37*(1), 678–687.

Yugma, C., S. Dauzère-Pérès, C. Artigues, A. Derreumaux, and O. Sibille (2012). A batching and scheduling algorithm for the diffusion area in semiconductor manufacturing. *International Journal of Production Research 50*(8), 2118–2132.

Yurtsever, T., E. Kutanoglu, and J. Johns (2009). Heuristic based scheduling system for diffusion in semiconductor manufacturing. In *Winter Simulation Conference*, pp. 1677–1685. Winter Simulation Conference.

Zhang, X. and S. van de Velde (2010). On-line two-machine open shop scheduling with time lags. *European Journal of Operational Research 204*(1), 14–19.

Ziarnetzky, T. and L. Mönch (2016). Incorporating engineering process improvement activities into production planning formulations using a large-scale wafer fab model. *International Journal of Production Research 54*(21), 6416–6435.

Index

École Nationale Supérieure des Mines de Saint-Étienne

NNT: 2016LYSEM014

Sebastian KNOPP

COMPLEX JOB-SHOP SCHEDULING WITH BATCHING IN SEMICONDUCTOR MANUFACTURING

Specialization: Industrial Engineering

Keywords:
Scheduling, Disjunctive Graphs, Semiconductor Manufacturing, Optimization

Abstract:
The integration of batching machines within a job-shop environment leads to a complex job-shop scheduling problem. Semiconductor manufacturing presumably represents one of the most prominent practical applications for such problems. We consider a flexible job-shop scheduling problem with p-batching, reentrant flows, sequence dependent setup times and release dates while considering different regular objective functions. The scheduling of parallel batching machines and variants of the job-shop scheduling problem are well-studied problems whereas their combination is rarely considered.

Existing disjunctive graph approaches for this combined problem rely on dedicated nodes to explicitly represent batches. To facilitate modifications of the graph, our new modeling reduces this complexity by encoding batching decisions into edge weights. An important contribution is an original algorithm that takes batching decisions "on the fly" during graph traversals. This algorithm is complemented by an integrated move to resequence and reassign operations. This combination yields a rich neighborhood that we apply within a GRASP based metaheuristic approach.

We extend this approach by taking further constraints into account that are important in the considered industrial application. In particular, we model internal resources of machines in detail and take maximum time lag constraints into account. Numerical results for benchmark instances of different problem types show the generality and applicability of our approach. The conciseness of our idea facilitates extensions towards further complex constraints needed in real-world applications.

École Nationale Supérieure des Mines de Saint-Étienne

NNT : 2016LYSEM014

Sebastian KNOPP

Ordonnancement d'ateliers complexes de type job-shop avec machines à traitement par batch en fabrication de semi-conducteurs

Spécialité : Génie Industriel

Mots clefs :
Ordonnancement, Graphe Disjonctif, Fabrication de semi-conducteurs, Optimisation

Résumé :
La prise en compte de machines à traitement par batch dans les problèmes d'ordonnancement d'ateliers complexes de type job-shop est particulièrement difficile. La fabrication de semi-conducteurs est probablement l'une des applications pratiques les plus importantes pour ce types de problèmes. Nous considérons un problème d'ordonnancement de type job-shop flexible avec « p-batching », des flux rentrants, des temps de préparation dépendant de la séquence et des dates de début au plus tôt. Le but c'est d'optimiser différentes fonctions objectives régulières.

Les approches existantes par graphe disjonctif pour ce problème utilise des nœuds dédiés pour représenter explicitement les batches. Afin de faciliter la modification du graphe conjonctif, notre nouvelle modélisation réduit cette complexité en modélisant les décisions de batching à travers les poids des arcs. Une importante contribution de cette thèse est un algorithme original qui prend les décisions de batching lors du parcours du graphe. Cet algorithme est complété par un déplacement (« move ») intégré qui permet de reséquencer ou réaffecter les opérations. Cette combinaison donne un voisinage riche que nous appliquons dans une approche méta-heuristique de type GRASP.

Nous étendons cette approche en prenant en compte de nouvelles contraintes qui ont un rôle important dans l'application industrielle considérée. En particulier, nous modélisons de manière explicite les ressources internes des machines, et nous considérons un temps maximum d'attente entre deux opérations quelconques d'une gamme de fabrication. Les résultats numériques sur des instances de la littérature pour des problèmes plus simples ainsi que sur de nouvelles instances montrent la généricité et l'applicabilité de notre approche. Notre nouvelle modélisation permet de faciliter les extensions à d'autres contraintes complexes rencontrées dans les applications industrielles.