Optimization in the Photolithography Bay
Scheduling and the Traveling Salesman Problem

Janssen, Teun

DOI
10.4233/uuid:12961f87-eeff-41b5-8688-df28e0ad9860

Publication date
2019

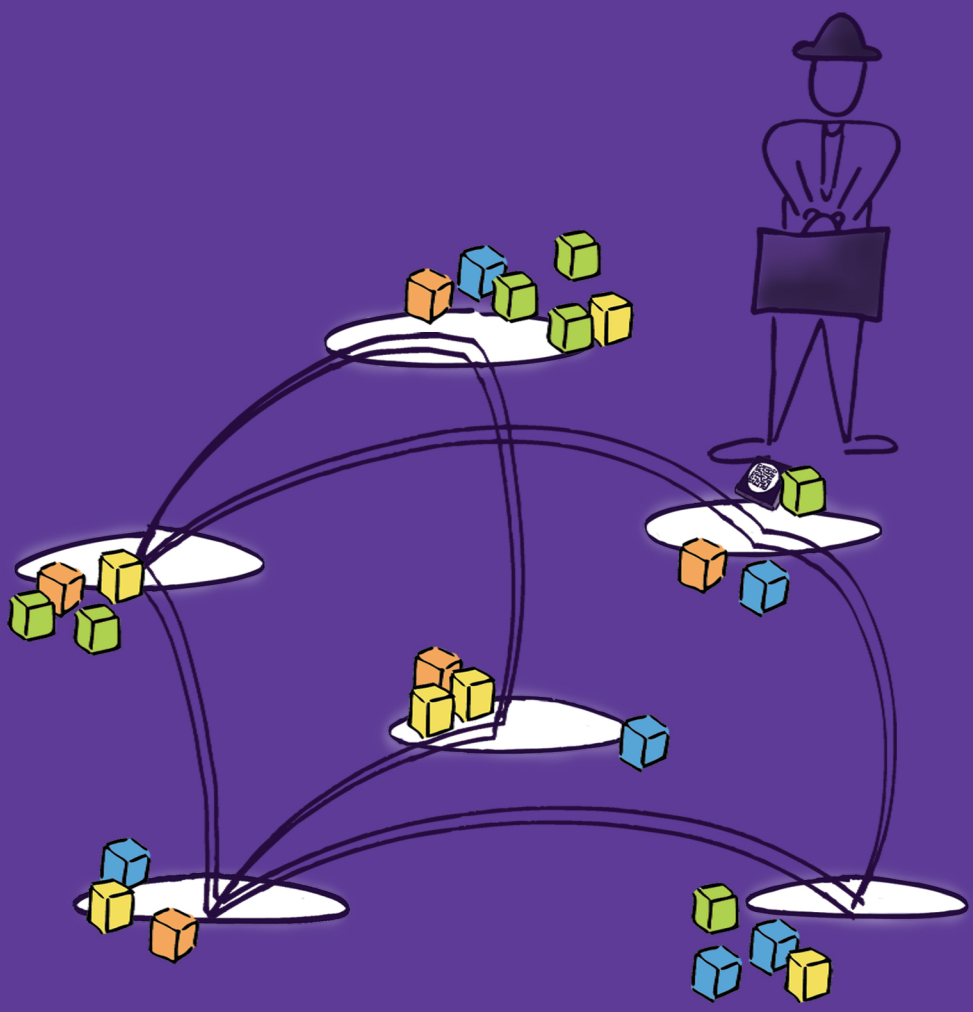
Document Version
Final published version

Citation (APA)
Janssen, T. (2019). *Optimization in the Photolithography Bay: Scheduling and the Traveling Salesman Problem*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:12961f87-eeff-41b5-8688-df28e0ad9860

Optimization in the Photolithography Bay

Scheduling and the Traveling Salesman Problem

Teun Janssen



Propositions

accompanying the dissertation

OPTIMIZATION IN THE PHOTOLITHOGRAPHY BAY

SCHEDULING AND THE TRAVELING SALESMAN PROBLEM

by

Teun Michiel Louis JANSSEN

1. There is no $o(\log n)$-approximation algorithm for the *a priori* traveling salesman problem in the scenario model.
2. The problem $P|partition|\sum_j C_j$ is $\mathcal{NP}$-hard.
3. The SPT-*available* rule gives a $\frac{4}{3}$-approximation for $P|partition|\sum_j C_j$.
4. The machine capacity required for the continuously arriving work in progress can be decreased by 1.67% in the semiconductor factory through the use of efficient scheduling algorithms.
5. In lens design, the performance of continuous optimization algorithms, in finding the optimum lens system for the application at hand, is increased by choosing the right model to calculate the objective function.
6. The performance of an optimization algorithm depends in practice much more on the actual application than theory suggests.
7. Applications increase the legitimacy[1] of scientific mathematical research.
8. A child's name should be researched before giving it to them.
9. Updates are not upgrades.
10. The quality of a board game is determined by the quality of its playtesting.

These propositions are regarded as opposable and defendable, and have been approved as such by the promotor prof. dr. ir. K. I. Aardal.

[1] As defined by M.C. Suchman. *Managing legitimacy: Strategic and institutional approaches.* Academy of management review 20(3): 571-610,1995.

Stellingen

behorende bij het proefschrift

OPTIMIZATION IN THE PHOTOLITHOGRAPHY BAY

SCHEDULING AND THE TRAVELING SALESMAN PROBLEM

door

Teun Michiel Louis JANSSEN

1. Er is geen $o(\log n)$- approximatiealgoritme voor het *a priori* handelsreizigersprobleem in het scenario model.
2. Het probleem $P|partition|\sum_j C_j$ is $\mathcal{NP}$-hard.
3. De SPT-*available* regel geeft een $\frac{4}{3}$-approximatie voor $P|partition|\sum_j C_j$.
4. Men kan de benodigde machinecapaciteit voor continu veranderende lopende werkzaamheden in een halfgeleiderfabriek verlagen met 1.67% door het gebruik van efficiënte scheduling-algoritmes.
5. Bij het ontwerpen van een lens kunnen de prestaties van het continue optimalisatiealgoritme bij het vinden van een optimale lensconfiguratie worden vergroot door het juiste model te kiezen waarmee de doelfunctie wordt berekend.
6. De prestaties van optimalisatiealgoritmes hangen in praktijk veel meer af van de daadwerkelijke applicatie dan door de theorie wordt gesuggereerd.
7. Toepassingen verhogen de legitimiteit[2] van wetenschappelijk wiskundig onderzoek.
8. Er moet eerst onderzoek gedaan worden naar de naam van een kind voor het aan hem of haar te geven.
9. Updates zijn geen upgrades.
10. De kwaliteit van een bordspel wordt bepaald tijdens het testen van het spelontwerp.

Deze stellingen worden opponeerbaar en verdedigbaar geacht en zijn als zodanig goedgekeurd door de promotor prof. dr. ir. K. I. Aardal.

[2] Zoals gedefinieerd door M.C. Suchman. *Managing legitimacy: Strategic and institutional approaches.* Academy of management review 20(3): 571-610,1995.

Optimization in the Photolithography Bay

Scheduling and the Traveling Salesman Problem

OPTIMIZATION IN THE PHOTOLITHOGRAPHY BAY

SCHEDULING AND THE TRAVELING SALESMAN PROBLEM

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus prof.dr.ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates
to be defended publicly on Thursday 21 March 2019 at 15:00 o'clock

by

Teun Michiel Louis JANSSEN

Master of Science Applied Mathematics,
Delft University of Technology, the Netherlands

born in Delft, the Netherlands.

This dissertation has been approved by the promotors.

Composition of doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. ir. K. Aardal,	Delft University of Technology, promotor
Dr. ir. L.J.J. van Iersel,	Delft University of Technology, copromotor
Independent members:	
Prof. dr. G. Schäfer,	Vrije Universiteit Amsterdam
Prof.dr. F.C.R. Spieksma,	Eindhoven University of Technology
Prof. dr. L. Stougie,	Vrije Universiteit Amsterdam
Prof. dr. ir. A. Verbraeck,	Delft University of Technology
Prof. dr. ir. A. W. Heemink,	Delft University of Technology, reserve member
Other members:	
Prof. dr. S. Dauzère-Pérès,	l'Ecole des Mines de Saint-Étienne

A part of the research described in this dissertation has been performed in the project INTEGRATE "Integrated Solutions for Agile Manufacturing in High-mix Semiconductor Fabs", co-funded by grants from France, Italy, Ireland, The Netherlands and the ECSEL Joint Undertaking.

Printed by: GVO printers & designers B.V. - Ponsen & Looijen, Ede, the Netherlands

Front & Back: Cover art by dr. ir. Sarah Bork.

ISBN 978-94-6332-460-1

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

voor jou

CONTENTS

SUMMARY

In a semiconductor factory, integrated circuits (or chips) are constructed on top of slabs of silicon, called wafers. The construction of these wafers is complicated and many different processing steps are needed to gradually building the chip layer by layer. Of these steps, photolithography uses the most expensive equipment. Therefore, the photolithography equipment is often the bottleneck of the factory.

Photolithography is used to transfer the geometric pattern of a chip on a wafer. First a light-sensitive photoresist is put on the wafer. Then UV light is sent through a photomask on the photoresist. The exposed parts of the photoresist will chemically react, creating the pattern. After the exposure, chemical reactions and metal depositions make a layer of circuits on the wafer.

In this thesis, we try to increase the production of the semiconductor factory by reducing the time needed for the photolithography. In the first part, we look at the machine level. The time to process a wafer on a lithography stepper machine is determined by different elements of the process (Chapter 2). It turns out that the blade movement required in the exposure step has a significant impact total time required to process a wafer. The blade movement in turn depends on the order in which the different images are processed. Hence we want to find an ordering of the images, such that the blade movement is minimized. This problem turns out to be equivalent to the *a priori* traveling salesmen problem in the scenario model. The practical problem instances found are solved a limited amount of time using an integer linear programming solver and the average blade movement is reduced by approximately 20%, which reduces the average exposure time 1.6%.

The *a priori* traveling salesmen problem (*a priori* TSP) in the scenario model is hard to solve in theory (Chapter 3). In *a priori* TSP in the scenario model, we are given a complete weighted graph $G = (V, E)$ and a set of scenarios $\mathcal{S}$ with $S_1, \ldots, S_m \subseteq V$. Scenario S_j has probability p_j of being the active set, where $\sum_j p_j = 1$. We begin by finding an ordering on V, called the first-stage tour. When an active set is released, the second-stage tour is obtained by shortcutting the first-stage tour on the vertices of the active set. The goal is to find a first-stage tour that minimizes the expected length of the second-stage tour. The best known approximation algorithm for *a priori* TSP has a ratio of $O(\log n)$. We show that the problem is $\mathcal{NP}$-hard even for scenarios with only four cities and there are constant-factor approximation algorithms for instances with small, big or nested scenarios. Furthermore we show that there is no polynomial-time approximation scheme for planar bipartite graphs.

We also look at a space filling curve heuristic for *a priori* TSP (Chapter 4). The Sierpiński curve solves TSP problems in the plane, but it is not easily extended to higher dimen-

sions. Therefore we use the Hilbert curve instead and test the algorithm performance on the instances found when minimizing the blade movement in the photolithography process. The algorithms is very fast and is on average less than 1% away from the optimal solution.

In the second part, we look at the photolithography bay as a whole. We will improve the scheduling of the work in progress (Chapter 5). Currently, this schedule does not take processing times into account. Detailed timing data is now available for the components of the machines with which we can accurately calculate the processing time of every job. We propose a two stage algorithm that uses the newly available processing times to improve the scheduling of photolithography (and metalization) bays. The algorithm focuses on minimizing the average completion time of the jobs, but it also allows the logistic manager to influence the schedule. We show that the new schedule reduces the average completion time by 6.02% and the machine capacity by 1.97%.

Minimizing the average completion time in the photolithography bay is equal to the problem of minimizing the total completion time while scheduling jobs that each use exactly one resource, $P|\mathit{partition}|\sum_j C_j$ (Chapter 6). We show that $P|\mathit{partition}|\sum_j C_j$ always has an optimal solution where jobs sharing the same resource are ordered by the processing time. While the complexity of $P|\mathit{partition}|\sum_j C_j$ remains an open problem, we show that similar problems such as $P|\mathit{partition}, \mathcal{M}_r|\sum_j C_j$, $P|\mathit{partition}(2), p_j = 1|\sum_j C_j$ and $P|\mathit{partition}, \mathit{unmovable}, p_j = 1|\sum_j C_j$ are $\mathcal{NP}$-hard. Furthermore, we approximate the problem using a list scheduling rule, denoted the shortest processing time *available* rule. This rule gives a $2 - \frac{1}{m}$-approximation.

SAMENVATTING

In een halfgeleiderfabriek worden geïntegreerde schakelingen (chips) bovenop schijven silicium gemaakt. Deze schijven worden wafers genoemd. De constructie van deze wafers is een gecompliceerd proces en er zijn veel verschillende verwerkingsstappen nodig om de chip stapsgewijs op te bouwen. Fotolithografie is de duurste stap in dit proces en daarom zijn de fotolithografie machines vaak de bottleneck van de fabriek.

Fotolithografie wordt gebruikt om het geometrische patroon van een chip op een wafer over te brengen. Eerst wordt een lichtgevoelig materiaal (fotoresist) op de wafer geplaatst. Vervolgens bestraalt men door een fotomasker de fotoresist met uv-licht. Op de delen van de fotoresist die aan het licht worden blootgesteld vindt een chemische reactie plaats, waardoor het geometrische patroon ontstaat. Met behulp van dit patroon kan vervolgens een laag met schakelingen op de wafer gecreëerd worden door middel van verdere chemische reacties en metaaldeposities.

In dit proefschrift proberen we de productie van de halfgeleiderfabriek te verhogen door de tijd die nodig is voor het fotolithografieproces te verkorten. In het eerste deel van de thesis doen we dit door te kijken naar het proces in de machines. De tijd om een wafer op een lithografiestepmachine te verwerken, wordt bepaald door verschillende elementen van het proces (Hoofdstuk 2). Het blijkt dat de bewegingen van lichtafschermende bladen, die vereist zijn in de belichtingsstap, een significante impact heeft op de totale procestijd van een wafer. De beweging van zo'n blad hangt op zijn beurt weer af van de volgorde waarin de verschillende afbeeldingen worden verwerkt. We zoeken daarom naar een volgorde van deze afbeeldingen zodanig dat de totale beweging van de bladen wordt geminimaliseerd. Dit probleem blijkt equivalent te zijn aan het *a priori* handelsreiziger probleem in het scenariomodel. Dit probleem kunnen we in de praktijk in beperkte tijd oplossen met behulp van een algoritme dat geheeltallige lineaire problemen oplost. De gemiddelde beweging van de bladen wordt hierdoor met ongeveer 20% gereduceerd, waardoor de gemiddelde belichtingstijd met 1,6% korter wordt.

Het *a priori* handelsreiziger probleem (*a priori* TSP) in het scenariomodel is in theorie moeilijk op te lossen (Hoofdstuk 3). Bij het probleem hebben we een volledig gewogen graaf $G = (V, E)$ en een reeks scenario's $\mathcal{S}$ met $S_1, \ldots, S_m \subseteq V$. Ieder scenario S_j heeft een kans p_j dat het de actieve set is, waarbij $\sum_j p_j = 1$. We beginnen met het vinden van een ordering van de knopen V. Deze ordering geeft de eerste fase toer. Wanneer een actieve set wordt vrijgegeven, wordt een tweede fase toer verkregen op de knopen in S_j door dezelfde ordering te nemen op deze knopen als ze hadden in de eerste fase toer. Het doel is om een toer in de eerste fase te vinden zodanig dat de verwachte lengte van de toer in de tweede fase wordt geminimaliseerd. Het beste approximatie algoritme voor *a priori* TSP heeft een ratio van $O(\log n)$. We laten zien dat het probleem $\mathcal{NP}$-moeilijk is, zelfs voor kleine scenario's met maar vier steden en we laten zien dat er algoritmen zijn die

de oplossing benaderen binnen een constante factor voor instanties met kleine, grote of geneste scenario's. Verder laten we zien dat er geen polynoom-tijd-benaderingsschema is voor planaire bipartiete grafen.

We kijken ook naar een heuristiek voor *a priori* TSP die gebruikt maakt van een ruimtevullende curve (Hoofdstuk 4). De Sierpiński-kromme lost TSP-problemen in het vlak op, maar de uitbreiding naar meerdere dimensies is niet evident. We gebruiken daarom de Hilbert-curve in plaats van de Sierpinski-curve. We testen het algoritme dat gebruik maakt van de Hilbert-kromme door het te gebruiken om de beweging van de bladen in het fotolithografieproces te minimaliseren. Het algoritme is snel en is gemiddeld maar 1% slechter dan de optimale oplossing.

In het tweede deel kijken we naar alle fotolithografiemachines als productiegroep in de halfgeleiderfabriek. We verbeteren de planning van het lopende werk zodanig dat de benodigde machinecapaciteit voor dit werk wordt geminimaliseerd (Hoofdstuk 5). Momenteel houdt de productieplanning geen rekening met procestijden. Gedetailleerde tijdgegevens zijn nu beschikbaar voor de verschillende componenten van de machines waarmee we de procestijd van iedere taak nauwkeurig kunnen berekenen. Om de planning van fotolithografie (en metallisatie) te verbeteren, stellen we een tweetrapsalgoritme voor dat wel gebruik maakt van nieuw beschikbare procestijden. Het algoritme focust zich op het minimaliseren van de gemiddelde doorlooptijd van de taken, maar stelt de logistiek manager ook in staat om de planning te beïnvloeden. We laten zien dat met de nieuwe planning de producten gemiddeld 6,02% eerder klaar zijn en dat er 1,97% minder machinecapaciteit nodig is.

Het minimaliseren van de gemiddelde doorlooptijd van de producten bij de fotolithografiemachines is gelijk aan het minimaliseren van de totale tijd benodigd voor taken die exact één grondstof nodig hebben. We definiëren dit probleem als $P|\mathit{partition}|\sum_j C_j$ (Hoofdstuk 6). We laten zien dat $P|\mathit{partition}|\sum_j C_j$ altijd een optimale oplossing heeft waarbij taken die dezelfde grondstof gebruiken gerangschikt zijn op basis van hun procestijd. De complexiteit van $P|\mathit{partition}|\sum_j C_j$ blijft een open probleem. We laten echter wel zien dat vergelijkbare problemen zoals $P|\mathit{partition}, \mathcal{M}_r|\sum_j C_j$, $P|\mathit{partition}(2), p_j = 1|\sum_j C_j$ en $P|\mathit{partition}, \mathit{unmovable}, p_j = 1|\sum_j C_j$ $\mathcal{NP}$-hard zijn. We kunnen bovendien de optimale oplossing benaderen met een planningsregel, die de taken met de kortste procestijd als eerste plant (SPT-*available* rule). Deze SPT *available* rule geeft een $2 - \frac{1}{m}$-benadering ten opzichte van de optimale oplossing.

1

INTRODUCTION

Nowadays, integrated circuits can be found in many devices, ranging from the traditional personal computers to home appliances and even airbags. An integrated circuit (IC or chip) is a collection of electronic circuits, mainly composed of transistors, on a piece of semiconductor material, normally silicon. One of the first integrated circuits was presented in 1958 by Jack Kilby. It was a small piece of germanium glued to a piece of glass with wires hanging out. When switched on, it produced a continuous sine curve on an oscilloscope screen. The transistors it used were cut from a square centimeter of germanium that contained 25 devices [9]. In the past decades, technological advancements enabled the further miniaturization of these circuits. Today, circuits can be made with a precision of 10 nanometers containing over 45 million transistors on a single square millimeter [14].

Chips are not produced individually, but multiple chips are made on a single disc of silicon, called a wafer. A single wafer can contain thousands of chips. These wafers are produced in a semiconductor factory (often called wafer fab). After the wafer is produced, electronic probing is performed to test the IC basic functionalities. The probed wafers are then sent to assembly, where they are cut into the individual ICs. These are then packaged such that they are protected and wired such that they can connect with the other parts of the electronic device. Next, a final test is performed to see if the chip is working, after which they are labelled and ready to be used. The wafer fab including the probing step are often called the *front end* and the assembly with the final testing are referred to as the *back end*. The front end and back end steps are usually done in different facilities. Of the four stages of chip production, the wafer production is by far the most time consuming and the most expensive.

1.1 THE SEMICONDUCTOR FAB

In the semiconductor fab the chips are manufactured on top of the wafer surface. The construction of the chips is done layer-by-layer. The wafer visits various areas in the fab (so-called bays) which contain different production equipment, thus building gradually the intricate electronic circuits. The most advanced technologies have up to 40 patterned layers and take up to 700 production steps to manufacture. The order in which these bays are visited depends on the design of the chip. Figure 1.1 gives a schematic representation of the production bays and the possible routes the wafer can follow.

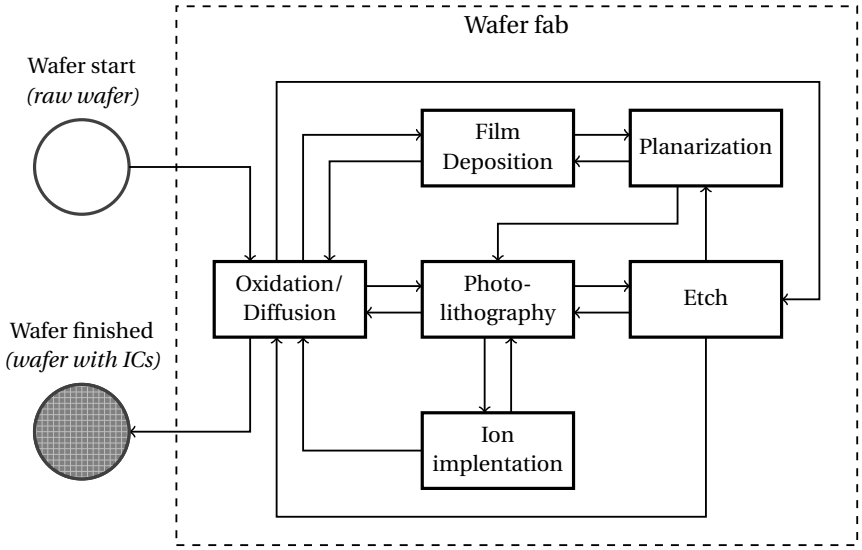


Figure 1.1: Process flow of a wafer through the different production bays of the wafer fab as described by Mönch et al. [11].

In the *film deposition* area a material is transferred to the wafer. The two most used techniques are physical and chemical vapor deposition. Physical vapor deposition is done using a plasma where accelerated gas ions sputter particles from a target onto the wafer in a low pressure plasma chamber. Chemical vapor deposition uses a chemical reaction of a gas mixture on the wafer surface at high temperatures. The materials deposited are quite diverse. The metallization tools put metals such as titanium, titanium-nitride and aluminum on the wafer surface to create conductive layers. The dielectric machines deposit a layer of silicon dioxide (glass) as an electrical insulator.

In the *photolithography* bay, a geometric pattern is transferred to the last deposited top layer. This is done by applying a light-sensitive material (or photoresist) on the wafer. Next, light is directed through a patterned photomask onto the wafer to create the geometric pattern. Section 1.2 covers this process in more detail.

In *implantation*, ionized atoms or molecules are accelerated towards the wafer such that they penetrate the wafer until they come to rest in the silicon layers. These atoms form

impurities in the crystalline structure of the silicon, which changes the electronic properties of the silicon. However, it also damages the crystalline structure. In order to restore this, the wafer has to be heated in a furnace in a process called *annealing*.

Furnaces are also used for *diffusion* and *oxidation*. If silicon is heated while oxygen or water is present, the silicon on the surface will react with oxygen to form silicon dioxide (glass), which serves as an insulating material. This process can be made more effective by increasing the amount of oxygen or water molecules in the atmosphere. Heating also increases the diffusion or movement of impurities, such as those created by ion implantation, in the silicon. These impurities will move out off the silicon as it tries to restore its crystalline structure, making diffusion an effective way to remove impurities.

Etching removes material. This is done either through *dry etch* where surface material is removed by bombarding it with ions using a plasma, or through *wet etch* where the wafer is put in a bath of volatile chemicals that react with the material. Using a patterned mask layer, typically created using a resist and photolithography, etching can be done selectively and the pattern of the photoresist can be transferred onto the wafer.

The combination of deposition, patterning and etching leads to an uneven surface topography. However, a flat and smooth surface is required for an optimal pattern transfer in the photolithograpy step. This constraint becomes more critical for the smallest miniaturizations. To achieve this one makes use of *chemical mechanical polishing/planarization* which flattens the surface of the wafer out with the help of a chemical slurry.

For economic reasons, *i.e.* the price of the processing equipment, a typical wafer fab is organized such that the (expensive) photolithography equipment serves as the bottleneck of the production line. Hence, the overall performance of the wafer fab can be improved by raising the equipment throughput on these photolithography tools.

1.2 PHOTOLITHOGRAPHY

Photolithography is a process step used in semiconductor manufacturing to transfer the geometric pattern of a chip-design onto a wafer as depicted in Figure 1.2. First, a light-sensitive photoresist is put on the wafer. Next, a light beam (visible or UV) is sent through a patterned photomask (the so-called reticle) onto the photoresist. The portion of the photoresist that is exposed to the light will be modified chemically. Additional steps are performed to remove the photoresist material that has been exposed (in case of positive photoresist) or not exposed (in case of negative photoresist). The pattern of the photoresist is transferred to the wafer in subsequent processing steps, such as *dry etch* or *implantation*.

In the semiconductor fab, of which the efficiency improvements are analyzed in this thesis, the photolithograpy process takes place inside a stepper. In the stepper, the reticle image is focused and reduced by a lens to a local rectangular spot (see Figure 1.2). The stepper gets its name from the fact that it moves or *steps* the wafer from one location to another. The local exposure of the wafer is repeated in a grid pattern, resulting in the full

patterning of the entire wafer.

In actual manufacturing, it is quite common that some areas of the wafer surface need to be shielded from exposure. For example, the alignment markers and wafer-ID information should not be exposed. To achieve this the reticle pattern is shielded partially. Any rectangular shape within the full reticle pattern can be selected by moving four blades (top/bottom/left/right) to block unwanted light. Putting such constraints on the full wafer exposure impacts the way in which the wafer is processed and introduces additional images (partial reticle exposure) that need to be transferred to the photoresist. The total time to pattern the full wafer depends critically on the order in which these images are processed as it influences the time needed for stepping, exposure and blading.

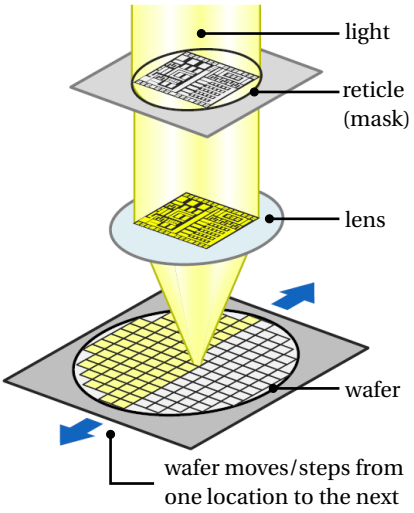


Figure 1.2: Photolithography process

In this thesis, we will focus on the time needed for the photolithography process in the wafer fab. We try to reduce the time needed by using techniques from the field of mathematical optimization. These techniques are analyzed using complexity theory. We will give a brief overview of complexity theory and the two basic versions of the mathematical problems we study in order to optimize the photolithography process: the Traveling Salesman Problem and Scheduling on unrelated parallel machines.

1.3 COMPLEXITY THEORY

Computational complexity theory is a field of mathematics that deals with the question how hard it is to solve a certain computational problem. It studies the resources required to solve a problem, *i.e.*, the memory and time. In this thesis, we will consider optimization problems. An *optimization problem* Π is given by a set of instances $\mathcal{I}$. For each instance $I \in \mathcal{I}$, we have a set $\mathcal{F}$ of feasible solutions for I and a goal function $c : \mathcal{F} \to \mathbb{R}$. The problem is to find the best solution among all feasible solutions, *i.e.*, find a feasible solution $F \in \mathcal{F}$, such that $c(F)$ is the minimized or maximized. Most theory however deals with *decision problems*; problems for which a yes or no answer is required. Every optimization problem can be written as a decision problem by taking a fixed optimization goal k. In this decision problem, we want to determine given an instance I whether or not there exist a feasible solution $F \in \mathcal{F}$, such that $c(F) \leq k$. If such a solution exist, we will call I a *yes-instance*, if such a solution does not exist, we will call I a *no-instance*.

There are many and diverse types of decision problems. To cope with this variety of problems the problems are grouped in certain classes. These are called *complexity classes*. The two main classes we will consider are the classes $\mathcal{P}$ and $\mathcal{NP}$. Intuitively, class $\mathcal{P}$

consist of all problem that can be solved efficiently, while $\mathcal{NP}$ consist of all problems where, if you are given a solution, you can verify it efficiently. More formally, the complexity class $\mathcal{P}$ consists of all decision problems for which there exists an algorithm that for every instance $I \in \mathcal{I}$ can determine whether I is a yes- or a no-instance in polynomial-time, i.e., the time it takes to compute a solution to the decision problem, is polynomial in the *input size*. The input size is number of symbols needed to describe the problem input. It is often the number of nodes or vertices of a graph, the size of a matrix or the number of variables. Note that this size can be dependent on the encoding scheme of a problem. The complexity class $\mathcal{NP}$ consists of all decision problems where given a yes-instance $I \in \mathcal{I}$ and a polynomial-size feasible solution F of this instance with $c(F) \leq k$, called *certificate*, we can check in polynomial time, that I is a yes-instance. Note that every problem that is in $\mathcal{P}$ is also in $\mathcal{NP}$, but not the other way around. Although no proof exists, it is generally assumed that $\mathcal{P} \neq \mathcal{NP}$, *i.e.*, not all problems in $\mathcal{NP}$ can be solved in polynomial time.

An important subclass of $\mathcal{NP}$ is the class $\mathcal{NP}$-complete. $\mathcal{NP}$-complete problems are the most difficult problems in $\mathcal{NP}$. A problem Π is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and for all other problems in $\Pi' \in \mathcal{NP}$ there is an algorithm that transforms Π' to Π in polynomial time. Such an algorithm is called a *polynomial-time reduction*.

Definition 1. *A* polynomial-time reduction *from a decision problem* Π_1 *to another decision problem* Π_2 *is a function* $\phi : \mathcal{I}_1 \to \mathcal{I}_2$ *that maps every instance* $I_1 \in \mathcal{I}_1$ *to an instance* $I_2 = \phi(I_1) \in \mathcal{I}_2$ *of* Π_2 *such that*

- *the time required for the mapping is bounded by a polynomial of the input size of* I_1*;*
- I_1 *is a yes-instance of* Π_1 *if and only if* $I_2 = \phi(I_1)$ *is a yes-instance of* Π_2*.*

The complexity class $\mathcal{NP}$-hard consist of all problems for which the second property of $\mathcal{NP}$-complete problems holds, *i.e.*, a problem is $\mathcal{NP}$-hard if there is a polynomial-time reduction form all problems in $\mathcal{NP}$ to the problem. We will often state that Π_1 can be *reduced to* Π_2 to indicate there is a polynomial time reduction from Π_1 to Π_2. These reductions are often used to show that a problem is $\mathcal{NP}$-hard, if Π_1 can be reduced to Π_2 and a polynomial-time algorithm for Π_2 is known then we can also solve Π_1 in polynomial-time. Furthermore if Π_1 can be reduced to Π_2 and Π_2 can be reduced to Π_3, then also Π_1 can be reduced to Π_3. In other words polynomial-time reductions are transitive. At first sight, proving that a problem is $\mathcal{NP}$-hard might seem hard, however, because of the transitivity of polynomial-time reductions, one can prove that a decision problem Π is $\mathcal{NP}$-hard by showing a reduction from a single $\mathcal{NP}$-hard problem.

Since it is generally assumed that $\mathcal{P} \neq \mathcal{NP}$, it is very unlikely that we can find efficient algorithms for $\mathcal{NP}$-hard problems. When considering these problems we therefore often look at three types algorithms: exponential algorithms, approximation algorithms and heuristics.
Exponential algorithms solve the problem, but are not efficient. Their running time cannot be bounded by a polynomial in the input. *Approximation algorithms* compute a solution efficiently (in time polynomial in the input size) and with a certain performance guarantee. *Heuristics* are all algorithms that find a solution for the problem without any

formal guarantee on the quality of the solution. All these algorithms might perform well in practice depending on the problem. One of the most (in)famous ones is the simplex algorithm for solving linear programming. In theory the simplex algorithm can run for exponential time in the input size, but in most practical instances it solves a linear program quite fast.

As mentioned, an approximation algorithm gives a certain performance guarantee. In general, we will compare its performance against the optimal solution.

Definition 2. *Let* $\text{OPT}(I)$ *and* $\text{ALG}(I)$ *be the objective value of the optimal and the algorithm's solution respectively. An algorithm for a maximization (or minimization) problem* Π *is an* α-approximation algorithm *($\alpha \leq 1$) if for every instance I it runs in polynomial time and finds a feasible solution such that* $\text{ALG}(I) \geq \alpha\text{OPT}(I)$ *(or* $\text{ALG}(I) \leq \alpha\text{OPT}(I)$ *for minimization problems).*

The value of α is called the approximation ratio of the algorithm.

For some $\mathcal{NP}$-hard problems, a lower bound (or upper bound) is known on how well they can be approximated. For example, the Max Cut problem cannot be approximated above a factor $\frac{16}{17}$, unless $\mathcal{P} = \mathcal{NP}$ [6]. Such a result is called an *inapproximability* result. Thus, an inapproximability result states that for an optimization problem there is function $f(n)$, where n is the input size, such that for the approximation ratio α of the problem, it must hold that $\alpha > f(n)$. More details on the dominant approaches for approximation algorithms can be found in the book by Vazirani [16].

One of the most general problems in $\mathcal{P}$ is linear programming. In linear programming, we are given a number of variables $(x_1, \ldots, x_n) \in \mathbb{R}^n_+$ and we want to optimize a linear objective function

$$c(x) = \sum_{i=1}^{n} c_i x_i$$

subject to a set of m linear constraints

$$\sum_{i=1}^{n} a_{ij} x_i \leq b_j, \ \forall j \in \{1, \ldots, m\}.$$

A problem instance of linear programming is called a linear program or LP. A linear program can be solved in polynomial time, for example, by the interior method proposed by Karmarkar [7].

One of the most general $\mathcal{NP}$-complete problems is integer linear programming. Its definition is similar to linear programming except that the variables are integer instead of real valued, *i.e.*, $(x_1, \ldots, x_n) \in \mathbb{Z}^n_+$. It can be shown, due to this difference, the problem becomes $\mathcal{NP}$-hard and can therefore not be solved in polynomial time, unless $\mathcal{P} = \mathcal{NP}$. Similar to linear programming, a problem instance of integer linear programming is called an integer linear program or ILP. Depending on the ILP, different techniques are used to solve it. For a detailed overview of (integer) linear programming and techniques used see Bertsimas and Tsitsiklis [2].

1.4 THE TRAVELING SALESMAN PROBLEM

The Taveling Salesman Problem (TSP) is a classical mathematical optimization problem. TSP states the problem faced by salesmen from the mid 19th until the early 20th century. Salesman sold their wares traveling from city to city. They had a list of cities that they visited on a regular basis. The travel time between cities often was long and hence such a salesman wanted an efficient route to visit all cities before returning back to their hometown. The problem is still relevant today for the delivery of for examples parcels or food.

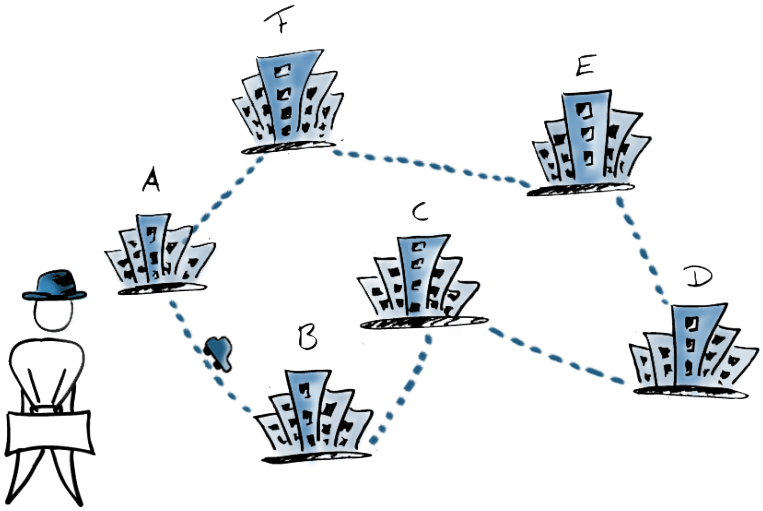


Figure 1.3: Artist impression of the TSP.

Mathematically, the problem is defined on a graph. The cities are defined as the vertices of the graph. The edges in the graph represented the connections between the cities and are assigned a value equal to the distance between two cities. More formally in the Traveling Salesman Problem, we are given a weighted graph $G = (V, E)$ and we want to find a minimum weight cycle (or tour) visiting every vertex exactly once. In the following, we will assume that the edge weights satisfy the triangle inequality and that they are symmetric. Furthermore, we will assume that if an edge between nodes u and v, *i.e.* $e = (u, v)$, is not contained in the graph its weight is equal to the shortest path in the graph between u and v.

The decision version of TSP is $\mathcal{NP}$-complete. This can be shown by a fairly straightforward reduction from the Hamiltonian cycle problem. The Hamiltonian cycle problem is one of the classic 21 problems shown by Karp to be $\mathcal{NP}$-complete [8]. A graph $G = (V, E)$ contains a Hamiltonian cycle if there is a tour visiting all vertices V exactly once using only edges of the graph. Given a graph $G = (V, E)$ the Hamiltonian cycle problem asks if such a tour exists.

The best known approximation algorithm for TSP is the algorithm which is proposed

by Christofides [4]. It gives a $\frac{3}{2}$-approximation. A detailed overview of the history and mathematical results about TSP can be found in the book by Applegate et al. [1].

1.5 SCHEDULING

Scheduling is a decision-making process that deals with the assignment of tasks (or jobs) to certain resources. The assignment is done in such a way that certain objectives are optimized. In our case, the resources are the machines (and possibly the reticles) in the fab and the tasks are the wafers waiting to be processed.

The goal of a scheduling problem is to find a schedule such that the objective is optimized. A schedule is an assignment of the jobs to the machines including an order in which the jobs are processed on the machines. In some schedules (such as a schedule for a problem with precedence constraints) a machine might become idle for some time. If this is the case, a schedule also contains the start and/or completion times of the jobs.

Scheduling problems have been studied in the field of mathematical optimization for over 50 years and many different problems have been considered. To identify these different problems, a framework has been created to name the different layouts, resources, constraints and objective function considered [5]. For each scheduling problem, we are given a set of m machines M and a set of n jobs J. For each job $j \in J$, we are given some data. This may include:

Processing time (p_{ij}) This represents time required to process job j on machine i. We omit the subscript i if the processing time does not depend on the machine.

Release date (r_j) Sometimes a job j is not available when we begin scheduling ($t = 0$). The job will then have a release date r_j assigned to it, representing the time at which the job becomes available.

Due date (d_j) Sometimes a job j is needed to be finished at a certain point in time. A due date is then introduced for that job. The objective function is chosen such that it penalizes jobs that are late.

Weight (w_j) Some jobs might be more important than others. When this is the case a priority or weight (w_j) is assigned to every job and taken into account in the objective function.

The scheduling problem itself is described by the triplet $\alpha|\beta|\gamma$. The α-field describes the machine environment. The β-field describes the environment further and gives the constraints imposed on the jobs and machines. The γ-field contains the objective function that should be optimized.

The α-field can describe very complicated machine configurations where a job might need to be processed more than once and with a specific route. However, in this thesis we will only consider the following machine environments where every job has to be processed once on one of the machines:

Single machine (denoted by a 1 in the α-field This is the simplest type of machine environment; there is a single machine on which jobs can be processed.

Parallel machines (Pm) In this environment we have m machines working in parallel. Each machine has the same speed at which a job is processed, *i.e.* $p_j = p_{ij}$.

Unrelated parallel machines (Rm) In this environment we have m machines working in parallel, but the speed at which a job is processed depends both on the job and the machine. There might also be machines on which a job cannot be processed. When this is the case, $p_{ij} = \infty$.

If the number of machines is part of the input, we will omit the m from the α-field, i.e., P instead of Pm for parallel machines. The β-field can contain many entries. It describes restrictions on the jobs such as restrictions on the processing time (*e.g.* $p_{ij} = 1$) or release dates r_j. But there are many more constraints that can be described in this field. The following classic constraints are considered in this thesis:

Preemptions ($prmp$) If $prmp$ is in the β-field of a scheduling problem, jobs are allowed to be interrupted when processing. The processing is not lost when the proces is interrupted. The jobs can be processed in job parts one after the other on different machines, but not at the same time. If $prmp$ is not in the β-field, jobs have to be processed from start to finish on the same machine and cannot be interrupted.

Precedence constraints ($prec$) When a precedence constraint is put on a job one or more jobs need to be completed before the job can start processing. Precedence constraints are often described in the form of a directed graph $G = (V, A)$. The nodes of this graph are the jobs. If there is an arc from node j to j', job j has to be processed before job j'.

Sequence dependent setup times ($s_{jj'}$) A sequence dependent setup time ($s_{jj'}$) describes the extra time that may be incurred when job j' is processed directly after job j. This happens for example in a furnace when two jobs need different temperatures for processing.

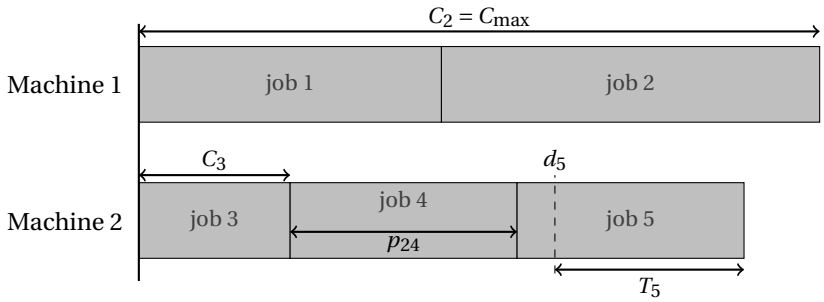


Figure 1.4: The most common variables and objective functions of a scheduling problem

The γ-field contains the objective function of the optimization problem. These objective functions are dependent on the completion time of job j, denoted by C_j. We will

consider the following objective functions which are visualized in Figure 1.4:

Total completion time $\left(\sum C_j\right)$ The total completion time (TCT) objective minimizes the sum of completion of the jobs. It is equivalent to minimizing the average completion time and sometimes also called the mean flow time.

Total weighted completion time $\left(\sum w_j C_j\right)$ The total weighted completion time (TWCT) objective function uses the weights of the jobs to prioritize the completion time of more important jobs.

Makespan ($C_{\max}$) The makespan of a schedule is equal to the completion time of the last job finishing in the schedule, *i.e.*, $C_{\max} = \max_{j \in J} C_j$. This objective will try to schedule the jobs on the machines as evenly as possible.

Total weighted tardiness $\left(\sum w_j T_j\right)$ The tardiness T_j of a job j measures the 'lateness' of a job compared to their due date. It is defined as $T_j = \max\{C_j - d_j, 0\}$. The total weighted tardiness minimizes weighted sum of the tardinesses of the jobs.

The complexity of a scheduling problem depends on all three fields. For example, the single machine problems $1||\sum w_j C_j$ and $1||\sum C_{\max}$ are polynomially solvable, while $1||\sum w_j T_j$ and $1|r_j|\sum C_j$ are $\mathcal{NP}$-complete [10, 15].
In the parallel machines case, $P||\sum C_{\max}$ and $P||\sum w_j C_j$ are $\mathcal{NP}$-complete. $R||\sum C_j$ is polynomially solvable [3]. The book by Pinedo [12] gives a more thorough overview of the different scheduling problems, their complexity and algorithms used to solve them.

1.6 OUTLINE

In this thesis, we will show how to speed up the wafer fab using the current infrastructure. Since the photolithography machines are the bottleneck in the wafer fab, we will focus on those tools. We will do this both at the machine level as well as consider the photolithography bay as a whole.

In the first part of the thesis, we will reduce the time needed to process a wafer inside the photolithography machine. In Chapter 2, we consider the exposure chamber of the photolithography machine and find that we can reduce the time needed, by reducing the time needed for the blade movements. This problem can be formulated as an adjusted for of the Traveling Salesman Problem; *a priori* TSP in the scenario model. We construct and implement a smart IT-solution in the fab, that uses an ILP to solve the problem.
In Chapter 3, we analyze the mathematical properties of *a priori* TSP. We look at it complexity and consider variants and subproblems. We construct approximation algorithms and derive inapproximability results for these.
In Chapter 4, we focus on a specific approximation algorithm for *a priori* TSP; the space-filling curve algorithm, which was first proposed by Platzman and Bartholdi III [13]. We adjust this algorithm such that we can use it to solve real life instances and compare it against the results found by the ILP solver.

In the second part of the thesis, we focus on the entire photolithograpy and metallization bays in the wafer fab. We will show that it is possible to increase the throughput by im-

proving the schedule of the work in progress. Currently, this schedule is constructed by using the priorities assigned by the logistic manager. Due to a recent effort in upgrading the IT-infrastructure, detailed timing data is now available for the components of these two types of machines. Chapter 5 describes how this data is obtained and used to accurately predict the processing time for a job on a specific machine. We will use these processing times to construct a two-stage scheduling algorithm that uses these processing times to find a schedule aimed at minimizing the total completion time, while taking into account the job priorities and the reticles needed for processing.
The scheduling problem found in the photolithography is modeled as $P|partition|\sum_j C_j$ or $R|partition|\sum_j C_j$, depending on the fab considered. The β-field entry *partition* stands for the constraints posed by the reticles. In Chapter 6, we derive complexity and approximation results for these and related problems.

REFERENCES

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[2] D. Bertsimas and J. N. Tsitsiklis. *Introduction to linear optimization*. Athena Scientific Belmont, MA, 1997.

[3] J. Bruno, E. G. Coffman, Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Commun. ACM*, 17(7):382–387, July 1974.

[4] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976.

[5] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[6] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.

[7] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, Dec 1984.

[8] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[9] J. S. Kilby. Invention of the integrated circuit. *IEEE Trans. on Electoron Devices*, 23 (7):651–652, 1976.

[10] J. K. Lenstra. Sequencing by enumerative methods. *MC Tracts*, 1985.

[11] L. Mönch, J. W. Fowler, and S. J. Mason. *Production planning and control for semiconductor wafer fabrication facilities: modeling, analysis, and systems*, volume 52. Springer Science & Business Media, 2012.

[12] M. L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2008.

[13] L. K. Platzman and J. J. Bartholdi III. Spacefilling curves and the planar travelling salesman problem. *Journal of the ACM (JACM)*, 36(4):719–737, 1989.

[14] Qualcomm. Qualcomm datacenter technologies announces commercial shipment of Qualcomm centriq 2400. https://www.qualcomm.com/news/releases/2017/11/08/qualcomm-datacenter-technologies-announces-commercial-shipment-qualcomm.

[15] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[16] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

2

MINIMIZING THE BLADE MOVEMENT IN PHOTOLITHOGRAPHY EQUIPMENT

The production process in a semiconductor factory (often called wafer fab) is complex. The wafer, which contains the chips, will visit different production bays multiple times during its production cycle. As described in Section 1.1 the (expensive) photolithography equipment serve as the bottleneck of the production line. Hence, the overall performance of the fab can be improved by raising the throughput on these litho tools. In this chapter, we investigate the time consumption of the lithography process. It is found that a significant part of the process time is needed for the blade movement to shield a selected part of the reticle image. This blading time depends critically on the order in which different images are processed. As such, this blading time can be modified by changing this order. Minimizing the blade movement can be seen as a new variant of the well-known optimization problem; the traveling salesman problem. We will use an integer linear programming formulation and solver to tackle this problem and investigate its real-time performance when implemented into the Fab Information and Control Systems (FICS) of an existing wafer fab.

2.1 DETAILED BREAKDOWN OF THE WAFER PROCESSING TIME

As described in Section 1.2, photolithography is used in semiconductor manufacturing to transfer the geometric pattern of a chip design onto a wafer. This process is often done by a stepper. The stepper exposes the wafer on rectangular spot a time. If certain parts of

This chapter contains joined work with Jan Driessen

the wafer should not be patterned four blades move in to block the unwanted light. This introduces additional images (partial reticle exposure) that need to be transferred to the photoresist and the total duration to pattern the whole wafer depends critically on the order in which these images are processed.

In a real wafer fab, the stepper is combined with a photoresist line in which multiple wafer operations are integrated to obtain a total wafer operation. This is done in three steps: coating (spinning & baking), exposure and development (resist removal & hard-bake). This is visualized in the schematic representation of Figure 2.1 showing a complete wafer operation, which is a series of 13 processing steps. Some steps are executed on a single component (*e.g.* stepper), while others can be performed in parallel on multiple components (*e.g.* developer).

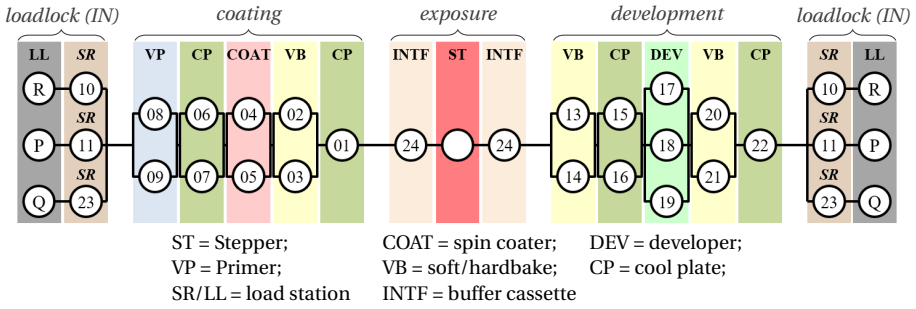


Figure 2.1: Schematic representation of the complete photolithography process (including the coating and development of the photoresist)

In this section, we investigate which variables influence the production speed in such a lithography machine. A batch of (at most) 25 wafers is placed on a loadlock (LL) and the wafers start their operation sequentially. Because the machine has multiple entry components, a new batch of wafers can start its operation in the slipstream of the previous batch. As such, the overall processing time of subsequent lots can overlap in time.

As depicted in Figure 2.1, different wafer operations are performed at different components. Because the exposure step is carried out on the most expensive piece of equipment (the stepper ST), it is essential to maximize its production rate (*i.e.* wafers-per-hour) in order to keep the production costs at its lowest level. To guarantee that the exposure step remains the bottleneck of the complete wafer operation, the coating and development of the photoresist can be done on multiple components. Therefore, when the tool layout is optimized the machine speed will be determined by the exposure step.

The time needed to perform a complete wafer exposure on the stepper depends on the stepper hardware and the actual patterning of the job that needs to be processed (which we call reticle job). The most important machine constants that influence the production time are the light intensity and the speed settings of the stepper components which need to move. Furthermore, the exposure time also depends on the reticle job. Foremost, it depends on the total number of flashes in the grid pattern. But, on top of that,

time is also needed to move the four blades to create the partial reticle exposures. These are needed to shield the wafer-ID information and the alignment markers. As visualized in Figure 2.2, the total blade movement depends critically on the order in which these partial exposures are processed.

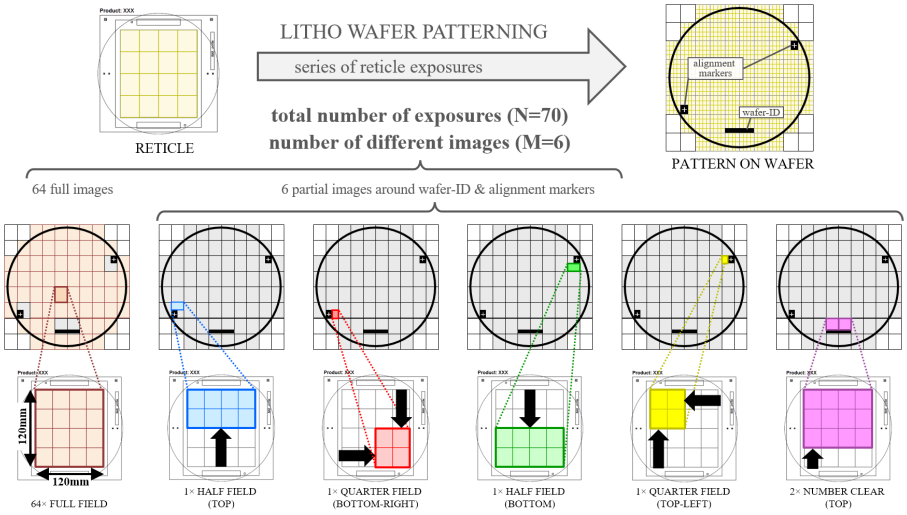


Figure 2.2: The litho wafer patterning of a wafer consists of a series of reticles exposures. Partial exposures around wafer-ID and alignment markers are obtained by moving blades. The total blade movement depends on the order of the images.

In an extensive study by Driessen [3], the observed lithography processing times obtained from the actual wafer fab have been analyzed in combination with the parameters which characterize the reticle job. This resulted in a simple model to express the total process time (T_P) to pattern the wafer

$$T_P = c_0 + c_1(ND/I) + c_2 N + c_3(M-1) + c_4 B \tag{2.1}$$

where the coefficients c_0, c_1, c_2, c_3 and c_4 are equipment constants which represent the stepper hardware. The reticle-job is described by the other parameters: N is the total number of flashes, D is the required energy dose, I is the light intensity at the wafer, M is the number of different images, and B is the total blade movement needed to create the series of partial exposures. Figure 2.2 shows a job with $N = 70$ and $M = 6$.

In the expression of Equation (2.1), we can identify five distinct contributions to the total process time. The first term c_0 is a machine-dependent overhead time. The second part $c_1(ND/I)$ represents the actual time that the wafer is being exposed to the light. The third item $c_2 N$ signifies the stepping time to move the wafer to the N positions in the exposure strategy. The fourth contribution $c_3(M-1)$ deals with overhead time of the stepper to change its machine settings to another partial reticle exposure (image). The final term $c_4 B$ stands for the total time to move the blades to obtain the different

partial images. The coefficients $c_0, \ldots, c_4$ have been determined by a least-squares approximation using Equation (2.1). We found that the 26 steppers in the actual fab can be separated in two hardware groups with different coefficients as listed in Table 2.1.

		16 'fast' tools	**10 'slow' tools**
		97466 lot-runs 7479 reticles 617 products	47016 lot-runs 4042 reticles 607 products
Coef.	Description	ASML PAS5500/250	ASML PAS5500/100 ASML PAS5500/80
c_0	Overhead (varies per tool)	Ranges from 3 to 11 sec	Ranges from 4 to 14 sec
c_1	Light exposure	1.119 (I = 2000 Watt)	1.303 (I = 1000 Watt)
c_2	Stepping speed	0.303 sec	0.284 sec
c_3	Image transition	0.850 sec	1.426 sec
c_4	Blading speed	0.99 sec / 100 mm	3.78 sec / 100 mm

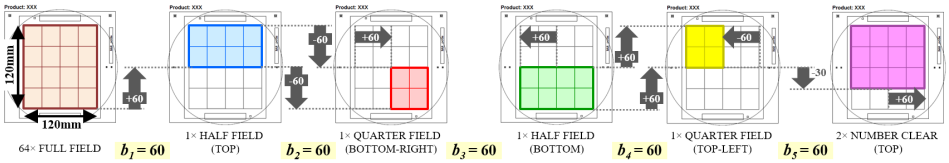
Table 2.1: Data has been analyzed per hardware class over a period of 168 days (Aug 25, 2014 - Feb 09, 2015), resulting in two groups with distinct coefficients.

2.2 BLADING AND THE TRAVELING SALESMEN PROBLEM

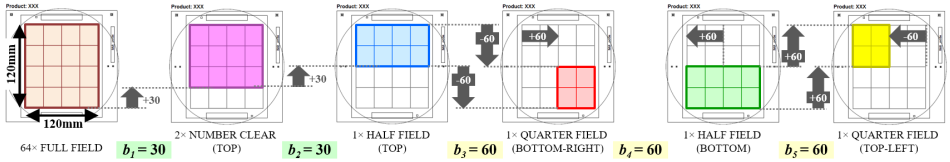
As discussed earlier, depending on the reticle job certain parts of the wafer should not be exposed, *e.g.* wafer-ID and/or alignment markers. Partial reticle exposures are required which are obtained by moving blades to block the light. For each individual image transition (i, j), moving from image i to image j, the blade movement b_{ij} is determined by the blade which needs to move the longest distance. To process a complete wafer a series of images is selected resulting in a total blade movement of $B_{\text{wfr}} = \sum b_{ij}$.

The blading time $c_4 B_{\text{wfr}}$ of Equation (2.1) depends strongly on the order in which the reticle images are processed. By changing the order in which the partial exposures are processed the total blade movement can be reduced. This is exemplified in Figure 2.3 where we have used the previous hypothetical reticle job with $N = 70$ and $M = 6$. In the top graph (a), the original image-order of Figure 2.2 is used resulting in a total blade movement of $B_{\text{wfr(act)}} = 300$ mm. By shuffling the image-order one can reduce the total blade movement resulting in an optimized value $B_{\text{wfr(opt)}} = 240$ mm. Using Equation (2.1) this means that the associated reduction in process time $\Delta t_B = c_4[B_{\text{wfr(act)}} - B_{\text{wfr(opt)}}]$ depends on the coefficient c_4 which represents the mechanical speed at which the blades can be moved. We find that the process time in this example is reduced by 0.6 seconds on the 'fast' tools and by 2.3 seconds on the 'slow' tools using the c_4 value from Table 2.1.

Because of the high-mix character of the fab with more than 600 products, there are more than 7000 different reticle jobs. The number of flashes (N) and images (M) for a typical reticle job are $N \sim 90$ and $M \sim 6$. However, these parameters are distributed over quite wide ranges; $N \in \{70, \ldots, 140\}$ and $M \in \{2, \ldots, 14\}$. All different reticle jobs are stored in a structured manner at the level of a complete product. In other words, all reticle jobs that



(a) The actual image-order results in a total blade movement $B_{\text{wfr}}(\text{act}) = \sum b_{ij} = 300\text{mm}$.



(b) With an optimized image-order, one finds a total blade movement $B_{\text{wfr}}(\text{opt}) = \sum b_{ij} = 240\text{mm}$.

Figure 2.3: Changing the order of the images, results in a reduction of $\Delta B_{wfr} = 60$ mm in blade movement.

are needed to manufacture a specific chip are managed in a single database. As such, the reticle jobs cannot be optimized on an individual basis, but rather on a product basis.

This is illustrated in Table 2.2 with an example of a reticle job matrix to make a certain product. All images are specified including their blading positions. The reticle jobs R_k for each technical stage are specified in the columns on the right. In this example there are 19 technical stages, each with its own reticle job. The number of images per job varies from 2 (`ALIGNMENT MARK`) to 8 (`METAL1/2/3`). The order in which these images are processed is defined by their index in the table. For example, the technical stage '`CONTACT`' has 5 images which are processed in the following order: `FIELD`, `HALF-FIELD-1`, `HALF-FIELD-2`, `HALF-FIELD-3`, and `HALF-FIELD-WN`. One can derive the required displacements from the blade positions of each image, resulting in a total blade movement of $B_{\text{wfr}} = 9.6 + 9.6 + 82.1 + 17.7 = 119.0$ mm. The blade movement is calculated in a similar manner for all other technical stages as listed at the bottom of Table 2.2.

The minimization of the total blade movement for each individual reticle job can be regarded as a variant of the traveling salesman problem. The optimization of the total blade movement at product level, B_{PROD}, comes down to calculating an order for the image superset such that the sum of all blade transitions over all reticle jobs $B_{\text{wfr}}(k)$ is minimized. The positions of the four blades can be regarded as points in a four-dimensional space. For a single reticle $B_{\text{wfr}}(k)$, the problem is to find a path between the points such that we visit every point exactly once and minimize the distance traveled. This problem is well known in literature and is called the Metric Traveling Salesman Path Problem (metric TSP path).

The difference lies in that any reordering of the superset of images will affect all reticle jobs R_k simultaneously. As such, the blade movement cannot be optimized for indi-

		BLADE POS.					TECHNICAL STAGE (= reticle R_k)																	
	IMAGE ID	LEFT (mm)	RIGHT (mm)	TOP (mm)	BOTTOM (mm)	# FLASHES	ALIGNMENT MARKER	ACTIVE PATTERNING	CONTACT HANDLE	CONTACT	VIA1	VIA2	N PLUS	P PLUS	NWELL DRIFT	PWELL DRIFT	SN DMOSS	SP DMOS	PASSIVATION	WAFER COAT	POLY PATTERNING	TRENCH ISOLATION	METAL 1	METAL 2	METAL 3
1	PM	0.8	-0.8	52.0	53.6	**1**	✓																		
2	PF	-8.5	-16.5	62.0	70.0	**1**																	✓	✓	✓
3	FIELD	54.4	-54.4	-48.5	48.5	**82**		✓	✓	✓	✓	✓									✓	✓			
4	CONV-PF	-8.5	-16.5	62.0	70.0	**2**															✓				
5	HALF-FIELD-1	44.8	-54.4	-48.5	48.5	**1**		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
6	HALF-FIELD-2	54.4	-44.8	-48.5	48.5	**1**		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
7	HALF-FIELD-3	54.4	-54.4	33.6	48.5	**1**		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
8	KLA-SMALL	38.8	36.8	64.9	66.9	**2**																✓			
9	HALF-FIELD-WN	54.4	-54.4	15.9	48.5	**1**		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
10	FIELD-IMP	54.4	-54.4	-48.5	48.5	**77**							✓	✓	✓	✓	✓	✓							
11	FIELD-MIN4	54.4	-54.4	-48.5	48.5	**77**													✓	✓					
12	KLA-BIG	10.8	9.2	52	53.6	**2**	✓																		
13	NR-CL1	27.8	-27.8	-70	-62	**1**																	✓	✓	✓
14	NR-CL2	27.8	-27.8	-70	-62	**1**																	✓	✓	✓
15	NR-CL3	27.8	-27.8	-70	-62	**1**																	✓	✓	✓
16	FIELDMET	54.4	-54.4	-48.5	48.5	**87**																	✓	✓	
17	FIELDMET3	54.4	-54.4	-48.5	48.5	**82**																			✓
18	NR-CL4	27.8	-27.8	-70	-62	**1**																	✓	✓	✓
19	NR-CL5	27.8	-27.8	-70	-62	**1**																	✓	✓	✓
20	NR-CL6	27.8	-27.8	-70	-62	**1**																	✓	✓	✓
	BLADE MOVEMENT $B_{\text{wfr}}(k)$ (mm)						10	119	119	119	119	119	174	174	174	174	174	174	174	174	330	284	353	353	353
	PRODUCT BLADE MOVEMENT B_{PROD}(original) = $\sum_k B_{\text{wfr}}(k)$ = 3670 mm																								

Table 2.2: Reticle jobs are managed collectively for each product in a single database. In this product-example the required images are listed vertically. The columns on the right specify which images are used for each technical stage. The order in which the images are processed is defined by their index in this table.

vidual reticles. Therefore, the optimization problem is re-defined to minimize the total blade movement at product level B_{PROD}, which is the sum over all reticle jobs k, $B_{\text{PROD}} = \sum_{k=1}^{m} B_{\text{wfr}}(k)$. In the analogy with the traveling salesman problem, one can think of the list of images as a list of cities. Each reticle job R_k can be regarded as an individual salesman who has to visit a subset of cities. In this manner, the total blade movement B_{PROD} is equal to the total distance traveled by multiple salesmen.

More formally, we are given a set V of n points, nodes or cities and the distance between all cities d_{ij} is according to the maximum metric (i.e. L_∞-norm or $\|\cdot\|_\infty$) and given $i, j \in V$. We have m salesmen. Every salesman k has to visit a subset $S_k \subseteq V$ of the cities where it visits every city exactly once. We will call these subsets scenarios. Let

$$x_{kij} = \begin{cases} 1 & \text{, if salesman } k \text{ visits city } j \text{ directly after city } i \\ 0 & \text{, otherwise} \end{cases} \tag{2.2}$$

Now, We want to find an ordering of the cities such that

$$\sum_{k=1}^{m} \sum_{i \in J_k} \sum_{j \in J_k, j \neq i} d_{ij} x_{kij}$$

is minimized.

If we divide the objective function by m, we are minimizing the expected value of the tour, where each scenario has probability $\frac{1}{m}$. The problem then transforms to a variant of the *a priori* traveling salesman problem (*a priori* TSP), where the salesman has a discrete distribution over the scenarios $S_1 \dots S_m$.

The traveling salesman problem has been studied in the context of photolithography on other occasions: A reticle is created using a plotter. This plotter can draw dots and lines. Minimizing the distance traveled by the plotter to create the dots can modelled as the symmetric TSP[5]. Furhtemore, solution strategies for the generalized asymmetric TSP problem are used by Kuijpers et al. [6] to find an optimal movement strategy for scanning the wafer when processing a single image. More recently, the geometric TSP is also use for finding a route for maskless photolithography without a dedicated microfabrication facility [7].

2.3 OPTIMIZATION ALGORITHM

In this section, an integer linear program (ILP) formulation is discussed for the scenario *a priori* TSP path problem. Next, this formulation is used as input for an ILP solver to find the optimized image order giving the minimized total blade movement B_{total} for each product that is manufactured in the semiconductor fab. Given an instance I of the scenario TSP path problem, we want to write it as an integer linear program (ILP). In order to do this, we introduce a dummy node p_0. This dummy node has $d_{0i} = d_{i0} = 0, \forall i \in V$. It will act as the begin and end point of all our TSP paths essentially turning them in tours. Let $\bar{S}_k = S_k \cup p_0, \forall k \in \{1, \dots m\}$. We consider the following ILP.

$$\min \sum_{k=1}^{m} \sum_{i \in \bar{S}_k} \sum_{j \in \bar{S}_k, i \neq j} d_{ij} x_{kij}$$

$$\text{s.t.} \quad \sum_{i \in \bar{S}_k, i \neq j} x_{kij} = 1, \qquad \forall j \in \bar{S}_k, \forall k \in \{1, \dots, m\}, \tag{2.3}$$

$$\sum_{j \in \bar{S}_k, i \neq j} x_{kij} = 1, \qquad \forall i \in \bar{S}_k, \forall k \in \{1, \dots, m\}, \tag{2.4}$$

$$u_i - u_j + n x_{kij} \leq n - 1, \forall i \in S_k, \forall j \in S_k \setminus \{i\}, \forall k \in \{1, \dots, m\}, \tag{2.5}$$

$$x_{kij} \in \{0, 1\}, \qquad \forall i \in S_k, \forall j \in S_k \setminus \{i\}, \forall k \in \{1, \dots, m\},$$

$$1 \leq u_i \leq n - 1, \qquad \forall i \in S_k. \tag{2.6}$$

Constraints (2.3) and (2.4) make sure that salesman k visits every node exactly once and leaves that node exactly once. Constraints (2.5) and (2.6) were first stated by Miller et al. [8]. They make sure that the nodes can be ordered in an ascending order (by the values of u_i). This order is identical for every salesman because the values of the u_i can only be set once.

The formulation by Miller et al. imposes an ordering of the cities, to which all salesmen have to comply. We use this formulation because most other classic formulations use

constraints on subsets to make sure the solution has no subcycles. These cycle canceling constraints do not impose that two salesmen follow the same ordering of the cities and thus need additional constraints to impose such an ordering. Hence, we use the formulation by Miller et al. [8], because it is one of the few that can be easily extended.

In the literature, there are formulations that incorporate the Miller-Tucker-Zemlin constraints found in Equation (2.5) and are stronger in the sense that their LP relaxation gives a better lower bound on the ILP. Desrochers and Laporte [2] use the relation of the MTZ formulation to vehicle routing problems to strengthen the formulation. Gouveia and Pires [4] introduce variables $v_{ij}, \forall i, j \in J$. Given an arbitrary but fixed begin node 1, $v_{ij} = 1$ if i lies on the path between 1 and j and is zero otherwise. Thus $u_j = \sum_{i \in J} v_{ij}$. Using the variables v_{ij}, the ILP is reformulated and the formulation is further strengthened with additional constraints.

When the above ILP formulation is applied to a real life instance such as the example of Table 2.2, the algorithm splits the problem in subproblems, when the images have no overlap. The algorithm can be made even more efficient by taking into account images with identical blade settings, e.g. grouping all `NR-CLx` images and/or grouping the set `FIELD`, `FIELD-IMP`, `FIELD-MIN4`, `FIELDMET` and `FIELDMET3`. Furthermore, we recognize several reticle jobs R_k (or technical stages) which use the same subset of images. Thus, the optimization problem can be further reduced by replacing M similar reticle jobs with only one, and give it a weighting factor w_k. We find that these replacements can often reduce the size of a large optimization problem (*e.g.* 30 images + 50 reticles) to a smaller problem (*e.g.* 10 images + 10 reticles). Next, the scaled-down problem is optimized with the ILP solver SCIP [1]. Afterwards, the optimized solution is scaled up to the original database size by duplicating the identical images and technical stages.

The example product of Table 2.2 has been optimized using the ILP. This results in a modified database which is given in Table 2.3. Using the above reduction mechanism, the original problem (20 images + 19 reticles) has been reduced to a smaller issue (11 images + 5 reticles) and optimized. The results have been expanded into the full problem resulting in 13 reticle jobs with a blade movement $B_{\text{wfr}}(k) = 101$mm and 3 reticle jobs with $B_{\text{wfr}}(k) = 221$mm.

The total blade movement has been reduced by -36% from $B_{\text{PROD}} = 3670$ mm (Table 2.2) to $B_{\text{PROD}} = 2342$ mm (Table 2.3) . For individual reticle-jobs R_k the improvement varies from $\Delta B_{\text{wfr}}(k) = 0$ mm (`ALIGNMENT MARKER`), 18 mm (*e.g.* `CONTACT`), 73 mm (*e.g.* `N PLUS`), 120 mm (`TRENCH ISOLATION`), 122 mm (*e.g.* `METAL1`) up to 166 mm (`POLY PATTERNING`).

Without the knowledge of the time-consumption associated with the image transitions, the litho reticle-jobs have been modified regularly over time to accommodate product changes, *e.g.* 'more-dies-per-wafer', 'critical dimension (CD) control' or 'solving process-integration issues'. When more images were needed, they were added to the bottom of the image list. This can be recognized from Table 2.2, where the images `FIELDMET` and `FIELDMET3` are copies of the full field image `FIELD`. In a similar manner, the images `NR-CLR4/5/6` have been added as repeats of `NR-CLR1/2/3`. As a result, the original list of images is not necessarily the optimized order to minimize the total blading distance.

		BLADE POS.					TECHNICAL STAGE (= reticle R_k)																	
	IMAGE ID	LEFT (mm)	RIGHT (mm)	TOP (mm)	BOTTOM (mm)	# FLASHES	ALIGNMENT MARKER	ACTIVE PATTERNING	CONTACT HANDLE	CONTACT	VIA1	VIA2	N PLUS	P PLUS	NWELL DRIFT	PWELL DRIFT	SN DMOSS	SP DMOS	PASSIVATION	WAFER COAT	POLY PATTERNING	TRENCH ISOLATION	METAL 1	METAL 2	METAL 3
1	PM	0.8	-0.8	52.0	53.6	**1**	✓																		
2	PF	-8.5	-16.5	62.0	70.0	**1**																	✓	✓	✓
3	HALF-FIELD-1	44.8	-54.4	-48.5	48.5	**1**		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
4	HALF-FIELD-2	54.4	-44.8	-48.5	48.5	**1**		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
5	FIELD	54.4	-54.4	-48.5	48.5	**82**		✓	✓	✓	✓	✓									✓	✓			
6	FIELD-IMP	54.4	-54.4	-48.5	48.5	**77**							✓	✓	✓	✓	✓	✓							
7	FIELD-MIN4	54.4	-54.4	-48.5	48.5	**77**													✓	✓					
8	FIELDMET	54.4	-54.4	-48.5	48.5	**87**																	✓	✓	
9	FIELDMET3	54.4	-54.4	-48.5	48.5	**82**																			✓
10	HALF-FIELD-WN	54.4	-54.4	15.9	48.5	**1**		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
11	HALF-FIELD-3	54.4	-54.4	33.6	48.5	**1**		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
12	KLA-BIG	10.8	9.2	52	53.6	**2**	✓																		
13	KLA-SMALL	38.8	36.8	64.9	66.9	**2**																✓			
14	CONV-PF	-8.5	-16.5	62.0	70.0	**2**															✓				
15	NR-CL1	27.8	-27.8	-70	-62	**1**																	✓	✓	✓
16	NR-CL2	27.8	-27.8	-70	-62	**1**																	✓	✓	✓
17	NR-CL3	27.8	-27.8	-70	-62	**1**																	✓	✓	✓
18	NR-CL4	27.8	-27.8	-70	-62	**1**																	✓	✓	✓
19	NR-CL5	27.8	-27.8	-70	-62	**1**																	✓	✓	✓
20	NR-CL6	27.8	-27.8	-70	-62	**1**																	✓	✓	✓
	BLADE MOVEMENT $B_{\text{wfr}}(k)$ (mm)						10	101	101	101	101	101	101	101	101	101	101	101	101	101	164	192	221	221	221
	PRODUCT BLADE MOVEMENT B_{PROD}(optimized) = $\sum_k B_{\text{wfr}}(k)$ = 2342 mm																								

Table 2.3: The reticle jobs of Table 2.2 are expressed with an optimized image-order. The blade movement $B_{\text{wfr}}(k)$ per reticle has been modified. At the product level the blade movement B_{PROD} has been minimized, resulting in significant reductions for the 5 reticles on the right (`POLY, TI, METALx`).

To estimate the potential efficiency improvement in the actual high-mix fab with more than 500 products, we analyzed in a preliminary study the impact of the blade movement reductions for the 46 products, that make up 45.54% of the total 'work-in-progress' (WIP). The optimization algorithm using the ILP solver was tested on a laptop using the MATLAB application. The results of the optimization can be found in Appendix A.2. For 37 of the 46 products the total blading distance was reduced. On average, the blading distance reductions are about 20%. In terms of total exposure time (see Equation (2.1)), this results in a time reduction of 1.5% on 'fast' tools and 1.7% on the 'slow' tools for the analyzed WIP. On average, the optimization takes 1.23 seconds and at most 15 seconds using the MATLAB application. Since the optimization only has to be done once and can be done before the product goes into production, 15 seconds (or even 10 minutes) is not an issue.

2.4 IMPLEMENTATION

Based on the preliminary analysis, there is a significant reduction in processing time by reducing when the blade movement is reduced. Therefore, it has been decided to con-

struct a smart manufacturing solution into FICS of the ICN8 fab. Figure 2.4 visualizes the optimization. The optimization algorithm has been integrated into the existing IT-architecture where the reticle-jobs are managed. On four stand-alone workstations the lithography information of more than 600 products is stored and maintained, including the historic information. It contains all the particular job-data for more than 40000 reticles. A new IT-solution has been added to this IT-architecture, the so-called LJM (Litho Job Manager). Through an intuitive GUI-interface, it has become easier for the engineer to create and/or modify the litho reticle jobs. Each time when a product library has been modified, the optimization algorithm is invoked to optimize the image order such that the total blade movement is minimized.

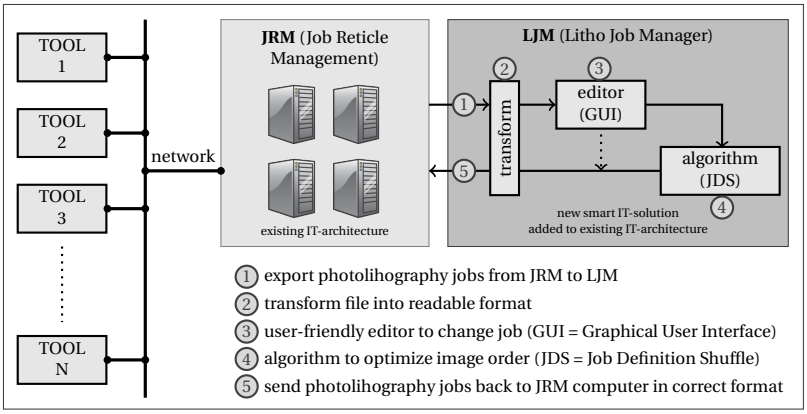


Figure 2.4: A new smart IT-solution has been integrated into the existing IT-architecture. Products are created and/or modified with a user-friendly GUI. The optimization algorithm is executed before the litho job information is stored on the JRM servers.

Next, the optimized litho job information is sent back to the JRM-computer. When a specific reticle will be processed on the production equipment, the reticle-job is sent to the tool. As explained earlier, when a reticle-job is dispatched on the tool the images are processed in the list-order stored in the database (see Tables 2.2 and 2.3).

In 2014Q3 this new smart solution has been implemented. Over the course of a two-month period all products stored on the four stand-alone workstations have been processed and optimized with the Litho Job Manager. The calculated reductions of the blade movement for each reticle job R_k ($\Delta B_{\text{wfr}}(k)$) can be transformed in a process time reduction $\Delta T_p^k = c_4 \Delta B_{\text{wfr}}(k)$ using the coefficient c_4 from Table 2.1. The predicted gains in process time have been compared to observed expose durations for the period 2014Q3. In total, 79000 lots have been processed using 8845 different reticles on the 16 tools listed in Table 2.1. On average each reticle has been used only 9 times in this three-month period, *i.e.* less than once per photolithography tool. Therefore, in this high-mix production environment the process time reduction cannot be recognized immediately from the raw equipment data.

To validate the predicted time gains ΔT_p^k we have analyzed the observed expose dura-

tions for large runners where reticle timing data on a specific photolithography tool is available both before and after the photolithography job optimization. An example of the gain in process time is shown in Figure 2.5, where the actual expose durations on five 'slow' tools of Table 2.1 are shown for two reticles. For the first reticle `N WELL` no blade movement reduction is found, which results in the same process time over the whole 3-month period. For the second reticle `METAL5` the blade movement reduction is $\Delta B_{\text{wfr}}(k) = 233$ mm, which gives a time reduction of 8.8 sec on photolithography tools with a slow blading speed $c_4 = 3.78$ sec/100 mm. This predicted value agrees with the observed process times as is shown in the bottom graph of Figure 2.5.

TECHNICAL STAGE (= reticle R_k)	# FLASHES (N)	# IMAGES (M)	actual $B_{\text{wfr}}(k)$ (mm)	optimized $B_{\text{wfr}}(k)$ (mm)	$\Delta B_{\text{wfr}}(k)$ (mm)	fast ΔT_p^k (sec)	slow ΔT_p^k (sec)
`MARKER` (1×)	2	2	13	13	0	0.0	0.0
`TRANSISTOR` (8×)	90	2	57	57	0	0.0	0.0
`POLY` (2×)	90	4	86	86	0	0.0	0.0
`ACTIVE` (1×)	92	5	202	172	31	0.3	1.2
`VIA1-4` (4×)	96	7	285	171	115	1.1	4.3
`SILICIDE` (1×)	98	8	361	228	133	1.3	5.0
`CONTACT` (1×)	96	7	200	171	29	0.3	1.1
`CONTROL GATE` (1×)	98	10	359	227	132	1.3	5.0
`METAL1-5` (5×)	100	12	517	285	232	2.3	8.8

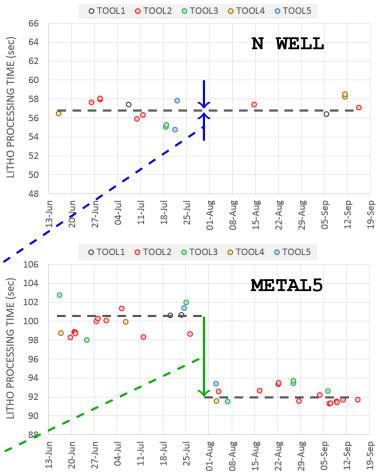


Figure 2.5: For each product, the Litho Job Manager optimizes the image order resulting in different blade movement reductions for each reticle job. Depending on the litho hardware (c_4) this translates in possible process time reductions. These predictions have been confirmed by measured durations. For large runners with many observations, the time changes can be evidenced easily from the raw data as shown here for `N WELL` *and* `METAL5`*.*

To estimate the photolithography capacity gain that has been achieved by implementing the LJM smart solution, we analyzed the impact of the blade movement reduction on all lot-runs in the period listed in Table 2.1 (Aug 25, 2014 - Feb 09, 2015). On average, the blade movement has been reduced by 21.0%. On the 16 fast tools the average exposure time has been reduced from 43.3 sec to 42.6 sec (−1.6%), and on the 10 slow tools the mean litho process time was lowered from 70.5 sec to 69.2 sec (−1.8%). In the same study we can compare the takt time of the litho equipment (takt time is equal to the process time plus the handling time) and the takt time of the photoresist line, and identify which lot-runs benefited from the LJM-optimization. From this analysis, we estimate that we have gained approximately 1.2% capacity on the 26 litho tools of Table 2.1. In other words, the equipment throughput has been raised and less machine time is needed to process a certain amount of WIP in comparison to the old situation without the LJM smart solution.

2.5 CONCLUSION

The time to process a wafer on a lithography stepper machine can be decomposed in non-overlapping time elements for light exposure, wafer movement (stepping), blade movement and machine overhead. The blading time makes up a significant fraction of the total time and depends crucially on the order in which the different images are processed. An ILP formulation has been used to optimize the image order. Minimizing the blading boils down to solving *a priori* TSP problem in the scenario model. Although this problem is very hard to solve in theory (see Chapter 3), the instances resulting from the blading inside the lithography machines are structured enough that they can be solved in a limited amount of time using an ILP solver. In this manner the average blading distance (and blading time) have been minimized. We estimate that per product the total blading movement has been reduced by approximately 20%, which results in a reduction of the average exposure time of around 1.6%. As a result, the overall throughput of the litho steppers (26 machines) has been raised by about 1.2%. Since the lithography machines are often the bottleneck of the semiconductor fab, this new smart solution including the ILP solver can be regarded as one of the key enablers to transform a manufacturing plant into a smart factory.

REFERENCES

[1] T. Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. `http://mpc.zib.de/index.php/MPC/article/view/4`.

[2] M. Desrochers and G. Laporte. Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. *Operations Research Letters*, 10(1):27–36, 1991.

[3] J. Driessen. An OEE increase of 10 percent on litho equipment. In *12th European Advanced Process Control and Manufacturing Conference*. APC|M, 2012.

[4] L. Gouveia and J. M. Pires. The asymmetric travelling salesman problem and a reformulation of the miller–tucker–zemlin constraints. *European Journal of Operational Research*, 112(1):134–146, 1999.

[5] M. Grötschel, M. Jünger, and G. Reinelt. Optimal control of plotting and drilling machines: a case study. *Zeitschrift für Operations Research*, 35(1):61–84, 1991.

[6] C. Kuijpers, C. Hurkens, and J. Melissen. Fast movement strategies for a step-and-scan wafer stepper. *Statistica Neerlandica*, 51(1):55–71, 1997.

[7] S. Y. Leigh, A. Tattu, J. S. Mitchell, and E. Entcheva. M 3: Microscope-based maskless micropatterning with dry film photoresist. *Biomedical microdevices*, 13(2):375–381, 2011.

[8] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, Oct. 1960. ISSN 0004-5411.

3

A priori TSP IN THE SCENARIO MODEL

In Chapter 2, we came across the problem of minimizing the distance blades have to travel during the expose step in the photolithography processes. This process is used in semiconductor manufacturing to transfer the geometric pattern of a chip onto a wafer. Since the expose step often influences the total processing time of a wafer in the lithography machine, minimizing the distance reduces the processing time. We showed that it can be formulated in such a way that it is precisely a form of the *a priori* TSP in the scenario model.
In this Chapter, we will look at the *a priori* TSP in the scenario model. In the classical traveling salesman problem as described in Section 1.4, we are given a set of cities and the distances between them and we want to find the shortest tour that visits all cities exactly once. We extend this classical routing problem to the case that the set of cities is uncertain. This uncertainty might be in the problem itself, such as in the case of the blades in the photolithography machines. It also might be that the city set changes regularly and that reoptimizing over and over again might be inconvenient or impossible. In the *A priori* TSP, we want to find a single tour for all cities. Given this tour and a set of cities, the active set, we shortcut the tour to the active set. In universal routing, the goal is to minimize the worst-case ratio of the value of the obtained solution and the deterministic optimal value of a tour on the active set. In *a priori* routing, we want to be good on average. The problem we consider in this chapter is formally defined as follows.

In the *a priori* traveling salesman problem (*a priori* TSP) in the scenario model, we are given a complete weighted graph $G = (V, E)$ and a set of scenarios $\mathcal{S}$ with $S_1, \ldots, S_m \subseteq V$. Scenario S_j has probability p_j of being the active set, where $\sum_j p_j = 1$. We begin by finding an ordering on V, called the first-stage tour. When an active set is released, the

This chapter is based on the paper that appeared in the journal of Discrete Applied Mathematics[27]

second-stage tour is obtained by shortcutting the first-stage tour on the vertices of the active set. The goal is to find a first-stage tour that minimizes the expected length of the second-stage tour. Throughout this chapter, we assume that the edge weights obey the triangle inequality.

A priori TSP has already been considered in the independent decision and black-box model. In the independent decision model, vertex i is active with probability p_i, independent of the other vertices. Shmoys and Talwar [26] showed that a sample-and-augment approach gives a randomized 4-approximation, which can be derandomized to an 8-approximation algorithm. This factor was improved by Van Zuylen [28] to 6.5. In the black-box model, we have no knowledge on the probability distribution over the vertices, but we are able to sample from it, i.e. to query the probability of any subset of the vertices. Schalekamp and Shmoys [25] showed that one can obtain a randomized $O(\log n)$-approximation even without sampling. A deterministic $O(\log^2 n)$-approximation can be obtained by using the result for universal TSP [14]. It was shown by Gorodezky et al. [13] that there is an $\Omega(\log n)$ lower bound for deterministic algorithms on general metrics. By using the result of [16] and Theorem 3 in [13], there is no deterministic algorithm with guarantee $o\left(\sqrt[6]{\log n/\log\log n}\right)$ for planar metrics. For randomized algorithms, no lower bound is known for the black-box model.

The above mentioned results give us the first results for *a priori* TSP in the scenario model. Firstly, we inherit the randomized $O(\log n)$-approximation. Secondly, we know that a deterministic algorithm that does not use the information given in the scenarios will not achieve an approximation guarantee better than $O(\log n)$. The main question is whether we can use the scenarios to improve upon the $O(\log n)$ upper bound and which restrictions we can put on the scenarios in order to obtain constant-factor approximability. This question will be considered in this chapter.

The scenario model has not been studied extensively for other optimization problems. Immorlica et al. [18] investigated scenario versions of Vertex Cover and Shortest Path. Ravi and Sinha [23] also looked at these problems and also defined scenario versions of Bin Packing, Facility Location and Set Cover. The problems in [23] differ from our setting in the sense that the weights used in the instance differ between scenarios. Further, the authors of [6] investigate a two-stage stochastic scheduling problem, where the set of jobs to be processed is uncertain. Finally, in [10], the classical scheduling problem of minimizing the makespan on two machines is considered in the *a priori* model with scenarios. It would be interesting to consider other stochastic combinatorial optimization problems in the *a priori* framework.

A priori TSP can be considered as a stochastic version of TSP. Alternatively, one could consider a robust version where we want to minimize the maximum length over all scenarios. We will refer to this problem as Min-Max TSP. When applicable, we will state to which extend the theorems for *a priori* TSP also hold for the Min-Max TSP. An easy observation is that the approximation ratios for universal TSP carry over directly to Min-Max TSP. Hence, we have an $O(\log^2 n)$-approximation algorithm.

We will first examine the most natural lower bound that we call the master tour lower bound. We use this lower bound to show that there exists a constant-factor approxima-

tion algorithm for the problem if the number of scenarios is fixed. However, we also show that this lower bound cannot be used to improve upon the $O(\log n)$-approximation. We then look at several natural restrictions on the scenarios, namely small, big and nested scenarios. For small scenarios, we give strong inapproximability results. After that, we analyze the performance of the optimal tour on V for big scenarios. For nested scenarios, we show that there exists a 9-approximation algorithm. Finally, we show that there exists an elegant connection to an *a priori* minimum spanning tree problem. We end with a discussion on some open problems.

3.1 MASTER TOUR LOWER BOUND

In this section, we explore the master tour lower bound. An instance of TSP has the master tour property if there is an optimum TSP tour on all vertices in the set S such that the optimal TSP tour of any subset of the vertices can be obtained by simply shortcutting the optimal TSP tour on the vertices that are not in the subset. A natural question one can ask is whether an optimal tour on all vertices performs well compared to the actual optimal tour for *a priori* TSP in the scenario model.

Here, we use that the contribution of scenario S_j to the objective value of an optimal solution, denoted by OPT, is at least $p_j T_j^*$, where T_j^* is the length of the optimal tour on S_j, so $\text{OPT} \geq \sum_j p_j T_j^*$. Two natural algorithms for *a priori* TSP in the scenario model are the following. For each scenario, find an α-approximate tour, where α is the best approximation ratio available for TSP, and sort the scenarios on their resulting tour lengths T_j. Rename the scenarios such that $T_1 \leq T_2 \leq \ldots \leq T_m$. Now traverse the tours $1, 2, \ldots, m$, skipping already visited vertices, resulting in tour τ_1. Alternatively, rename the scenarios such that $p_1 \geq p_2 \geq \ldots \geq p_m$ and traverse the tours $1, 2, \ldots, m$, skipping already visited vertices, resulting in tour τ_2. We get the following result.

Theorem 3.1. *Tours τ_1 and τ_2 are $(2m-1)$-approximations for* a priori *TSP in the scenario model, where $m \geq 2$ is the number of scenarios.*

Proof. Let us analyze tour τ_1. Consider an arbitrary scenario S_j. Let D_j be the diameter of G restricted to S_j, so we have $T_j^* \geq 2D_j$. Note that when analyzing the contribution of scenario S_j, we only have to consider tours that contain vertices in S_j. Further, it might happen that two tours, say T_x and T_y, with $x, y < j$, $S_x \cap S_j \neq \emptyset$ and $S_y \cap S_j \neq \emptyset$, belong to disjoint scenarios. In this case, we have to go from T_x to T_y. If $d(A, B)$ denotes the maximum distance between a vertex in A and a vertex in B, then this move costs us at most an extra $d(S_x \cap S_j, S_y \cap S_j)$. In the worst case, all scenarios before S_j have a non-empty intersection with S_j. For $j = 1$, the contribution is just $p_1 T_1 \leq \alpha p_1 T_1^*$. For $j \geq 2$, the contribution of S_j to the objective value of our solution is at most

$$\begin{aligned} &p_j(T_1 + d(S_1 \cap S_j, S_2 \cap S_j) + T_2 + \ldots + d(S_{j-2} \cap S_j, S_{j-1} \cap S_j) + T_{j-1} + T_j) \\ \leq &p_j(jT_j + (j-2)D_j) \leq p_j\left(\alpha j T_j^* + (j-2)\frac{1}{2}T_j^*\right) = \left(\left(\alpha + \frac{1}{2}\right)j - 1\right)p_j T_j^*. \end{aligned}$$

Note that you do not have to incur an extra distance from S_{j-1} to S_j, since they have a non-empty intersection. In general, this holds for the last scenario that intersects with S_j. The objective value is at most

$$\alpha p_1 T_1^* + \sum_{j=2}^{m} \left(\left(\alpha + \frac{1}{2} \right) j - 1 \right) p_j T_j^* \leq \left(\left(\alpha + \frac{1}{2} \right) m - 1 \right) \text{OPT}.$$

Since $\alpha = 1.5$, see [7], we get a $(2m-1)$-approximation algorithm. The analysis for τ_2 is similar and the proof is omitted here. □

Since in the proof of Theorem 3.1 we bound the length of each tour by $2m-1$ times the optimal tour for that scenario, it is obvious that τ_1 and τ_2 are also $(2m-1)$-approximations for Min-Max TSP.

It turns out that the master tour lower bound will not give a constant-factor approximation for *a priori* TSP on general metrics. This can be deduced from Theorem 2 in [13], which roughly states the following. Suppose you are given a d-regular Ramanujan graph G on n vertices with girth $g \geq \frac{2}{3} \log_{d-1} n$. Take a random walk of length $70g$ in G and let S be the vertices visited in this walk. Now, consider a TSP-tour on the vertices of G. Theorem 2 in [13] states that for each of the first $g/2$ steps of the tour restricted to S, the probability that the edge has length $\Omega(\log n)$ is bounded from below by a constant.

Theorem 3.2. *There is an instance of* a priori *TSP in the scenario model such that* $\text{OPT} = \Omega(\log n) \sum_j p_j T_j^*$ *and* $\text{OPT} = \Omega(\log m) \sum_j p_j T_j^*$.

Proof. We use Theorem 2 from [13] as discussed above. Let G be a d-regular Ramanujan graph on n vertices with girth $g \geq \frac{2}{3} \log_{d-1} n$. The set of scenarios is the set of all vertex sets of walks of length $70g$. The probability p_j of scenario S_j is equal to the probability that S_j is the vertex set of a random walk of length $70g$. For a fixed first-stage tour, Theorem 2 in [13] states that in each of the first $g/2$ steps of the second-stage tour, there is a constant probability that the second-stage tour uses an edge of length $\Omega(\log n)$. This implies that the expected length of the first $g/2$ steps of the tour have expected length $\Omega(\log n)$. Since $T_j^* = O(g)$, the first $g/2$ steps are a constant fraction of all the steps and so the lower bound also holds for the entire tour. Hence, we have an instance such that $\text{OPT} = \Omega(\log n) \sum_j p_j T_j^*$. The number of scenarios is equal to the number of possible walks of length $70g$. This is equal to $n \cdot d^{70g} = O(nd^{\log n}) = O(n^{\log d + 1})$. Since d is a constant, this number is polynomially bounded. Hence, we have $\Theta(\log m) = \Theta(\log n)$, which gives us the second lower bound. □

Another natural question one can ask is whether a given instance has an optimal value that is equal to the master tour lower bound. Stated differently, is there a tour such that if we shortcut on the vertices of a scenario, we get the optimal solution for that scenario? Deineko et al. [8] studied this problem for the case where every possible subset is a scenario. They called this the master tour problem and showed that it is polynomially

solvable. We can reformulate the problem to the case where we are given a set of scenarios and we only have to be optimal for these scenarios. It turns out that this problem is Δ_2^p-complete [9].

3.2 SMALL SCENARIOS

We will continue by considering *a priori* TSP in the scenario model with further bounds on the scenarios. We start with looking at scenarios where the number of vertices per scenario is small. We begin by showing that *a priori* TSP is still NP-hard when all scenarios are very small. We reduce from the Max Cut problem [19]. Here, we are given a graph $G = (V, E)$ and our goal is to find a set $X \subseteq V$ such that $|\delta(X)|$ is maximized, where $\delta(X)$ is the set $\{(i, j) \in E : i \in X, j \notin X\}$.

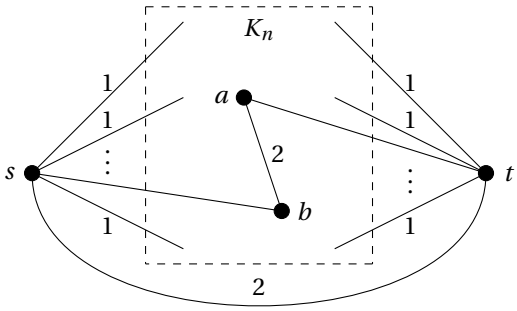


Figure 3.1: Graph G' as in the proof of Theorem 3.3.

Theorem 3.3. A priori *TSP is NP-hard even when* $|S_j| \leq 4$ *for all* j.

Proof. We reduce from the Max Cut problem. Given an instance $G = (V, E)$ of Max Cut, we create an instance of *a priori* TSP by making a complete graph G' on $V \cup \{s, t\}$. All edges with s or t as endpoint, except edge (s, t), have length 1 and all other edges have length 2 (see Figure 3.1). For every edge $(a, b) \in E$, we create a scenario $\{a, b, s, t\}$. All scenarios have equal probability. Note that the second-stage tour on a scenario either has a length of 4 or length 6. We say that a scenario is satisfied if its resulting tour has length 4. Hence, minimizing the expected length is equivalent to maximizing the number of satisfied scenarios. We will show that $\text{OPT}_{\text{TSP}} = 6|E| - 2\text{OPT}_{\text{CUT}}$, where OPT_{TSP} and OPT_{CUT} are the optimal sum of tour lengths of *a priori* TSP in the created instance and the optimal value of Max Cut in the original instance respectively.

Suppose there is a cut of size at least k in G, say $Q \subseteq V$. First, visit s. Then, visit the vertices of Q in arbitrary order. After that, we visit t. Finally, we visit the vertices not in Q in arbitrary order. It is easy to see that every scenario corresponding to an edge in the cut has length 4, whereas other scenarios have length 6. Hence, there is a tour satisfying at least k scenarios.

On the other hand, suppose that we have a tour in G' satisfying at least k scenarios. Without loss of generality, the tour can be written as (s, R_1, t, R_2), where R_1 and R_2 are sequences of vertices. The only way to satisfy a scenario $\{a, b, s, t\}$ is by putting one vertex of $\{a, b\}$ in R_1 and one vertex in R_2. Hence, the k satisfied scenarios correspond to edges in the cut $R_1 \subseteq V$ which has size at least k. □

By adjusting the proof of Theorem 3.3, we can prove that the master tour problem with scenarios is NP-complete when $|S_j| \leq 5$. This is done by reducing from Set Splitting instead of Max Cut and using that 3-Set Splitting is NP-complete [21]. This also shows that Min-Max TSP is NP-hard when $|S_j| \leq 5$ for all j. Moreover, when $|S_j| \leq 5$ for all j, we cannot approximate Min-Max TSP within a factor of $\frac{4}{3}$, unless P=NP. This is because a split set will correspond to a scenario with tour length 6, whereas an unsplit set corresponds to a scenario with tour length 8. The master tour problem with scenarios is still open for $|S_j| \leq 4$.

Note that the graph we used in the proof of Theorem 3.3 is obtained by taking the metric completion of $K_{2,n}$. This graph is planar, bipartite and it has treewidth and pathwidth equal to 2. Deterministic TSP would be polynomially solvable on such a graph with bounded treewidth. Furthermore, there is a PTAS for deterministic TSP in planar graphs [2]. The next theorem shows that this is not the case for *a priori* TSP (since the proof uses the same graph as before, a metric completion of $K_{2,n}$). This theorem relies on the fact that Max Cut, given the unique games conjecture (UGC), cannot be approximated by a factor above the Goemans-Williamson [12] constant, i.e. approximately 0.878567, unless P=NP [20]. Without this conjecture, Håstad [17] showed that it cannot be approximated above a factor $\frac{16}{17}$, unless P=NP.

Theorem 3.4. *There is no* 1.0117*-approximation for* a priori *TSP with* $|S_j| \leq 4$*, unless P=NP. Assuming UGC, there is no* 1.0242*-approximation, unless P=NP.*

Proof. Consider the reduction from the proof of Theorem 3.3. As a result, we have $\text{OPT}_{\text{TSP}} = 6|E| - 2\text{OPT}_{\text{CUT}}$. If we have an $(1+\alpha)$-approximation algorithm, we get a tour with total length at most $(1+\alpha)(6|E| - 2\text{OPT}_{\text{CUT}})$. This implies that there are at least η satisfied scenarios, where

$$\begin{aligned} 4\eta + 6(|E| - \eta) &= (1+\alpha)(6|E| - 2\text{OPT}_{\text{CUT}}) \\ -2\eta &= -2(1+\alpha)\text{OPT}_{\text{CUT}} + 6\alpha|E| \\ \eta &= (1+\alpha)\text{OPT}_{\text{CUT}} - 3\alpha|E|. \end{aligned}$$

These correspond to edges in the cut, hence we have

$$\begin{aligned} \text{Size of cut} &\geq (1+\alpha)\text{OPT}_{\text{CUT}} - 3\alpha|E| \\ &\geq (1+\alpha)\text{OPT}_{\text{CUT}} - 6\alpha\text{OPT}_{\text{CUT}} \\ &= (1-5\alpha)\text{OPT}_{\text{CUT}}, \end{aligned}$$

where the second inequality follows from $\text{OPT}_{\text{CUT}} \geq |E|/2$. Hence, assuming $\mathcal{P} \neq \mathcal{NP}$, there is no $(1+\alpha)$-approximation for $1 - 5\alpha \geq \frac{16}{17}$, i.e., there is no 1.0117-approximation.

If we also assume that the unique games conjecture holds, there is no $(1+\alpha)$-approximation for $1-5\alpha \geq 0.878567$, i.e., there is no 1.0242-approximation. □

Since graph G' in Figure 3.1 used in Theorem 3.4 is the metric completion of $K_{2,n}$, we get the following corollary.

Corollary 3.5. A priori *TSP in the scenario model on planar bipartite graphs does not admit a PTAS, unless P=NP.*

When $|S_j| \leq 6$, we can slightly strengthen the result of Theorem 3.4, by reducing from Max E4-Set Splitting, which cannot be approximated with a factor above $\frac{7}{8}$, unless P=NP [17]. This gives an inapproximability of 1.0265 when $|S_j| \leq 6$.

One could also consider the path-version of *a priori* TSP. In fact, the application on photolithography is modeled as the path-version. It is easy to see that this problem is trivial when $|S_j| \leq 2$ for all j. If we delete t from the graph created in the reduction of Theorem 3.3, we can use this graph and the same reduction to show that the path-version of *a priori* TSP is NP-hard when $|S_j| \leq 3$. It is easy to see that this graph can be obtained by taking the metric completion of the star graph. Note that we can also adjust Theorem 3.4 to the path-version which will give the same inapproximability result, i.e. there is no 1.0117-approximation, unless P=NP, and there is no 1.0242-approximation if we also assume that the UGC holds.

We can strengthen the inapproximability of *a priori* TSP by using strong results on Permutation Constraint Satisfaction Problems [15]. We will call the problem that we need 4-Undirected Cyclic Ordering (4-UCO). To the best of our knowledge, the problem has never been considered. In this problem, we are given a ground set U and a set of 4-tuples Δ^{UCO} using elements from U. Our goal is to construct an ordering on U that maximizes the number of satisfied 4-tuples. We say that 4-tuple (a,b,c,d) is satisfied if one of the following sequences is a subsequence of the total ordering: (a,b,c,d), (b,c,d,a) , (c,d,a,b), (d,a,b,c), (d,c,b,a), (c,b,a,d), (b,a,d,c), (a,d,c,b). In other words, we get a collection of cycles and we want to find an ordering maximizing the number of cycles that can be embedded in it. For completeness, we first show that deciding whether all 4-tuples can be satisfied is NP-complete by using a reduction from Cyclic Ordering. In this problem, we are given a set of ordered triples Δ^{CO} of ground set U. The question is whether there exists a cyclic ordering on all elements such that each triple is ordered in the right direction. This problem is NP-complete [11].

Theorem 3.6. *4-Undirected Cyclic Ordering is NP-hard.*

Proof. Given an instance of Cyclic Ordering, we create elements a_1 and a_2 for every element $a \in U$ and three additional elements, x, y and z. For every element $a \in U$ we create ordered sets (x,y,a_1,a_2), (x,z,a_1,a_2) and (y,z,a_1,a_2). For every triple in Δ^{CO}, we create one set by splitting an arbitrary element. For example, we create set (a_1,b_1,b_2,c_1) for triple (a,b,c).

If there exists a cyclic ordering, say $(a,b,\ldots,q)$, we can construct the following satisfying solution for 4-UCO: $(x,y,z,a_1,a_2,b_1,b_2,\ldots,q_1,q_2)$.

On the other hand, suppose that we have a satisfying solution for 4-UCO. Without loss of generality, we may assume that (x, y, a_1, a_2) is visited in this direction. We will show that x, y and z are visited consecutively. Suppose this is not the case and x, y and z are placed at different positions on the solution. This splits the solution into three segments. It is easy to see that for a any $u \in U$, we must have u_1 and u_2 in the same segment. Now, suppose that these elements are visited in the segment between x and y. This implies that the tour has to visit (x, u_2, u_1, y) in this order. However, this conflicts with scenario (y, z, u_1, u_2). Similarly, placing u_1 and u_2 between y and z implies visiting (y, u_2, u_1, z) in this order. This conflicts with scenario (x, y, u_1, u_2). Thus, we know that the solution visits x, y and z consecutively. We now fix the positions of u_1 for all $u \in U$ and we move u_2 to the position next to u_1. This does not conflict with any of the scenario's. The resulting arrangement of the u_1 vertices corresponds to an arrangement consistent with Δ^{CO}. □

In [15], it is shown that every Permutation CSP of constant arity is approximation resistant. This means that, under the unique games conjecture, the best we can do is constructing a random ordering. Classical problems like Cyclic Ordering and Betweenness are in this class of problems. It is easy to see that 4-UCO is also in this class. A corollary of the work of Guruswami et al. [15] is that for any $\epsilon > 0$ it is hard to distinguish between instances where at least a $(1-\epsilon)$ fraction of the 4-tuples can be satisfied from instances where at most a $(\frac{1}{3}+\epsilon)$ fraction of the 4-tuples can be satisfied, assuming the unique games conjecture is true. The natural generalization of 4-UCO is 5-UCO. For this problem, this result implies that there is no algorithm having a guarantee larger than $\frac{1}{12}$. This gives the following results.

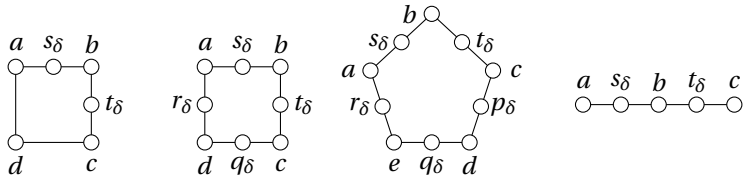


Figure 3.2: Gadgets used in proofs of Theorem 3.7 and 3.8.

Theorem 3.7. *Under UGC, there is no α-approximation for* a priori *TSP with*

(a) $\alpha < \frac{10}{9}$ *when* $|S_j| \leq 6$,

(b) $\alpha < \frac{4}{3}$ *when* $|S_j| \leq 8$,

(c) $\alpha < \frac{41}{30}$ *when* $|S_j| \leq 10$,

unless P=NP.

Proof. (a) Given an instance of 4-UCO, we create $|U| + 2|\Delta^{\text{UCO}}|$ vertices, one for each element of U and 2 for each 4-tuple in Δ^{UCO}. We create edges that correspond to 4-tuples in Δ^{UCO} in the following way. For 4-tuple $\delta = (a, b, c, d)$, we have vertices a, b, c, d and vertices s_δ and t_δ. We create edges $(a, s_\delta), (s_\delta, b), (b, t_\delta), (t_\delta, c), (c, d)$

and (d,a), as in Figure 3.2. The scenarios correspond to these six vertices for every tuple. Finally, the distances correspond to the shortest path distances in the created graph. A tuple is satisfied if and only if the tour restricted to the scenario has length 6. A solution satisfying $\frac{1}{3}$ of the scenarios has value at least $\frac{1}{3}\cdot 6+\frac{2}{3}\cdot 7=\frac{20}{3}$. A solution satisfying all scenarios has a value of 6. Since it is hard to distinguish between these two cases, we obtain an inapproximability of $\frac{20}{18}=\frac{10}{9}$ for *a priori* TSP with $|S_i|\leq 6$.

(b) We use a similar reduction. Instead of adding two vertices per tuple, we create four new vertices. In Figure 3.2, these vertices are called s_i, t_i, q_i and r_i. The scenarios will therefore have size 8. Again, a tuple is satisfied if and only if the tour restricted to the scenario has length 8. However, if we restrict the tour to a scenario corresponding to a non-satisfied tuple, it must have length at least 12. A similar calculation gives an inapproximability of $(\frac{1}{3}\cdot 8+\frac{2}{3}\cdot 12)/8=\frac{4}{3}$.

(c) We now reduce from 5-UCO. We add 5 dummy vertices for each scenario and place them between consecutive elements on the cycles. The scenarios will therefore have size 10. Again, a tuple is satisfied if and only if the tour restricted to the scenario has length 10. If we restrict the tour to a scenario corresponding to a non-satisfied tuple, it must have length at least 14. A similar calculation gives an inapproximability of $(\frac{1}{12}\cdot 10+\frac{11}{12}\cdot 14)/10=\frac{41}{30}$.

□

For the path-version, we can strengthen previous results by using hardness results for Betweenness. In this problem, we are given a set of triples Δ^{B} from elements of U. The triple (a,b,c) is satisfied if (a,b,c) or (c,b,a) is a subsequence of the total ordering. The goal is to find an ordering on U maximizing the number of satisfied triples. By [15], the best approximation ratio is $\frac{1}{3}$, assuming UGC. Without this conjecture, there is no approximation for Max Betweenness with a factor better than $\frac{1}{2}$, unless P=NP [3].

Theorem 3.8. *There is no $\frac{9}{8}$-approximation for* a priori *path-TSP with $|S_j|\leq 5$, unless P=NP. Assuming UGC, there is no $\frac{7}{6}$-approximation, unless P=NP.*

Proof. Given an instance of Betweenness, we create a graph with $|U|+2|\Delta^{\mathrm{B}}|$ vertices. A scenario contains the elements used in a triple and two extra vertices. The edges are drawn in the following way. For triple $\delta=(a,b,c)$, we add edges $(a,s_\delta),(s_\delta,b),(b,t_\delta)$ and (t_δ,c) (Figure 3.2). A triple is satisfied if and only if the path restricted to the scenario has length 4. Assuming UGC, we get that there is no approximation algorithm with guarantee smaller than $(\frac{1}{3}\cdot 4+\frac{2}{3}\cdot 5)/4=\frac{7}{6}$ for *a priori* path-TSP with $|S_j|\leq 5$, unless P=NP. Without assuming UGC, there is no approximation algorithm with guarantee smaller than $(\frac{1}{2}\cdot 4+\frac{1}{2}\cdot 5)/4=\frac{9}{8}$, unless P=NP. □

Finally, we note that by using twice the diameter of a scenario as a lower bound, we can show that taking an arbitrary tour as a solution is a $c/2$-approximation when $|S_j|\leq c$. A random tour gives a value of at most $(c^2-3c+4/2c-2)$ times the optimal value in

expectation. This factor approaches $c/2$ for c large. Similar results hold for the path-version.

3.3 BIG SCENARIOS

In this section, we investigate the special case of big scenarios, i.e. the case when each scenario has size at least $n-c$, for small c. One would expect that simply taking the optimal tour on the entire vertex set V would perform well on these instances. Here, we analyze this option. Let us denote $\text{OPT}(S)$ for the optimal value of a tour on $S \subseteq V$. Further, let $\text{OPT}(V)|_S$ denote the value of the optimal tour on V shortcutted to S. As before, let D_S denote the diameter of the graph restricted to S.

Lemma 3.9. *For $S \subset V$ and $1 \le c \le n/2$ such that $|S| = n-c$, we have*

$$\text{OPT}(V)|_S \le \text{OPT}(S) + cD_S.$$

Proof. Suppose $S = V \setminus \{a_1, \dots, a_c\}$. Let $\mathcal{D}_S^{a_i} = \min_{u \in S} d(u, a_i)$ for $i = 1, \dots, c$, where d denotes the edge length. Since we can extend our tour on S to V by going back and forth to each a_i, we have

$$\text{OPT}(V) \le \text{OPT}(S) + 2\sum_{i=1}^{c} \mathcal{D}_S^{a_i}. \tag{3.1}$$

Furthermore, suppose w.l.o.g. that b_i and d_i are the two vertices in S that are visited before and after a_i in the optimal tour of V. If two consecutive vertices on the tour are not in S, then one can reconstruct the tour accordingly without increasing the length of the tour restricted to S. This is true since vertices not in S do not influence the tour restricted to S and since $c \le n/2$. Hence, we can assume that there are no two consecutive vertices on the tour that are not in S. Then

$$\begin{aligned}\text{OPT}(V) &= \text{OPT}(V)|_S + \sum_{i=1}^{c} (d(b_i, a_i) + d(a_i, d_i) - d(b_i, d_i)) \\ &\ge \text{OPT}(V)|_S + \sum_{i=1}^{c} (2\mathcal{D}_S^{a_i} - d(b_i, d_i)) \ge \text{OPT}(V)|_S - cD_S + 2\sum_{i=1}^{c} \mathcal{D}_S^{a_i}. \end{aligned} \tag{3.2}$$

Combining Equations (3.1) and (3.2) we get

$$\text{OPT}(V)|_S \le \text{OPT}(V) + cD_S - 2\sum_{i=1}^{c} \mathcal{D}_S^{a_i} \le \text{OPT}(S) + cD_S.$$

□

The inequality is tight for the graph in Figure 3.3 with $c = 2$. We can generalize this tight instance for $c \le n/2$ by adding more diagonal paths.

Theorem 3.10. *The optimal solution on V is a $(1 + \frac{c}{2})$-approximation for* a priori *TSP with $|S_i| \ge n - c$, where $1 \le c \le \frac{n}{2}$.*

Obviously, these results extend to Min-Max TSP.

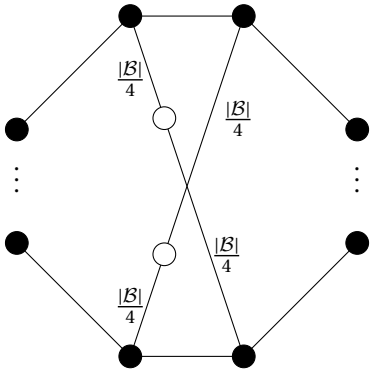


Figure 3.3: Instance for which inequality of Lemma 3.9 is asymptotically tight for $c = 2$, where $\mathcal{B}$ is the set of black (non-white) vertices.

3.4 NESTED SCENARIOS

Let us now consider the case of nested scenarios, i.e. $S_1 \subseteq S_2 \subseteq \ldots \subseteq S_m$. Here, the following algorithm gives a constant-factor approximation. First, compute an 1.5-approximate tour T_j for scenario S_j for all j. Let $\alpha_1 = 1$. Next, for $h = 2,3,\ldots$ let α_h be the largest number $k > \alpha_{h-1}$ for which $T_k \leq 2T_{\alpha_{h-1}}$. If no such k exists then let $\alpha_h = \alpha_{h-1} + 1$. The first-stage tour is obtained by visiting vertices in the order $T_{\alpha_1}, T_{\alpha_2}, \ldots$.

Theorem 3.11. *The algorithm above is a 9-approximation for nested scenarios.*

Proof. Consider scenario S_j. The last vertices of this scenario will be visited on the tour T_{α_h}, where h is the smallest index such that $\alpha_h \geq j$. Note that for any $h \geq 2$, we have $T_{\alpha_h} > 2T_{\alpha_{h-2}}$. Hence, we can decompose the concatenated tour up to T_{α_h} into two parts which correspond to even and odd h respectively, such that both parts have geometrically increasing tour lengths. The length of the concatenated tour up to T_{α_h} is therefore at most

$$2T_{\alpha_{h-1}} + 2T_{\alpha_h}.$$

If $\alpha_h = j$, then the length of the tour is at most $2T_{\alpha_{h-1}} + 2T_{\alpha_h} \leq 4T_{\alpha_h} = 4T_j \leq 6T_j^*$.

If $\alpha_h > j$, then we must have $T_{\alpha_h} \leq 2T_{\alpha_{h-1}}$, so the length of the tour is at most $2T_{\alpha_{h-1}} + 2T_{\alpha_h} \leq 6T_{\alpha_{h-1}} \leq 9T^*_{\alpha_{h-1}} \leq 9T_j^*$. □

Finding a constant-factor approximation is still open for laminar scenarios, i.e., when for each i, j, either $S_i \cap S_j = \emptyset$ or $S_i \subseteq S_j$ or $S_j \subseteq S_i$. It is even open in the case when the scenarios have the following star-like structure.

$$S_i \cap S_j = \emptyset \text{ for } i \neq j, i, j = 1, \ldots, m-1, \text{ and } S_m = \bigcup_{j=1}^{m-1} S_j. \tag{3.3}$$

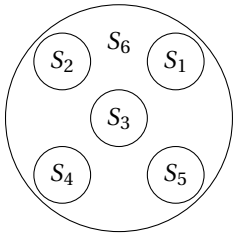


Figure 3.4: Star-like instance with 6 scenarios.

It would be interesting to know if one could get a constant-factor approximation for these instances. Finally, observe that the Min-Max TSP for laminar scenarios reduces to standard TSP since the largest scenario determines the value of the solutions.

3.5 RELATION WITH MINIMUM SPANNING TREE PROBLEMS

It would be nice to have a similar relation between *a priori* TSP and *a priori* MST as in the deterministic setting between TSP and MST. We consider two versions of *a priori* MST. The first one is defined by Bertsimas [4], who called it *a priori* MST, although it seems more natural to call it *a priori* Steiner Tree. The second problem is defined by Boria et al. [5], who called it Probabilistic MST under Closest Ancestor (PMST-CA). In both problems, we have a graph $G = (V, E)$ and a probability distribution over subsets of vertices. The second problem also has a root r that is always active. This is optional in the first problem. The goal is to construct a tree on the entire vertex set in the first stage. A subset S of the vertices, drawn according to the probability distribution, is revealed in the second stage. In the *a priori* MST, the second-stage tree will be obtained by deleting inactive vertices, provided that the remaining tree stays connected. In the PMST-CA, the second-stage tree only contains active vertices. This is done by taking an edge between an active vertex and its closest active ancestor in the rooted first-stage tree. In both problems, the goal is to construct a first-stage tour that minimizes the expected length of the second-stage tree.

Unfortunately, it turns out that the expected length of the optimal *a priori* MST defined by Bertsimas is not smaller than the optimal *a priori* TSP in general. The gap between the optimal values of *a priori* MST and *a priori* TSP can be arbitrarily large.

Theorem 3.12. *There are instances such that the optimal value of the* a priori MST*-solution is arbitrarily larger than the optimal value of the* a priori *TSP-solution.*

Proof. Take a 3-regular graph with girth g. Sachs [24] showed that these graphs exist. Define a scenario for each edge by the endpoints of the edge. All scenarios have the same probability. Any tour on this graph will be shortcut to a tour of length 2 for each scenario, so the objective value of *a priori* TSP is 2. Consider the optimal *a priori* MST. Since this is a tree, it uses $n-1$ edges. If an edge is in the tree, the corresponding scenario gets value 1. If an edge is not in the tree, the corresponding scenario gets value at least

$g-1$. Since there are $3n/2$ edges (and scenarios), we get at least the following objective value.

$$\left(\frac{3n/2-(n-1)}{3n/2}\right)(g-1)+\frac{n-1}{3n/2}=\frac{g+1}{3}+\frac{2g-4}{3n}\geq\frac{g+1}{3}.$$

Now, we can take g arbitrarily large, which makes the objective value arbitrarily large and hence the gap with the objective value of *a priori* TSP. □

Unlike the *a priori* MST, the PMST-CA can be used as a lower bound for *a priori* TSP. In fact, we only lose a factor 2. Note that this only works for the rooted case, since PMST-CA is defined with a root vertex.

Theorem 3.13. *If there is an α-approximation for the PMST-CA, then there is a 2α-approximation for the* a priori *TSP, and vice versa.*

Proof. First, we show that the following inequalities are valid, where OPT_{MST} and OPT_{TSP} denote the optimal values of PMST-CA and *a priori* TSP respectively.

$$\text{OPT}_{\text{MST}}\leq\text{OPT}_{\text{TSP}}\leq 2\text{OPT}_{\text{MST}}.$$

The first inequality can be proven by taking the optimal *a priori* TSP-tour and deleting one edge. This gives a spanning tree on V, called T. If we look at a specific active set S, then the optimal *a priori* TSP-tour restricted to S will have exactly one edge less than before. Namely, if we delete edge (a,b) from tour $(1,\ldots,a,b,\ldots,n)$, only edge $(\max\{k\in S:k\leq a\},\min\{k\in S:k\geq b\})$ will disappear from the restricted tour on S. Note that for active set S, the tour without this edge is the same as T shortcutted to S. Hence, this is a feasible solution for PMST-CA with cost no larger than the optimal value of *a priori* TSP, and the first inequality has been proven.

The second inequality is proven by doubling the optimal tree and shortcutting the obtained Eulerian tour. In each scenario, the cost of the edges is at most twice the cost of the edges in the tree restricted to the scenario.

Now, if there is an α-approximation for PMST-CA, we double the tree and shortcut the Eulerian tour to obtain a tour on V. This tour has a value of at most

$$2\alpha\text{OPT}_{\text{MST}}\leq 2\alpha\text{OPT}_{\text{TSP}}.$$

Given an α-approximation for *a priori* TSP, we take the tour and delete one edge. The resulting tree has a value of at most

$$\alpha\text{OPT}_{\text{TSP}}\leq 2\alpha\text{OPT}_{\text{MST}}.$$

□

Recall that there is a randomized 4-approximation for *a priori* TSP in the independent decision model [26]. There is also a deterministic 6.5-approximation [28] for this problem. Using Theorem 3.13, we obtain the following corollary.

Corollary 3.14. *There is a randomized* 8*-approximation and a deterministic* 13*-approximation for PMST-CA in the independent decision model. There is also a* $O(\log n)$*-approximation in the black-box model.*

Unfortunately, Theorem 3.13 does not imply a 2-approximation for *a priori* TSP, since we can prove that PMST-CA is NP-hard in the scenario model. For this, we need the following lemma. This lemma holds for both the scenario and the independent decision model.

Lemma 3.15. *If PMST-CA is NP-hard in the non-metric case, then it is NP-hard in the metric case.*

Proof. One can turn a graph into a graph satisfying the triangle inequality by adding a sufficiently large number M to all distances. In the PMST-CA, this affects every solution by an additive constant equal to $\sum_S p(S)(|S|-1)M$, where $p(S)$ is the probability that set S is the active set. Hence, the complexity of the problem is preserved in the metric case. □

Boria et al. [5] showed that PMST-CA is NP-hard in the independent decision model, but only for the non-metric case. Using Lemma 3.15, we obtain the following corollary.

Corollary 3.16. *PMST-CA is NP-hard in the independent decision model, even if the triangle inequality is satisfied.*

Theorem 3.17. *PMST-CA in the scenario model is NP-hard.*

Proof. We reduce from the NP-complete problem Exact Cover by 3-Sets [19]. In this problem, we are given $3q$ elements, $X = \{x_1, \ldots, x_{3q}\}$, and m subsets, $Y = \{y_1, \ldots, y_m\}$, with $y_i \subseteq X$ and $|y_i| = 3$ for all i. The problem asks whether there are q sets that together cover all elements. Create the graph as in Figure 3.5. There are m scenarios with probability $1/m$. Define $S_i = X \cup \{r, s, y_i\}$.

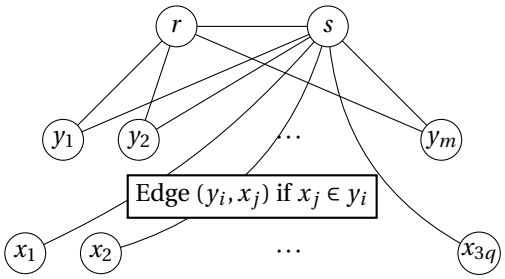


Figure 3.5: Graph used in proof of Theorem 3.17. Edges (r, s) *and* (r, y_i) *have length 0. Edges* (s, y_i) *and* (y_i, x_j) *have length 1. Edges* (s, x_j) *have length 2. All other edges have length* M*, where* M *is a large number.*

If there is an exact cover, then construct the following solution. If set y_i is chosen in the cover, then use edge (s, y_i) and the edges from vertex y_i to the corresponding elements of y_i. If set y_i is not in the cover, then use edge (r, y_i). Finally, use edge (r, s). For any y_i in the cover, consider the subtree containing s, y_i and the x_j's corresponding to elements from subset y_i. In scenario S_i, the resulting subtree has value 4. In all other scenarios, vertex y_i will not be present and this subtree will contain three edges from s to the vertices of the elements. Hence, this solution has expected value equal to $q(1/m \cdot 4 + (m-1)/m \cdot 6) = q(6 - 2/m)$.

Note that an optimal tree will never use edges with weight M or a combination of edges that enforce using an edge of weight M in the shortcut solution. This leaves five ways of connecting a specific set vertex y_i and element vertex x_j, where j is in set i, to r and s. The five subtrees are depicted in Figure 3.6.

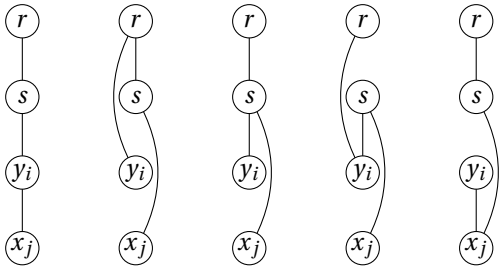


Figure 3.6: Form left to right, subtrees T_1 up to T_5.

Tree T_3 is dominated by T_1, since T_1 only has cost 2 for connecting x_j when y_i is inactive while T_3 always has cost 2. Similarly, T_4 is dominated by T_2 and T_5 is dominated by T_1. So, an optimal tree is a combination of T_1 and T_2. Suppose that the tree connects k set vertices to s which connect ℓ elements vertices. The other set vertices are connected to r whereas the other element vertices are connected to s. Number the k set vertices connected to s as $1, \ldots, k$ and say that set vertex i connects ℓ_i element vertices. This tree has an expected value of

$$\frac{1}{m}\sum_{i=1}^{k}((\ell_i + 1) + 2(3q - \ell_i)) + \frac{m-k}{m}6q = 6q + \frac{1}{m}(k - \ell),$$

which is equal to $q(6 - 2/m)$ if and only if $k = q$ and $\ell = 3q$. Hence, there is a tree with expected value at most $q(6 - 2/m)$ if and only if there is an exact cover. Using Lemma 3.15 completes the proof. □

3.6 CONCLUSION

In this chapter, we looked at *a priori* TSP in the scenario model, since the problem of minimizing the blading in photo-lithography machines is a direct practical application of it. We showed how to get constant-factor approximation algorithms for instances of

the *a priori* TSP with small, big or nested scenarios. An interesting question that remains unanswered is whether there exists a constant-factor approximation for *a priori* TSP with laminar scenarios. More specifically, it is still open whether we can do this on star-like scenarios as defined in Equation (3.3). Next to restricted scenarios we also considered restricted metrics. In Section 3.2 we showed that there is no PTAS for planar bipartite graphs. We do not have such results in the Euclidean plane. It would be interesting to settle the approximability of the problem in this metric. It is easy to construct examples where the optimal solution crosses itself and hence the non-crossing property does not hold. This property was a crucial ingredient of the PTAS by Arora [1] for the deterministic problem. So far, we have not been able to show any lower bound or improve the upper bound for this special case.

We did not succeed in improving the $O(\log n)$-approximation for the general problem. In fact, we conjecture that there is no $o(\log n)$-approximation algorithm for *a priori* TSP in the scenario model in the general case. Hence, we expect any given polynomial approximation algorithm for the blading problem to have at best an $O(\log n)$-approximation guarantee. The problem instances produced by the problem of minimizing the blading in photo-lithography machines have no restrictions on the size and amount of scenarios. Thus, the best approximation guarantee we can have for this problem is $O(\log n)$. Algorithms do not need to use information from the scenarios to achieve such a guarantee, such as the randomized algorithm by Schalekamp and Shmoys [25] or, as we will see in Chapter 4, a spacefilling curve heuristic such as the one by Platzman and Bartholdi [22].

REFERENCES

[1] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[2] S. Arora, M. Grigni, D. R. Karger, P. N. Klein, and A. Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 33–41. ACM/SIAM, 1998.

[3] P. Austrin, R. Manokaran, and C. Wenner. On the NP-hardness of approximating ordering-constraint satisfaction problems. *Theory of Computation*, 11:257–283, 2015.

[4] D. Bertsimas. *Probabilistic combinatorial optimization problems*. PhD thesis, Massachusetts Institute of Technology, 1988.

[5] N. Boria, C. Murat, and V. Paschos. On the probabilistic min spanning tree problem. *Journal of Mathematical Modelling and Algorithms*, 11(1):45–76, 2012.

[6] L. Chen, N. Megow, R. Rischke, and L. Stougie. Stochastic and robust scheduling in the cloud. In N. Garg, K. Jansen, A. Rao, and J. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015*, volume 40 of *LIPIcs*, pages 175–186. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[7] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976.

[8] V. G. Deineko, R. Rudolf, and G. J. Woeginger. Sometimes travelling is easy: The master tour problem. *SIAM Journal of Discrete Mathematics*, 11(1):81–93, 1998. doi: 10.1137/S0895480195281878. URL `http://dx.doi.org/10.1137/S0895480195281878`.

[9] M. v. Ee and R. Sitters. On the complexity of master problems. In G. Italiano, G. Pighizzini, and D. Sannella, editors, *Proceedings of the 40th Symposium on Mathematical Foundations of Computer Science*, volume 9235 of *Lecture Notes in Computer Science*, pages 567–576. Springer, 2015.

[10] E. Feuerstein, A. Marchetti-Spaccamela, F. Schalekamp, R. Sitters, S. v. d. Ster, L. Stougie, and A. v. Zuylen. Scheduling over scenarios on two machines. In Z. Cai, A. Zelikovsky, and A. Bourgeois, editors, *Proceedings of the 20th International Conference on Computing and Combinatorics*, volume 8597 of *Lecture Notes in Computer Science*, pages 559–571. Springer, 2014.

[11] Z. Galil and N. Megiddo. Cyclic ordering is NP-complete. *Theoretical Computer Science*, 5(2):179–182, 1977.

[12] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[13] I. Gorodezky, R. D. Kleinberg, D. B. Shmoys, and G. Spencer. Improved lower bounds for the universal and *a priori* TSP. In M. Serna, R. Shaltiel, K. Jansen, and J. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010*, volume 6302 of *Lecture Notes in Computer Science*, pages 178–191. Springer, 2010.

[14] A. Gupta, M. T. Hajiaghayi, and H. Räcke. Oblivious network design. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 970–979. ACM, 2006. URL `http://dl.acm.org/citation.cfm?id=1109557.1109665`.

[15] V. Guruswami, J. Håstad, R. Manokaran, P. Raghavendra, and M. Charikar. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM Journal on Computing*, 40(3):878–914, 2011.

[16] M. T. Hajiaghayi, R. Kleinberg, and T. Leighton. Improved lower and upper bounds for universal TSP in planar metrics. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 649–658. ACM, 2006.

[17] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.

[18] N. Immorlica, D. R. Karger, M. Minkoff, and V. S. Mirrokni. On the costs and benefits of procrastination: Approximation algorithms for stochastic combinatorial optimization problems. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 691–700. SIAM, 2004.

[19] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[20] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007.

[21] L. Lovász. Coverings and colorings of hypergraphs. In *Proceedings of the 4th Southeastern Conference on Combinatorics, Graph Theory and Computing*, pages 3–12, 1973.

[22] L. K. Platzman and J. J. Bartholdi III. Spacefilling curves and the planar travelling salesman problem. *Journal of the ACM (JACM)*, 36(4):719–737, 1989.

[23] R. Ravi and A. Sinha. Hedging uncertainty: Approximation algorithms for stochastic optimization. *Mathematical Programming*, 108(1):97–114, 2006.

[24] H. Sachs. Regular graphs with given girth and restricted circuits. *Journal of the London Mathematical Society*, 1(1):423–429, 1963.

[25] F. Schalekamp and D. B. Shmoys. Algorithms for the universal and *a priori* TSP. *Operations Research Letters*, 36(1):1–3, 2008.

[26] D. B. Shmoys and K. Talwar. A constant approximation algorithm for the *a priori* traveling salesman problem. In A. Lodi, A. Panconesi, and G. Rinaldi, editors, *Proceedings of the 13th International Conference on Integer Programming and Combinatorial Optimization*, volume 5035 of *Lecture Notes in Computer Science*, pages 331–343. Springer, 2008.

[27] M. vanEe, L. vanIersel, T. Janssen, and R. Sitters. A priori tsp in the scenario model. *Discrete Applied Mathematics*, 2018.

[28] A. v. Zuylen. Deterministic sampling algorithms for network design. *Algorithmica*, 60(1):110–151, 2011.

4

THE SPACEFILLING CURVE ALGORITHM

In Chapter 2, we came across the problem of minimizing the distance blades have to travel during the expose step in the photolithography processes and its direct relation to *a priori* TSP. This process is used in semiconductor manufacturing to transfer the geometric pattern of a chip onto a wafer. Since the expose step often influences the total processing time of a wafer in the photolithography machine, minimizing the blade movements reduces the processing time. The blade movements are determined by the order of the reticle images. We showed that this can be formulated in such a way that it is precisely a form of the *a priori* TSP in the scenario model. In Chapter 3, we looked at properties of *a priori* TSP and we conjectured that there is no $o(\log(n))$-approximation algorithm for the problem.
An $O(\log(n))$-approximation algorithm is given by Schalekamp and Shmoys [9] for the black-box model of *a priori* TSP. This algorithm does not use any information of the probability distribution and is constructed by (probabilistically) embedding the metric in tree metrics.

Another $O(\log(n))$-approximation algorithm which would work for *a priori* TSP in the plane is the spacefilling curve heuristic proposed by Platzman and Bartholdi [8]. The idea behind the algorithm is fairly simple: construct a space-filling curve $\phi : C \to S$, which is a continuous mapping from the unit interval $C = [0,1]$ onto the unit square $S = [0,1]^2$. After constructing the curve, we visit the points in ascending order in which they occur on the curve. An example is shown in Figure 4.1.

In this chapter, we will look at the spacefilling curve algorithm and its application to the problem of minimizing the distance blades have to travel in the photolithography processes. We begin by formally describing the algorithm. Since in our application the distances of the TSP are based on 4-dimensional points, we will extend the algorithm of

This chapter contains joined work with Céline Swennenhuis

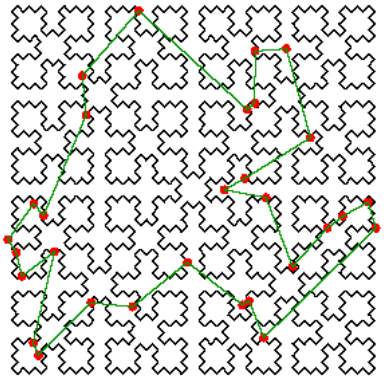


Figure 4.1: Example of the spacefilling curve heuristic for a planar TSP instance with 30 cities.

Bartholdi and Platzman to 4-dimensional points. Since the Sierpiński curve used cannot (easily) be extended to four dimensions, we will consider the Hilbert curve. Lastly, we will see the algorithm performs very well in practice on the real life *a priori* TSP instances found when minimizing the blade movement in the photolithography machines.

4.1 THE SPACEFILLING CURVE HEURISTIC FOR PLANAR TSP

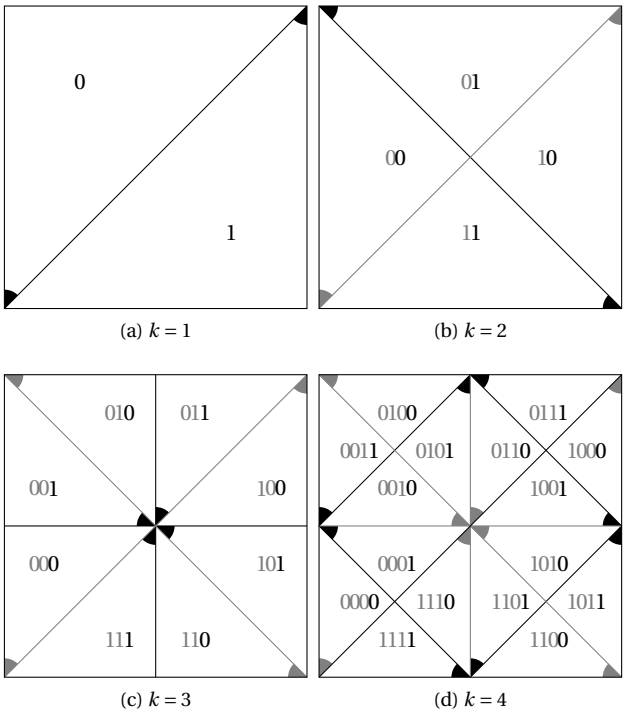
In planar TSP we are given a set N of points in the Euclidean plane and we want to find a tour visiting all points exactly once that minimizes the distance travelled. Distance between points are given by the Euclidean distance between two points in the plane (i.e. if $i = (x_i, y_i)$, $j = (x_j, y_j) \in N$, the distance $d_{ij} = \|i - j\| = \sqrt{((x_i - x_j)^2 + (y_i + y_j)^2)}$). In this section, we will discuss the spacefilling curve algorithm given by Platzman and Bartholdi [8].

As mentioned earlier, the algorithm is based in a spacefilling curve $\phi : C \to S$, where $C = [0,1]$ and $S = [0,1]^2$. This curve is a continuous mapping from C onto S. The mapping is surjective but it does not have to be injective. Hence, there might be $t, t' \in C$ such that $\phi(t) = \phi(t')$. We find the order in which the points are visited by doing the following two steps.

1. For each point $i \in S$ to be visited, compute a $t \in C$ such that $i = \phi(t)$.
2. Sort the points in ascending order of their corresponding t and construct a tour visiting the points in this order.

The spacefilling curve used is the Sierpiński curve. This curve is defined on the triangle. A tour on the unit square is constructed by joining the two curves at their begin- and endpoints. The Sierpiński curve on the triangle is constructed by successively dividing the triangle into smaller triangles and marking the corners of the triangle. We label the triangles with binary numbers. In every partition, we number the triangle which contains the original marked vertex with an additional right hand 0 and the triangle without a marked vertex with a 1. We mark in this triangle the vertex that was divided by the par-

tition. Thus, the kth partition of S consists of $2k$ identical triangles, each labeled with the binary representation of an integer $0,\ldots,2k$. Figure 4.2 shows the first four partitions.



(a) $k = 1$ (b) $k = 2$

(c) $k = 3$ (d) $k = 4$

Figure 4.2: First four partitions of S in the spacefilling curve construction for the Sierpiński

This construction of te space-filling curve ϕ maps each subinterval of $C = [0,1]$ onto the subtriangle of S with the corresponding label. The starting point of the subinterval will be mapped to the marked vertex of the triangle. The Sierpiński curve ϕ is not one-to-one. This can easily be seen by looking at $(\frac{1}{2},\frac{1}{2})$, for which it holds that $\phi(\frac{1}{8}) = \phi(\frac{3}{8}) = \phi(\frac{5}{8}) = \phi(\frac{7}{8}) = (\frac{1}{2},\frac{1}{2})$. There are eight possible encodings for this point, namely, 001, 011, 101, 111 and 000111…, 010111…, 100111… and 110111….
Although ϕ is not one-to-one, we still define the mapping $\lambda : S \to C$ and only require that for every $p \in S$, $\phi(\lambda(p)) = p$. The non-uniqueness of λ turns out not to have any theoretical or practical implications.
In the first step of the algorithm, we need to compute a $t \in C$ such that $i = \phi(t)$ for each point $i \in S$. For the Sierpiński curve, this boils down to find a division in subtriangles such that each subtriangle contains at most one point. Hence, we can stop partitioning when this occurs and construct a tour using the binary encoding of the triangles.

The main problem with this approach for our problem is that a Sierpiński curve is not easily extended to three dimensions, let alone to four. The natural extension for a triangle to a three dimensional object is the tetrahedron and six tetrahedrons could cover the unit

cube. We could even mark one edge of every tetrahedron such that they form a tour. The difficulty lies in how to split a single tetrahedron. Splitting the tetrahedron in half on the marked edge, just like in the two dimensional case, will give two congruent tetrahedrons. These tetrahedrons will however be heavily distorted and the ratio between different edges of the tetrahedron will become unbounded as splitting continues as shown by Bader [2].

Next to the Sierpiński curve, Bartholdi III and Platzman [3] also propose two different curves to tackle TSP in the Euclidian space; the Hilbert curve and the 'zig-zag'-curve. For these curves, we do not have an approximation ratio, although Bertsimas and Grigni [4] show an example were the heuristic tour is $O(\log(n))$ from the optimal tour. We will continue by considering the Hilbert curve as a basis for our heuristic, since it is easy construct in four dimensions and it is easy to check the position of points on the curve.

4.2 HILBERT CURVE

The Hilbert Curve was first described by David Hilbert [6]. In this section, we will describe the four dimensional curve and adjust the curve to form a tour. The Hilbert curve describes a continuous mapping from the unit line to the unit square. It is constructed by iteratively dividing every square in four smaller squares. Every square contains a $\sqcap$-shaped part of the curve and these are rotated and connected through the edges of each square to form a continuous line. We call the starting curve the first level of the curve or basis curve and the curve after the first iteration, the second level curve, continuing in this way for every iteration. Figure 4.3 shows the first 3 levels of the curve.

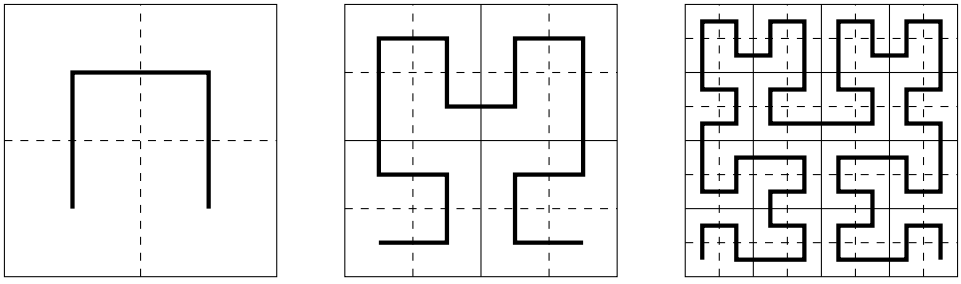


Figure 4.3: First 3 levels of the Hilbert curve in 2D

Because we are solving *a prior* TSP in the scenario model, we want a tour instead of a path. We therefore rotate the first and fourth square of the tour in the second level and connect their endpoints. The rotation of the first and the fourth square is taken into account, when construction the tour for higher iterations. This does however not change the substructure of these levels, only its orientation. See Figure 4.4 for the first 3 levels of this curve. Note that we could have a slightly different curve by rotating the square 90 degrees.

The three dimensional curve is less straight forward than the two dimensional one. According to Bader [2] there are three characteristics that we want to preserve:

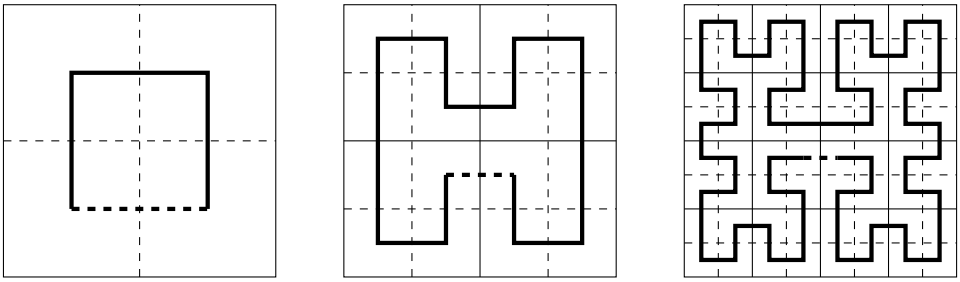


Figure 4.4: Adjusted Hilbert Curve in 2D, so it forms a circuit

- It should fill the unit cube.
- We want to be able to construct it recursively and it should be based on recursive substructuring of cubes into subcubes with halve the side length of the original cube.
- It should be face-connected, so that we can form a curve.

The basis (i.e. the first level) of the two dimensional Hilbert curve is the cup. Given start- and endpoints of the curve in the two dimensional case, there is only one curve possible that visits all squares of the next iteration. In the three dimensional case, there are more options for the start- and endpoints and given these there are multiple options to construct a tour. A Hilbert curve can have three different basis forms of the curve in three dimensions, which can be rotated to form even more basis curves. Figure 4.5 shows these. Similarly, while the two dimensional Hilbert curve has only a single substructure, there are multiple substructures in the three dimensional case.

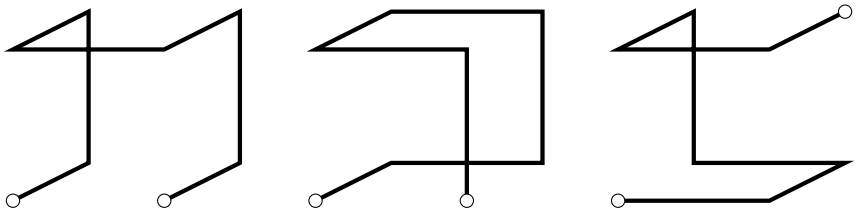


Figure 4.5: The 3 different basis forms for the 3D Hilbert curve.

Again, we can form a tour by rotating two cubes at the second iteration. We can therefore only use the first two basis forms of Figure 4.5. We can rotate each of these forms resulting in 6 different basis curves per basis form and each of these can be used for the 3-dimensional circuit.

The four dimensional case is very similar to the three dimensional case, only with more basic form ans more rotations. We therefore fix the basic form we use to construct the curve. This form is shown in Figure 4.6. We can rotate this basis curve. In total, this results in 24 different starting curves. In our algorithm, we will consider all these starting structures. There are even more substructures, but we will only use one substructure and its rotations, namely one similar to our basis curve shown in Figure 4.6. To form

a tour we rotate the first and the sixteenth hypercube in our second iteration to form a tour instead of a path.

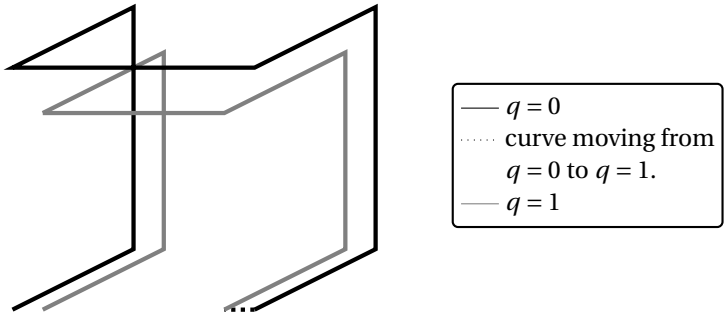


Figure 4.6: An impression of a four-dimensional Hilbert curve.

4.3 THE SPACEFILLING CURVE HEURISTIC FOR 4D *a priori* TSP

In the 4-dimensional *a priori* TSP in the scenario model, we are given a set of n points V in the metric space $S = [0,1]^4$ and a set of scenarios $\mathcal{S}$ with $S_1, \ldots, S_m \subseteq V$. Each scenario S_j has probability p_j of being the active set and $\sum_j p_j = 1$. We begin by finding an ordering on the points in V, called the first-stage tour. When an active set is released, the second-stage tour is obtained by shortcutting the first-stage tour on the vertices of the active set. The goal is to find a first-stage tour that minimizes the expected length of the second-stage tour.
To tackle the 4-dimensional *a priori* TSP in the scenario model, we will use the same approach as found in Section 4.1, but now we will use the Hilbert curve $\mathcal{H} : C \to S$, where $C = [0,1]$ and $S = [0,1]^4$ instead of the Sierpiński curve. Hence, we construct a tour in the following way:

1. For each point $i \in V$ to be visited, compute a $t \in C$ such that $i = \mathcal{H}(t)$.
2. Sort the points in ascending order of their corresponding t and construct a tour visiting the points in this order.

To calculate t for points $i \in V$, we use the same method as proposed by Jin and Mellor-Crummey [7]: The Hilbert curve is a self-similar space filling curve that can be constructed recursively. Therefore, if we know all basis curves that occur in our curve and use these as first level curves to construct a second level curve, we can find a table with the relation between the parent hypercube and its subhypercubes. This relation is the same at every level of the curve. We collected these relations in a 'movement specification table'. This table can be used to iteratively construct the third and higher levels of the curve. To construct the first and second level curves, we use the algorithm from Butz [5].

Since we restricted ourselves to the hypercube found in Figure 4.6, we can identify each curve in a (sub)hypercube by their movement in the first, second, fourth and eighth di-

rection of the curve. Let x, y, z and q denote the directions of the axes and let x^+ denote an increase and x^- denote a decrease in the x-direction and similarly for y, z and q. We can then identify the basis curve in Figure 4.6 as $x^+z^+y^+q^+$. We then use the algorithm by Butz to find the 16 subhypercubes of $x^+z^+y^+q^+$ and the movement specification table. The 'movement specification table' is given in Table 4.1.

Subhypercube	Orientation
1	z^+, y^+, q^+, x^+
2	y^+, q^+, x^+, z^+
3	y^+, q^+, x^+, z^+
4	q^+, x^-, z^-, y^+
5	q^+, x^-, z^-, y^+
6	y^-, q^+, x^+, z^-
7	y^-, q^+, x^+, z^-
8	x^-, z^+, y^-, q^+
9	x^-, z^+, y^-, q^+
10	y^-, q^-, x^+, z^+
11	y^-, q^-, x^+, z^+
12	q^-, x^-, z^-, y^-
13	q^-, x^-, z^-, y^-
14	y^+, q^-, x^+, z^-
15	y^+, q^-, x^+, z^-
16	z^+, y^+, q^-, x^-

Table 4.1: Movement specification table for x^+, z^+, y^+, q^+

By mapping each subhypercube back to x^+, z^+, y^+, q^+, we can use the same table for every iteration and construct a Hilbert curve up until every given level j. To construct a tour we rotate the first and sixteenth hypercube at the second level. Note that we do not change the movement specification table.

Thus, using the above method, we can identify for every point $i \in V$ in which (sub)hypercube it is contained up until a given maximum level j. Let C_i^l denote the hypercube in which hypercube $(1,\ldots,16)$ point i is contained in level l. Our algorithm then works as follows:

1. For each point $i \in V$ compute in which of the 16 hypercubes C_i^l it is contained in levels $l = 1,\ldots,j$.
2. Set $t_i = \sum_{l=1}^{j} (C_i^l - 1)/(16)^l$.
3. Sort the points $i \in V$ in ascending order of their corresponding t_i and construct a tour visiting the points in this order.

The algorithm takes $\mathcal{O}(n\log(n))$ time. Given a j, the maximum deviation is smaller than $\mathcal{O}(n \cdot 2^{-j})$. One could choose j, based on the minimum distance between the points. Given this distance, j can be chosen such that no two points share a subhypercube at

level j (unless they have the same coordinates). Finding this distance takes $\mathcal{O}(n\log(n))$ time.
For our application, we need a path instead of a tour. Therefore, when minimizing the blading, we add a fourth step to the algorithm. Let $d(x,y)$ denote the distance measure of the metric space S for $x,y \in S$.

4 Find the maximum $d(u,v)$ with $u,v \in V$ where u is visited right before v on the constructed tour and is maximal. Construct a path by starting at v and ending at u on the constructed tour.

In the above algorithm, we use a single level one curve, namely x^+, z^+, y^+, q^+. We will therefore call this algorithm `Hilbert1`. When first using this algorithm, we concluded that it was very fast, but also that the choice of the level one curve has a big influence on the quality of the solution. Because of this dependence, we also constructed an algorithm where we construct a tour (or path) using the above algorithm with 24 different level one curves. From the 24 tours found, we take the best solution. The 24 level one curves are obtained by rotating the x^+, z^+, y^+, q^+ curve. We will call this algorithm `Hilbert24`.

Note that in the description of the algorithm, we never used that the set of scenarios with associated probabilities are given. The algorithm does not use the probability distribution, hence it actually works for any kind of distribution on the points in the 4-dimensional space.

4.4 APPLICATION AND RESULTS

The algorithm was tested on the real-world data from the semiconductor manufacturer to minimized total blade movement. We evaluate its performance against the original ordering of the images and the ordering found by ILP solver SCIP[1] used in Section 2.4 for the pilot study. This study contains 46 products which account for 33.9% of the total work in progress.

The points defining the blading positions are not in the unit hypercube. We therefore multiply them by a factor c, where $c = \frac{(c-c_{min})}{c_{max}-c_{min}}$, with c_{max} and c_{min} the maximum and minimum values of all coordinates in our data set. This resizing makes sure all values are between 0 and 1, while the relative distances stay the same.

We will test two versions of our spacefilling curve heuristic against the ILP found in Section 2.4. We take $j = 8$, since after resizing the minimum distance between two points over all instances is 0.00714 and $2^{-8} < 0.00714$. The approximation results can be found in Table 4.2. We will also try to speed up the ILP by using the solution found by the `Hilbert24` algorithm as an initial solution for the ILP. The time required for each algorithm can be found in Table 4.3.

The `Hilbert24` algorithm performs quite well for the blading optimization problem. Compared to the original solution, the solution found by `Hilbert24` is only worse for four of the job optimization problems. In three of those cases, the original ordering was already optimal (MFS00, MYA00,MH021). In the last case (NKH02), the original was only

Instance			Objective value (B_{total})				Approximation ratio	
Product	Images	Layers	Original	ILP	Hilbert1	Hilbert24	Hilbert1	Hilbert24
MAA28	20	22	3124.62	3124.62	3196.67	3124.62	1.02	1
MAA04	20	25	3174.15	3174.15	3222.88	3174.15	1.02	1
MFS00	27	30	3253.83	3253.83	4134.13	3362.09	1.27	1.03
MAA01	18	24	3766.23	3517.08	4093.08	3577.15	1.16	1.02
MAA50	22	46	4271.1	3198.15	3365.3	3227	1.05	1.01
MAA29	20	22	3854.13	3298.6	3298.6	3298.6	1	1
SZS08	19	18	3968	3356.42	3440.17	3356.42	1.02	1
SQB01	30	14	5589.74	4640.15	4932.11	4665.15	1.06	1.01
MA936	23	49	5262.5	4374.2	4389.12	4389.12	1	1
SAC19	17	21	2677.03	2065.25	2219.63	2065.25	1.07	1
MAA45	19	28	6757.42	5392.8	5614.93	5392.8	1.04	1
MX028	26	48	6642.52	4339.3	5015.6	4339.3	1.16	1
RMP01	6	15	2589.9	2589.9	3299	2589.9	1.27	1
BGY00	7	22	3973.5	3973.5	4884.9	3973.5	1.23	1
MA924	22	29	6817	5582.1	5938.2	5597.22	1.06	1
XX055	26	46	4798.95	3922.35	4292.15	3922.35	1.09	1
DWB12	36	27	9177.64	6467.53	7384.78	6467.53	1.14	1
MAE00	16	18	3017.27	2403.75	2504.72	2403.75	1.04	1
MGY01	8	22	2848.63	2480.03	3170.78	2480.03	1.28	1
SZS05	18	21	3389.15	2646.79	2871.54	2646.79	1.08	1
XX040	24	62	4124.55	2644.89	2909.78	2644.89	1.1	1
MAA47	20	22	5551.7	4294.3	4548.35	4294.3	1.06	1
MX031	26	38	5323.07	3886.75	4227.58	3886.75	1.09	1
MAB00	22	42	3924.05	2536.6	2877.02	2536.6	1.13	1
XJ203	52	61	6104.29	3932.45	4466.73	3965.23	1.14	1.01
SAC03	19	17	2393.15	1991.68	2027.9	1991.68	1.02	1
DWB57	32	30	8827.86	6542.26	7060.97	6542.26	1.08	1
NKH02	19	29	7324.75	7294.33	7891.14	7561.34	1.08	1.04
MAA48	21	29	2373	2373	2422.03	2373	1.02	1
MAE03	20	18	3300.7	2298.57	2406.05	2298.57	1.05	1
XXW61	20	36	6632.37	5739.22	5806.05	5787.29	1.01	1.01
DWB14	36	27	9177.64	6467.53	7384.78	6467.53	1.14	1
XX056	23	56	6639.62	4333.05	4996.72	4333.05	1.15	1
MAA54	21	32	4662.35	3721.67	4218.22	3721.67	1.13	1
MAA62	16	23	3845.97	3105.01	3167.8	3105.01	1.02	1
MAA32	19	27	3857.08	3857.08	4387.63	3857.08	1.14	1
MYA00	26	22	5300.35	5300.35	5655.2	5655.2	1.07	1.07
MAB04	18	18	2938.18	2346.35	2413.8	2346.35	1.03	1
DWB06	36	36	9177.64	6467.53	7384.78	6467.53	1.14	1
SAB21	18	18	3419.88	2817.12	2935.2	2817.12	1.04	1
DWB40	36	31	9892.7	7550.32	7591.89	7591.89	1.01	1.01
MH021	19	28	3997.99	3997.99	4483.59	4075	1.12	1.02
MX032	19	30	4442.72	3261.03	3315.13	3261.03	1.02	1
MAA49	23	34	5424.43	4613.83	5053.8	4668.3	1.1	1.01
RKH05	17	27	8526.28	5302.28	6046.98	5407.03	1.14	1.02
MAA25	23	37	3726.45	3040.18	3352.63	3112.4	1.1	1.02
Average	22.28	29.93	4997	3989.47	4354.35	4017.84	1.09	1.01

Table 4.2: Results of the blading minimization for the ILP solver and the spacefilling curve heuristics, compared to the original order used in the Fab.

0.4% from the optimal solution. On average, the solution found by `Hilbert24` is 19.6% smaller than the original solution. This results in a gain in total time needed for the current work in progress similar to the gain of using the ILP (1.5% on 'fast' tools and of 1.7% on the 'slow' tools).
Compared to the optimal solution, it is on average only 1% away from the optimal solution. It is optimal for 31 of the 46 instances. In the worst case, it is 7% from the optimal solution (MYA00).

The `Hilbert1` algorithm performs worse than the `Hilbert24` algorithm for the blading optimization problem. This is of course to be expected, since the solution of `Hilbert1` is one of the solutions considered in `Hilbert24`, but the difference is quite large. The `Hilbert1` algorithm is worse than the original solution for 12 of the instances. On average, the solution found by `Hilbert1` is 12.9% smaller than the original solution. Hence, even using this algorithm to minimize the blade movement would result in a gain of 1.0% on 'fast' tools and of 1.1% on the 'slow' tools in total time needed for the current work in progress. It is at most 27% larger than the optimal as well as from the original solution (MFS00). On average, it is 12% larger than the optimal solution.

Instance			Time (sec)[1]				
Product	Images	Layers	ILP	Hilbert1	Hilbert24	ILP initial	Δ ILP
MAA28	20	22	0.632	0.042	0.062	0.838	-0.206
MAA04	20	25	0.174	0.005	0.056	0.161	0.013
MFS00	27	30	2.297	0.006	0.096	2.123	0.174
MAA01	18	24	0.776	0.005	0.052	0.962	-0.185
MAA50	22	46	0.267	0.005	0.069	0.287	-0.020
MAA29	20	22	0.450	0.004	0.057	0.371	0.079
SZS08	19	18	0.399	0.004	0.053	0.434	-0.035
SQB01	30	14	2.324	0.007	0.115	5.792	-3.468
MA936	23	49	0.613	0.005	0.074	0.698	-0.085
SAC19	17	21	0.294	0.034	0.049	0.608	-0.314
MAA45	19	28	0.849	0.005	0.054	0.636	0.213
MX028	26	48	0.733	0.006	0.092	1.024	-0.290
RMP01	6	15	0.048	0.002	0.011	0.018	0.030
BGY00	7	22	0.015	0.002	0.012	0.018	-0.003
MA924	22	29	0.451	0.005	0.069	0.684	-0.233
XX055	26	46	0.994	0.006	0.092	0.921	0.074
DWB12	36	27	4.086	0.008	0.163	2.251	1.835
MAE00	16	18	0.486	0.004	0.041	0.541	-0.056
MGY01	8	22	0.034	0.003	0.016	0.035	-0.001
SZS05	18	21	0.185	0.004	0.049	0.205	-0.020
XX040	24	62	0.795	0.005	0.079	1.217	-0.423
MAA47	20	22	0.187	0.004	0.058	0.309	-0.121
MX031	26	38	0.045	0.006	0.089	0.096	-0.051
MAB00	22	42	0.522	0.005	0.073	0.605	-0.083
XJ203	52	61	0.908	0.016	0.325	0.810	0.099
SAC03	19	17	0.054	0.004	0.053	0.040	0.014
DWB57	32	30	1.021	0.007	0.130	1.072	-0.051
NKH02	19	29	5.983	0.004	0.055	4.107	1.876
MAA48	21	29	0.482	0.004	0.061	0.610	-0.128
MAE03	20	18	0.260	0.004	0.057	0.371	-0.111
XXW61	20	36	0.743	0.004	0.058	1.132	-0.390
DWB14	36	27	4.051	0.009	0.162	2.225	1.826
XX056	23	56	0.819	0.005	0.077	0.538	0.282
MAA54	21	32	0.558	0.004	0.061	0.660	-0.103
MAA62	16	23	0.769	0.004	0.042	0.459	0.310
MAA32	19	27	0.078	0.004	0.054	0.082	-0.004
MYA00	26	22	0.236	0.006	0.090	0.403	-0.167
MAB04	18	18	0.696	0.004	0.049	0.665	0.030
DWB06	36	36	4.071	0.009	0.161	2.250	1.821
SAB21	18	18	0.548	0.004	0.049	0.495	0.053
DWB40	36	31	14.976	0.009	0.162	10.043	4.933
MH021	19	28	0.066	0.004	0.053	0.073	-0.007
MX032	19	30	0.357	0.004	0.053	0.143	0.214
MAA49	23	34	0.372	0.005	0.075	0.685	-0.313
RKH05	17	27	1.377	0.004	0.044	0.661	0.716
MAA25	23	37	0.288	0.005	0.073	0.181	0.106
Average	22.283	29.935	1.226	0.008	0.077	1.068	0.158

Table 4.3: Time required for blading minimization by the ILP solver and spacefilling curve heuristics. Δ ILP is time required for the ILP without initial solution minus time with initial solution.

Both `Hilbert1` and `Hilbert24` are very fast. Table 4.3 shows that `Hilbert1` takes at most 0.042 seconds (MAA28) and `Hilbert1` takes at most 0.325 seconds (XJ203). On average, `Hilbert24` is almost 16 times faster than the ILP. We also tried to use solution found by `Hilbert24` as an initial solution to the ILP. In general, the ILP with initial so-

lution takes around the same time as the ILP without. There are some instances for which it even generates a significant decrease in time (DWB12, NKH02, DWB14, DWB06, DWB40). There is however one instance (SQB01), where it increases the time required for the ILP by 3.468 seconds. This could be because the `Hilbert24` solution is close to the optimal, but not in the neighborhood of the optimal solution.

We fixed $j = 8$ since 0.00714 is the minimum distance after resizing over all problem instances that a blade has to travel between two images. This minimum is only obtained in MX028, XX055 and MX031. We could speed up the algorithm for some of the other instances by first calculating the required j and using this j for the algorithm. Appendix A.3 shows the result per instance. On average it reduces the time of `Hilbert1` by 5.3% and `Hilbert24` by 6.0% (including the time required for finding j).

4.5 CONCLUSION

We looked at a space filling curve heuristic for the *a priori* TSP problem. We saw that the Sierpinski curve solves TSP problems in the plane, but it is not easily extended to higher dimensions. Since the Hilbert curve is easily computed in higher dimensions, we used that curve instead, but we lose the approximation guarantee. Although we used the heuristic on *a priori* TSP, it can be used for a variety of optimizations problems that are optimizing over points in a multidimensional space.
We tested the algorithm on instances from the problem of minimizing the blade movement in the photolithography process. The `Hilbert24` algorithms is very fast (it requires less than 0.5 seconds) for the instances and on average it is less than 1% away from the optimal solution. Although the ILP is fast enough for our application, one might however also include the `Hilbert24` algorithm in the programming when building the Litho Job Manager described in Section 2.5, to give a good approximate solution for larger instances.

REFERENCES

[1] T. Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. `http://mpc.zib.de/index.php/MPC/article/view/4`.

[2] M. Bader. *Space-filling curves: an introduction with applications in scientific computing*. Springer Science & Business Media, 2012.

[3] J. J. Bartholdi III and L. K. Platzman. Heuristics based on spacefilling curves for combinatorial problems in euclidean space. *Management Science*, 34(3):291–305, 1988.

[4] D. Bertsimas and M. Grigni. Worst-case examples for the spacefilling curve heuristic for the euclidean traveling salesman problem. *Oper. Res. Lett.*, 8(5):241–244, 1989.

[1] The construction of the ILP is done in Matlab and solved using SCIP[1]. The `Hilbert1` and `Hilbert24` algorithms are implemented in Matlab. Calculations are done using an Intel core i7-6700k CPU, 4.0 GHz with 16 GB of RAM.

[5] A. R. Butz. Alternative algorithm for hilbert's space-filling curve. *IEEE Trans. Comput.*, 20(4):424–426, 1971.

[6] D. Hilbert. Über die stetige abbildung einer linie auf ein flachenstück. *Mathematische Annalen*, 38:459–460, 1891.

[7] G. Jin and J. Mellor-Crummey. Sfcgen: A framework for efficient generation of multi-dimensional space-filling curves by recursion. *ACM Transactions on Mathematical Software (TOMS)*, 31(1):120–148, 2005.

[8] L. K. Platzman and J. J. Bartholdi III. Spacefilling curves and the planar travelling salesman problem. *Journal of the ACM (JACM)*, 36(4):719–737, 1989.

[9] F. Schalekamp and D. B. Shmoys. Algorithms for the universal and *a priori* TSP. *Operations Research Letters*, 36(1):1–3, 2008.

5

SCHEDULING IN THE METAL AND PHOTOLITHOGRAPHY BAYS

In this chapter, we look at the scheduling of jobs in the metal and photolithography areas of the semiconductor fab. We analyze the possible benefits of replacing the existing dispatching rules by an optimization algorithm. The dispatching rules give a priority to every job. Operators use these priorities to decide which job will be put on an available machine first. The process times for a single job can be different for the candidate machines due to the high-mix nature of the fab. These tool speed differences are not exploited with the current rules. Through an enlarged data-collection and an improved IT-architecture, we can now accurately predict the process times for each job in the WIP on all candidate tools. This enriched information is used to construct a two stage algorithm, which exploits the speed differences of the candidate tools. This new scheduler solution can give a load distribution which is balanced over the possible equipment while it optimizes the overall equipment throughput. In the second stage the job priorities are taken into account and reticle constraints in case of lithography.

5.1 INTRODUCTION

European IC manufacturers are typical suppliers of customized products for equipment manufacturer companies, which means high-mix (product diversity), low volumes (many customers) and high flexibility. In order to collect the market advantages of high-mix, low-volume production the semiconductor industry needs to solve the associated complex logistic manufacturing challenges. The processing of different wafer products requires thousands of recipes. On top of that, additional complications arise from the

This chapter contains joined work with Jan Driessen

hardware differences (because of product-dedicated tool layout and/or wafer fab expansion with second-hand equipment).

In existing legacy 200mm fabs, the data-collection from the manufacturing equipment is quite often insufficient to obtain reliable process times for all recipes. The speed differences over the candidate tools for each recipe are usually not known accurately, *e.g.* in a simple model the wafers-per-hour production rate is scaled linearly with the number of available components. As a result, the scheduling (*i.e.* assignment) of lots in the WIP (work-in-progress) to the various candidate tools is generally done using a basic set of dispatching rules, very similar to the approach by Dabbas and Fowler [5].Each product or lot is assigned a priority rating by the logistic manager. This priority rating is used by the logistic manager to take control of the expected fab-out dates of each product (*i.e.* sense of urgency for delivery). In the daily fab operation the lot with the highest priority in front of the machine is handled first by the operator and loaded onto the equipment.

In general, when a job enters the factory it is given its priority rating. However, these priorities may shift over time due to a couple of reasons. First of all, the priority may change when a job is running late. Furthermore, the priority may be raised when there are time constraints between consecutive steps and the batch is not expected to make all process steps in time. Additionally, sometimes the process step may result in a few wafers with errors. These wafers are split up in a new batch, called the child batch, and are given additional process steps for repairs. It may be beneficial to unite this child batch with the original (parent) batch. For this, the priority of the child batch is increased. Finally, the equipment performance is monitored by regular inspection of the processed jobs. Only a fraction of the jobs in the fab will be inspected. It is important that the interval between consecutive lot-inspections for each tool does not become too large. When this interval becomes too large, the priority of the next inspection lot is increased. Of course, it is possible that lots in the WIP have the same priority. For these situations certain tiebreaker rules are in place, which may vary per tool group.

Dispatching strategies which rely predominantly on priority ratings, and which lack detailed and accurate timing input for all different recipes on all candidate tools, are not necessarily suited to maximize the fab-efficiency. However, the integration of modern IT-solutions in existing legacy semiconductor fabs has resulted in 'smart solutions' which can predict accurate process times for the WIP in front of the tool. In this chapter we will use this enriched information as model input to develop and validate improved dispatching strategies to raise the overall throughput of the critical tool-sets in the fab.

5.2 MOTIVATION

The processing in an high-mix, low-volume environment is irregular and difficult to model and/or predict. More processing details are needed (on wafer and component level) as well as a raised level of manufacturing science (complex mathematical models) to accurately describe and model high-mix, low-volume manufacturing. In an ideal IT-architecture where all relevant data is collected and analyzed almost 'live', it is possible to predict the next-load and next-unload timestamps for each tool. Moreover, using a

component-based timing model it becomes possible to accurately predict the process time for each lot in the queue on all possible candidate tools.

In a recent paper by Driessen et al. [8], it has been demonstrated that a raised level of manufacturing science can be realized in existing legacy 200mm fabs based on a robust data-collection compatible with outdated hardware and software. A mathematical timing model has been developed which is capable to predict accurate recipe-times per tool in high-mix, low-volume manufacturing based on the historic data-collection. Because the implementation of the smart manufacturing strategies is quite labor intensive, the feasibility of this approach has been demonstrated in the pilot phase on two tool-families, 1) the metallization tool-set (AMAT Endura) with the most complex diversity of component-utilization per recipe, and 2) the lithography tool-set (ASML stepper/scanner with SVG track) which is supposed to serve as the bottleneck of the manufacturing line.

For these two pilot tool-sets the smart manufacturing solution has matured into a reliable timing model which is capable to generate accurate tool-recipe predictions. The process flow of a wafer through such a so-called 'multi-path cluster tool' is modeled logistically using the SEMI standard E10-0304E. Based on the process times and handling times, one can determine the throughput times at each functional step depending on the number of available components. The overall takt time at tool level is determined by the bottleneck step, which can be readily identified in such a logistic model.

In Figure 1, this timing model is exemplified for the metallization toolset. In the actual fab, these multi-path cluster tools are laid out with various configurations. The collected timing information at wafer/component-level is analyzed and used as the input data for the timing model. Using the actual up/down status of individual components, it is now possible to accurately predict the recipe times of all lots in the WIP on all candidate tools.

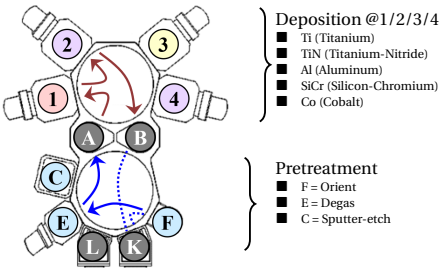
In case of lithography, the exposure time at the stepper also depends on the reticle parameters. As described in a previous paper by Driessen [7] and discussed in Section 2.1, the total process time (T_P) to pattern the wafer can be expressed in a simple model

$$T_P = c_0 + c_1(ND/I) + c_2 N + c_3(M-1) + c_4 B \tag{5.1}$$

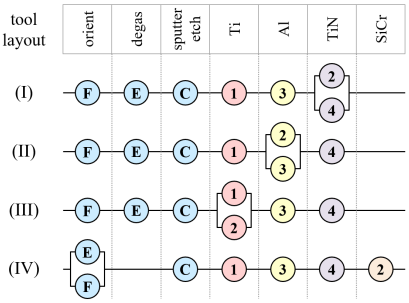
where the coefficients c_0, c_1, c_2, c_3 and c_4 are equipment constants which represent the stepper hardware. The reticle-job is described by the other parameters: N is the total number of flashes, D is the required energy dose, I is the light intensity at the wafer, M is the number of different images, and B is the total blade movement needed to create the series of partial exposures.

In order to use the predicted process times obtained from the new smart manufacturing solution, we have to validate the accuracy of the calculated recipe times on the candidate tools. This essential step has been performed for all tool-sets where our smart IT-solution has been implemented. In Figure 5.2 the validation results are shown for the most critical tool-set in the semiconductor fab that we consider.

We have analyzed one month of production data processed on 26 lithography tools. The



(a) tool-layout AMAT ENDURA



(b) process flow modeling (SEMI E10-0304E)

Recipe & Stack		Equipment (layout) Tool1 (I)	Tool2 (II)	Tool3 (II)	Tool4 (III)	Tool5 (III)	Tool6 (IV)
A	TiN	94	145	147	143	145	142
B	Ti-Al-TiN	108	82	80	100	102	103
C	Ti-Al-TiN	183	103	104	189	192	184
D	Ti-Al	263	130	126	256	254	257
E	Ti-Al-Ti-TiN		166	164	118	119	
F	Ti-Al-Ti-TiN		157	153	134	135	
G	SiCr						192

1) mathematical timing model

@ component level
- process time – recipe dependent
- handling time – recipe independent

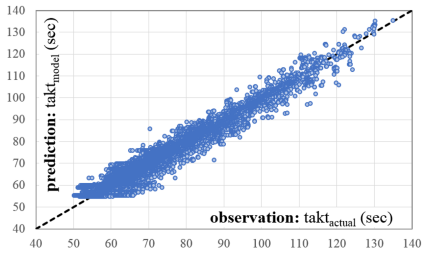
@ equipment level
- determine limiting component
- calculate effective takt time

2) data-collection → determine and validate model input → accurate prediction of actual and golden recipe times.

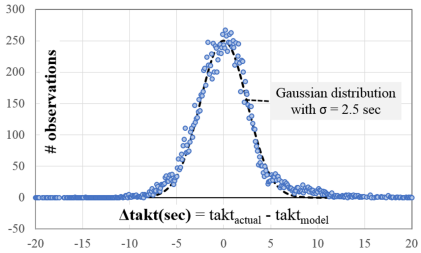
Figure 5.1: The process time on a so-called 'multi-path cluster tool' depends on the actual tool-layout. Using the SEMI-standard E10-0304E a mathematical model is set up. The process times and handling times are used as input and can be validated regularly with historical data from the equipment. Taking into account the actual tool-status (e.g. components up or down), one can accurately predict the recipe times needed for scheduling optimization.

high-mix character is obvious from the large amount of different reticle jobs: on average a specific exposure job is performed only 6.5 times on one of the 26 candidate tools. In Figure 5.2a the predicted takt times are compared to the actual observed values. Even though these takt times are distributed over a wide range (from 50 to 130 seconds), we are capable of making reliable predictions. The accuracy is obtained from the distribution of the difference $\Delta\text{takt} = \text{takt}_{\text{actual}} - \text{takt}_{\text{model}}$. This accuracy is shown in Figure 5.2b where a Gaussian distribution with a standard deviation of $\sigma = 2.5$ seconds is found. On top of this, the model is capable of detecting changes which impact the process time. This is illustrated in Figure 5.2c where the number of available coater components in the lithography changed from 1 to 2. We see that our timing model adapts correctly to the new tool speed settings.

With the process times for all different tool-recipes combinations now available, the challenge lies in finding an algorithm that makes use of this new data to find a better schedule for the current WIP the fab. The goal of such an algorithm would be to produce the jobs as fast as possible while still taking into product restrictions, reticle conflicts and time constraints.

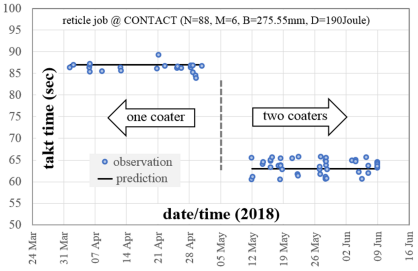


(a) observation versus prediction



(b) accuracy of timing model

- 20684 runs (1 month data, 26 tools)
- 3250 reticle jobs (≈ 6.5 run/reticle)
- exposure time = function(N, M, B, D)
- accurate predictions ($\sigma = 2.5$ sec)
- for new reticles and small runners
- dynamic model adapts to comp. up/down



(c) adaptive model (auto-adjust)

Figure 5.2: In case of photolithography, the wafers-per-hour production rate depends not only on the tool layout (see Fig. 5.1) but also on the parameters of the reticle job (see Eq. 5.1). The observed takt times are distributed over a wide range (a), for which the model can make accurate predictions (b). Moreover, the model can detect tool changes which impact the equipment throughput (c).

5.3 ALIGNMENT OF NAMING CONVENTION

As described in Section 1.5, scheduling problems are often studied in the field of mathematical optimization. The names and definitions used are not necessarily the same as the terminology used in the daily operation of a manufacturing line. In Figure 5.3 we make a comparison of the naming conventions in these two fields by visualizing their definitions in a Gantt chart.

In manufacturing, we can identify three important metrics; cycle time, lead time and capacity time. Cycle time is defined as the total time from the beginning to the end of a process. In the graph it corresponds with the time interval between lot-start (podPlaced) and lot-finish (podUnlocked). Lead time is explained as the time required to manufacture an item, including queue time and run time. It is the time interval between the lot-finish of the job in the previous process step and the lot-finish of the current step, as illustrated in our Gantt visualization. Finally, capacity time is related to the production speed of the equipment. It is inversely proportional to the equipment throughput.

Both cycle time and capacity time are recipe dependent, and they may differ over the various processing equipment. As described in section 5.2, in a high-mix manufacturing line detailed data is needed to cover all possible tool-recipe combinations. The lead time

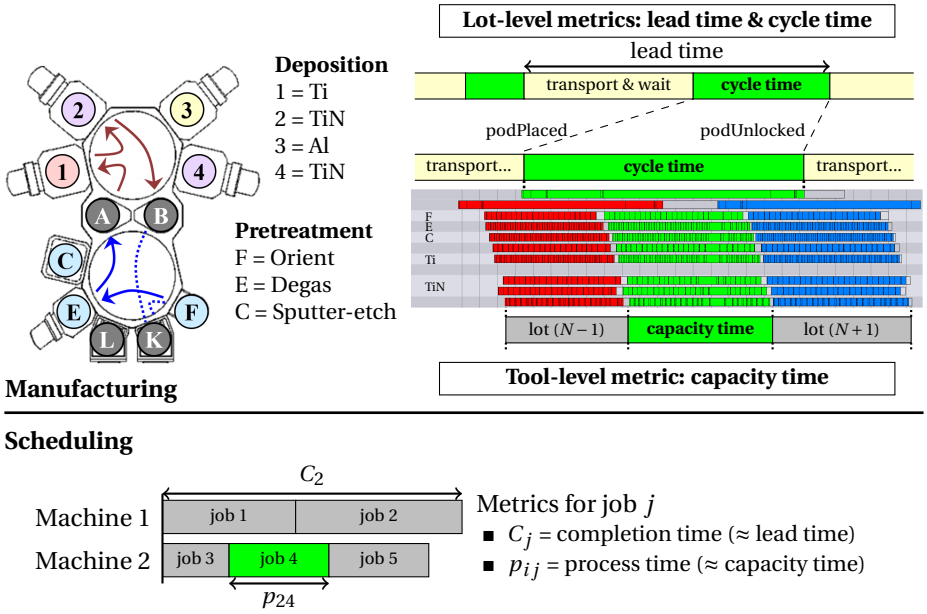


Figure 5.3: The terminology used in the field of manufacturing differs from the naming convention applied in scheduling research. However, it is possible to identify the terms with similar meaning. The completion time C_j of a job can be compared to the lead time of the manufactured product (i.e. delivery speed of the lot), whereas the process time p_{ij} is equal to the capacity time of the production line (i.e. equipment throughput or tool-speed).

depends strongly on the WIP-levels in front of the available tools, and how the logistic transport of lots in the fab is organized. It is well-known that a great amount of WIP will result in large average lead times.

In scheduling one can distinguish two important metrics - completion time and process time. Completion time is the calculated amount of time required to complete a particular task. In the Gantt visualization it is indicated by the total time span from 'start' until 'job finish'. The process time is the specific amount of time needed on a certain machine to process an individual job. In Figure 5.3 it correlates with the length of the individual building blocks making up the complete schedule. From this graph it is obvious that it matches the capacity time in manufacturing.

Scheduling optimization impacts both metric measurements. A reduction of the average completion time per job can be interpreted in the field of manufacturing as a shortened lead time, which is beneficial for lot delivery (*i.e.* customer satisfaction). On the other hand, a reduction of the average process time per job corresponds to a raised overall throughput of the equipment (*i.e.* increased fab effiency). Both points of view will be discussed in this chapter when interpreting the optimization calculations for metal and photolithography.

5.4 KNOWN APPROACHES

Many scheduling problems encountered in a semiconductor fab have been considered in literature. The photolithography bay is often the focus of these studies since it is the bottleneck of the fab. The literature that tackles logistic problems in lithography can be roughly divided in two types of techniques: dispatching rules or scheduling algorithms.

Dispatching rules have been studied for quite some time in literature. Through these rules a priority is assigned to a job, and machines select jobs with the highest priority. Classical examples are i) first-in-first-out, where jobs are processed in the order in which they are received; ii) shortest processing time first; iii) critical ratio, which is calculated by dividing the time remaining to complete a job by the expected time needed to finish the job; and iv) earliest due date, by which jobs are processed according to their delivery dates. These rules perform reasonably well for some problems [12]. Sarin et al. [13] give a detailed overview of the basic dispatching rules used in the semiconductor industry and their performance.

Geiger et al. [9] conclude after many years of research on dispatching rules that there is no dispatching rule that outperforms consistently all the other rules and that the performance of every rule is very dependent on the problem considered. Therefore, they propose a machine-learning algorithm that dynamically generates effective dispatching rules automatically from a set of given rules.

Scheduling algorithms for the problems found in the photolithography areas have to be able to deal with the reticle requirements of the jobs. Cakici and Mason [4] model this as an identical parallel machine scheduling problem with auxiliary resources and release dates. The objective is to minimize the total weighted completion time. They use an ILP formulation reminiscent of the one found in Section 2.3 to find optimal solution to the problem with at most 3 machines, 15 jobs and 6 reticles.

Ham and Cho [11] consider parallel machines with auxiliary resources, which require additional setup times. They use a two stage model. First, they use an ILP to find an assignment of the jobs and resources to the machines. They find a reduction of 2.7% in total completion time for instances with 30 machines, 500 jobs and 800 reticles. In the second stage, the jobs assigned to each tool are sequenced using a set of dispatching rules.

Doleschal et al. [6] formulate the lithography scheduling as a problem with unrelated machines with auxiliary resources. They use an ILP with three objective functions, that they optimize in three stages. In sequential steps the high priority jobs, the total number of jobs and load balance are optimized. The result of this ILP is an allocation of the resources such that every resource is assigned to at most 1 machine. The assignment is then tested in an online environment using a simulation model and evaluated against a dispatching rule. In their test it yields especially good results in terms of cycle time and lead times (*e.g.* tardiness).

Bitar et al. [2] propose a memetic algorithm. They consider the problem of unrelated parallel machines with release dates, sequence dependent setup times and auxiliary re-

sources. The algorithm optimizes two objective functions: it minimizes the total weighted completion time and maximizes the total number of jobs produced in a fixed time window. The memetic algorithm generates solutions using a genetic algorithm, which are then improved using a local search algorithm (in this case simulated annealing). They analyze the impact of different parameters on the performance of the algorithm.

All of the above described scheduling algorithms use a priority rating (or weight factor) for every job. This priority is used either as part of the objective function or as a second stage, the so-called post-processor. In this step the jobs that have been assigned to a machine are sequenced in order of priority. In this manner the logistic manager still has a means of control to influence the actual processing order of individual jobs. Note, such a means of control is often highly appreciated in the daily operation of a manufacturing line, because the scheduling algorithms are often perceived as 'black boxes'.

As a final remark, all of the above described scheduling algorithms focus on the constraints due to the limited availability of reticles (resources). For instance, individual reticles cannot be assigned to multiple machines simultaneously, which is referred to as a reticle conflict. However, when the high-mix character of a fab increases, more reticles are available and the chance of encountering reticle conflicts is reduced. Looking at the data provided in Figure 5.2, we consider these reticle requirements less of a constraint in the fab that we consider. Therefore, we propose an algorithm that focuses more on the production time differences and less on the reticle requirements.

5.5 MODEL AND ALGORITHM

Given the new detailed data that have become available as described in Section 5.2, we are able to model the production bays and devise an algorithm that optimizes the scheduling. We will use a static model where we are given a number of machines and the current work in progress. For each job j we know where it can be processed (machine i) and with what process time (p_{ij}). Furthermore, we know the job priority from the dispatching model and (in case of photolithography) the reticle that is required. Finally, we know which jobs are already on a machine. The goal is to find an algorithm that minimizes the machine time required for the current WIP, while still balancing the load. Optimizing the machine time only, may result in an overloading of the fastest machine and an idle situation on the slow equipment. This would wreak havoc for the line balance, resulting in large completion times. Therefore, a smarter way of optimizing is needed to satisfy the customer needs (*i.e.* shorter lead times)

In the more formal notation of Graham et al. [10], we formulate the problem as a scheduling problem with unrelated parallel machines. We are given a set of m unrelated machine M, a set of n jobs J and a set of l resources R. We have the following parameters:

- a process time $p_{ij}, \forall i \in M, j \in J$. $p_{ij} = \infty$ if job j cannot be processed on machine i.
- a priority (or weight) $w_j, \forall j \in J$.
- a resource (or reticle) $r_j \in R, \forall j \in J$ required when scheduling in the lithography bay.

- a partial schedule S' containing the jobs in J that are already on a machine.

Note that, we will denote r^q as the qth resource and r_j as the resource used by job j. The proposed algorithm works in two stages, similar as described by Ham and Cho [11]. First, an integer linear program (ILP) is used to find an optimized assignment of jobs in the WIP to the available machines. In the second stage, dispatching rules and priority ratings are used to determine the ordering of the jobs on each machine.

As an objective function for the ILP, we will use the total completion time objective (TCT = $\sum C_j$). As explained in section 5.3, this objective minimizes the lead time of the jobs. It achieves our goal of finding a balanced schedule that minimizes the total machine time by preferring to put a job on a faster machine. Overloading the fastest machines results in large completion times, thus penalizing the planning of jobs far ahead in time. As a result, jobs are moved to the other equipment and the proposed load distribution becomes more balanced.

First, we calculate $d_i, \forall i \in M$ from S', which is the timestamp when all jobs already loaded onto machine i are finished. Therefore, at time d_i the machine i becomes available for scheduling of jobs waiting in the WIP in front of the tool. Next, We use the following ILP formulation:

$$\begin{aligned}
\min \quad & \textstyle\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{n}(kp_{ij}+d_i)x_{ijk} \\
\text{s.t.} \quad & \textstyle\sum_{i=1}^{m}\sum_{k=1}^{n} x_{ijk} = 1, && \forall j \in \{1,\ldots,n\}, \\
& \textstyle\sum_{j=1}^{n} x_{ijk} \leq 1, && \forall i \in \{1,\ldots,m\}, \forall k \in \{1,\ldots,n\}, \\
& x_{ijk} \in \{0,1\}, && \forall i \in \{1,\ldots,m\}, \forall j \in \{1,\ldots,n\}, \forall k \in \{1,\ldots,n\}.
\end{aligned}$$

Here $x_{ijk} = 1$ when job j is the kth job to be processed on machine i after d_i, and $x_{ijk} = 0$ otherwise. Bruno et al. [3] show that the problem of minimizing the total completion time on unrelated machines $\left(R|\,|\sum C_j\right)$ can be solved efficiently (in polynomial time of the input size) using a minimal-cost-flow formulation. The ILP formulation we use, is very similar to the formulation obtained from this minimal-cost-flow formulation. The matrix obtained by both formulations is totally unimodular. Therefore, one can drop the binary constraints and solve the now linear program in polynomial time of the input size.

After solving the ILP, we fix the machine assignment of each job, according to the value of x_{ijk}. Let $J_i \subseteq J$ denote all jobs assigned to machine i by the ILP scheduler. In a second step, a post-processor uses the priority ratings provided by the logistic manager to determine the order in which these jobs will be processed.

For the metallization tool-set the post-processor is very straightforward. We sort the jobs j on every machine according to their priority rating w_j. When the jobs have the same weight factor, they are ordered according to their processing time p_{ij}.

In case of lithography, the post-processor becomes more complex. We also have to take into account the reticle constraints. Two jobs using the same reticle cannot be processed at the same time. A reticle conflict occurs in the proposed schedule S when two jobs

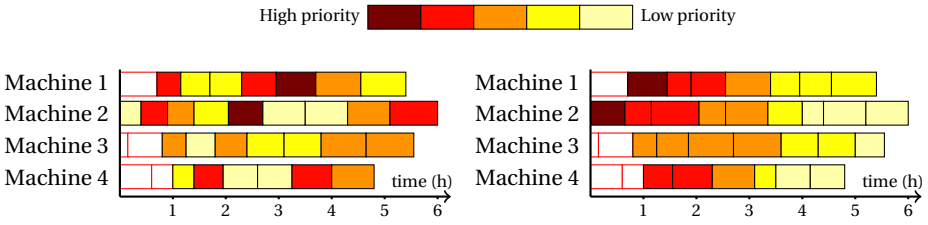
using the same reticle have overlapping intervals on the timeline. The post-processor tries to find a schedule without these conflicts.

Algorithm 1: Post-processor photolithography

Input: m machines, jobs $J = J_1 \cup \ldots \cup J_m$, processing times $p_j = p_{ij}, j \in J_i, \forall i \in M$, weights $w_j, \forall j \in J$, resources $r_j \in R, \forall j \in J$ and a partial schedule S'.
Output: A schedule S containing the machine and position for all jobs j

1 *initialization:* Set $\bar{J} = J \setminus J_{S'}$ and set $\bar{M} = M \setminus M_\emptyset$, where $J_{S'}$ are all jobs scheduled in S' and $M_\emptyset$ are all machines that have no jobs in $\bar{J}$ that can be assigned to them. Set the current schedule S equal to the partial schedule S'. Set $r^q(t)$ equal to the completion time of the last job using reticle q in S' and set $m_i(t)$ to the completion time of the last job on machine i in S'.;
2 **while** $\bar{J} \neq \emptyset$ **do**
3 **find** $i \in \bar{M}$ such that $m_i(t) = \min_{i \in \bar{M}} m_i(t)$;
4 **find** $j \in \bar{J}_i$ with $r_j(t) \leq m_i(t)$ such that j has maximum w_j;
5 **if** j *does not exist* **then**
6 **find** $j \in \bar{J}_i$ such that j has maximum w_j;
7 **end**
8 Add j to S on i on the first available position;
9 Set $r_j(t) = m_i(t) + p_{ij}$, $m_i(t) = m_i(t) + p_{ij}$ and remove j from $\bar{J}$;
10 **if** $\bar{J}_i = \emptyset$ **then**
11 Remove i from $\bar{M}$;
12 **end**
13 **end**
14 **return** S

The post-processor algorithm 1 schedules the jobs on each machine according to their weight. It does this one job after the other. For the next job to be considered, it first considers the jobs with a reticle that is not in use on another equipment. Among these it finds the job with the highest priority and schedules it. If such a job does not exist, it schedules a job with its reticle in use, which create a reticle conflict. In this manner all reticle conflicts will occur at the end of the schedule. Figure 5.4 shows a small example of influence of the post-processor on the schedule.



(a) Without post-processor (b) With post-processor

Figure 5.4: The post-processor sorts the jobs according to their weight. In the figures, dark red represents high and yellow low priority. Jobs already on the tool are white.

Although not currently implemented in the post-processor, one could make another adjustment to Algorithm 1 to cluster scheduling jobs which use the same reticle one after the other. In the sorting process the next job is selected with an adjusted weight $\bar{w}_j = w_j + b_j$, where b_j equals a bonus value when job j uses the same reticle as the previous job scheduled on i and zero otherwise.

The construction of the ILP is done in Matlab as well as the implementation of the post-processor. The ILP itself is solved using SCIP[1]. The results shown are obtained using an Intel core i7-4600 CPU, 2.10 GHz with 8 GB of RAM.

5.6 ANALYSIS STRATEGY TO IDENTIFY POSSIBLE GAIN

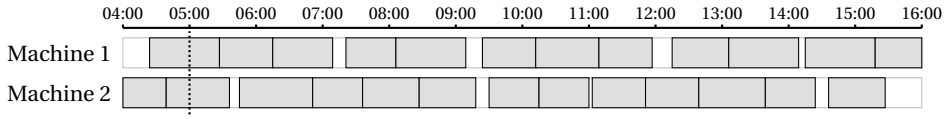
The scheduling algorithm makes an assignment of all the lots in the WIP to the available tools. This proposed load distribution is a static advice. In actual manufacturing, the WIP evolves dynamically, with new lots being added to the WIP when they finish the previous step. Depending on their priority settings, these new lots may even overtake jobs with a lower priority. Therefore, it is quite complicated to determine the possible benefits of the new smart solution.

The implementation of the new scheduling algorithm into the existing manufacturing line requires a major effort to make all the necessary modifications not only in the IT-architecture, but also in the logistic organization of the fab (*e.g.* lot transport). Before committing resources to such a big task, it is crucial to have an accurate estimate of the expected benefits.

To make an assessment of the improvement opportunity, an analysis method is proposed which compares the static result of the scheduling optimization with the actual historical loading of the equipment in the high-mix fab. In Figure 5.5, this strategy is illustrated for an example with 2 tools.

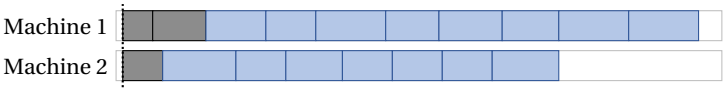
The historical loading of these 2 machines is plotted on a timeline in Figure 5.5a. Every hour we take a snapshot of the jobs already on a machine, as well as the jobs waiting in front of the tool-set, including their priority rating. At a later moment we can identify on which machine these individual jobs have been processed as is shown in Figure 5.5b. Next, we construct a reference loading scenario by eliminating the idle gaps as well as the lots which were not present in the initial WIP list. This reference schedule is depicted in Figure 5.5c.

As explained in Section 5.2, the new smart manufacturing solution is capable to predict accurate recipe-times per tool for all jobs in the WIP. This enriched information is included in the snapshot, which serves as input for the scheduling algorithm. The optimized loading is presented in Figure 5.5d and can be compared directly with the reference schedule. This snapshot analysis can be repeated for every instance. By aggregating these results over a longer period (*i.e.* many snapshots), we expect that the impact of the continuously changing WIP levels will balance out. In the next sections, we will apply this analysis strategy to the historical data collected for the two pilot tool-sets in metal and photolithography.

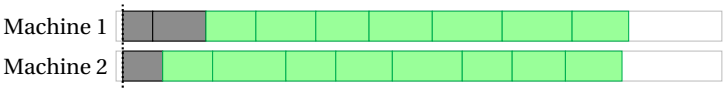


(a) historical loading of two machines as observed in actual production

(b) identify the lots which were present in the WIP at 05:00



(c) eliminate idle gaps → reference loading scenario at 05:00



(d) scheduling optimization → improved loading scenario at 05:00

Figure 5.5: Strategy to compare the load distribution in actual manufacturing with scheduling optimization

5.7 RESULTS METAL

In the manufacturing line that we consider, a total of 22 multi-path cluster tools are used for metal deposition. Using the timing model of Figure 5.1 and the different tool-layouts, we can make accurate predictions of the actual process times for all possible tool-recipe combinations.

Because of some functional constraints and hardware differences, not all recipes are interchangeable over all 22 machines. In fact, we can split the metal tool-set in three functional groups in which the job processes can be exchanged: 1) full wafer-coverage - 10 tools, 2) shielded marker - 9 tools, and 3) CVD-deposition - 3 tools. For all three subsets the analysis strategy has been applied using a large time window of historical loading (period = 7 months).

The hourly reference scenarios derived from actual manufacturing can be used in scheduling calculations with alternative optimization criteria. When we proposed to exploit the tool-speed differences to optimize the machine loading, many stakeholders were worried that the *fast tools* would become overloaded, and worse, that the *slow tools* would receive no load and remain idle. For this reason we have included two different metrics in the analysis: 1) the average completion time (aCT) and 2) the required machine capacity (MC). The impact of alternative optimization strategies on these two measurement values will be evaluated in this analysis.

In literature, the normally used optimization criterion is the *minimization of completion*

time. As explained in section 5.3 a reduced average completion time per job leads to a shortened lead time and faster lot delivery to the customer. In contrast, to address the worries of the stakeholders we will also use two optimization criteria in which the machine capacity is either minimized or maximized. In this manner we can illustrate the impact on manufacturing when overloading either the *fast tools* or the *slow tools*.

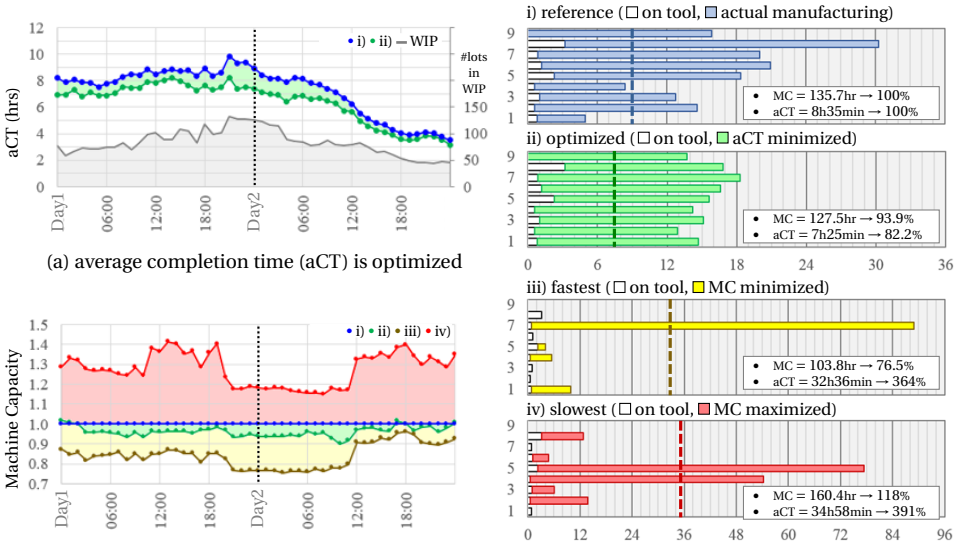


Figure 5.6: The historical loading of 9 metal tools with exchangeable processing in functional group 'shielded marker' is used for analysis. As explained in section 5.6 every hour a snapshot of the WIP is available for analysis to determine the impact on a) average completion time (aCT) and b) machine capacity (MC = $\sum p_{ij}$). Depending on the optimization goal, one can arrive at various alternative loading scenarios as shown in Figure c for timestamp (DAY 2, t=00:00).

The complete analysis for the 7-month period contains 5103 hourly WIP snapshots. The three aforementioned optimization strategies have been performed for all instances and for all three functional metal groups. In Figure 5.6 we present the analysis results over a two-day period for sub-set 2 (shielded marker). In Figure 5.6.a the average completion time is visualized on a scale of 0-12 hours. Because minimizing and/or maximizing the machine capacity produced very large aCT-values, they are not visible on this scale. The WIP-level of the snapshot is also included in this graph. The mathematical strategy which minimizes the aCT-value gives the best results in this graph.

In Figure 5.6.b the relative machine capacities of the three optimized scheduling results are presented, where a normalization is applied with respect to the reference scenario. Now, all three optimization strategies are depicted in the graph. The optimization results where the machine capacity has been minimized or maximized serve as the range of possible values for any alternative loading scenario. Even though the aCT-optimization

does not directly influence this machine capacity, there is obviously an indirect benefit. This can be understood, because the completion time of an individual job contains the sum of the process times of all previous jobs. Thus, the aCT-value improves when the previous processing is done with the shortest capacity times possible.

To illustrate the impact of these three optimization strategies in detail, four different loading scenarios are shown in Figure 5.6.c for one specific WIP-snapshot, taken at an arbitrary timestamp (DAY 2, t=00:00). In the reference scenario (i) the machine-loading is displayed as found in actual manufacturing. The corresponding aCT-value and MC-value serve as normalization (100%) for the three optimization strategies. In the optimized scenario (ii) the aCT-value is minimized giving a reduction of 17.8% compared to the reference scenario. As mentioned earlier, the machine capacity is reduced indirectly as well resulting in MC = 93.9%. In the fastest scenario (iii) all jobs are processed on the fastest machine leading to MC = 76.5%, but the scheduling distribution becomes severely unbalanced characterized by the high aCT-value of 364%. In the slowest scenario (iv) all processing is done on the slowest machines, which results in the worst case MC (114%) and poor values for aCT (391%).

5.7.1 OVERALL RESULTS FOR A 7-MONTH PERIOD

The optimization results have been collected for all WIP snapshots in a 7-month period. Because the WIP-levels in front of the metallization tools is relatively low, the variation from one snapshot to the next one can be quite large. In other words, in one hour of processing, the changes in WIP and tool-situation can lead to significant jumps in the results as can be seen in Figure 5.6. The scheduling optimization gives a loading distribution which holds for a *static* situation with no new jobs arriving over time. However, in actual manufacturing we deal with a *dynamic* situation where new jobs are added to the WIP-list over time in a random manner (much like an online optimization problem). This results in a error in the estimate for the capacity gain ΔMC. By averaging the optimization results over many instances, we expect that the contribution of this error becomes negligible.

The overall statistical results have been brought together in Table 5.1. These results suggest that the tool-speed differences can be exploited in all three functional groups in metallization. If the transport logistics can be aligned with this scheduling optimization, it may be possible to free up to 4.5% of machine capacity. Depending on the machine capacity of the other toolsets in the fab, this increase will contribute to a raise of the overall fab efficiency.

5.8 RESULTS PHOTOLITHOGRAPHY

The toolset for lithography consists of a total of 37 machines. The differences in material used such as photoresist and exposure hardware (stepper vs. scanner) lead to four different functional groups where the jobs are interchangeable from a hardware point of view. Within these four functional groups, additional constraints for job-exchange may

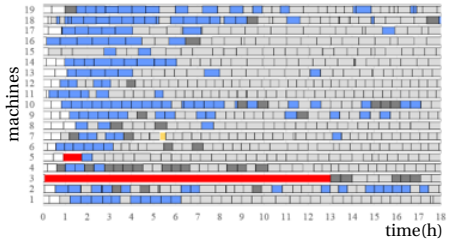
		Tool group		
		Full Coverage (10 tools)	Shielded Marker (9 tools)	CVD (3 tools)
	instances	4394	2913	3402
WIP	average	39.6	28.3	15.1
WIP	median	29	25	11
WIP	max	197	132	66
aMC	ref	0:42:35	0:56:56	0:48:50
aMC	opt	0:41:23	0:55:24	0:46:39
aMC	gain	97.18%	97.33%	95.52%
aCT	ref	3:12:13	3:24:00	3:35:06
aCT	opt	2:50:15	2:57:02	3:18:22
aCT	gain	88.57%	86.78%	92.2%
	calc. time	332 msec	259 msec	36 msec

Table 5.1: Optimization results for metal obtained from a 7-month period with 5103 instances. Only WIP snapshots have been taken into account which contain more jobs than the number of machines.

be in place since not all reticles have been qualified on all possible machines: the specifications for critical dimension control require additional tweaking and tuning per machine for each reticle. Three of the four functional groups are small (< 4 tools) and have negligible differences in hardware layout and tool speed. We will focus on the largest functional group (24 tools) which has significant differences in hardware (*e.g.* number of coater or hardbake components, lamp power, blading speed). As a result, the tool speed differences for each individual job can be quite large. Moreover, this functional group is considered the bottleneck of the fab based on the historical data for the WIP levels and the machine loading.

As explained in Section 5.6, we construct reference scenarios for each WIP snapshot using the historical loading of the lithography equipment. This is visualized in Figure 5.7. The jobs which were present in the WIP snapshot at time $t = 0$ have been identified in the Gantt visualization of Figure 5.7a. We can distinguish lots already placed *on the tool* (□) and lots waiting to be dispatched. These waiting lots can be separated in *fixed jobs* (■) which need to be processed on a specific machine and *flexible jobs* (■) which can be dispatched on multiple machines. The fraction of fixed jobs is relatively high (~20%) because the critical dimension constraints often require that successive photolithography operations are performed on the same equipment. In the timelines of Figure 5.7a, we can identify many lots which were not in the WIP list at $t = 0$ (□) but which overtook the previous lots because of priority settings and/or lot availability at the tool location.

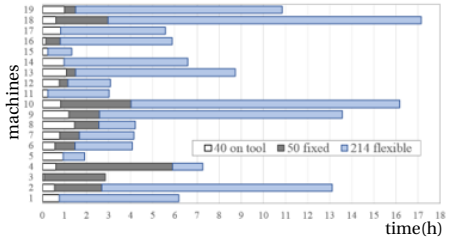
In Figure 5.7b the reference scenario is constructed by eliminating the idle gaps and the jobs which were not present in the WIP list at $t = 0$. The resulting load distribution appears quite unbalanced. In fact, one could interpret this scenario as if some tools were overloaded at the expense of other tools. However, from the timeline of Figure 5.7a we find that all tools are completely loaded. This dynamic evolution of the historical loading means that new WIP arrives with the same rate at which it is processed, keeping WIP



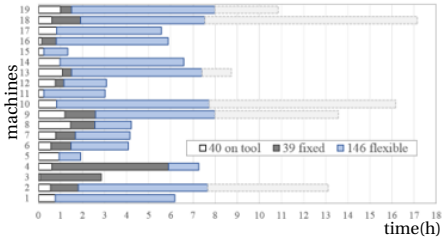
304 lots in WIP at $t = 0 \rightarrow$ identified in historical load
□ 40 on tool; ■ 50 fixed (no alternative);
■ 214 flexible (multiple candidate tools).

- continuous changing WIP-level and tool-status:
  ■ tool down; ■ tool idle; □ lots arriving later.
- reference scenario (b) → unbalanced machine loading
  □ 40 on tool; ■ 50 fixed; ■ 214 flexible
- reference scenario (c) → ignore lots with large CT
  □ 40 on tool; ■ 39 fixed; ■ 146 flexible

(a) historical loading (dynamic)



(b) unbalanced reference scenario



(c) reference scenario with cut-off after 8h

Figure 5.7: The historical loading of the litho tools (a) is analyzed into a reference scenario (b) which suggests that the machine loading is unbalanced, whereas all tools are completely loaded in Figure (a). Using a cut-off time the lots with a large completion time (CT) are eliminated in Figure (b) resulting in a more balanced reference scenario (c). The specific timestamp of this WIP snapshot is also used in the next Figures 5.8 and 5.9

levels high, such that tools never need to be idle in practice. On average, we find that per hour approximately 35 ~ 40 jobs are added to the WIP list and about the same amount disappears after processing.

Because the average completion time (aCT) of the unbalanced scenario in Figure 5.7b is relatively high, the results of the scheduling optimization may suggest a gain in aCT (or lead time) which is too optimistic. For this reason we have applied a second analysis strategy in which a more balanced reference scenario is created by eliminating the jobs with a large reference completion time. In Figure 5.7c this modified reference scenario is depicted for a cut-off time of 8 hours.

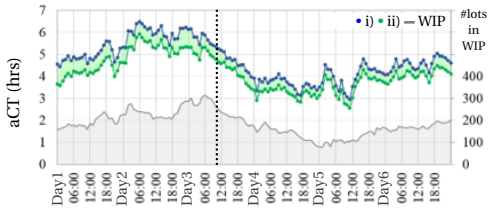
Both reference scenarios have been analysed for the chosen litho toolset using the hourly WIP snapshots. The number of lots in the WIP snapshot for lithography is much larger than for metal, resulting in longer calculation times. We analyzed a period of 5 months for the scheduling optimization in lithography.

5.8.1 RESULTS UNBALANCED REFERENCE SCENARIO

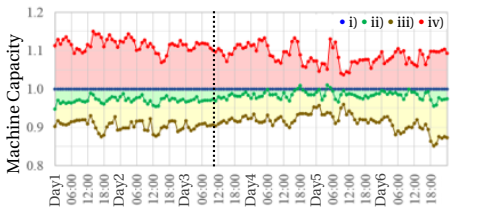
The complete analysis for the 5-month period contains 3676 WIP snapshot. In only 27 instances the WIP level was smaller than the number of tools (*i.e.* $n < 24$). Because of the high WIP levels the hourly changes in WIP resulted in relatively small changes of the

analysis results. The effect of changes in tool status (*tool up* versus *tool down*) was found to be larger than the evolving WIP levels.

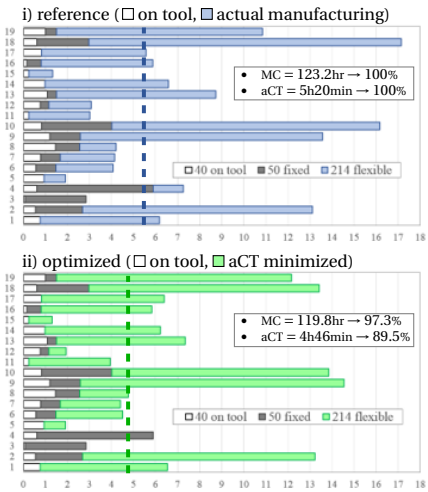
In Figure 5.8, the lithography results are presented using the unbalanced reference scenarios for a 6-day period with a similar graph layout as in Figure 5.6. In Figure 5.8a, the aCT-value is depicted on a scale of 0-7 hours, as well as the WIP level of the snapshot. The best results for average completion time are found with the optimized scenario, which minimizes aCT.



(a) average completion time (aCT) is optimized



(b) machine capacity (MC) w.r.t. reference scenario



(c) different scenarios at (DAY3, t = 10:00)

Figure 5.8: The historical loading of 19 litho tools is used to determine the impact on a) completion time (CT) and b) machine capacity (MC). The results are visualized in a similar manner as the metallization example of Figure 5.6. The average completion time (aCT) is optimized for timestamp (DAY 3, t=10:00), as visualized in Figure (c) by the aCT-lines in the reference and optimized scenarios.

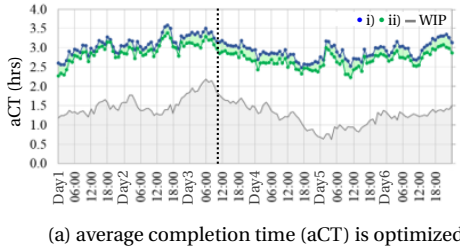
In Figure 5.8b the relative machine capacities of the three different optimizations are shown. Again, the results have been normalized with respect to the reference scenario. The results where the machine capacity have been minimized and maximized provide a window of possibilities for any alternative loading scenario. Similar as for the metal analysis, the aCT-optimization results in indirect benefits for the machine capacity (MC).

In Figure 5.8c the reference scenario for one specific WIP snapshot is compared to the optimized scenario where aCT has been minimized. The arbitrary timestamp has been taken (at DAY3, t=10:00). The reference scenario (i) displays the machine loading as found in actual manufacturing. In the optimized scenario (ii) this aCT-value has been minimized giving a relative value of aCT = 89.5%. The machine capacity is reduced indirectly resulting in reduction of 2.7%.

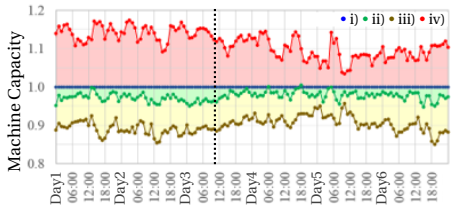
5.8.2 RESULTS REFERENCE SCENARIO WITH CUT-OFF VALUE OF 8 HOURS

In the second optimization analysis for lithography, we eliminated jobs from the reference scenario when their completion time was above a certain cut-off value. In Figure 5.9 the optimization results are presented for a 6-day period using a cut-off value of 8 hours. The graph layout is the same as in Figure 5.8.

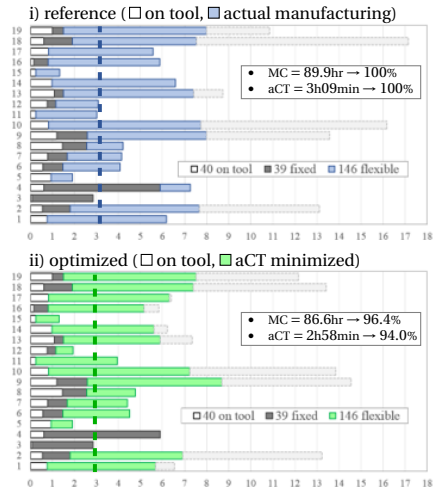
In Figure 5.9a the aCT-value is depicted on a scale of 0-4 hours, as well as the WIP level in the different snapshots. As expected the best results for average completion time are found for the optimized scenario which minimizes aCT. Note that, the gain in average completion time is smaller than in Figure 5.8. This is understandable since the cut-off reference scenario is better balanced.



(a) average completion time (aCT) is optimized



(b) machine capacity (MC) w.r.t. reference scenario



(c) different scenarios at (DAY3, t = 10:00)

Figure 5.9: The optimization strategy of Figure 5.8 is repeated for the cut-off reference scenario of Figure 5.7. The lots with large completion times (CT > 8 hours) have been removed, resulting in a more balanced reference scenario. Optimizing the average completion time now gives a smaller gain (aCT → 94.0%), whereas the machine gain is improved (MC → 96.4%) in comparison to Figure 5.8. The corresponding aCT-value is displayed as a vertical line.

The relative machine capacities of the three different optimizations are shown in Figure 5.9b, where the results have been normalized with respect to the cut-off reference scenario. The results show the same trend as Figure 5.8b, but with an enlarged range. The reason for this is that in the original reference scenario the *fast tools* were loaded more than the *slow tools*. This is a direct consequence of the fact that the fast tools are the preferred tools for qualifying new jobs, resulting in relative long queuing times in front of these tools. Therefore, we expect that the cut-off strategy will give a better estimate for the possible gain in machine capacity.

Finally, in Figure 5.9c the reference and optimized scenarios are compared for the same timestamp as in Figure 5.8c. The reference scenario (i) displays the machine loading as

found in actual manufacturing. In the optimized scenario (ii) this aCT-value has been minimized giving a relative value of aCT = 94.0%. The machine capacity is reduced indirectly resulting in a normalized value of MC = 96.4%.

5.8.3 OVERALL RESULTS FOR A 5-MONTH PERIOD

The optimization results for lithography have been collected for all WIP snapshots in a 5-month period. Because of the large WIP-levels in front of the lithography machines, the variation from one snapshot to the next one is relatively small. The comparison of the *static* scenario of the scheduling optimization and the *dynamic* scenario of historical loading again results in error in the estimate for the capacity gain. We expect the contribution of this stochastic error is nullified by averaging the optimization results over many instances, obtaining a realistic estimate for the capacity gain in real manufacturing.

		Optimization Strategy	
		no cut-off (24 tools)	cut-off = 8 hours (24 tools)
	instances	3649	3649
WIP	average	233	174.8
WIP	median	216	169
WIP	max	651	337
aMC	ref	0:27:49	0:27:55
aMC	opt	0:27:20	0:27:22
aMC	gain	98.26%	98.03%
aCT	ref	5:26:17	3:09:53
aCT	opt	4:52:24	2:57:22
aCT	gain	89.61%	93.98%
	Av. conflicts	1.36	1.59
	calc. time	25.82 sec	3.26 sec

Table 5.2: Optimization results for lithography obtained from a 5-month period with 3676 instances. Only snapshots have been taken into account which contain more jobs than machines ($n > 24$).

The overall statistical results have been brought together in Table 5.2. These results suggest that the tool-speed differences can be exploited in the bottleneck group in lithography. If the transport and dispatching logistics can be aligned with this scheduling optimization, it may be possible to free up almost 2.0% of machine capacity for the toolset which is considered the major bottleneck of the fab. Furthermore, the schedule has on average less than two reticle conflicts and these occur at the end of the schedule. We therefore thinks these will not play any part when scheduling with the the continuously changing WIP in the actual fab.

5.9 CONCLUSION AND FUTURE WORK

The proposed two stage algorithm uses the newly available accurate process times for all possible tool-recipe combinations to improve the overall throughput in the metalization and photolithography bays. The algorithm focuses on minimizing the average completion time of the jobs, but it also allows the logistic manager to influence the dispatching order of the lots assigned to a specific machine. In the pilot studies, we found improvement opportunities that suggest a reduced lead time as expressed by the average completion time (aCT) as well as an overall raised equipment throughput indicated by the reduced machine capacity (MC). For the metalization tools the expected benefits are an aCT-reduction of about 7.8% and a capacity gain of ΔMC = 1.67%, whereas for photolithography we find an average aCT-benefit of 6.02% and a capacity gain of ΔMC = 1.97%. These results are obtained considering static WIP and therefore the reductions might defer in practice. However, since we averaged over many instances we expect the implementation of the proposed scheduling optimization to yield similar results and contribute to a raise of the overall fab efficiency.

In order to deploy the optimization algorithm in the daily fab operation, it needs to be integrated with the current IT-architecture. A graphical user interface (GUI) or dashboard needs to be created to inform the operators about the scheduling decisions. The GUI needs to be up-to-date with the latest information, since it needs to cope with continuously evolving WIP-levels and changing tool statuses. The calculation time required to perform the scheduling optimization is dominated by the LP solver, which needs $O(n^{3.5})$ operations, where n is the number of jobs in the WIP. Therefore, one might want to reduce the number of jobs in the optimization input, much like we did with the cut-off strategy in photolithography. To create a reduced input file one can take into account the job priorities and a minimum amount of candidate jobs for each available machine.

For the operators to work efficiently, the proposed job-machine allocations should not change for the jobs that are being transported for imminent dispatching. This can be achieved by increasing the time interval between optimization calculations for modified WIP levels. Alternatively, one could also fix the lot-tool assignment for jobs planned for imminent dispatching. In such a strategy, the first part of the proposed load distribution should become fixed once the lots have been put on transport to their assigned tool.

REFERENCES

[1] T. Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. `http://mpc.zib.de/index.php/MPC/article/view/4`.

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

[3] J. Bruno, E. G. Coffman, Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Commun. ACM*, 17(7):382–387, 1974. ISSN 0001-0782.

[4] E. Cakici and S. Mason. Parallel machine scheduling subject to auxiliary resource constraints. *Production Planning and Control*, 18(3):217–225, 2007.

[5] R. M. Dabbas and J. W. Fowler. A new scheduling approach using combined dispatching criteria in wafer fabs. *IEEE Transactions on semiconductor manufacturing*, 16(3):501–510, 2003.

[6] D. Doleschal, G. Weigert, A. Klemmt, and F. Lehmann. Advanced secondary resource control in semiconductor lithography areas: From theory to practice. In *Simulation Conference (WSC), 2013 Winter*, pages 3879–3890. IEEE, 2013.

[7] J. Driessen. An OEE increase of 10 percent on litho equipment. In *12th European Advanced Process Control and Manufacturing Conference*. APC|M, 2012.

[8] J. Driessen, R. Sjardijn, F. van Heukelom, C. van Roest, and M. Mom. Wait-time-waste improvement opportunities and 'smart manufacturing'in legacy 200mm fabs. In *Advanced Semiconductor Manufacturing Conference (ASMC), 2016 27th Annual SEMI*, pages 440–445. IEEE, 2016.

[9] C. D. Geiger, R. Uzsoy, and H. Aytuğ. Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *Journal of Scheduling*, 9(1): 7–34, 2006.

[10] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[11] A. M. Ham and M. Cho. A practical two-phase approach to scheduling of photolithography production. *IEEE Transactions on Semiconductor Manufacturing*, 28 (3):367–373, 2015.

[12] M. L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2008.

[13] S. C. Sarin, A. Varadarajan, and L. Wang. A survey of dispatching rules for operational control in wafer fabrication. *Production Planning and Control*, 22(1):4–24, 2011.

6

PARALLEL MACHINE SCHEDULING WITH A SINGLE RESOURCE PER JOB

In this chapter, we study a variant of the problem of scheduling jobs on parallel machines with the objective to minimize the total completion time, i.e. the sum of the completion times of all jobs. In this variant, each job uses exactly one resource, thus partitioning the jobs. There is only one unit of each resource available at any time, so jobs using the same resource cannot be processed simultaneously. Using the classification of Graham et al. [8] as described in Section 1.5, we will denote the problem as $P|partition|\sum_j C_j$.

The problem is motivated by a scheduling problem found in the semiconductor industry as seen in Chapter 5. As described in Chapter 1 the wafer, which contains the chips, will visit different production bays multiple times during its production cycle. The expensive photolithography pieces of equipment are often the bottleneck of the production line. Hence, the overall performance of the factory can be improved by raising the equipment throughput on these tools (which is achieved by minimizing $\sum_j C_j$). For the photolithography however we also need to ensure the reticle (the resource) is available when a job is processed.

In the scheduling problem $P||\sum_j C_j$, one wants to minimize the total completion time while scheduling on a set of parallel machines. It is well-known from the literature that $P||\sum_j C_j$ is polynomially solvable by using the shortest processing time first (SPT) order on the earliest available machine (Smith [12]). This rule makes sure that every time a machine finishes a job, it will be assigned, from among the jobs waiting, the job with a shortest processing time.

Our problem adds auxiliary resource constraints. Blazewicz et al. [2] describe the resource requirements with the entry res$\lambda\delta\rho$ in the β field of the scheduling problem. The

This chapter is based on the paper that appeared as a preprint on Arxiv [10]

number of different resources is given by $\lambda \in \{\cdot, c_\lambda\}$. If $\lambda = c_\lambda$, the number of resources is given by c_λ. If $\lambda = \cdot$ it is part of the input. The resource capacities are denoted by $\delta = \{\cdot, c_\delta\}$. If $\delta = c_\delta$, there is exactly c_δ of every resource available. If $\delta = \cdot$, the total amount available of a resource is part of the input. The resource requirements per job are denoted by $\rho = \{\cdot, c_\rho\}$. If $\rho = c_\rho$, every job needs exactly c_ρ of a resource it requires. If $\rho = \cdot$, the amount required is part of the input.

The type res·11 implies that per resource type, there is one resource available at any given time and this resource will be used entirely if a job needing this resource is processed. Thus, jobs that share the same resource cannot be processed simultaneously. When only res·11 is in the β field of the scheduling problem, a job can need any number of resources. This does not capture that every job in the lithography bay needs only one resource, the reticle. Therefore, we indicate the problem within this chapter by *partition* in the β field of the scheduling problem. Hence, *partition* is a special case of res $\cdot$ 11 resources.
We know from Blazewicz et al. [2] that if the number of machines is $m \geq 3$ and there are no further restrictions on the $P|\text{res} \cdot 11, p_j = 1|\sum_j C_j$, the problem is $\mathcal{NP}$-hard. The proof is based on a reduction from partition into triangles and uses multiple resources per job. It is proven by Garey and Johnson [6] that $P2|\text{res} \cdot 11, p_j = 1|\sum_j C_j$ can be solved in polynomial time by a reduction to the matching problem.

The problem can also be viewed as a special case of $PD|\text{res}\,1 \cdot 1|\sum_j C_j$. In $PD|\text{res}\,1 \cdot 1|\sum_j C_j$, we have dedicated machines and are given only one resource of a certain quantity c_δ and every job needs exactly 1 from that resource. We can rewrite $P|\textit{partition}|\sum_j C_j$ to $PD|\text{res}\,1 \cdot 1|\sum_j C_j$ by taking c_δ equal to the number of machines and introducing a dedicated machine for every resource, such that all jobs that share a resource have to be processed on the same machine. One could also view $P|\textit{partition}|\sum_j C_j$ as a special version of scheduling with conflicts. In scheduling with conflicts, jobs cannot be processed at the same time if they share an edge in the conflict graph $G = (J, E)$ with an edge between two jobs if they share the same resource. In our problem, G is a collection of cliques.

Our contribution is as follows. First we prove that allowing preemptions to the problem, does not change the problem. Using that, we conclude that in each optimal schedule for $P|\textit{partition}|\sum_j C_j$, all jobs sharing the same resource must be processed in non-decreasing order of their processing time. If we restrict the problem to $p_j = 1$, we can proof the problem is polynomially solvable. Thereafter we look at an approximation algorithm for $P|\textit{partition}|\sum_j C_j$, based on a variant of the shortest processing time (SPT) rule that takes the partition constraints into account. We prove that it gives a $\left(2 - \frac{1}{m}\right)$-approximation and show that it cannot give an α-approximation with $\alpha < \frac{4}{3}$. In the last three sections we look at three related problems and show that they are $\mathcal{NP}$-hard. The first problem has additional processing set restrictions for resources, meaning each resource has a set $\mathcal{M}_r$ of machines on which it can be used. From this, we can also conclude that the problem with unrelated machines, i.e. $R|\textit{partition}|\sum_j C_j$, is also $\mathcal{NP}$-hard. This is the situation of the scheduling of photolithography machines in practice. The second related problem assumes that resources are unmovable, meaning that once a resource is used on a machine, it can thereafter only be used on that specific machine.

In the last related problem, each job has at most q resources with $q \geq 2$ a constant.

6.1 DEFINITIONS

Before we begin our analysis of the problem and its solutions, we first formally define *partition* as part of the β field and some intermediary concepts. We have m identical machines and let J be the set of jobs that are to be scheduled. Each $j \in J$ has a *processing time* p_j and each machine can only process a single job at a time. We will denote C_j as the *completion time* of job j in a feasible schedule for an instance. We want to minimize the sum of the completion times (total completion time).

Definition 3. *If* partition *is in the β field, there is a partition R of J, i.e., there is a collection of subsets $R = \{r^1, \ldots, r^s\}$ with $r^k \subseteq J$, where every job is contained in exactly one of the subsets. If $j, j' \in r^k$, j and j' cannot be processed at the same time. Furthermore, we want to define which resource is used by which job. Let $r_j = \{r^k \in R \mid j \in r^k\}$, i.e., the subset that contains job j. If two jobs share the same resource, we will denote this by $r_j = r_{j'}$.*

When we look at a job, we will often consider the other jobs that share the same resource. We will, therefore, introduce the concept of slack, which, intuitively, is the amount of time before and after the job that its resource is not being used.

Definition 4. *A job j has* positive slack $d^+ \geq 0$, *which is the largest non negative number, such that in a given schedule all jobs $j' \in J$ which have the same resource and start after job j, start at least d^+ time units after j finishes. More formally, we define d^+ as*

$$d^+ := \min\{C_{j'} - p_{j'} - C_j \mid j' \in J \text{ satisfies } r_{j'} = r_j \text{ and } C_{j'} > C_j\}$$

where we define $+\infty$ as the minimum over the empty set.
Similarly, a job j has negative slack $d^- \leq 0$, *which is the largest non negative number, such that all jobs $j' \in J$ which have the same resource and start before job j, finish at least d^- time units before j starts. More formally,*

$$d^- := \min\{C_j - p_j - C_{j'} \mid j' \in J \text{ satisfies } r_{j'} = r_j \text{ and } C_{j'} < C_j\}$$

where we define $+\infty$ as the minimum over the empty set.
The slack $d > 0$ *of a job j, is defined as $d = \min\{d^+, d^-\}$.*

We defined the slack of a job by considering all jobs that share the same resource, but often we are only interested in the last job before and the first job after a job j that use the same resource. We therefore introduce the following concept.

Definition 5. *Let d^+ be the positive slack of j, we call a job pair (j, j') a* blocking pair *if $r_j = r_{j'}$ and $C_j = C_{j'} - p_{j'} - d^+$. Thus, j' is the first job to start after C_j that uses the same resource as job j. A blocking pair (j, j') is* tight *if $d^+ = 0$.*

Given a tight pair where the two jobs are not on the same machine, it can be advantageous to construct a schedule where they actually are on the same machine. We will call this operation 'untangling'. Before we define it properly however we need to first define a job's suffix.

Definition 6. *Let job j be processed on machine i. The suffix of job j, $\mathcal{S}(j)$, is the set of jobs $j' \in J$ that are also processed on machine i with $C_{j'} > C_j$.*

Definition 7. *Given a tight pair (j, j'), let i be the machine on which j is processed and i' be the machine on which j' is processed.* Untangling (j, j') *is the operation that changes the schedule by swapping suffices between the machines i and i', i.e., we move j' and $S(j')$ to machine i and $S(j)$ to machine i'.*

Since we work with parallel machines, untangling will not change any of the start or completion times of the jobs. Hence, it will not create any resource conflicts and the objective function remains the same.

6.2 PROBLEM PROPERTIES

In this section, we consider the structure of optimal solutions. We show that there is no non-trivial idle time in an optimal solution and that given a resource, the jobs using that resource are scheduled from shortest to longest. We will continue by looking at the complexity of the problem. Whether or not $P|partition|\sum_j C_j$ is $\mathcal{NP}$-hard remains an open problem, but we can show that when $p_j = 1$ the problem is polynomially solvable. We also show that the problem with preemptions is equal to the problem without preemptions.

We first note that if $|R| < m$ the problem becomes trivial. In that case, one will put all jobs which use the same resource in shortest processing time order on one machine. We continue by looking at idle times in a solution.

Lemma 6.1. *For every instance of $P|partition|\sum_j C_j$ there exists an optimal solution that contains no idle times.*

Proof. Suppose that we have an optimal schedule with idle times, we begin by untangling all tight pairs. If an idle time remains, we consider the last idle time, which appears on machine i that starts on time t_1 and ends at time t_2. Since we have untangled all tight pairs, the resource used by the job on machine i starting at time t_2 is not used until time t_2. Hence, we can start the processing of this job earlier. We can then schedule the job either at time t_1 or at the last time before t_2 its resource was used. This would reduce the completion time of this job. Therefore, after untangling, there cannot be any idle times. Since untangling does not change completion times there is an optimal schedule without idle times. □

In the proof, we saw it is easy to turn an arbitrary schedule into a schedule where tight pairs (j, j') are processed on the same machine. We call such a schedule a *tight* schedule.

Definition 8. *A* tight schedule *is a schedule without any idle time and in which each tight blocking pair (j, j') is executed on the same machine.*

Notice that untangling results in jobs using the same resource being processed one after another on the same machine. We will call these job sequences *trains*.

Definition 9. *A* train sequence $T(j_1)$ *in a schedule is a maximal sequence of consecutively jobs* $j_1, j_2, ..., j_c$ *on the same machine using the same resource.*

Notice that a tight schedule only consists of train sequences $T(j_k)$ with nonzero slack between the train sequences of the same resource, where the j_k are the first jobs to be scheduled when a machine changes resource.

We continue by looking at the order in which jobs that use the same resource are processed. We will prove that this is from shortest to longest processing times. We will prove this by first looking at the problem with preemptions, denoted by $P|partition, prmp|\sum_j C_j$. In a preemptive schedule, the total amount of processing done on the job needs to be equal to its processing time (p_j), but jobs can be interrupted at any time and the processing done is not lost. A job can thus be split into multiple parts, possibly processed on different machines. We begin by defining these more precisely.

Definition 10. *A* job part j^l *is the* l*th maximal part of the job* j *that is processed without interruption on a single machine with positive length. The superscript* l *will be omitted when it is of no importance. A pair of job parts* (j, j') *is called a* blocking pair *if* $r_j = r_{j'}$ *and* j' *is the first job part to start after* j *that uses the same resource as job part* j*. A blocking pair* (j, j') *is* tight *if job part* j' *starts at the time* j *ends, i.e.,* $d^+ = 0$.

The definitions for blocking pair, slack, suffix, train sequence, tight schedule and untangling can easily be extended to the case of job parts.

Lemma 6.2. *In an optimal schedule for* $P|partition, prmp|\sum_j C_j$ *all jobs sharing the same resource must be processed in SPT-order, i.e., if job* j *and* j' *both use resource* $r \in R$ *and* $p_j < p_{j'}$ *then* $C_j < C_{j'}$*. Furthermore, if* $C_j < C_{j'}$*, all job parts of* j *will be processed before any job parts of* j'.

Proof. Suppose we have an optimal schedule S where this is not the case. Then there is a resource $r \in R$ and two jobs using this resource ($r_j = r_{j'}$), job j and job j', with $C_j < C_{j'}$ and $p_j > p_{j'}$. From S we get an ordering of the jobs using resource r. Let $j_{S(r,p)}$ denote the lth job finishing in S using resource r.
Create a new schedule S' which is identical to S except for all jobs using resource r. We remove from S all job parts using resource r. This will remove $t = \sum_{j \in J | r_j = r} p_j$ units of processing from the schedule. We fill these units of processing again with the jobs using resource r but now we process them in an SPT-order. We process the first job in the ordering $j_{S'(r,1)}$ in the first $p_{j_{S'(r,1)}}$ units of t. We schedule the second job in the ordering $j_{S'(r,2)}$ in the first $p_{j_{S'(r,2)}}$ units of t after $C_{j_{S'(r,1)}}$ and so on until all jobs using resource r are scheduled. Resource r will be used in the same time as in S by only a single job, hence S' is a feasible schedule. Furthermore, it holds that $C_{j_{S(r,1)}} \leq C_{j_{S(r,p)}} \forall p$, since

$$p_{j_{S'(r,1)}} + p_{j_{S'(r,2)}} + \ldots + p_{j_{S'(r,p)}} \leq p_{j_{S(r,1)}} + p_{j_{S(r,2)}} + \ldots + p_{j_{S(r,p)}}. \tag{6.1}$$

Since $p_j > p_{j'}$, equation (6.1) is satisfied with inequality for the lth job and S cannot be optimal. □

Theorem 6.3. *There is an optimal schedule for $P|partition, prmp|\sum_j C_j$ without any preemptions.*

Proof. Assume not, then take any optimal tight schedule with a minimal amount of preemptions. Let t_1 be the time of the last occurring preemption, let job j be the job that is being interrupted with resource r on machine i, let j^l be the respective job part. Let t_2 be the time that job part j^{l+1} starts on machine i'. Let j' be the job part on machine i that starts at t_1 and let r' be its resource.
We know that $t_2 > t_1$, otherwise we would not have a tight schedule. Furthermore, following from Lemma 6.2, resource r cannot be used by another job between t_1 and t_2.
We also know that $i \neq i'$ by the following argument illustrated in Figure 6.1. Assume $i = i'$. Take $\epsilon > 0$ as the minimal negative slack of all train sequences between t_1 and t_2 on machine i. Then, one can move all jobs on machine i between t_1 and t_2 ϵ to the front and then split j^l on $t_1 - \epsilon$ and move the second part from $[t_1 - \epsilon, t_1]$ to $[t_2 - \epsilon, t_2]$. Since there is no preemption after t_1, at least one job finishes earlier in this new situation and no jobs is completed later. Thus, the original schedule was not optimal.

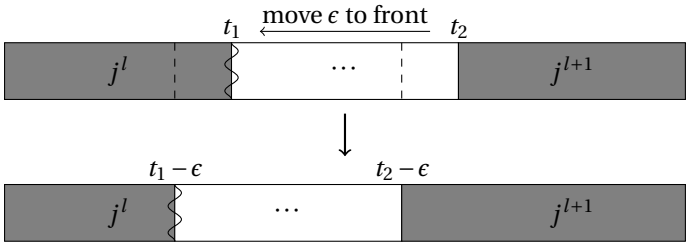


Figure 6.1: Situation where $i = i'$. The squiggly line represents a preemption.

We know that job j' cannot end before or on t_2. As illustrated in Figure 6.2, if it would, one could move job j' $\epsilon > 0$ to the front, where ϵ is the negative slack of job j'. This would split the job part j^l on $t_1 - \epsilon$ and moving the part that was executed during the interval $[t_1 - \epsilon, t_1]$ to the back of j'. This leads to a feasible schedule, since we defined t_2 as the

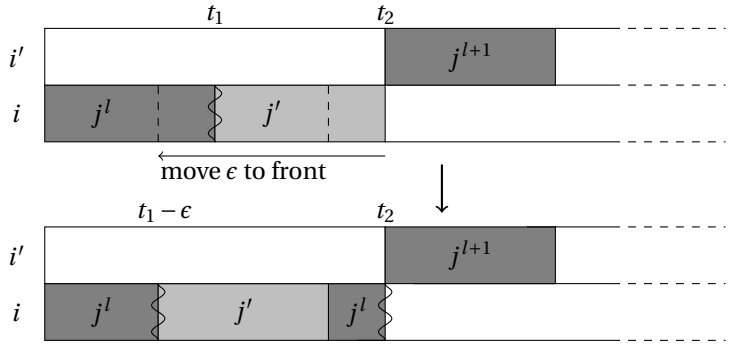


Figure 6.2: Finding a better solution if $C_{j'} \leq t_2$

time that job part j^{l+1} starts and resource r is not used between t_1 and t_2. Furthermore, since j' is a finishing job part and it finishes ϵ earlier, this will lead to a schedule with better objective value.

As a result, there is only one possible situation that can occur with the last preemption: The last preemption is at a different machine than where it later continues and job j' starts at t_1 on machine i and does not finish before or at t_2. The partial schedule is shown in Figure 6.3.

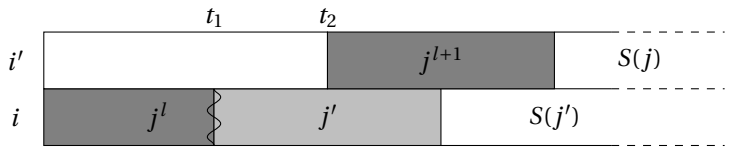


Figure 6.3: Only possible situation in an optimal schedule with preemption.

Let $S(j)$ be job part j^{l+1} on machine i' and its suffix and let $S(j')$ be job part j' on machine i and its suffix. The jobs in these sets all finish, as j^l is the last preemption. When we look at the number of jobs in both sets, there are two possibilities:

- $|S(j)| < |S(j')|$. Figure 6.4 illustrates this case. Take a maximal $\epsilon > 0$ such that all train sequences in $S(j)$ can start ϵ later (i.e. have $d^+ \geq \epsilon$) and all train sequences in $S(j')$ can start ϵ earlier (i.e. have $d^- \geq \epsilon$), while staying in a feasible schedule. Move the sets in the mentioned directions and move the interval $[t_1 - \epsilon, t_1]$ of job j on machine i to machine i' in the interval $[t_2, t_2 + \epsilon]$. Also, move j^{l+1} ϵ to the back and j' ϵ to the front. Clearly, this is a feasible schedule. All job parts in the sets $S(j)$ and $S(j')$ have no preemptions, thus the objective value changes by $\epsilon(|S(j)| - |S(j')|)$, and therefore becomes smaller. Hence, this situation cannot happen in an optimal solution.

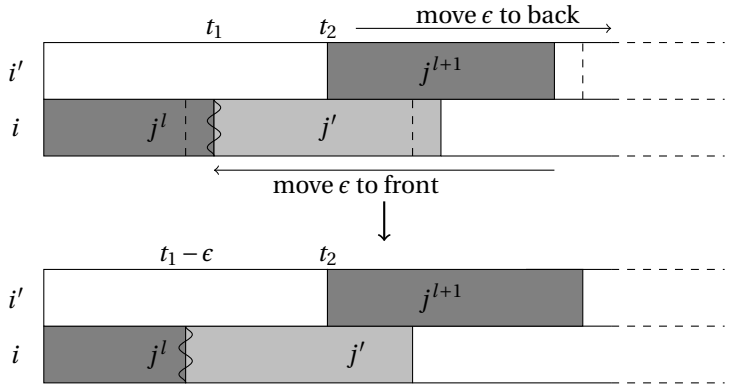


Figure 6.4: Finding a better solution if $|S(j)| < |S(j')|$

- $|S(j)| \geq |S(j')|$. Figure 6.5 illustrates this case. Take a maximal $\epsilon > 0$ such that all train sequences in $S(j)$ can start ϵ earlier (i.e. have $d^- \geq \epsilon$) and all train sequences in $S(j')$

can start ϵ later (i.e. have $d^+ \geq \epsilon$), while staying a feasible schedule. Move the sets in the mentioned directions and move the interval $[t_2, t_2+\epsilon]$ of job j on machine i' to machine i on the interval $[t_1, t_1+\epsilon]$. Also move job j^{l+1} to the front and j' to the back. Clearly, this is a feasible schedule. All job parts in the sets $S(j)$ and $S(j')$ have no preemptions, thus the objective value changes by $\epsilon(|S(j')| - |S(j)|)$. Hence, $|S(j)| > |S(j')|$ cannot happen in an optimal solution.

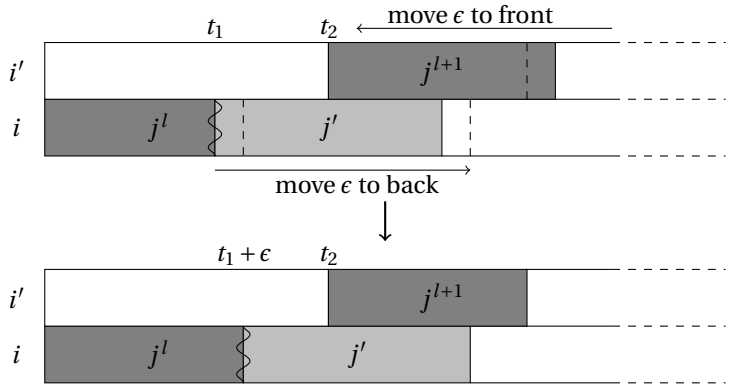


Figure 6.5: Finding a better solution if $|S(j)| > |S(j')|$

Therefore, in an optimal solution, $|S(j)| = |S(j')|$. Since the ϵ was chosen maximal in the $|S(j)| \geq |S(j')|$ case, there must be a new tight pair, created by moving the suffices backward and forward. Untangle new tight pairs, such that the schedule becomes tight again. In the new schedule, it must hold that again $|S(j)| = |S(j')|$, because otherwise the solution was not optimal. It is possible that (one of the) newly tight pairs was (j^l, j^{l+1}), in that case a preemption was removed contradicting the assumption that the number of preemptions was minimal. If not, keep repeating moving the suffices and job parts as described and untangling. This can be done only a finite number of times since there is a maximum number of tight pairs (bounded by n) and because during this process, tight pairs remain tight and no new job parts are created (and with it new preemptions). Hence, there is an optimal schedule containing no preemptions. □

Following from Theorem 6.3 and Lemma 6.2, we know that $P|partition, prmp|\sum_j C_j$ and $P|partition|\sum_j C_j$ have the same objective value and we obtain the following result.

Theorem 6.4. *In an optimal schedule for* $P|partition|\sum_j C_j$*, all jobs sharing the same resource are processed in SPT-order, i.e., if job* j *and* j' *both use resource* $r \in R$ *and* $p_j < p_{j'}$ *then* $C_j < C_{j'}$*.*

We continue by looking at $p_j = 1$, since $P3|res.11, p_j = 1|\sum_j C_j$ is $\mathcal{NP}$-hard. Surprisingly, with at most one resource per job, the problem becomes polynomially solvable.

Theorem 6.5. *$P|partition, p_j = 1|\sum_j C_j$ is polynomially solvable.*

Proof. The problem can be reduced to an instance of the Min-Cost Flow problem. Next to the source node s and the target node t, we construct three sets of nodes V_J, $V_{\text{res,pos}}$, $V'_{\text{res,pos}}$ and $V_{M,\text{pos}}$. The set V_J corresponds to the jobs. It has n nodes; one for every job. The set $V_{\text{res,pos}}$ corresponds to the resource needed and the completion time/position of the job on a machine in the schedule. The completion time and position on a machine are in this case equal since $p_j = 1$ and we may assume no idle times from Lemma 6.1. The set $V_{\text{res,pos}}$ has $n|R|$ nodes. The set $V'_{\text{res,pos}}$ is a duplicate of these nodes. The set $V_{\text{M,pos}}$ corresponds to the used machine and the completion time/position of the job on the machine in the schedule. It has mn nodes.
We start by constructing arcs with cost 0 and capacity 1. The first set of arcs is constructed from s to every node in V_J. From every node $v_j \in V_J$, we construct an arc to every node in $V_{\text{res,pos}}$ that corresponds to the resource needed by job j. We construct from every node $v_{r,p} \in V_{\text{res,pos}}$ an arc to the corresponding node with the same resource and position $v'_{r,p} \in V'_{\text{res,pos}}$. Next, we construct from every node $v'_{r,p} \in V'_{\text{res,pos}}$ an arc to every node in $V_{M,\text{pos}}$ having the same position p. Lastly, we construct arcs with capacity 1 and cost equal to position p from every node $v_{m,p} \in V_{M,\text{pos}}$ to t. We thus have $n(1+n+|R|+|R|m+m)$ arcs. Lastly, we require that we have at least n units of flow from s to t.

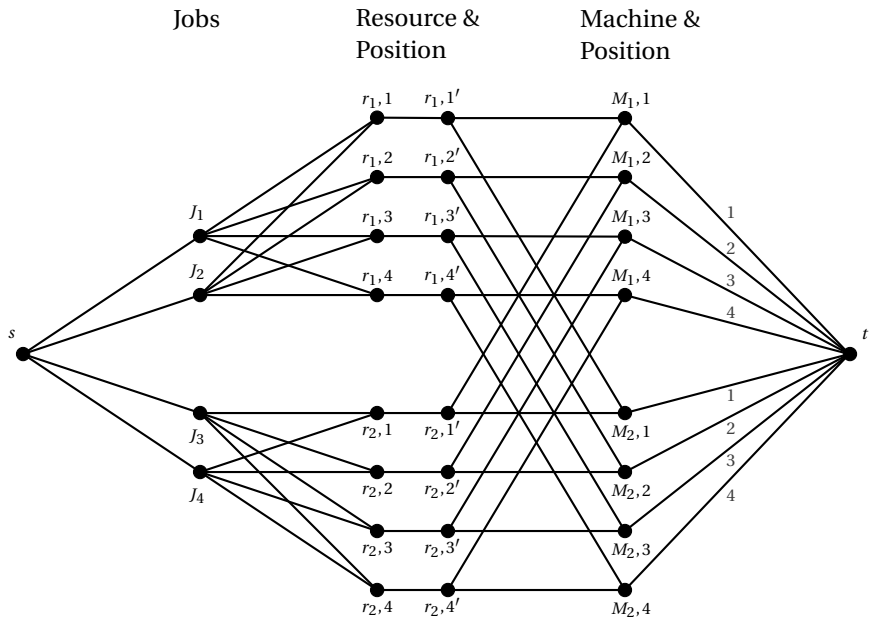


Figure 6.6: Min Cost-Flow instance for $P|partition, p_j = 1|\sum_j C_j$ with 4 jobs and 2 resources.

Suppose we have an instance of $P|partition, p_j = 1|\sum_j C_j$ with objective value c and consider its corresponding Min-Cost Flow instance. We put one unit of flow in the network

for every job in the schedule of $P|partition, p_j = 1|\sum_j C_j$ corresponding to the job, its position, which machine and which resource used. For example, for a job j on position p machine M using resource r, we put one unit of flow from s to t through nodes v_J, $v_{r,p}$, $v'_{r,p}$ and M, p. Since no jobs share the same machine or resource at a given position, every node $v \in V \setminus \{s, t\}$ will have at most one unit of flow going in and going out. Thus, we have a feasible flow. Furthermore, we only put flow on edges from $v_{m,p} \in V_{M,\text{pos}}$ to t, if and only if we also have a job with corresponding machine m and position p, thus we have a flow of cost c. The reverse is also true by this construction and hence we have an objective value of c for $P|partition, p_j = 1|\sum_j C_j$ if and only if we have an objective value of c for its corresponding Min-Cost Flow instance. □

Note that, the above proof can easily be adjusted to the problem where one has more than one unit of a resource available, by setting the capacities of the arcs between $V_{\text{res,pos}}$ and $V'_{\text{res,pos}}$ equal to the amount of resources one has. Furthermore, notice that one could put the costs on a different set of edges. In this way, one can also show that $P|partition, p_j = 1|\sum_j w_j C_j$ is polynomially solvable by setting the cost of the edges between v_J and $v_{r,p}$ equal to w_j times the position.

Since we know that $P|partition, p_j = 1|\sum_j C_j$ is polynomially solvable, one might wonder what happens when the processing time of each job is bounded, i.e. $1 \leq p_j \leq c$, where c is a constant. A simple Shrinking algorithm would be to create an instance of $P|partition, p_j = 1|\sum_j C_j$ by setting all processing times in our original problem to one. We then solve this problem using the construction in Theorem 6.5 to obtain an optimal schedule $S^{\text{OPT}}_{p_j=1}$. We can then construct a feasible schedule for $P|partition, 1 \leq p_j \leq c|\sum_j C_j$ in the following way: All jobs in $S^{\text{OPT}}_{p_j=1}$ start at a given integer $s_j \in \mathbb{Z}^+$, since $p_j = 1$. We can create a feasible solution S_{ALG} from $S^{\text{OPT}}_{p_j=1}$ by setting the starting time for all jobs to cs_j. In this way, in every time interval $ci \leq t \leq c(i+1)$ with $i \in \mathbb{Z}^+$, every machine will only work on one product. There also will not be any resource conflicts, since there were none in $S^{\text{OPT}}_{p_j=1}$ in the corresponding interval $i \leq t \leq (i+1)$.

Proposition 6.6. *The Shrinking algorithm gives a c-approximation for $P|partition, 1 \leq p_j \leq c|\sum_j C_j$.*

Proof. Let OPT denote the optimal value for $P|partition, 1 \leq p_j \leq c|\sum_j C_j$ and S^{OPT} the optimal schedule and $\text{OPT}_{p_j=1}$ denote the optimal value for $P|partition, p_j = 1|\sum_j C_j$. The Schrinking algorithm gives a feasible solution of cost $c\text{OPT}_{p_j=1}$, so it suffices to show that $\text{OPT} \geq \text{OPT}_{p_j=1}$. Using S^{OPT}, we can find a schedule $S_{p_j=1}$ for our constructed instance of $P|partition, p_j = 1|\sum_j C_j$. This is done by scheduling all jobs 1 time unit before there completion time in S^{OPT}. Thus all completion times in $S_{p_j=1}$ will be the same as in S^{OPT}. Furthermore, since $1 \leq p_j \leq c$ in S^{OPT}, there will be no resource conflict in $S_{p_j=1}$, since there were none in S^{OPT}. □

Note that one can remove all idle time in S_{ALG} by using the untangling operation described in the proof of Lemma 6.1.

6.3 SHORTEST PROCESSING TIME FIRST

The shortest processing time first (SPT) rule is optimal for a few scheduling problems, one of which is $P||\sum_j C_j$. In this section, we will look at how well the rule performs for $P|partition|\sum_j C_j$. Before we can do this however we need to adjust the rule slightly to cope with the resources.

Definition 11. *The SPT-available rule schedules the jobs according to a list. This list contains all jobs ordered from shortest to largest processing times. At any point in time, when a machine is available for processing. The rule selects the first job in the list for which the resource is not in use. It then removes the job from the list. If multiple machines are available at time t and a job is selected of which the resource was not available just before time t, the algorithm will put this job on the machine that was previously using this resource. Otherwise the rule will choose an arbitrary available machine. No job is added to a machine that is available if the resource is in use of all jobs on the list.*

Because of the way we defined the SPT-*available* rule, jobs that share the same resource and that are processed one after the other will be scheduled on the same machine. In other words, the SPT-*available* rule produces a tight schedule. This schedule also has no idle times, since if at time t a job j of resource r is scheduled on machine m which would create an idle time, then it could not be scheduled on that machine earlier due to some other job j' using the same resource on some machine m'. Job j can only be scheduled at time t since j' just finished. But the rule states that job j then has a preference for machine m' over m (creating a tight schedule).
The SPT-rule is optimal for $P||\sum_j C_j$. Hence, one might wonder whether this is also the case with the SPT-*available* rule for $P|partition|\sum_j C_j$. Example 1 shows that this is not the case.

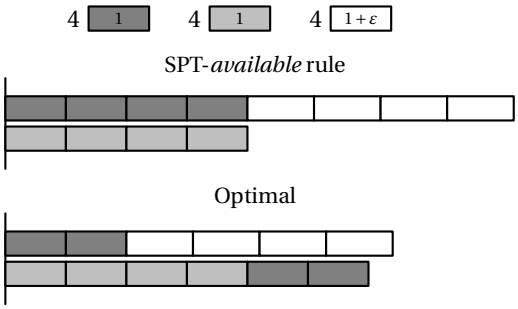


Figure 6.7: Optimal and SPT-available schedule for Example 1

Example 1: (SPT-*available* not optimal)
Consider 2 machines with 12 jobs that use 3 resources. We divide the jobs J into 3 groups depending on the resource used, $J = J_1 \cup J_2 \cup J_3$. We have 4 jobs in $J_1 = \{j_1, \ldots, j_4\}$ with $p_1 = \ldots = p_4 = 1$ using the first resource. We have another 4 jobs in $J_2 = \{j_5, \ldots, j_8\}$ with $p_5 = \ldots = p_8 = 1$ using the second

resource. Lastly, we have 4 jobs in $J_3 = \{j_9, \ldots, j_{12}\}$ with $p_9 = \ldots = p_{12} = 1+\varepsilon$ with $\varepsilon > 0$ using the third resource.

The SPT-*available* rule will schedule the jobs in $J_1 \cup J_2$ first on the two machines and will then at time 4 schedule the jobs in J_3 on one machine one after the other. This will result in an objective value of $46+10\varepsilon$. An optimal schedule would be to schedule first two jobs from J_1 on the first machine and then all jobs from J_3. On the second machine all jobs from J_2 are scheduled first and then the last two jobs from J_1. This results in a schedule with objective value $42+10\varepsilon$. Hence SPT-*available* is not optimal.

The SPT-*available* rule is not optimal for $P|partition|\sum_j C_j$, but it might give a good approximation. Example 1 gives a type of instance that is hard to tackle for the SPT-*available* rule. We generalize this example to find a lower bound on the approximation factor.

Lemma 6.7. *The SPT-available rule does not give an α-approximation for $P|partition|\sum_j C_j$ with $\alpha < \frac{4}{3}$.*

Proof. Consider the instance $\mathcal{I}$ with 3 machines and job set $J = J_A \cup J_B$. J_A consists of $3c$ jobs of length 1, that all use their own resource, with $c \in \mathbb{Z}^+$ even. The set J_B consist of $3c$ jobs of length $1+\varepsilon$ with $\varepsilon > 0$, that all share the same resource, i.e., $r_j = r_{j'}, \forall\, j, j' \in J_B$. The SPT-*available* rule will first schedule c jobs from J_A on every machine. Then, it will schedule at time c all jobs from J_B on one machine. Let $\texttt{ALG}_{\mathcal{I}}$ be the objective value of the SPT-*available* rule for instance $\mathcal{I}$. Then,

$$\begin{aligned}\texttt{ALG}_{\mathcal{I}} &= \tfrac{3}{2}c(c+1) + 3c^2 + \tfrac{3}{2}c(3c+1)(1+\varepsilon)\\ &= 9c^2 + 3c + \tfrac{1}{2}(9c^2+3c)\varepsilon\end{aligned}$$

An optimal schedule would schedule the J_b jobs on a single machine and schedule the J_A jobs evenly on the remaining two machines. Let $\texttt{OPT}_{\mathcal{I}}$ be the objective value of the optimal schedule for instance $\mathcal{I}$. Then,

$$\begin{aligned}\texttt{OPT}_{\mathcal{I}} &= \tfrac{3}{2}c(\tfrac{3}{2}c+1) + \tfrac{3}{2}c(3c+1)(1+\varepsilon)\\ &= \tfrac{27}{4}c^2 + 3c + \tfrac{1}{2}(9c^2+3c)\varepsilon\end{aligned}$$

Thus,

$$\begin{aligned}\frac{\texttt{ALG}_{\mathcal{I}}}{\texttt{OPT}_{\mathcal{I}}} &\geq \lim_{c\to\infty}\lim_{\varepsilon\to 0}\frac{9c^2+3c+\frac{1}{2}(9c^2+3c)\varepsilon}{\frac{27}{4}c^2+3c+\frac{1}{2}(9c^2+3c)\varepsilon}\\ &= \lim_{c\to\infty}\frac{9c^2+3c}{\frac{27}{4}c^2+3c} = \frac{4}{3}\end{aligned}$$

□

We will proceed by giving an upper bound to the approximation ratio by using an approach similar to the approach used by Chekuri et al. [4] to show a 2-approximation for

the problem of minimizing the total weighted completion time on m parallel machines with in-tree precedence constraints. We begin by defining the minimum completion time of every job, based on the fact that by Theorem 6.4 all jobs sharing the same resource must be processed in SPT-order. Without loss of generality we can assume that the jobs are ordered according to their processing times, $j_1 \prec j_2 \prec \cdots \prec j_n$, breaking ties arbitrarily.

Definition 12. *The* minimum completion time k_j *for each job* j *is given by*

$$k_j = p_j + \sum_{j' | r_{j'} = r_j \text{ and } p_{j'} \prec p_j} p_{j'}.$$

Define OPT^m as the optimal value for a given instance of jobs on m machines and let $\text{OPT}^m_{\text{res}}$ be the optimal value for the instance of jobs on m machines with *partition* constraints. Clearly, $\text{OPT}^1 = \text{OPT}^1_{\text{res}}$ for each instance since the additional constraints do not interfere with the optimal schedule for one machine. Notice that the optimal schedule for one machine and for parallel machines is created by the SPT rule [5]. Let C^1_j denote the completion time of job j in an optimal schedule using one machine (with or without *partition* constraints) and C^m_j the completion time of job j in an optimal schedule for m machines without *partition* constraints.

Lemma 6.8. $\frac{1}{m} OPT^1 \leq OPT^m \leq OPT^m_{res}$ *for each instance.*

Proof. Clearly, the last inequality holds, as an optimal schedule for the problem with constraints is always a feasible solution to the problem without the partition constraints. Hence, we only have to show the first inequality.
Sort the jobs from small to large processing times. Let j be an arbitrary but fixed job and let $P_j = \sum_{i=1}^{j} p_i$ be the sum of all processing times of the jobs that have a smaller or equal processing time than j. Clearly $C^m_j \geq \frac{1}{m} P_j$, since $\frac{1}{m} P_j$ is the earliest time j can finish. We also have that $P_j = C^1_j$, since the SPT-rule is optimal. Thus, for every job j we also have that $\frac{1}{m} C^1_j \leq C^m_j$ from which the first inequality follows. □

We can now prove the upper bound for the SPT-*available* rule.

Theorem 6.9. *The SPT-available rule gives a* $\left(2 - \frac{1}{m}\right)$*-approximation for* $P|partition|\sum_j C_j$.

Proof. Let C^G_j be the completion time of job j in a schedule created by the SPT-*available* rule. We begin by proving by induction that

$$C^G_j \leq \left(1 - \frac{1}{m}\right) k_j + \frac{1}{m} C^1_j, \quad \forall j \in J. \tag{6.2}$$

The jobs that are the first to be scheduled on a machine are also the first of their resource. Therefore,

$$\begin{aligned} C_j^G &= p_j \\ &= \left(1 - \frac{1}{m}\right) p_j + \frac{1}{m} p_j \\ &= \left(1 - \frac{1}{m}\right) k_j + \frac{1}{m} C_j^1 \end{aligned}$$

in that case.
Now assume that Equation (6.2) holds for any job $j' \prec j$. Then, in particular, it also holds for the job j' that is scheduled right before job j on the same machine. We distinguish the following two cases:

- Job j' *and job* j *use the same resource.* Job j is scheduled right after j' and thus

$$\begin{aligned} C_j^G &= C_{j'}^G + p_j \\ &\leq \left(1 - \frac{1}{m}\right) k_{j'} + \frac{1}{m} C_{j'}^1 + p_j && \text{(by induction)} \\ &= \left(1 - \frac{1}{m}\right) (k_{j'} + p_j) + \frac{1}{m} (C_{j'}^1 + p_j) \\ &\leq \left(1 - \frac{1}{m}\right) k_j + \frac{1}{m} C_j^1. \end{aligned}$$

- Job j' *and job* j *use different resources.* This implies that j was scheduled at the first possible free machine and not at a later possibility because of the resource constraints. Define job j'' as the job that is right before j in the schedule for one machine, i.e. the last j'' in the ordering such that $j'' \prec j$. For the starting time of job j (equal to $C_{j'}^G$) it then holds that $C_{j'}^G \leq \frac{1}{m} C_{j''}^1$. Using this we see that:

$$\begin{aligned} C_j^G &= C_{j'}^G + p_j \\ &\leq \frac{1}{m} C_{j''}^1 + p_j \\ &= \left(1 - \frac{1}{m}\right) p_j + \frac{1}{m} (C_{j''}^1 + p_j) \\ &\leq \left(1 - \frac{1}{m}\right) k_j + \frac{1}{m} C_j^1 && \text{(since } p_j \leq k_j \, \forall \, j\text{)}. \end{aligned}$$

Hence, we can conclude that (6.2) holds for all jobs $j \in J$.

Note that $\sum_j k_j \leq OPT^m_{res}$, since $k_j \leq C_j$ in any feasible schedule for $P|partition|\sum_j C_j$ by the definition of the minimal completion time k_j. Using equation (6.2) we get:

$$\begin{aligned}
\sum_j C^G_j &\leq \sum_j \left(1-\frac{1}{m}\right)k_j + \sum_j \frac{1}{m}C^1_j && \text{(using (6.2))}\\
&= \left(1-\frac{1}{m}\right)\sum_j k_j + \frac{1}{m}\sum_j C^1_j \\
&\leq \left(1-\frac{1}{m}\right)OPT^m_{\text{res}} + OPT^m_{\text{res}} && \text{(Lemma 6.8)}\\
&\leq \left(2-\frac{1}{m}\right)OPT^m_{\text{res}}
\end{aligned}$$

□

6.4 MACHINE SUBSET CONSTRAINTS

Since we do not know the complexity of $P|partition|\sum_j C_j$, it is interesting to look at related problems. We will look at several of these related problems. We begin by considering the problem where jobs that share the same resource can only be processed on a subset of the machines.

We can add processing set restrictions by adding $\mathcal{M}_j$ to the β field of a scheduling problem, as found in [11]. This means that for each job j, there is a set $\mathcal{M}_j \subseteq \{1,...,m\}$ such that j can only be scheduled on machines in $\mathcal{M}_j$. Let us define a variation called processing set restrictions for resources as follows: For each resource $r \in \mathcal{R}$ there is a set $\mathcal{M}_r \subseteq \{1,...,m\}$ such that all jobs sharing resource r can only be scheduled on machines in $\mathcal{M}_r$. We denote these restrictions as $\mathcal{M}_r$ in the β field.

Corollary 6.10. *$P|partition, \mathcal{M}_r, p_j = 1|\sum_j C_j$ is polynomially solvable.*

This is a consequence of Theorem 6.5. One could use the same algorithm, but only include edges $v'_{r,p}$ to $v_{i,p}$ if $i \in \mathcal{M}_r$.

Consider the following NP-complete problem from [7].

Definition 13. *3-PARTITION Given positive integers m and b, and a multiset of $3m$ integers A with $\sum_{a\in A} a = mb$, and $b/4 \leq a \leq b/2$ for all $a \in A$, does there exist a partition $(A_1,...,A_m)$ of A into 3 element sets such that for each i, $1 \leq i \leq m$, $\sum_{a\in A_i} a = b$?*

This problem is $\mathcal{NP}$-hard in the strong sense for $m \geq 3$. Using this, we will prove the following theorem.

Theorem 6.11. *$P|partition, \mathcal{M}_r|\sum_j C_j$ is NP hard in the strong sense.*

Proof. Assume we have an instance of 3-PARTITION. Since 3-PARTITION is NP-complete in the strong sense, we may assume that mb is bounded by a polynomial in m, which is

crucial for our proof. Define $N_c = 2mb$ and $C = 8mb$. Create an instance of $P|partition, \mathcal{M}_r|\sum_j C_j$ with $2m$ machines and the following jobs:

- for all $a \in A$ make job a_j, with processing time $p_{a_j} = a$, a unique resource $r(a_j)$ and $\mathcal{M}_{r(a_j)} = \{1, 2, ..., m\}$. These jobs represent the integers that should be partitioned over the first m machines.
- for all $1 \leq i \leq m$ make N_c jobs called 'C'-jobs with processing time C, resource i and $\mathcal{M}_i = \{i, m+i\}$, so for each i, there are N_c jobs with length C, all sharing the same resource, that can only be scheduled on machines i and $m+i$.
- for all $1 \leq i \leq m$ make job r_i, also called a release date job with processing time $p_{r_i} = b$ and resource i, so it shares its resource with a sequence of 'C'-jobs and can only be scheduled on machines i and $m+i$.
- for all $1 \leq i \leq m$ make job D_i, also called a 'D'-job, with processing time $p_{D_i} = N_C^2 C$, resource $r(D_i)$ and $\mathcal{M}_{r(D_i)} = \{m+i\}$, so its resource is unique and can only be scheduled on machine $m+i$.

Define $Z^+ = mb + m\left(N_C b + (C + N_C C)\frac{N_C}{2}\right) + m(b + N_c^2 C) + 2mb$. We will show that the optimal schedule for the scheduling problem has objective value $Z^* \leq Z^+$ if and only if the 3-PARTITION instance is a yes-instance.

Assume that the 3-PARTITION instance is a yes-instance, then the following schedule is a feasible solution: We can find A_i with $|A_i| = 3$ s.t. $\sum_{a \in A_i}^m a = b$ for all i. Schedule each of these A_i at the beginning of one of the first m machines. Process the jobs in non-decreasing order of their processing times per machine. Start the release date jobs r_i at machines $m+i$ at $t = 0$. Process the 'C'-jobs from $t = b$ and onwards at the first m machines. Start each 'D' job D_i at machine $m+i$ at $t = b$. For a visualization of this schedule see Figure 6.8. The objective value of such a solution is equal to

$$Z_{feas} = \underbrace{m \cdot b}_{\text{release date jobs}} + \underbrace{m\left(N_C \cdot b + (C + N_C \cdot C)\frac{N_C}{2}\right)}_{\text{'}C\text{' jobs}} + \underbrace{m\left(b + N_C^2 \cdot C\right)}_{\text{'}D\text{' jobs}} + \underbrace{Z_A}_{a_j \text{ jobs}},$$

with $\frac{7}{4}mb \leq Z_A \leq 2mb$ since the following holds: $\frac{b}{4} \leq a_j \leq \frac{b}{2}$, and the a_j jobs are sorted in non-decreasing order of their processing times per machine, so the worst case scenario is if A_i only has jobs of processing times $\frac{b}{3}$ and the best case scenario is if A_i has jobs of processing times $\frac{b}{4}$, $\frac{b}{4}$ and $\frac{b}{2}$. Since $Z_{feas} \leq Z^+$, we can conclude that $Z^* \leq Z^+$.

We will show that if the 3-PARTITION instance is a no-instance, the optimal schedule has an objective value $Z^* > Z^+$. Let $Z^* = Z_r^* + Z_C^* + Z_D^* + Z_a^*$ with Z_r^*, Z_C^*, Z_D^* and Z_a^* the sum of the completion times of the release date jobs, 'C'-jobs, 'D'-jobs and a_j-jobs respectively. Notice that mb is a lower bound on Z_r^* since the jobs cannot start before $t = 0$. In the same way, $mN_C^2 C$ is a lower bound on Z_D^*. A lower bound on Z_a^* is $\frac{7}{4}mb$, this is because if only the a_j-jobs were to be scheduled on m machines, the SPT-order would be optimal and would have 3 jobs on every machine. Suppose not, then there is a machine i_1 with 4 jobs or more. Then there is another machine i_2 with 2 or less

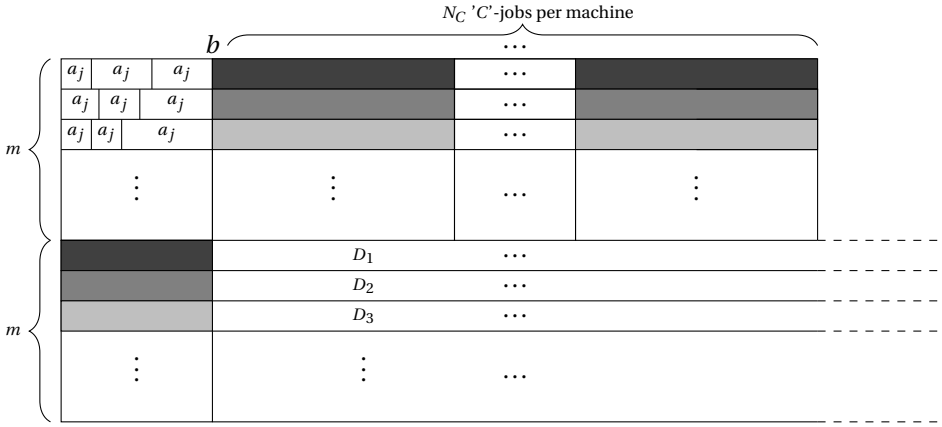


Figure 6.8: Feasible solution in the case of a yes-instance.

jobs. Moving the first job j on machine i_1 to machine i_2 would result in at least 3 jobs finishing p_j earlier and at most 2 jobs finishing p_j later at machine i_2. This leads to a contradiction that SPT is optimal. So we may assume each machine has exactly 3 jobs for finding the lower bound of Z_a^*. Then $\sum_{i=1}^{m}(3a_{i1}+2a_{i2}+a_{i3})$ is minimal if one chooses all a_{i1} and a_{i2} to be $\frac{b}{4}$, i.e. as small as possible. This implies $a_{i3}=\frac{b}{2}$, as $\sum_{a\in A} a = mb$ and $\frac{b}{4} \le a \le \frac{b}{2}$ for all $a \in A$. This leads to a total completion time and therefore a lower bound of $m\left(\frac{b}{4}+\frac{b}{2}+b\right)=\frac{7}{4}mb$ for Z_a^*.

If we have a no-instance, we argue that in the optimal schedule $\exists a_j$ with completion time larger than b. Assume not, then all first m machines are processing a_j jobs until b, since $\sum_{a\in A} a = mb$. If there is a machine processing more than three a_j jobs, it would have to be four a_j jobs of length $\frac{b}{4}$, so that the a_j jobs are all finished before or at b. But then, there is also a machine processing only two a_j jobs of length $\frac{b}{2}$, otherwise another machine would have to finish its a_j jobs after b. Switching a $\frac{b}{2}$ job with two $\frac{b}{4}$ jobs would then result in a smaller objective value. Hence, all machines are processing exactly three a_j jobs with $\sum_{a\in A_i} a = b$ for all $1 \le i \le m$. Then we would find a partition, leading to a contradiction. So there must be an a_j finishing after b. Since $a_j \in \mathbb{N}$ for all j, we can find an a_j with completion time $\ge b+1$. We distinguish 4 cases:

- *At least one 'C'-job is scheduled before the r_i job with the same resource.* Let N_i be the number of 'C'-jobs on machines i and $m+i$ scheduled before the corresponding release job r_i. Then r_i starts at N_iC or later. The lower bound on Z_r^* becomes $mb + C\cdot\sum_{i=1}^{m} N_i$. Then $\sum_{i=1}^{m}\left((N_C-N_i)b+(C+N_CC)\frac{N_C}{2}\right)$ is a lower bound of Z_C^*. Using the other lower bounds for Z_a^* and Z_D^*, we obtain the following lower bound for Z^*:

$$mb + C\cdot\sum_{i=1}^{m} N_i + \sum_{i=1}^{m}\left((N_C-N_i)b+(C+N_CC)\frac{N_C}{2}\right) + mN_C^2C + \frac{7}{4}mb.$$

Then

$$Z^* - Z^+ \geq (C-b)\sum_{i=1}^{m} N_i - mb - \frac{1}{4}mb > 0,$$

using that $C = 8mb$ and $\sum_{i=1}^{m} N_i > 0$.

- *All 'C'-jobs are scheduled after the r_i job with the same resource, but at least one 'C'-job is scheduled on one of the last m machines.* We split this up into two subcases:
  - At least one such 'C'-job is scheduled *before* a 'D'-job on the same machine. We know all 'C'-jobs start after b, hence $Z_C^* \geq m(N_C b + (N_C C + C)\frac{N_C}{2})$. Then $Z_D^* \geq mN_C^2 C + b + C$ since one 'D'-job starts after $b + C$. Using the other lower bounds we get
  $$Z^* - Z^+ \geq b + C - mb - \frac{1}{4}mb > 0,$$
  using that $C = 8mb$.
  - At least one such 'C'-job is scheduled *after* a 'D'-job on the same machine. Let i be the resource of one such 'C'-job and D_i the corresponding 'D'-job. Then the 'C'-job finishes at $N_C^2 C + C$ or later, while any 'C'-job scheduled *not* after a D'-job would have a maximum completion time of $\sum_{j=1}^{3m} a_j + b + N_C C$, which is the sum of all processing times of jobs that could possibly be scheduled on machine i. Hence $Z_C^* \geq m(N_C b + (N_C C + C)\frac{N_C}{2}) + (N_c^2 C + C - (mb + b + N_C C))$. Using the lower bounds for Z_r^*, Z_a^* and Z_D^*, we get
  $$Z^* - Z^+ = N_C^2 C + C - (mb + b + N_C C) - mb - \frac{1}{4}mb > 0,$$
  using that $C = 8mb$ and $N_C = 2mb$.
- *All 'C'-jobs are scheduled after the r_i job with the same resource, all 'C'-jobs are scheduled on the first m machines, but at least one 'C'-job is scheduled before an a_j job on the same machine.* All 'C'-jobs are scheduled after b, hence $Z_C^* \geq m(N_C b + (C + N_C C)\frac{N_C}{2})$. At least one a_j has a completion time larger than $C + b$, while any a_j *not* scheduled after an 'C'-job has a completion time smaller than or equal to $\sum_{j=1}^{3m} a_j + b = mb + b$, so $Z_a^* \geq \frac{7}{4}mb + (C + b - mb - b)$. Using the lower bounds for Z_r^* and Z_D^*, we get
$$Z^* - Z^+ \geq (C - mb) - mb - \frac{1}{4}mb > 0$$
using that $C = 8mb$.
- *All 'C'-jobs are scheduled after the r_i job with the same resource, all 'C'-jobs are scheduled on the first m machines and all 'C'-jobs are scheduled after the a_j jobs on the same machine.* Notice that the feasible solution in Figure 6.8 is structured in a similar way. At least one machine i should have an a_j job with completion time at least $b + 1$. So the sequence of 'C'-jobs on machine i should start at $b + 1$ or later. This means that $Z_C^* \geq m\left(N_C b + (C + N_C C)\frac{N_C}{2}\right) + N_C$. Using the lower bounds on Z_r^*, Z_a^* and Z_D^*, we get:
$$Z^* - Z^+ \geq N_c - mb - \frac{1}{4}mb > 0,$$

using that $N_C = 2mb$.

So if the 3-PARTITION instance is a no-instance, $Z^* > Z^+$, hence the reduction is complete. □

We can use Theorem 6.11 to show that our original problem with unrelated machines instead of parallel machines is $\mathcal{NP}$-hard. This problem is actually the problem found in the photolithography bays of European semiconductor factories [1].

Corollary 6.12. *$R|partition|\sum_j C_j$ is $\mathcal{NP}$-hard in the strong sense.*

Proof. We can reduce any decision variant instance I_P of $P|partition, \mathcal{M}_r|\sum_j C_j$, asking whether there exists a feasible solution with total completion time smaller than T, to a decision variant instance I_R of $R|partition|\sum_j C_j$ asking the same question. This is done by simply removing the processing set restrictions for resources and changing the processing times to:

$$p_{ij} = \begin{cases} p_j & \text{if } i \in \mathcal{M}_{r(j)} \\ T & \text{if } i \notin \mathcal{M}_{r(j)} \end{cases}$$

where $\mathcal{M}_{r(j)}$ denotes the machine restriction for $r(j)$, the resource of job j. Clearly, any feasible schedule for I_P is also a feasible schedule for the mapped instance I_R with the same total completion time. Hence if we have a yes-instance for I_P, we also have a yes-instance for I_R. However, if we have a no instance for I_P, all feasible solutions for I_P have a total completion time at least T. This means that all schedules for I_R processing only j on $i \in \mathcal{M}_{r(j)}$ for all j, also have a total completion time that is at least equal to T. However, any schedule processing at least one j on an $i \notin \mathcal{M}_{r(j)}$ also has a total completion time that is at least equal to T, because of such a job j. Hence I_R is also a no-instance. □

6.5 UNMOVABLE RESOURCES

Moving resources can be a costly operation. Thus one might also consider the case were the resources are also fixed on a machine. We therefore consider the problem where every resource can only be used on one machine. We define *unmovable* as an addition to the *partition* constraint, where all jobs $j \in r^k$ have to be processed on the same machine.

Theorem 6.13. *$P|partition, unmovable, p_j = 1|\sum_j C_j$ is $\mathcal{NP}$-hard.*

Proof. We give a polynomial time reduction from the 3-Partition problem as defined in Definition 13. We introduce in $P|partition, unmovable, p_j = 1|\sum_j C_j$ m machines and a number of jobs equal to $n = \sum_{a \in A} a$ and a number of resources equal to $3m$, where M is a large number. For every element of $a \in A$, we associate a number of jobs equal to a sharing the same resource. If we have a yes-instance of 3-Partition, then, $\forall i \in \{1, \ldots, m\}$, we can schedule all jobs associated with $a \in A_i$ to machine i. All machines will then be busy processing until time b. This will give us an objective value of $\frac{m}{2} b(b+1)$. If we have a no-instance of 3-Partition, we cannot distribute the jobs evenly over the machines and

thus the objective value will be greater than $\frac{m}{2}b(b+1)$. Hence, there exists a solution to 3-Partition if and only if $P|partition, unmovable, p_j = 1|\sum_j C_j$ as constructed above has an objective value of $\frac{1}{2}mb(b+1)$.
Note that, one might have a yes-instance of $P|partition, unmovable, p_j = 1|\sum_j C_j$, where on one machine there are 4 resources being used. If this is the case, these resources all have $b/4$ associated jobs and since it is a yes-instance, there also must be a machine using only 2 resources with $b/2$ associated jobs. An easy switch of the last $b/2$ units of processing of these two machines, will also lead to a yes-instance for 3-Partition. □

6.6 TWO RESOURCES PER JOB

Because the problem was motivated by the scheduling problem found in the photolithography bays of the semi-conductor industry, we are mainly interested in the case that there is only one resource per job. However, one might also wonder what happens if there is more than one resource needed per job. We will therefore analyze instances with at most q resources per job. We introduce $partition(q)$ for the β field of the scheduling problem. If $partition(q)$ is in the β field, there is a collection of subsets $R = \{r^1, \ldots, r^R\}$ with $r^k \subseteq J$, where every job is contained in at most q subsets. Let $r_j = \{r^k \in R \mid j \in r^k\}$, i.e., all subsets that contain job j. If two jobs share the same resource, we will denote this by $r_j = r_{j'}$, which implies that $r_j \cap r_{j'} \neq \emptyset$ and that j and j' cannot be processed at the same time.
The problem $P|partition(q), p_j = 1|\sum_j C_j$ is a special case of $P|res\cdots, types = \mathcal{R}, p_i < p|f$ with $f \in \{\sum_j w_j C_j, \sum_j T_j, \sum_j U_j\}$. Here, s is the number of resources and there are $\mathcal{R}$ types of jobs. A type of a job j is defined as the tuple $(p_j, \mathcal{R}_1(j), \ldots, \mathcal{R}_s(j))$, where $\mathcal{R}_u(j)$ is the amount of resource u required by job j. Note that, in our case, $|R| = \mathcal{R} = s$. [3] show that it can be solved in $O(\mathcal{R}(p+s)n^{\mathcal{R}p} + \mathcal{R}^2 pn^{\mathcal{R}(p+2)})$, resulting in the following corollary.

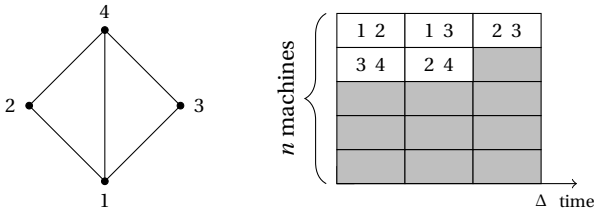
Corollary 6.14. *$P|partition(q), p_j = 1|\sum_j C_j$ is polynomially solvable for every $q \in \mathbb{Z}$, if the number of resources, $|R|$, is bounded.*

However, we will now show that the problem becomes NP-hard when the number of resources is not bounded, even with $q = 2$.

Theorem 6.15. *$P|partition(q), p_j = 1|\sum_j C_j$ is NP-hard for every $q \geq 2$, if the number of machines m and resources $|R|$ are unbounded.*

Proof. We will prove this by a reduction from edge coloring. In the edge coloring problem, one assigns colors (or labels) to the edges of a graph $G = (V, E)$, such that no two incident edges have the same color. Let Δ be the maximum node degree in the graph G, then [9] shows that it is NP-hard to decide for an arbitrary graph G whether or not it can be colored using only Δ colors.
We can reduce this problem to $P|partition(2), p_j = 1|\sum_j C_j$ as follows. Suppose we are given a graph $G = (V, E)$ with $|E| = m$. Take the number of machines equal to m. Introduce a resource for every node $u \in V$, $|R| = |V|$. We also introduce a job (with $p_j = 1$) for

every edge $e = \{u, v\}$ and these jobs require the resources that are associated with nodes it connects (i.e. u and v). Thus every resource will be used at most Δ times and every job uses exactly 2 resources. Lastly, we introduce $(\Delta - 1)n$ dummy jobs (with $p_j = 1$), that do no require a resource. Figure 6.9 shows an example of the reduction.



(a) Graph $G = (V, E)$ (b) Optimal schedule for the instance.

Figure 6.9: Example of the reduction from edge coloring to $P|partition(2), p_j = 1|\sum_j C_j$ with a graph with $\Delta = 3$. Gray colored jobs represent dummy jobs and numbers represent the resources.

We claim that there exists an edge coloring of graph G using only Δ colors if and only if the instance of $P|partition(2), p_j = 1|\sum_j C_j$ has an optimal value of $\frac{1}{2}\Delta(\Delta+1)m$. Suppose we are given a solution to the edge coloring problem which uses Δ colors, then we can put each color on a different time slot. So all jobs associated with an edge of the first color will be put on machines in the first time slot. We fill up all unused machine time until time Δ with dummy jobs. Since jobs only share a resource if they were incident in the graph, we will not have any resource conflict and all machines will be filled with jobs until time Δ, thus resulting in an objective value of $\frac{1}{2}\Delta(\Delta+1)m$.
Suppose we have a solution of the above instance of $P|partition(2), p_j = 1|\sum_j C_j$ with objective value $\frac{1}{2}\Delta(\Delta+1)m$. Then in each time slot we look for the jobs associated with a node and give them the same color. Since the objective value is $\frac{1}{2}\Delta(\Delta+1)m$ there are no jobs after time Δ, hence there are only Δ colors. Since all jobs associated with incident edges share a resource, no two incident edges will share the same color and hence we have found an edge coloring using Δ colors. □

Exercise 1:
Show using a reduction from vertex 3-colorability with maximum node degree 4, that $P|partition(4), p_j = 1|\sum_j C_j$ is $\mathcal{NP}$-hard for every $q \geq 4$, if the number of machines m and resources $|R|$ are unbounded.

6.7 CONCLUSION

In this chapter, we considered the scheduling problem of minimizing the total completion time on parallel machines when each job uses exactly one resource, $P|partition|\sum_j C_j$. Although the complexity of $P|partition|\sum_j C_j$ is open, we proved that similar problems such as $P|partition, \mathcal{M}_r|\sum_j C_j$, $P|partition(2), p_j = 1|\sum_j C_j$ and $P|partition, unmovable, p_j = 1|\sum_j C_j$ are $\mathcal{NP}$-hard. Therefore, we conjecture that $P|partition|\sum_j C_j$

is $\mathcal{NP}$-hard as well.
The problem $P|partition|\sum_j C_j$ always has an optimal solution where jobs sharing the same resource are ordered in non-decreasing order of their processing time. Such an optimal solution might even be more structured. For example, it remains open whether or not there is always an optimal solution that yields the SPT order property on each machine for all jobs on that machine.
We showed that the SPT-*available* rule gives a $\left(2-\frac{1}{m}\right)$-approximation. This bound may not be tight. There is a lower bound of $\frac{4}{3}$ on the approximation factor. Closing this gap is another interesting open problem, as well as designing other approximation algorithms with better approximation ratios.

REFERENCES

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

[2] J. Blazewicz, J. Lenstra, and A. R. Kan. Scheduling subject to resource constraints : Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

[3] P. Brucker and A. Krämer. Polynomial algorithms for resource-constrained and multiprocessor task scheduling problems. *European Journal of Operational Research*, 90(2):214–226, 1996.

[4] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31(1):146–166, 2001.

[5] R. Conway, W. Maxwell, and L. Miller. *Theory of scheduling*. Addison-Wesley, 1967.

[6] M. R. Garey and D. S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

[7] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to NP-completeness*. WH Freeman and Company, San Francisco, 1979.

[8] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[9] I. Holyer. The np-completeness of edge-coloring. *SIAM Journal on computing*, 10 (4):718–720, 1981.

[10] T. Janssen, C. Swennenhuis, A. Bitar, T. Bosman, D. Gijswijt, L. van Iersel, S. Dauzére-Pérès, and C. Yugma. Parallel machine scheduling with a single resource per job. arXiv:1809.05009, 2018.

[11] J. Y.-T. Leung and C.-L. Li. Scheduling with processing set restrictions: A survey. *International Journal of Production Economics*, 116(2):251–262, 2008.

[12] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

A

APPENDIX

A.1 EXERCISE 1

We will prove that $P|partition(q), p_j = 1|\sum_j C_j$ is $\mathcal{NP}$-hard for every $q \geq 4$, if the number of machines m and resources $|R|$ are unbounded, by a reduction to 3-colorability with node degree at most 4. This problem is shown to be $\mathcal{NP}$-hard by Garey et al. [2]. In this problem, we are given a graph $G = (V, E)$, where every node has at most 4 adjacent arcs. The goal is to find a labeling using only 3 different labels (i.e. colors), such that no two vertices who share an edge share the same label.
We can reduce this problem to $P|partition(4), p_j = 1|\sum_j C_j$ as follows. Suppose we are given graph $G = (V, E)$ with node degree at most 4, with $|V| = n$. Take the number of machines equal to n. Introduce a resource for every edge, $|R| = |E|$. We also introduce a job (with $p_j = 1$) for every node and these jobs require the resources that are associated with the edges adjacent to the node. Thus every resource will be used twice and every job uses at most 4 resources since the node degree is at most 4. Lastly we introduce $2n$ dummy jobs (with $p_j = 1$), that do no require a resource.
We have a 3-coloring of graph G if and only if the instance of $P|partition(4), p_j = 1|\sum_j C_j$ has an optimal value of $6n$. Suppose we are given a solution to 3-coloring, then we can put each label on a different time slot. So all jobs associated with a vertex of the first label will be put on machines in the first time slot. We fill up all unused machine time until time 3 up with dummy jobs. Since jobs only share a resource if they were adjacent in the graph, we will not have any resource conflict and all machines will be filled with jobs until time 3, thus resulting in an objective value of $6n$.
Suppose we have a solution of the above instance of $P|partition(4), p_j = 1|\sum_j C_j$ with $\sum_j C_j = 6n$. Then in each time slot we look for the jobs associated with a node and give them the same label. Since the objective value is $6n$ there are no jobs after time 3, this there are only 3 labels. Since all jobs associated with nodes share a resource with adjacent jobs, no two adjacent jobs will share the same label and hence we have found a three coloring.

A.2 RESULTS BLADE MOVEMENT OPTIMIZATION PILOT

We use the ILP formulation from Section 2.3 to optimize instances obtained from the semiconductor fab. The goal is to reduce the blade movement as to reduce the machine time needed for production. We obtained instances from the file that is used to define the images. We converted the data in these files to the needed tables with the blading positions and the image use per recipe step. Then, we used our algorithms to solve the problem instances.

In total, data was obtained for 46 (of 575) products. These 46 products account for 45.54% of the total work in process (WIP). The order of the images was optimized using the ILP solver. The results of the optimization can be found in Table A.2.

The construction of the ILP is done in Matlab and solved using SCIP[1]. Calculations are done using an Intel core i7-6700k CPU, 4.0 GHz with 16 GB of RAM.

Optimization problem (details per product)				Objective value (B_{PROD})			Performance
Product	WIP (%)	# images	# reticles	Original	Optimized	ΔB (%)	calc.time (sec)
MAA28	2.73	20	22	3124.62	3124.62	0	0.63
MAA04	2.40	20	25	3174.15	3174.15	0	0.17
MFS00	2.09	27	30	3253.83	3253.83	0	2.30
MAA01	1.97	18	24	3766.23	3517.08	6.62	0.78
MAA50	1.79	22	46	4271.1	3198.15	25.12	0.27
MAA29	1.76	20	22	3854.13	3298.6	14.41	0.45
SZS08	1.73	19	18	3968	3356.42	15.41	0.40
SQB01	1.55	30	14	5589.74	4640.15	16.99	2.32
MA936	1.45	23	49	5262.5	4374.2	16.88	0.61
SAC19	1.17	17	21	2677.03	2065.25	22.85	0.29
MAA45	1.15	19	28	6757.42	5392.8	20.19	0.85
MX028	1.12	26	48	6642.52	4339.3	34.67	0.73
RMP01	1.09	6	15	2589.9	2589.9	0	0.05
BGY00	1.06	7	22	3973.5	3973.5	0	0.01
MA924	1.06	22	29	6817	5582.1	18.12	0.45
XX055	1.06	26	46	4798.95	3922.35	18.27	0.99
DWB12	1.03	36	27	9177.64	6467.53	29.53	4.09
MAE00	1.01	16	18	3017.27	2403.75	20.33	0.49
MGY01	1.00	8	22	2848.63	2480.03	12.94	0.03
SZS05	0.87	18	21	3389.15	2646.79	21.90	0.19
XX040	0.87	24	62	4124.55	2644.89	35.87	0.79
MAA47	0.86	20	22	5551.7	4294.3	22.65	0.19
MX031	0.86	26	38	5323.07	3886.75	26.98	0.05
MAB00	0.81	22	42	3924.05	2536.6	35.36	0.52
XJ203	0.76	52	61	6104.29	3932.45	35.58	0.91
SAC03	0.74	19	17	2393.15	1991.68	16.78	0.05
DWB57	0.73	32	30	8827.86	6542.26	25.89	1.02
NKH02	0.67	19	29	7324.75	7294.33	0.42	5.98
MAA48	0.64	21	29	2373	2373	0	0.48
MAE03	0.64	20	18	3300.7	2298.57	30.36	0.26
XXW61	0.64	20	36	6632.37	5739.22	13.47	0.74
DWB14	0.63	36	27	9177.64	6467.53	29.53	4.05
XX056	0.63	23	56	6639.62	4333.05	34.74	0.82
MAA54	0.59	21	32	4662.35	3721.67	20.18	0.56
MAA62	0.59	16	23	3845.97	3105.01	19.27	0.77
MAA32	0.58	19	27	3857.08	3857.08	0	0.08
MYA00	0.58	26	22	5300.35	5300.35	0	0.24
MAB04	0.58	18	18	2938.18	2346.35	20.14	0.70
DWB06	0.56	36	36	9177.64	6467.53	29.53	4.07
SAB21	0.53	18	18	3419.88	2817.12	17.62	0.55
DWB40	0.53	36	31	9892.7	7550.32	23.68	14.98
MH021	0.53	19	28	3997.99	3997.99	0	0.07
MX032	0.50	19	30	4442.72	3261.03	26.60	0.36
MAA49	0.47	23	34	5424.43	4613.83	14.94	0.37
RKH05	0.47	17	27	8526.28	5302.28	37.81	1.38
MAA25	0.46	23	37	3726.45	3040.18	18.42	0.29
Total	45.54	1025	1377	229862.03	183515.83	-	56.37
Average	0.99	22.28	29.93	4997	3989.47	20.16	1.23

Table A.1: Blade movement optimization results for 46 products comparing the original blade movement against the now optimized distance

A.3 SPACEFILLING CURVE ALGORITHM WITH INSTANCE DEPENDENT CURVE LEVEL

In Section 4.4, we look at the performance of space filling curve algorithms, `Hilbert1` and `Hilbert24`, on problem instances found in the photolithography bay. We used them to minimize the blade movement for 46 reticle jobs. In the algorithms, we fixed the maximum level of the curve, $j = 8$, since 0.00714 is the minimum movement between two images after resizing over all problem instance. This however is only the minimum for three reticle job instances; MX028, XX055 and MX031. We could speed up the algorithm for some of the other instances by first calculating the required j and using this j for the algorithm. Table A.2 shows the results per instance for the different algorithms including the preprocessor that finds the required j. On average, it reduces the time of `Hilbert1` by 5.3% and `Hilbert24` by 6.0% (including the time required for the preprocessor).

The `Hilbert1` and `Hilbert24` algorithms are implemented in Matlab. Calculations are done using an Intel core i7-6700k CPU, 4.0 GHz with 16 GB of RAM.

Instance			Preprocessor		Time (ms)				
Product	Images	Layers	$\min \Delta B$	j	Preproc.	`Hilb1`(8)	`Hilb24`(8)	`Hilb1`(j)	`Hilb24`(j)
MAA28	20	22	0.0896	4	2.9	41.9	61.9	29.3	55.5
MAA04	20	25	0.0896	4	1.2	5.3	56.4	5.5	51.5
MFS00	27	30	0.0219	6	0.4	6.0	95.5	5.6	90.0
MAA01	18	24	0.0836	4	0.3	5.0	52.1	3.7	44.5
MAA50	22	46	0.0896	4	0.4	4.8	69.5	4.4	60.9
MAA29	20	22	0.0896	4	0.3	4.1	56.5	3.9	50.5
SZS08	19	18	0.1230	4	0.3	4.2	53.2	3.7	47.4
SQB01	30	14	0.0575	5	0.5	7.0	115.3	6.7	114.8
MA936	23	49	0.0896	4	0.4	5.2	74.3	4.6	66.0
SAC19	17	21	0.0555	5	0.3	33.5	49.1	3.5	40.3
MAA45	19	28	0.0896	4	0.3	5.1	53.8	3.8	47.3
MX028	26	48	0.0071	8	0.4	5.9	91.9	5.6	96.4
RMP01	6	15	0.1086	4	0.3	2.4	10.8	2.3	9.0
BGY00	7	22	0.1099	4	0.3	2.4	12.1	2.3	10.8
MA924	22	29	0.0896	4	0.3	4.9	68.7	4.4	67.9
XX055	26	46	0.0071	8	0.4	5.7	91.7	5.8	92.6
DWB12	36	27	0.0732	4	0.5	8.5	163.0	8.0	151.6
MAE00	16	18	0.0896	4	0.3	3.5	41.0	3.5	38.0
MGY01	8	22	0.0774	4	0.3	2.5	15.7	2.4	13.4
SZS05	18	21	0.1230	4	0.3	3.9	48.8	3.9	44.1
XX040	24	62	0.0071	8	0.4	5.1	78.8	5.0	76.5
MAA47	20	22	0.0896	4	0.3	4.2	58.1	4.0	51.3
MX031	26	38	0.0071	8	0.4	5.6	89.0	5.7	86.8
MAB00	22	42	0.0896	4	0.4	4.8	73.0	4.5	60.7
XJ203	52	61	0.0457	5	0.6	15.8	325.0	14.8	305.2
SAC03	19	17	0.0179	6	0.4	4.1	52.6	4.1	51.5
DWB57	32	30	0.0732	4	0.5	7.1	129.9	7.5	123.1
NKH02	19	29	0.0814	4	0.3	4.5	55.1	3.8	47.9
MAA48	21	29	0.0687	4	0.3	4.5	61.3	4.1	57.8
MAE03	20	18	0.0418	5	0.3	4.5	56.8	4.0	51.9
XXW61	20	36	0.0896	4	0.3	4.3	58.1	3.9	51.0
DWB14	36	27	0.0732	4	0.5	8.8	162.3	8.0	150.4
XX056	23	56	0.0071	8	0.4	5.3	76.6	5.0	73.3
MAA54	21	32	0.0896	4	0.3	4.5	61.2	4.1	56.1
MAA62	16	23	0.0747	4	0.3	3.5	41.6	3.3	35.9
MAA32	19	27	0.0896	4	0.3	4.2	54.2	3.8	48.3
MYA00	26	22	0.0732	4	0.4	5.9	89.8	5.3	87.2
MAB04	18	18	0.0616	5	0.3	4.1	48.9	4.0	45.9
DWB06	36	36	0.0732	4	0.5	8.5	161.4	8.9	152.9
SAB21	18	18	0.0896	4	0.3	3.8	48.8	3.7	43.3
DWB40	36	31	0.0732	4	0.5	8.5	162.5	8.1	150.3
MH021	19	28	0.0896	4	0.3	4.2	52.5	3.7	47.2
MX032	19	30	0.0896	4	0.3	4.3	53.5	3.7	46.8
MAA49	23	34	0.0896	4	0.4	4.8	74.6	4.6	68.6
RKH05	17	27	0.0962	4	0.4	3.6	44.0	3.7	40.8
MAA25	23	37	0.0896	4	0.4	5.2	72.5	4.5	66.0
Average	22.3	29.9	0.0071	8	0.9	6.6	76.6	5.4	71.1

Table A.2: Time required for blade movement minimization by the spacefilling curve algorithms. $\min \Delta B$ *is the minimum distance between points and* j *is the associated maximum level required such that every point is contained in its own subhypercube. The time required is shown for the preprocessor,* `Hilbert1` *&* `Hilbert24` *with* $j = 8$ *fixed and* `Hilbert1` *&* `Hilbert24` *with* j *depending on the required* j*.*

REFERENCES

[1] T. Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. http://mpc.zib.de/index.php/MPC/article/view/4.

[2] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976.

ACKNOWLEDGEMENTS

Although its my name on the cover, this thesis wouldn't not have been possible without the help of others. I will therefore make a feeble attempt at thanking everyone who has helped, inspired or influenced me in the past five years. I know this is not an exhaustive list and conjecture that if this were the case, it would be longer than the credits of The Lord of the Rings movies.

First of all, I would like to thank my promotor Karen, my copromotor Leo and my NXP-supervisor Jan. You have been an excellent team of supervisors, each with your very own knowledge and style. You gave me the opportunity to grow and learn as a researcher and as a person. I could not have done this without your support. Karen, thank you for having trust in my potential as a PhD candidate. You made me feel at home in the optimization group and were always had my back, which I very much appreciated. Leo, thank you for all your patience and support. Your door was always open for a quick talk or a long blackboard discussion. Although I did not know a priori, I am very glad you joined my supervisory team. Jan, thank you for introducing me to the wondrous world of semiconductor manufacturing. Your analytic look at the Fab gave me my research topics and an opportunity to really apply my research. Next to being a superb research partner, you also helped me to improve and structure my writing as well as help me understand the importance of visual aids.

During my PhD I had the pleasure of working with many talented researchers. I especially want to thank my co-authors who contributed to the chapters in this thesis. Abdel, Claude and Stéfane, you welcomed me in Gardanne and helped me get up to speed on the state of the art of scheduling in semiconductor manufacturing. You, Jacky and the other EMSE-CMP staff made my four stays there informative, inspiring and fruitful, while at the same time make it feel a little bit like a vacation.
Céline, Dion, Martijn, Rene and Thomas, I would like to thank you for the many invigorating discussions that let to such wonderful research and for the relaxing moments we had at conferences with 'prima pils'.

Also a word of gratitude to the (remaining) Integrate partners, in particular Barri, Bas, Jan-Eite, Leon & Martijn, for all the help and insight in developing working Fab solutions. Together we made Integrate into a successful project, which we can be proud of.

Also a special word of thanks to the other members of the optimization group at the TU Delft for creating a stimulating and supportive daily work environment. I would like to thank my office mates, Pieter & Theresia and later Jacobo, Remy, Ren Fei & Yuki. Our discussions on both research and life were the perfect supplement to doing research. You especially made my PhD feel less stressful and I promise I will attend a boardgame evening in the near future.

I would like to thank Sarah for the many enjoyable discussions on PhD life during our Graduate School courses and for the the beautiful cover art made on such short notice. Furthermore a word of thanks for Ambi, Jarno & Merel, for proofreading parts of this thesis.

I would also like to thank my friends and family for the support they gave me and for keeping me in good health both mentally and physically.
Jan & Josephine, als ik meerdere dagen in Nijmegen moest zijn, gaven jullie mij onderdak. Het was heel fijn om bij jullie te mogen eten, slapen en genieten van de gezelligheid. Lieve schoonfamilie, bedankt voor alle hulp bij nevenzaken en voor alle bemoedigende woorden.
Davy, Jacob, Jarno, Mark, Menk, Olivier, Rens, Yannick & WP, jullie waren er altijd om mij te steunen, om naar mijn onderzoeksverhalen te luisteren en om een overvloed aan afleiding te verstrekken. Jullie tomenloze enthousiasme is aanstekelijk en ik voel me vereerd dat twee van jullie mij ook tijdens de verdediging bijstaan.
De basis van deze thesis ligt bij de familie Janssen aan de keukentafel, waar ik mijn eerste stappen als doctor in spe mocht zetten. Marieke, bedankt voor de grote betrokkenheid die je altijd hebt bij alle dingen die ik doe en onze discussies die alles in perspectief plaatsen. Reinier, ik mocht de afgelopen vijf jaar bij jou afkijken hoe ik een PhD moest doen. Bedankt voor alle goede tips en bedankt dat je mijn buddy wilde zijn op het (slag)veld. Ik ga zeker onze werkoverleggen op de TU missen waarin we vooral bezig waren met de nieuwste legerlijst. Papa, ondanks dat je dit niet mee hebt mogen maken, ben je toch van grote invloed geweest op deze thesis. Bedankt voor alle mooie momenten en dat je aan mij je interesse voor wiskunde hebt overgedragen. Mama, toen ik je vertelde dat ik een PhD wilde doen, reageerde je super enthousiast en dat enthousiasme is nooit weggegaan. Je hebt mij de hele weg gesteund en was altijd beschikbaar voor al mijn vragen en zorgen. Ik wil je bedanken dat je nog steeds een thuishaven voor mij bent. Steffy, ik ben heel dankbaar voor jouw advies, liefde en steun. Je gaf mij rust en een basis waarop ik kon terugvallen, zodat ik (samen met jou) kon groeien.

Teun Janssen
Delft, February 2019

CURRICULUM VITÆ

Teun Michiel Louis JANSSEN

29-11-1987	Born in Delft, the Netherlands.
2000 - 2006	Grammar School St. Stanislas College, Delft
2006 - 2012	B. Sc. Applied Mathematics Delft University of Technology *Bachelor Thesis:* Geoptimaliseerde Chaos (Optimized Chaos) *Supervisor:* Dr. R. J. Fokkink.
2012-2013	M. Sc. Applied Mathematics Delft University of Technology *Master Thesis:* Noise minimization on houses around airports *Supervisor:* Prof. G. Schäfer
2013	Intern at the National Aerospace Laboratory of the Netherlands (NLR)
2013	Intern at the National Research Center for Mathematics and Computer Science in the Netherlands (CWI)
2014-2018	Ph. D. Research Delft University of Technology *Thesis:* Optimization in the Photolithography Bay: Scheduling and the Traveling Salesman Problem *Promotor:* Prof. dr. ir. K. Aardal *Copromotor:* Dr. ir. L.J.J. van Iersel
2018 -	Post Doctoral Researcher VU University Amsterdam

LIST OF PUBLICATIONS

T. Janssen, C. Swennenhuis, A. Bitar, T. Bosman, D. Gijswijt, L. van Iersel, S. Dauzère-Pérès, C. Yugma. *Parallel Machine Scheduling with a Single Resource per Job*, `arXiv:1809.05009v1`

N. Hesam Mahmoudi Nezhad, M. Ghaffarian Niasar, A. Mohammadi Gheidari, T. Janssen, C.W. Hagen, P. Kruit. *Multi-electrode Lens System Optimization Using Genetic Algorithms*, under review

M. van Ee, L. van Iersel, T. Janssen, R. Sitters. *A priori TSP in the Scenario Model*. Discrete Applied Mathematics, 2018.

N. Hesam Mahmoudi Nezhad, M. Ghaffarian Niasar, A. Mohammadi Gheidari, T. Janssen, C. Hagen, P. Kruit. *Optimization of Electrostatic Lens Systems Using Genetic Algorithms*. Proceedings of the 16th International Seminar on Recent Trends in Charged Particle Optics and Surface Physics Instrumentation, 2018.